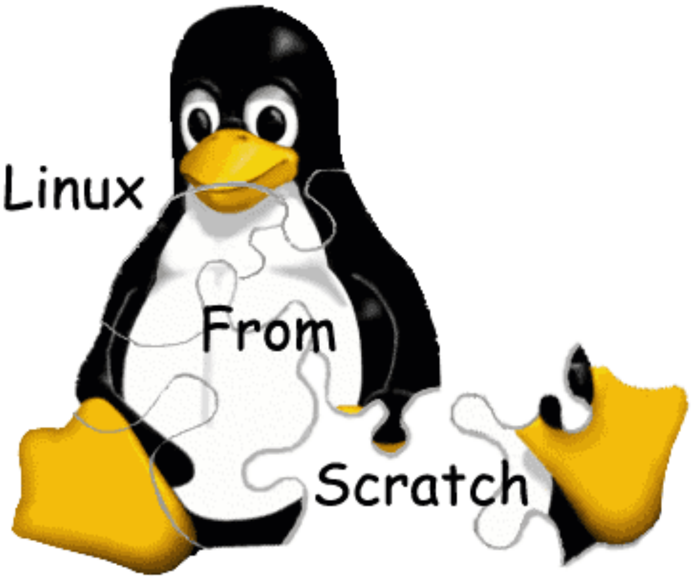

Linux From Scratch HOWTO

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Linux From Scratch HOWTO

Gerard Beekmans

Version 2.0−beta2

This document describes the process of creating your own Linux system from scratch from an already installed Linux distribution, using nothing but the source code of software that we need

---

1. Introduction

1.1 What's this all about?

I started this document around May 1999. I tried a few Linux distributions and came to the conclusion that there's wasn't a distribution I totally liked. Every distribution has it's own advantages and disadvantages, but I was never satisfied with what I had (although Debian comes very close to what I want), so I decided to explore the possibility of building my own Linux distribution using nothing but source code of programs. As I found out there's quite a bit of work involved, but it's also a lot of fun and you really learn a lot by doing it, since you need to configure every single aspect of the system. This forces you to read a lot of manuals on how to configure various software. It also gives you total control over your system (well, that's the idea). You know exactly what software is installed, how it is configured and where all the configuration files reside.

I started writing a series of articles for a Dutch/Belgium E−zine on this subject. Not soon after I got stuck getting a compiler to work. I decided to give this project a rest at that point, since a lot of things at that time needed my attention (I was about to move from The Netherlands to Canada to get married. There were a lot of things to arrange regarding the move abroad and a lot of immigration stuff to sort out).

A few months after my arrival in Canada and getting married, I decided to continue my work on this project. Pretty much starting all over again from scratch and following a different approach, I got things to work out finally. The end result is what you are reading right now.

1.2 New versions

The latest version of the document can always be found at http://huizen.dds.nl/~glb/

1.3 Version history

2.0−beta2 − Unknown date, sorry folks

- New Glibc and compiler installation method
- Totally revised software installation method − eliminated the need of all the statically linked packages in former chapter 6.1.
- Various fixed in software installation
- Installed a few more programs from the util−linux package
- Explained in greater detail what the $LFS is all about − how to and how not to use it.
- Section 5.2: Fixed reversed symlink (was ln −s dest. source in stead of ln −s source dest)

1.3 − February 11th, 2000

- Two mailinglists are available. Read section 1.6 for more details
- Changed the compiler setup. Gcc−2.95.2 no longer is being used. In stead gcc−2.7.2.3 is the new C compiler and egcs−2.91.60 is the C++ compiler.
- Updated sections that contained compile instructions by running make CC=/usr/gcc2723/bin/gcc. A simple 'make' suffices now since gcc−2.7.2.3 is our default C compiler now.
- Changed the 'abstract' line to be more accurate.
- Fixed typos that were left behind in the previous versions
- Moved section 1.4 (TODO) to section 1.5.
- Section 2: Added the version numbers of the software that are known to work with this document.
- Section 2: Mawk link was broken (thanks to David McCauley (and various other people) for informing me about this).
- Section 2: Sysklogd link was broken (thanks to David McCauley for informing me about this).
- Section 2: divided the list into mandatory and optional software (the separation is software for section 13 and above)
- Inserted new section 1.4: Current projects.
- Inserted new section 3.1: How we are going to do things. In this section I briefly explain that you need to already have Linux installed to use this HOWTO and also explained there is no need for any kind of boot disk.
- Section 3.3: Clarified the currently used kernel image is to be used (thanks to Andrew Blais for pointing this out).
- Section 3.4: Added the usr/share directory to the list of directories that need to be created.
- Section 5: Clarified that the kernel source tree must be copied to the LFS partition
- Section 6.1.2: Pointed out availability of fixed package in case compilation fails
- Section 6.1.4: Rather than renaming ginstall to install we create a symlink install
- Section 6.1.8: Pointed out availability of fixed package in case compilation fails
- Section 6.1.10: Pointed out availability of fixed package in case compilation fails
- Section 6.1.13: Pointed out availability of fixed package in case compilation fails
- Section 6.1.14: Pointed out availability of fixed package in case compilation fails
- Section 6.1.17: Pointed out availability of fixed package in case compilation fails
- Section 6.1.18: Pointed out availability of fixed package in case compilation fails
- Section 7.2: The gcc−2.7.2.3 compiler needs to be linked statically at first (to avoid possible Library conflicts between the normal and LFS system).
- Inserted section 11.1: Reinstalling GCC 2.7.2.3
- Section 11.13: Pointed out availability of fixed package in case compilation fails

- Section 11.19: Pointed out availability of fixed package in case compilation fails
- Section 14.4: Failed to mention that the package needs to be configured prior to compilation.
- Section 15.2: Failed to mention that the package needs to be configured prior to compilation.
- Inserted a new section 19 (old section 19 has become section 20) that contains migrating information, in case you need to do some re−modeling to change a setup (like migrating to the new compiler setup in this version).

1.2 – January 9th, 2000

- Section 2: Owen Cook pointed out that the link for the sysvinit package was wrong. It said cistron.nl. It should be ftp.cistron.nl
- Section 3.4: Added the usr/include directory to the list of directories that need to be created
- Section 4.3: Made a notion of the possibility that somebody's system might be using shadowed passwords.
- Section 6.1.3: The majority of the files that need to be copied was missing (the files that need to be copied to $LFS/usr/bin).
- Section 6.1.4: Forgot to mention that the mv program needs to be copied as well
- Section 6.1.14: Forgot to mention that the cmp program needs to be copied as well
- Section 7: Just to make sure nobody runs into problems, I added the comment that all file systems must be unmounted and the root file system must be mounted read−only before the computer is rebooted
- Section 7.2: Added the −−local−prefix=/usr/gcc2723 switch to the configure command line
- Section 11.7: Fixed a typo in one of the programs: patchchk should be pathchk
- Section 11.9: Added compilation and copying of the mkswap program

1.1 – December 20th, 1999

- Fixed a few typos
- Modified section 18 (Configuring system for Internet) from just a reference to the ISP−Hookup−HOWTO to a basic explanation on how to setup Internet
- Fixed error in /etc/syslog.conf (in section 12.8.2)

1.0 – December 16th, 1999

- Initial release

1.4 Mailinglists

There are two mailinglists you can subscribe to. The lfs−discuss and the lfs−announce list. The former is an open non−moderated list discussing anything that has got anything to do with this HOWTO (asking questions, inform about mistakes in this HOWTO and so on). The latter is an open moderated list. Anybody can subscribe to it, but you cannot post messages to it (only the moderator(s) can). This list is primarily used for announcements of new versions of the HOWTO.

