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

December 16th, 1999

This document describes the process of creating your own Linux system from scratch, using nothing but the sources of needed software.

---

1. Introduction

1.1 What's this all about?

I started this document about 6 months ago. 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

1.1 − December 20th, 1999

- Fixed a few typo's
- 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 TODO

Things that need to be done for future releases

- Adding a section about Pam
- Adding a section about SSH

1.5 Contact info

You can reach me, Gerard Beekmans, at tts−sol@dds.nl

2. Software 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, unless mentioned otherwise. The list is ordered on usage, meaning that the first program you see in the list is the first program we'll build in this document.

Sysvinit : ftp://cistron.nl/pub/people/miquels/sysvinit/

Bash : ftp://ftp.gnu.org/gnu/bash/

Linux Kernel : ftp://ftp.kernel.org/

Make : ftp://ftp.gnu.org/gnu/make/

Sed : ftp://ftp.gnu.org/gnu/sed/

Shell Utils : ftp://ftp.gnu.org/gnu/sh−utils/

File Utils : ftp://ftp.gnu.org/gnu/fileutils/

Util Linux : ftp://ftp.win.tue.nl/pub/linux/utils/util−linux/

Text Utils : ftp://ftp.gnu.org/gnu/textutils/

Tar : ftp://ftp.gnu.org/gnu/tar/

Gzip : ftp://ftp.gnu.org/gnu/gzip/

Binutils : ftp://ftp.gnu.org/gnu/binutils/

Grep : ftp://ftp.gnu.org/gnu/grep/

Bison : ftp://ftp.gnu.org/gnu/bison/

Mawk : ftp://ftp.gnu.org/gnu/mawk/

Find Utils : ftp://ftp.gnu.org/gnu/findutils/

Diff Utils : ftp://ftp.gnu.org/gnu/diffutils/

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

Perl : ftp://ftp.gnu.org/gnu/perl/

M4 : ftp://ftp.gnu.org/gnu/m4/

Texinfo : ftp://ftp.gnu.org/gnu/texinfo/

Automake : ftp://ftp.gnu.org/gnu/automake/

Autoconf : ftp://ftp.gnu.org/gnu/autoconf/

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

Glibc−crypt−2.0.7pre6 : 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/

GCC−2.95.2 : ftp://ftp.gnu.org/gnu/gcc/

GCC−2.7.2.3 : ftp://ftp.gnu.org/gnu/gcc/

Ncurses : ftp://ftp.gnu.org/gnu/ncurses/

Vim : ftp://ftp.vim.org/pub/vim/

Readline Library : ftp://ftp.gnu.org/gnu/readline/

Termcap Library : ftp://ftp.gnu.org/gnu/termcap/

Flex : ftp://ftp.gnu.org/gnu/flex/

Less : ftp://ftp.gnu.org/gnu/less/

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

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

Libtool : ftp://ftp.gnu.org/gnu/libtool/

Modutils : ftp://ftp.ocs.com.au/pub/modutils/v2.3/

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

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

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

Sysklogd : ftp://sunsite.unc.edu/pub/Linux/system/daemon/

Groff : ftp://ftp.gnu.org/gnu/groff/

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

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

Procinfo : ftp://ftp.cistron.nl/pub/people/svm/

Procmisc : ftp://lrcftp.epfl.ch/pub/linux/local/psmisc/

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

libstdc++ : ftp://ftp.gnu.org/gnu/libstdc++/

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

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

Procmail : ftp://ftp.procmail.org/pub/procmail/

Sendmail : ftp://ftp.sendmail.org/pub/sendmail/

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

Mutt : ftp://ftp.mutt.org/pub/mutt/

Fetchmail : http://www.tuxedo.org/~esr/fetchmail/

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

Proftpd : ftp://ftp.tos.net/pub/proftpd/

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

Apache : http://www.apache.org/dist/

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

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

Lynx : http://www.slcc.edu/lynx/release/

Xfree86 : ftp://ftp.xfree86.org/pub/XFree86/

libPropList : ftp://ftp.windowmaker.org/pub/libs/

libXpm : ftp://sunsite.unc.edu/pub/Linux/libs/X/

libpng : http://www.cdrom.com/pub/png/

libtiff : ftp://ftp.sgi.com/graphics/tiff/

libjpeg : http://www.ijg.org/

libungif : ftp://prtr−13.ucsc.edu/pub/libungif/

WindowMaker : ftp://ftp.windowmaker.org/pub/release/

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

3. Preparing a new partition

3.1 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. If you already have a Linux Native partition available, you can skip this subsection.

Start the `fdisk` program (or `cfdisk` if you prefer that program) 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 (c)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.

3.2 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. From now on I'll refer to this newly created partition as $LFS. $LFS should be substituted with the partition you have created. If you created your partition on /dev/hda4, you mounted it on /mnt/hda4 and this document tells you to copy a file to $LFS/usr/bin then you need to copy that file to /mnt/hda4/usr/bin.

To create a new ext2 file system we use the `mke2fs` command. Give $LFS as the only option and the file system will be created.

3.3 Adding an entry to LILO

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

```
image=<image>
   label=<label>
   root=$LFS
   read-only
```

Replace <image> by an already existing kernel image file. For now, use the kernel image you're using at the moment to boot your 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.

3.4 Creating directories

Let's create a minimal directory tree on the $LFS partition. issuing the following commands will create the necessary directories. Make sure you first mount the $LFS partition before you attempt to create the directories.

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

As you see, on the LFS system the /usr/local directory points to /usr. I am aware that this is in violation with the FHS (File Hierarchy Standard – http://www.pathname.com/fhs/) but my idea is that the usr/local directory doesn't apply on a completely self–built system, since every software package is installed locally anyway and there's no part installed by a vendor's CD–ROM or something similar. Therefore I chose to make /usr/local and /usr one–and–the–same directory.

Also, /usr/etc and /usr/var point to /etc and /var. This is just another of my preferences.

3.5 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 owner ships:

```
cp -av /dev $LFS
```

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

4. Installing Sysvinit

4.1 Preparing 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 program found on most Linux systems is called Sysvinit and that's the program we're going to install on our LFS system.

- Unpack the Sysvinit archive
- Go to the src directory
- Edit the `Makefile` file
- Somewhere in this file, but before the rule *all:* putt his variable: *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
```

- Compile the package by running `make LDFLAGS=-static`
- Install the package by running `make install`

4.2 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
```

4.3 Copying 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 root's password. This means we need to have at least a passwd file present on the LFS system. We'll use the passwd and group files from the current running Linux system. Since the passwords are encoded it's just easier to copy the already present file and use that, instead of retyping the encoded password. Mistakes are easily made and this way we can avoid extra hassle afterwards.

- Copy the `/etc/passwd` and `/etc/group` files to `$LFS/etc/`
- Edit the `$LFS/etc/passwd` file and remove every line, except the line for the user root
- Edit the `$LFS/etc/group` file and remove every line, except the line for the group root

4.4 Installing a root 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
- Configure the package by running `configure --enable-static-link`
- Compile the package by running `make`
- Copy the binary `bash` to `$LFS/bin`
- Create a symlink that links `$LFS/bin/sh` to `$LFS/bin/bash`

4.5 Testing the system

After you've completed this section, we can test the system and see if we can logon to it. Please note that you will get errors regarding the init program not being able to start the rcS and rc scripts. We will install these scripts in a later stage.

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." 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 also will be installed in a later stage. 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).

5. Installing a kernel

5.1 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).

5.2 Configuring the kernel

- Unpack the Kernel archive
- 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