If you're subscribed to the lfs−discuss list you don't need to be subscribed to the lfs−announce list as well. Everything that is sent over the lfs−announce list is also sent over the lfs−discuss list.

Subscribing

To subscribe to a list, send an email to majordomo@fist.org and type in the body either *subscribe lfs−discuss* or *subscribe lfs−announce*

Majordomo will send you a confirmation−request email. This email will contain an authentication code. Once you send this email back to Majordomo (instructions are provided in that email) you will be subscribed.

Unsubscribing

To unsubscribe from a list, send an email to majordomo@fist.org and type in the the body either *unsubscribe lfs−discuss* or *unsubscribe lfs−announce*

1.5 Contact info

Direct all your questions preferably to the mailinglist. If you need to reach me personally, you can reach me, Gerard Beekmans, at tts−sol@dds.nl

2. Conventions used in this HOWTO

2.1 About $LFS

Please read the following very carefully: throughout this document you will very frequently see the variable name $LFS. $LFS must at all times be replaced by the directory where the partition that will contain the LFS system is mounted on. The creating and where to mount the partition will be explained in full detail in chatper 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 ;)

2.2 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.

2.3 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, you best install it using your distribution's installation tool.

- Start by *copying* the package from wherever you have stored it to the *$LFS/usr/src* directory
- When you have a file that is tar'ed and gzip'ed, you unpack it by running: `tar xvfz filename.tar.gz; rm filename.tar.gz` or `tar xvfz filename.tgz; rm filename.tgz`
- 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; rm filename.tar.bz2`
- When you have a file that is only tar'ed, you unpack it by running `tar xvf filename.tar; rm filename.tar`

Note that immediately after we have unpacked the archive, we delete the packed file as we don't need it anymore. That's why you have to copy the file and not move it. If you move it and then delete it, you will need to re−download it when you need it again.

When the archive is unpacked a new directory will be created under the current directory (and this howto 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 sumarized as 'Unpack the xxx archive'. So, when you read it, you copy the package to $LFS/usr/src, 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.

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.

3.1 Mandatory software

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/

Make (3.77) : 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/

File Utils (4.0) : ftp://ftp.gnu.org/gnu/fileutils/

Util Linux (2.9z) : ftp://ftp.win.tue.nl/pub/linux/utils/util−linux/

Text Utils (1.22) : ftp://ftp.gnu.org/gnu/textutils/

Tar (1.12) : ftp://ftp.gnu.org/gnu/tar/

Gzip (1.2.4) : ftp://ftp.gnu.org/gnu/gzip/

Binutils (2.9.1) : ftp://ftp.gnu.org/gnu/binutils/

Grep (2.2) : ftp://ftp.gnu.org/gnu/grep/

Bison (1.25) : 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/

Diff Utils (2.7) : ftp://ftp.gnu.org/gnu/diffutils/

Ld.so (1.9.10) : ftp://tsx−11.mit.edu/pub/linux/packages/GCC/

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/

Automake (1.3) : ftp://ftp.gnu.org/gnu/automake/

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

Glibc (2.0.7pre6) : ftp://ftp.gwdg.de/pub/linux/glibc/2.0.7pre6/

Glibc−crypt (2.0.pre6) : ftp://ftp.gwdg.de/pub/linux/glibc/2.0.7pre6/

Glibc−linuxthreads (2.0.7pre6) : ftp://ftp.gwdg.de/pub/linux/glibc/2.0.7pre6/

Libstdc++−2.91.60 : ftp://ftp.debian.org/debian/dists/slink/main/binary−i386/base/

Libstdc++−2.91.660−dev : ftp://ftp.debian.org/debian/dists/slink/main/binary−i386/devel/

GCC (2.7.2.3) : ftp://ftp.gnu.org/gnu/gcc/

G++−2.91.60 : ftp://ftp.debian.org/debian/dists/slink/main/binary−i386/devel/

Ncurses (4.2) : ftp://ftp.gnu.org/gnu/ncurses/

Vim (5.5) : ftp://ftp.vim.org/pub/editors/vim/

Readline Library (4.0) : ftp://ftp.gnu.org/gnu/readline/

Termcap Library (1.3) : ftp://ftp.gnu.org/gnu/termcap/

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

Less (332) : ftp://ftp.gnu.org/gnu/less/

E2fsprogs (1.12) : ftp://tsx−11.mit.edu/pub/linux/packages/ext2fs/

File (3.26) : ftp://ftp.debian.org/debian/dists/slink/main/source/utils/

Libtool (1.2) : ftp://ftp.gnu.org/gnu/libtool/

Modutils (2.3.7) : ftp://ftp.ocs.com.au/pub/modutils/

Linux86 (0.14.3) : ftp://ftp.debian.org/debian/dists/slink/main/source/devel

Lilo (21) : ftp://sunsite.unc.edu/pub/Linux/system/boot/lilo/

DPKG (1.4.0.35) : ftp://ftp.debian.org/debian/dists/slink/main/source/base/

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

Groff (1.11.1) : ftp://ftp.gnu.org/gnu/groff/

Man−db (2.3.10) : ftp://ftp.debian.org/debian/dists/slink/main/source/doc/

Procps (2.0.6) : ftp://tsx−11.mit.edu/pub/linux/sources/usr.bin/

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

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

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

3.2 Optional software

All software below is used in sections 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 : ftp://ftp.uk.linux.org/pub/linux/Networking/netkit−devel/

Net−tools (1.53) : http://www.tazenda.demon.co.uk/phil/net−tools/

Procmail (3.13.1) : ftp://ftp.procmail.org/pub/procmail/

Sendmail (8.9.3) : ftp://ftp.sendmail.org/pub/sendmail/

Mailx (8.1.1) : ftp://ftp.debian.org/debian/dists/slink/main/source/mail/

Mutt (1.0i) : ftp://ftp.mutt.org/pub/mutt/

Fetchmail (5.2.0) : http://www.tuxedo.org/~esr/fetchmail/

Netkit−telnet : ftp://ftp.uk.linux.org/pub/linux/Networking/netkit−devel/

Proftpd (1.2.0pre9) : ftp://ftp.tos.net/pub/proftpd/

Netkit−ftp : ftp://ftp.uk.linux.org/pub/linux/Networking/netkit−devel/

Apache (1.3.3) : http://www.apache.org/dist/

Slang Library (1.3.10) : ftp://space.mit.edu/pub/davis/slang/

Zlib Library (1.1.3) : http://www.cdrom.com/pub/infozip/zlib/

Lynx (2.8.1) : http://www.slcc.edu/lynx/release/

Xfree86 (3.3.2.3) : 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/

PPP (2.3.10) : ftp://cs.anu.edu.au/pub/software/ppp/

4. Preparing the new system

4.1 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 HOWTO 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).

4.2 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 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 parition's designation is. It could be something like hda5 (as it is in my case). This newly created partition will be refered to as the *LFS partition* in this document.

4.3 Creating an 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`

4.4 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 HOWTO 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 practise 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.

4.5 Creating directories

Let's create a minimal directory tree on the LFS partition. issuing the following commands will create the necessary directories.

```
cd $LFS
mkdir boot etc home mnt proc root tmp var usr
cd $LFS/usr
mkdir bin sbin src man include share lib
cd $LFS/man
mkdir man1 man2 man3 man4 man5 man6 man7 man8
cd $LFS/usr
ln -s . local
ln -s ../etc etc
ln -s ../var var
cd $LFS/usr/share
ln -s ../man man
cd $LFS
ln -s usr/lib lib
ln -s usr/bin bin
ln -s usr/sbin sbin
```

I am aware that a number of directories you have created above are in total violation with the FHS (File Hierarchy Standard – http://www.pathname.com/fhs/). The reason why I do this is just a preference. I want to keep certain files all together. For example the old standard was that man pages go in /usr/man and /usr/local/man. The most recent standard dictates that man pages should go in /usr/share/man (and possibly /usr/local/share/man). I just want them all to be in /usr/man so I know exactly in what directory a certain man page is and I don't have to start looking in various directories to find out where it is (although I can simply find a file with the 'locate' command I still prefer the way I do things).

If you want to create a file system that it completely according the FHS, then I urge you to take a look at www.pathname.com/fhs and create your directories accordingly.

4.6 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
```

Feel free to strip down the $LFS/dev directory, only leaving the devices you really need.

5. Making the LFS system bootable

5.1 Installing Sysvinit

Under normal circumstances, after the kernel's 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
- Edit the `Makefile` file
- Somewhere in this file, but before the rule *all:* put this line: *ROOT = $LFS*
- Precede every /dev on the last four lines in this file by *$(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 LDFLAGS=-static; make install
```

5.2 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:

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

~~: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
z6:6:wait:/sbin/sulogin

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

1:2345:respawn:/sbin/sulogin

# End /etc/inittab
```

5.3 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 string above is: *lfs123*

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

5.4 Installing a 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
cp bash $LFS/usr/bin
cd $LFS/usr/bin; ln -s bash sh
```

5.5 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=$LFS
   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.

Now run the `lilo` program to update the boot loader.

5.6 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. Please note that you will get errors regarding the init program not being able to start the rcS and rc scripts. Ignore those errors for now. It is normal. We will install these scripts in a later chapter.

Also note that 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 NSSlibrary (Name Server Switch) which is part of the GNU C Library, which also 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 ok).

6. Installing a kernel

6.1 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).

6.2 Configuring the kernel

- 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
- Unpack the Kernel archive in the `/usr/src/` directory (this will create a new /usr/src/linux directory)
- 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, run `make dep`
- Compile the kernel by running `make bzImage`
- Copy the `arch/<cpu>/boot/bzImage` file to the `/boot` directory (or some place else if your Linux system uses a different convention where kernel images and the like are stored)
- Optionally you can rename the `/boot/bzImage` file to something like `/boot/lfskernel`
- Copy the entire kernel source tree from 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
```

6.3 Updating LILO

- Edit the `/etc/lilo.conf` file and go to the LFS section
- Change the image name to `lfskernel` (or whatever you've named the originally called bzImage file)
- Run `lilo` to update the boot loader.

6.4 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).

7. Installing the GNU C and C++ Libraries

7.1 Installing the GNU C Library

We're not going to installed the latest Glibc version (2.1.2 at the time of writing), but version 2.0.7pre6. The reason is that glibc 2.1.2 requires at least gcc 2.8 (or egcs 1.1). My system has gcc 2.7.2.3 thus I can't compile the glibc2.1.2 library. A second reason is that I have used the 2.1.2 C library for a while and found out that certain software doens't run smoothly when linked against that library version. In the future we might undergo the transition to Glibc−2.1.x when we have been able to fix those programs. For now, we'll stick with the library and compiler in which I have most confidence.

A note on the glibc−crypt package. The following is quoted from the glibc−crypt−README file on ftp://ftp.gnu.org/gnu/glibc:

```
−*−*−*−*−*−
The add−on is not included in the main distribution of the GNU
C library because some governments, mostly notable 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 license, including via the Internet. So please do not
download it from the main FSF FTP site at ftp.gnu.org if you
are outside of 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/2.0.7pre6/. 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 it from that German site.

- Unpack the Glibc archive
- Copy the glibc−crypt and glibc−linuxthreads archives into the unpacked glibc directory
- Unpack the glibc−crypt and glibc−linuxthreads there, but don't enter these directories. Just ungzip and untar them.
- Configure the package by running `configure −−with−gnu−binutils −−enable−shared −−enable−add−ons=linuxthreads,crypt`
- Create a new file `configparms` containing the following:

```
# Begin configparms
prefix=/usr
slibdir=/lib
sysconfdir=/etc
localtime=localtime
# End configparms
```

You have to select the current timezone that you are in. What you need to enter here must be a file relative to the /usr/share/zoneinfo directory. You can determine what your timezone file is, by running `ls −l /etc/localtime`. The output of this command will tell you to what file this symlink is pointing. If your timezone file is /usr/share/zoneinfo/EST5EDT, you enter 'localtime=EST5EDT' in the configparms file. If your timezone file is /usr/share/zoneinfo/Canada/Eastern you enter 'localtime=Canada/Eastern' in the configparms file.

- Install the package by running:

```
make; make install_root=$LFS install
```

7.2 Installing the GNU C++ Library

This HOWTO used to install the C++ library from sources, but that has been changed and is an exception. The C++ library is installed from pre−compiled binaries. The reason is that I have not been able to find the sources for the C++ library version that I prefer to use. So until then we'll use pre−compiled binaries. My preference are Debian packages.

Installing the libstdc++2.9_2.91.66−0slink2.deb package

- Extract the archive files by running `ar x libstdc++2.9_2.91.66-0slink2.deb`
- Extract the package itself by running `tar xvfz data.tar.gz`
- Copy the package by running `cp -av usr $LFS`
- Remove the following files and directory: `usr data.tar.gz control.tar.gz debian-binary`

Installing the libstdc++2.9−dev_2.91.66−0slink2.deb package

- Extract the archive files by running `ar x libstdc++2.9-dev_2.91.66-0slink2.deb`
- Extract the package itself by running `tar xvfz data.tar.gz`
- Copy the package by running `cp -av usr $LFS`
- Remove the following files and directory: `usr data.tar.gz control.tar.gz debian-binary`

8. Installing the GNU C and C++ compilers

This HOWTO used to install the C++ compiler from source, but that has been changed and is an exception. The C++ compiler is installed using pre−compiled binaries. The reason is that I have not been able to find the sources for the C++ compiler version that I prefer to use. So until then we'll use pre−compiled binaries.

We also will link the C compiler statically. Although Glibc is installed on our LFS system, we still are compiling the compiler on our normal Linux system. The normal Linux system may contain a different version of Glibc and the compiler will be linked against that version. Therefore we will link the compiler statically and later on when all statically linked software is being re−installed we also will re−install the compiler. This procedure is not necessary if both your normal Linux system and the LFS system use the same Library version, but since I don't know that we will do it this way.

8.1 Installing GCC 2.7.2.3

- Unpack the gcc−2.7.2.3 archive
- Build the compiler by running:

```
./configure
make LANGUAGES=c
make stage1
make CC="stage1/xgcc -Bstage1/" CFLAGS="-g -O2" LDFLAGS=-static LANGUAGES=c
```

```
make stage2
make CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O2" LDFLAGS=-static LANGUAGES=c
make compare
```

Please note that you will see errors during the "make stage1" and "make stage2" processes regarding files not being found. This is perfectly ok, as these errors are c++/objc/g77 compiler related files. We specify that we only want to install the C compiler, so you'll get those errors.

- If make compare doesn't report any differences, the compiler is build successfully.
- Install the package by running `make CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O2" LDFLAGS="-static" LANGUAGES=c prefix=$LFS/usr local-prefix=$LFS/usr install`

8.2 Installing the g++_2.91.66−0slink2.deb package

- Extract the archive files by running `ar x g++_2.91.66-0slink2.deb`
- Extract the package itself by running `tar xvfz data.tar.gz`
- Copy the files by running `cp -av usr $LFS`
- Remove the following files and directory: `usr data.tar.gz control.tar.gz debian-binary`

Note that this g++ compiler is already pre−compiled and linked against glibc 2.0.7. Therefore there is no need to re−install this package in a later stage.

8.3 Creating necessary symlinks

The system needs a few symlinks to ensure every program is able to find the compiler. Some programs run the 'cc' program, others run the 'gcc' program, some programs expect the cpp program to be in /lib (which is /usr/lib on the LFS system) and others expect to find it in /usr/bin.

- Create those symlinks by running:

```
cd $LFS/usr/lib; ln -s gcc-lib/<host>/2.7.2.3/cpp cpp
cd $LFS/usr/bin
ln -s ../lib/gcc-lib/<host>/2.7.2.3/cpp cpp
ln -s gcc cc
```

Replace <host> with the directory where the gcc−2.7.2.3 files were installed (i686−unknown−linux in my case). You will most likely find two different directories. One of these is i484−linux. This is the directory that contains the C++ compiler files that we installed before. The directory for the gcc compiler will probably have a different name.

9. 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 install the ppp daemon to give an example.

There are a number of packages that need to be linked statically 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. During the installatin 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.

9.1 Preparing LFS system for installing basic system software

Installing Binutils

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

```
./configure
make LDFLAGS=-all-static
cp gas/as-new $LFS/usr/bin/as
cp gas/gasp-new $LFS/usr/bin/gasp
cp ld/ld-new $LFS/usr/bin/ld
cd binutils
cp addr2line ar c++filt nm-new objcopy objdump ranlib size strings strip-new $LFS/usr/bin
mv $LFS/usr/bin/nm-new $LFS/usr/bin/nm
mv $LFS/usr/bin/strip-new $LFS/usr/bin/strip
```

Install Diffutils

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

```
./configure
make LDFLAGS=-static
cp cmp diff diff3 sdiff $LFS/usr/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 fixed package from http:/tts.ookhoi.dds.nl/lfs−howto/download/diffutils−2.7−lfs.tar.gz

Installing Fileutils

- Unpack the fileutils archive
- Install the pacakge by running:

```
./configure
make LDFLAGS=-static
cd src
cp chgrp chmod chown cp dd  df dir dircolors du ginstall ln ls mkdir mkfifo mknod mv rm rmdir sync
cd $LFS/bin
mv ginstall install
ln -s install ginstall
```

Installing grep

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

```
./configure
make LDFLAGS=-static
cd src
cp egrep fgrep grep $LFS/usr/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 fixed package from http:/tts.ookhoi.dds.nl/lfs–howto/download/grep–2.4–lfs.tar.gz

Installing gzip

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

```
./configure
make LDFLAGS=-static
cp gunzip gzip $LFS/bin
```

This package is known to cause compilation problems on all platforms we have access to. If you're having trouble compiling this package as well, you can download a fixed package from http:/tts.ookhoi.dds.nl/lfs–howto/download/gzip–1.2.4–lfs.tar.gz

Installing Make

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

```
./configure
make LDFLAGS=-static
cp make $LFS/usr/bin
```

Installing Sed

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

```
./configure
make LDFLAGS=-static
cp sed/sed $LFS/usr/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 fixed package from http:/tts.ookhoi.dds.nl/lfs–howto/download/sed–3.02–lfs.tar.gz

Installing Sh–utils

- Unpack the sh–utils archive
- Install the package by running:

```
./configure
make LDFLAGS=-static
cd src
cp basename chroot date dirname echo env expr factor false groups hostid hostname id logname nice
```

Installing Tar

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

```
./configure
make LDFLAGS=-static
cp src/tar $LFS/bin
```

Installing Textutils

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

```
./configure
make LDFLAGS=-static
cd src
cp cat cksum comm csplit cut expand fmt fold head join md5sum nl od paste pr ptx sort split sum ta
```

Installing Util−linux

- Unpack the util−linux archive
- Install the package by running:

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

9.2 Reboot into LFS

Before we install the rest of the basic system software, you first have to reboot into the LFS system before continuing. Once you're rebooted and logged in, remount the root partition in read−write mode by running: `/sbin/mount −n −o remount,rw / /`

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.

Installing Bison

- Unpack the bison archive and install it by running:

```
./configure --disable-nls
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 on all platforms we have access to. If you're having trouble compiling this package as well, you can download a fixed package from http:/tts.ookhoi.dds.nl/lfs–howto/download/findutils–4.1–lfs.tar.gz

Installing Ncurses

- Unpack the ncurses archive and install it by running:

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

Installing Less

- Unpack the Less achive and install it by running:

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

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 this moment the system isn't ready yet for running the perl test. At this time we'll trust that perl compiled fine.

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 Automake

- 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
```

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 Diffutils

- 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
```

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
rm ginstall; ln -s install ginstall
```

Installing GCC

- Unpack the GCC archive and install it by running:

```
./configure
make LANGUAGES=c
make stage1
make CC="stage1/xgcc -Bstage1/" CFLAGS="-g -O2" LANGUAGES=c
make stage2
make CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O2" LANGUAGES=c
make compare
make CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O2" LANGUAGES=c install
```

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
```

This package is known to cause compilation problems on all platforms we have access to. If you're having trouble compiling this package as well, you can download a fixed package from http:/tts.ookhoi.dds.nl/lfs–howto/download/gzip–1.2.4–lfs.tar.gz

Installing Ld.so

- Unpack the Ld.so archive and install it by running:

```
cd util; make ldd ldconfig
cp ldd /bin; cp ldconfig /sbin
```

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; 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 Shadow Password Suite

- Unpack the Shadow 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
```

Installing Man−db

- Unpack the Man−db archive and install it by running:

```
groupadd −g 1 man
useradd −u 1 −g man man
./configure
make; make install
```

Installing Modutils

- Unpack the Modutils archive and install it by running:

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

Installing Termcap

- Unpack the Termcap archive and install it by running:

```
./configure
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:

```
make; make XSCPT="" install
```

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
```

Installing Sh−utils

- Unpack the Sh−utils archive and install it by running:

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

Installing start−stop−daemon

- Unpack the Dpkg archive and install it by running:

```
cd scripts; make start-stop-daemon
cp start-stop-daemon /usr/sbin
cp start-stop-daemon.8 /usr/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
```

Install Tar

- Unpack the Tar archive and install it by running:

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

Installing Textutils

- Unpack the Textutils archive and install it by running:

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

Installing Util−linux

- Unpack the Util−linux package and install it by running:

```
./configure
cd lib; make
cd clock; make; make install
cd ../disk-utils; make; make install
cd ../fdisk; make; make install
cd ../login-utils; make agetty
cp agetty /sbin; cp agetty.8 /usr/man/man8
cd ../mount; make; make install
cd ../sys-utils; make; make install
cd ../text-utils; make MOREHELPDIR=/usr/share/more
make MOREHELPDIR=/usr/share/mor install
```

Installing Vim

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

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

10. Creating initial boot scripts

10.1 Preparing the directories and master files

You need the Sysvinit package again for this section.

Create the necessary directories by issuing these commands:

```
   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 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
```

10.2 Creating the reboot script

- Create a new file `reboot` containing the following:

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

   echo -n "System reboot in progress..."

   /sbin/reboot -d -f -i

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

10.3 Creating the halt script

- Create a new file `halt` containing the following:

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

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

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

10.4 Creating the mountfs script

- Create a new file `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..."
```

```
/sbin/mount -n -o remount,rw /
check_status

> /etc/mtab
/sbin/mount -f -o remount,rw /

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

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

10.5 Creating the umountfs script

- Create a new file `umountfs` containing the following:

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

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}
echo -n "Unmounting file systems..."
/sbin/umount -a -r
check_status

echo -n "Remounting root file system in read-only mode..."
/sbin/mount -o remount,ro /
check_status

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

10.6 Creating the sendsignals script

- Create a new file `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
```

10.7 Set file permissions and create symlinks

- Set the proper file permissions by running `chmod 755 reboot halt mountfs umountfs sendsignals`
- Create the necessary symlinks by running:

```
cd ../rc6.d; ln -s ../init.d/umountfs S90umountfs
ln -s ../init.d/reboot S99reboot
ln -s ../init.d/sendsignals S80sendsignals

cd ../rc0.d; ln -s ../init.d/umountfs S90umountfs
ln -s ../init.d/halt S99halt
ln -s ../init.d/sendsignals S80sendsignals

cd ../rcS.d; ln -s ../init.d/mountfs S10mountfs
```

10.8 Creating the /etc/fstab file

- Create a file `/etc/fstab` containing the following:

```
/dev/<LFS-partition device> / ext2 defaults 0 1
/dev/<swap-partition device> none swap sw 0 0
proc /proc proc defaults 0 0
```

10.9 Testing the system

You can test the system by restarting your computer and boot into LFS again. Any errors should be gone now and your root partition should be mounted in read–write mode automatically.

You can now finally restart your computer with a command like `shutdown -r now`

11. Installing the rest of the basic system software

The rest of the software that's part of our basic system will be installed in this section. You don't need all the software, but it's recommended to have it.

11.1 Installing E2fsprogs

Installing E2fsprogs

- Unpack the E2fsprogs archive
- Configure the package by running `configure`
- Compile the package by running `make`

When compiling I'm getting this error: mke2fs.c:142:SCSI_DISK_MAJOR not defined. I solved it the following way:

- Edit the `misc/mke2fs.c` file and find the first occurrence of *SCSI_DISK_MAJOR*
- Change this to: *SCSI_DISK0_MAJOR*

Please note that I have no idea what this does when you're using a SCSI system, but I can guess not a heck of a lot of good. Since I'm using an IDE system this doesn't harm me. If you're using SCSI you're on your own I'm afraid since I have no idea on how to fix this. Perhaps you don't even get it when using (a) SCSI disk(s).

- Install the package by running `make install`

Creating the checkroot bootscript

We'll create a checkroot bootscript so that whenever we boot our LFS system, the root file system will be checked by fsck.

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

```
#!/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
  mount -n -o remount,ro /
  if [ $? = 0 ]
  then
    if [ -f /forcecheck ]
    then
      force="-f"
    else
      force=""
    fi
```

```
    echo "Checking root file system..."
    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 "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
```

Updating /etc/init.d/umountfs

- Edit the `/etc/init.d/umounts` file and put these lines as the first commands (under the "# Begin /etc/init.d/umountfs" line)

```
echo "Deactivating swap..."
/sbin/swapoff -av
```

Creating proper permissions and creating symlink

- Set the proper permissions on the checkroot file by running `chmod 755 /etc/init.d/checkroot`
- Create the proper symlink by running `cd /etc/rcS.d; ln -s ../init.d/checkroot S05checkroot`

11.2 Installing File

- Unpack the File archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

11.3 Installing Libtool

- Unpack the Libtool archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

11.4 Installing Modutils

- Unpack the Modutils archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running make install

11.5 Installing Linux86

- Unpack the Linux86 archive
- Go to the as directory and compile the programs there by running `make`
- Copy the following binary to /usr/bin: `as86`
- Go to the ld directory and compile the programs there by running `make`
- Copy the following binary to /usr/bin: `ld86`

11.6 Installing Lilo

Installing Lilo

- Unpack the Lilo archive
- Compile the package by running `make`
- Install the package by running `make install`

Configuring Lilo

- Copy the `/etc/lilo.conf` file from your normal Linux system to the /etc directory on the LFS system

Copying kernel image files

- Copy the kernel images from the `/boot` directory from your normal Linux system to /boot on the LFS system

11.7 Installing DPKG

We don't install the Debian Package manger itself, but a small program that is shipped with this package; the start−stop−daemon program. This program is very useful in boot scripts so we're going to use it.

- Unpack the DPKG archive
- Go to the scripts directory
- Compile the start−stop−daemon program by running `make start−stop−daemon`
- Copy the following binary /sbin: `start−stop−daemon`
- Copy the following file to /usr/man/man8: `start−stop−daemon.8`

11.8 Installing Sysklogd

Installing Sysklogd

- Unpack the Sysklogd archive
- Compile the package by running `make`
- Install the package by running `make INSTALL=/bin/install install`

Configuring Sysklogd

- Create the /var/log directory
- Create a new file `/etc/syslog.conf` containing the following:

Please note that the white spaces must be tabs and not just hitting the space bar a few times.

```
#!/bin/sh
# Begin /etc/syslog.conf

auth,authpriv.*            /var/log/auth.log
*.*;auth,authpriv.none     /var/log/syslog
daemon.*                   /var/log/daemon.log
kern.*                     /var/log/kern.log
mail.*                     /var/log/mail.log
user.*                     /var/log/user.log

mail.info                  /var/log/mail.info
mail.warn                  /var/log/mail.warn
mail.err                   /var/log/mail.err

*.=info;*.=notice;*.=warn; \
  auth,authpriv.none; \
  daemon.none              /var/log/messages

*.emerg                    *

# End /etc/syslog.conf
```

Creating the Sysklogd bootscript

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

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

test -f /usr/sbin/klogd || exit 0
test -f /usr/sbin/syslogd || exit 0

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

- Set the proper permissions by running `chmod 755 /etc/init.d/sysklogd`
- Create the proper symlinks by running the following commands:

```
cd /etc/rc2.d; ln -s ../init.d/sysklogd S03sysklogd
cd ../rc6.d; ln -s ../init.d/sysklogd K90sysklogd
cd ../rc0.d; ln -s ../init.d/sysklogd K90sysklogd
```

11.9 Installing Groff

- Unpack the Groff archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

11.10 Installing Man−db

- Unpack the Man−db archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

11.11 Installing Procps

- Compile the package by running make
- Edit the Makefile file and comment out the variable: *XSCPT*
- Install the package by running `make install`

11.12 Installing Procinfo

- Compile the package by running `make`
- Install the package by running `make install`

11.13 Installing Procmisc

- Compile the package by running `make`
- Install the package by running `make install`

11.14 Installing Shadow Password

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, pop3d, 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.

- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`
- Copy the following files from the etc directory to /etc: `limits login.access login.defs.linux shells suauth`
- Rename the `/etc/login.defs.linux` to `/etc/login.defs`

Now is a very good moment to read section #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, but you can avoid a hassle by testing properly and reading manuals ;)

12. Setting up basic networking

12.1 Installing Netkit−base

- Unpack the Netkit−base archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`
- Copy the following files from the etc.sample directory to the /etc/ directory: `services protocols`

12.2 Installing Net−tools

- Unpack the Net−tools archive
- Compile the package by running `make`
- Install the package by running `make install`

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 permissions by running `chmod 755 /etc/init.d/localnet`
- Create the proper symlinks by running `cd /etc/rcS.d; ln −s ../init.d/network 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 somealiases
```

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.lfs.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).

Here's the `/etc/hosts` file if you don't configure a network card:

```
# 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)
```

Here's the `/etc/hosts` file if you do configure a network card:

```
# 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 sub section only applies if you are going to configure a network card. If not, skip this sub section and read on.

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
}

/sbin/ifconfig eth0 <ipaddress>
check_status

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

Setting up permissions and symlink for /etc/init.d/ethnet

- Set the proper permissions by running `chmod 755 ethnet`
- Create the proper symlinks by running `cd ../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 /etc/hosts is properly setup by running:

```
ping <your FQDN>
ping <what you choose for hostname>
ping localhost
ping 127.0.0.1
ping 192.168.1.1 (only when you configured your network card)
```

All these five ping command's should work without failures. If so, the basic network is working.

13. Setting up Email sub system

13.1 Preparing system for Email sub system

Creating extra groups and user

We need to add a few groups and a user which will be used by the email utilities.

- Create the bin group by running `groupadd -g 1 bin`
- Create the kmem group by running `groupadd -g 2 kmem`
- Create the mail group by running `groupadd -g 3 mail`
- Create the bin user by running `useradd -u 1 -g bin -d /bin -s /bin/sh bin`

Creating directories

There are two directories used by the email sub system, thus we need to create them and give them the proper permissions.

- Create the `/var/spool` directory
- Create the `/var/spool/mqueue` directory
- Create the `/var/spool/mail` directory
- Set permissions on /tmp by running `chmod 777 /tmp`
- Set permissions on /var/spool/mqueue by running `chmod 700 /var/spool/mqueue`
- Set permissions on /var/spool/mail by running `chmod 775 /var/spool/mail`
- Put /var/spool/mail in the mail group by running `chgrp mail /var/spool/mail`

13.2 Installing Procmail

- Unpack the Procmail archive
- Compile the package by running `make`
- Install the package by running `make install`
- Set the proper permissions on the Procmail utilities by running `make install-suid`

13.3 Installing Sendmail

Installing Sendmail

- Unpack the Sendmail archive
- Go to the src directory
- Compile the package by running `Build`
- Install the package by running `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, but this is not the right article. 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 the following:

```
OSTYPE(LFS)
FEATURE(nouucp)
define(`LOCAL_MAILER_PATH', /usr/bin/procmail)
MAILER(local)
MAILER(smtp)
```

- Create an empty file `ostype/lfs.m4` by running `touch ostype/lfs.m4`
- Compile the lfs.mc file by running `m4 m4/cf.m4 cf/lfs.cf > cf/lfs.cf`
- Copy the cf/lfs.cf to `/etc/sendmail.cf`
- Create an empty /etc/aliases file by running `touch /etc/aliases`
- Initialize this (empty) alias database by running `sendmail -v -bi`

13.4 Installing Mailx

- Unpack the Mailx archive
- Compile the package by running `gcc *.c -o mail`

Ignore possible 'comparison between pointer and integer' and 'assignments makes integer from pointer without a cast' warnings. You'll probably get quite a few of these. Though, the program seems to work just fine nevertheless.

- Copy the following binary to /usr/bin: `mail`
- Place the /usr/bin/mail program in the mail group by running `chgrp mail /usr/bin/mail`
- Let the /usr/bin/mail program be executed sgid by running `chmod 2755 /usr/bin/mail`

13.5 Creating /etc/init.d/sendmail bootscript

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

```
#!/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 -p /var/run/sendmail.pid \
        -x /usr/sbin/sendmail -- -bd
    check_status
    ;;

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

    sleep 1

    echo -n "Starting Sendmail..."
    start-stop-daemon -S -q -p /var/run/sendmail.pid \
        -x /usr/sbin/sendmail -- -bd
    check_status
    ;;

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

esac

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

13.6 Setting up permissions and symlinks

- Set the proper permissions by running `chmod 755 /etc/init.d/sendmail`
- Create the proper symlinks by running:

```
cd /etc/init.d/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
```

13.7 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
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

13.8 Installing Fetchmail

- Unpack the Fetchmail archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

13.9 Testing the Email sub system

It's time to test the email system now.

- Start Sendmail by running `/usr/sbin/sendmail -bd` (you need to start sendmail using the full path. If you don't, you can't let sendmail reload the sendmail.cf by with kill −1 <sendmail pid>).
- Send yourself an email by running `echo "this is an email test" | mail -s test root`
- Start the `mail` program and you should see your email there.
- Create a new user by running `useradd -m testuser`
- Send an email to testuser by running `echo "test mail to testuser" | mail -s test testuser`
- Login as testuser, try to obtain that email (using the mail program) and send an email to root in the same way as you send an email to testuser.

If this all worked just fine, you have a working email system for local email. It's not necessarily ready for Internet yet. You can remove the testuser by running userdel −r testuser

14. Installing Internet Servers

In this section we're going to install three of the most used Internet servers, together with the necessary clients. These are going to be installed:

telnetd with the standard telnet client

proftpd with the standard ftp client

apache with lynx as client

14.1 Installing telnet daemon + client

- Unpack the Netkit−telnet archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

14.2 Installing Proftpd

- Unpack the Proftpd archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

14.3 Installing Netkit−ftp

- Unpack the Netkit−ftp archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

14.4 Installing Apache

Apache isn't that easily configured. Like with Sendmail, a lot depends on your own preference and system setup. Therefore, once I again I stick with a very basic installation. If this doesn't work well enough for you, read the documentation and modify whatever you need to.

- Unpack the Apache archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

14.5 Installing Slang Library

The Slang library is an alternative to the Ncurses library. We're going to use this library to link Lynx against. Though Lynx works fine with the Ncurses library, people recommend using the Slang library. I myself can't find a difference between a Lynx linked against the Slang library or against the Ncurses library. However, I'll just follow that advise and use Slang.

- Unpack the Slang archive
- Configure the package by running `configure`
- Compile the package by running `make elf`

- Install the package by running `make install-elf`
- Create extra symlinks for the library by running `make install-links`

14.6 Installing Zlib

Zlib is a compression library, used by programs like PKware's zip and unzip utilities. Lynx can use this library to compress certain files.

- Unpack the Zlib archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

14.7 Installing Lynx

- Unpack the Lynx archive
- Configure the package by running `configure --libdir=/etc --with-zlib --with-screen=slang`
- Compile the package by running `make`
- Install the package by running `make install`
- Install the helpfile by running `make install-help`
- Install other documentation by running `make install-doc`

14.8 Configuring the daemons

It's possible to run the daemons in either stand–alone mode or via the Internet Server daemon (inetd). Where possible, I choose to run the daemons in stand–alone mode. This makes it easier to start and stop individual processes without modifying the /etc/inetd.conf file constantly.

However, in the telnetd case it's better to run it via inetd, since telnetd doesn't seem to respawn itself when the last user logs out. This would mean as soon as the last person logs out from the telnet session, the telnet daemon stops as well. This isn't desirable, so we let telnetd run using inetd to spawn a telnet process whenever somebody logs on.

14.9 Configuring telnetd

Creating the /etc/inetd.conf configuration file

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

```
# Begin /etc/inetd.conf
```

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

# End /etc/inetd.conf
```

Creating the /etc/init.d/inetd bootscript

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

```
#!/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 -p /var/run/inetd.pid \
         -x /usr/sbin/inetd
    check_status
    ;;

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

    sleep 1

    echo -n "Starting Internet Server daemon..."
    start-stop-daemon -S -q -p /var/run/inetd.pid \
        -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 necessary symlinks by running

```
cd /etc/rc2.d; ln -s ../init.d/inetd S30inetd
cd ../rc0.d; ln -s ../init.d/inetd K30inetd
cd ../rc6.d; ln -s ../init.d/inetd K30 inetd
```

14.10 Configuring proftpd

Creating necessary groups and users

- Create the necessary groups by running:

```
groupadd -g 65534 nogroup
groupadd -g 4 ftp
```

- Create the necessary users by running:

```
useradd -u 65534 -g nogroup -d /home nobody
useradd -u 4 -g ftp -m ftp
```

Creating the /etc/init.d/proftpd bootscript

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

```
#!/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 -x /usr/sbin/proftpd
    check_status
    ;;

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

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

    sleep 1

    echo -n "Starting Pro FTP daemon..."
    start-stop-daemon -S -q -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 necessary symlinks by running:

```
cd /etc/rc2.d; ln -s ../init.d/proftpd S40proftpd
cd ../rc0.d; ln -s ../init.d/proftpd K40proftpd
cd ../rc6.d; ln -s ../init.d/proftpd K40proftpd
```

14.11 Configuring apache

Editing apache configuration file

Edit the files in the /usr/apache/etc directory and modify them according to your own needs.

- Edit the `httpd.conf` file and find the variable: *Group*
- Replace the current value (if any) with: *nogroup*

Creating /etc/init.d/apache bootscript

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

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

case "$1" in
  start)
    echo -n "Starting Apache HTTP daemon..."
    /usr/apache/sbin/apachectl start
    ;;

  stop)
    echo -n "Stopping Apache HTTP daemon..."
    /usr/apache/sbin/apachectl stop
    ;;

  restart)
    echo -n "Restarting Apache HTTP daemon..."
    /usr/apache/sbin/apachectl restart
    ;;

  force-restart)
    echo -n "Stopping Apache HTTP daemon..."
    /usr/apache/sbin/apachectl stop

    sleep 1

    echo -n "Starting Apache HTTP daemon..."
    /usr/apache/sbin/apachectl start
    ;;

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

esac

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

Setting up permissions and symlinks

- Set the proper permissions by running `chmod 755 /etc/init.d/apache`
- Create the necessary symlinks by running:

```
cd /etc/rc2.d; ln -s ../init.d/apache S50apache
cd ../rc0.d; ln -s ../init.d/apache K50apache
cd ../rc6.d; ln -s ../init.d/apache K50apache
```

14.12 Testing the daemons

The last step in this section is testing the just installed and configured daemons.

- Start the Internet Server daemon (and with it telnetd) by running `/etc/init.d/inetd start`
- Start a telnet session to localhost by running `telnet localhost`
- Login and logout again.
- Start the Pro ftp daemon by running `/etc/init.d/proftpd start`
- Start a ftp session to localhost by running `ftp localhost`
- Login as user anonymous and logout again.
- Start the Apache http daemon by running `/etc/init.d/apache start`
- Start a http session to localhost by running `lynx http://localhost`
- Exit lynx.

If these tests ran without trouble, the daemons are all working fine.

15. Installing X Window System

15.1 Installing X

- Unpack the X archive
- Compile the package by running `make World`

During the compilation process you will encounter a few errors about the "makedepend" script not being able to find the stddef.h stdarg.h and float.h header files. The script just isn't as smart as the compiler is apparently, since the compilation itself does work fine without compilation errors. Though, creating a few temporary symlinks won't solve the problem; they only will cause more problems for some reason.

So you just ignore the many makedepend errors you most likely will be getting. Also errors similar to "pointer targets in passing arg x of somefunction differ in signedness". You can rewrite those files if you feel like it. I won't.

- Install the package by running `make install`
- Install the man pages by running `make install.man`

15.2 Creating /etc/ld.so.conf

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

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

/lib
/usr/lib
```

```
/usr/X11R6/lib

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

- Update the dynamic loader cache by running `ldconfig`

15.3 Modifying /etc/man_db.config

- Edit the /etc/man_db.config file and look for this line: *MANDATORY_MANPATH /usr/man*
- Under that line put the following one: *MANDATORY_MANPATH /usr/X11R6/man*

15.4 Creating the /usr/include/X11 symlink

- In order for the pre–processor to find the X11/*.h files (which you encounter in #include statements in source code) create the following symlink: `ln -s /usr/X11R6/include/X11 /usr/include/X11`

15.5 Creating the /usr/X11 symlink

Often software copies files to /usr/X11 so it doesn't have to know which release of X you are using. This symlink hasn't been created by the X installation, so we have to create it by ourselves.

- Create the /usr/X11 symlink by running `ln -s /usr/X11R6 /usr/X11`

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

There are a few ways to add the /usr/X11/bin path to the $PATH environment variable. One way of doing so is the following:

- Create a new file `/root/.bashrc` with it's contents as follows: *export PATH=$PATH:/usr/X11/bin*

You need to login again for this change to become effective. Or you can update the path by running `export PATH=$PATH:/usr/X11/bin manually`

15.7 Configuring X

- Configure the X server by running `xf86config`

If the XF86Config file created by xf86config doesn't suffice, then you better copy the already existing XF86Config from your normal Linux system to /etc. Cases wherein you need to make special changes to the file which aren't supported by the xf86config program force you to do this. You can always modify the created XF86Config file by hand. This can be very time consuming, especially if you don't quite remember what needs to be changed.

15.8 Testing X

Now that X is properly configured it's time for our first test run.

- Start the X server by running `startx`

The X server should start and display 3 xterm's on your screen. If this is true in your case, X is running fine.

16. Installing Window Maker

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.

16.1 Preparing the system for the Window Maker installation

Installing libPropList

- Unpack the libPropList archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

Installing libXpm

- Unpack the libXpm archive
- Prepare the compilation by running `xmkmf; make Makefiles; make includes; make depend`

Ignore the warning about not being able to find the X11/xpm.h file from make depend.

- Compile the package by running `make`

The compilation process will abort because the X11/xpm.h file cannot be found. So we install this file now and then recompile.

- Go to the lib directory
- Install the libraries and header files by running `make install`
- Go to the top level directory and recompile the package by running `make`
- Install the rest of the package by running `make install`

Installing libpng

- Unpack the libpng archive
- Compile the package by running `make -f scripts/makefile.lnx`
- Install the package by running `make -f scripts/makefile.lnx install`

Installing libtiff

- Unpack the libtiff archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

Installing libjpeg

- Unpack the libjpeg archive
- Configure the package by running `configure --enable-shared --enable-static`
- Compile the library by running `make libjpg.la`
- Compile the tools and install the package by running `make install`

Installing libungif

- Unpack the libungif archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

Installing WindowMaker

- Unpack the WindowMaker archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

16.2 Updating dynamic loader cache

- Update the dynamic loader cache by running `ldconfig`

16.3 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`

16.4 Testing WindowMaker

- Start the X server and see if the WindowMaker Window Manager starts properly by running `startx`

17. Configuring system for Internet

17.1 Configuring 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.

17.2 Creating groups and directories

- Create the daemon group by running `groupadd -g 5 daemon`
- Create the /var/lock directory by running `mkdir /var/lock`

17.3 Installing PPP

- Unpack the PPP archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

17.4 Creating /etc/resolv.conf

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

```
# Begin /etc/resolv.conf

nameserver <IP address of your ISP's primairy DNS server>
nameserver <IP address of your ISP's secundairy DNS server>

# End /etc/resolv.conf
```

17.5 Creating the connect and disconnect scripts

- Create a new file `/usr/bin/pon` file containing the following:

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

/usr/sbin/pppd call provider

# End /usr/bin/pon
```

- Create a new file `/usr/bin/poff` file containing the following:

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

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/poff
```

17.6 Creating /etc/ppp/peers/provider

- Create the `/etc/ppp/peers` directory
- Create a new file `/etc/ppp/peers/provider` containing the following:

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

noauth
connect "/usr/sbin/chat -v -f /etc/chatscripts/provider"
/dev/ttyS1
115200
defaultroute
noipdefault

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

17.7 Creating /etc/chatscripts/provider

- Create the `/etc/chatscripts` directory
- Create a new file `/etc/chatscripts/provider` containing the following:

```
# 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
```

17.8 Note on password authentication

As you see from the sample scripts (these are the actual scripts I'm using myself) above I logon to my ISP using this chatscripts in stead of authenticating via pap or chap. Though my ISP supports pap, I choose to do it this slightly different way which has it's disadvantages and advantages. In my case the advantages outweigh the disadvantages. This way I have more control over my logon procedure and I can see closer what is happening when.

For example most times when I connect I have a window running `tail -f /var/log/syslog` so I can keep an eye on when things like the username and password are sent.

17.9 Other resources

For a far more detailed guide on how to set up Internet, I refer to Egil Kvaleberg's *ISP−Hookup−HOWTO* which is available from the LDP site at http://www.linuxdoc.org/

18. Migrations from old to new setups

This section is only to be used by people who have installed an LFS system using previous versions of this HOWTO. If a major change in an installation approach has taken place, you first need to take some actions such as removing existing files from a package before you can re−install that package. This section is used to assist people who obtained an old, obsolete version of this HOWTO and after installing the LFS system noticed that there's a new HOWTO (like this one) fixing things that went wrong in the older versions.

18.1 Migrating from old C++ Library setup to the new setup

This section only applies to people who have previously installed the C++ Library using LFS−HOWTO version 1.0, 1.1 or 1.2.

- Remove the following directory and symlinks: /usr/include/g++*

18.2 Migrating from old compiler setup to the new setup

This section only applies to people who have previously installed compilers using LFS−HOWTO 1.0, 1.1 or 1.2.

- Remove the following files from the $LFS/usr/bin directory: cc cpp c++ gcc gcj gcjh g77 g++ protoize unprotoize
- Remove the following directories: $LFS/usr/lib/gcc−lib $LFS/usr/gcc2723

19. Copyright & Licensing Information

Copyright (C) 1999 by Gerard Beekmans. This document may be distributed only subject to the terms and conditions set forth in the LDP License at http://www.linuxdoc.org/COPYRIGHT.html.

It is not necessary to display the license notice, as described in the LDP License, when only a small part of this document (the HOWTO) is quoted for informational or similar purposes. However, I do require you to display with the quotation(s) a line similar to the following line: "Quoted from the LFS−HOWTO at http://huizen.dds.nl/~glb/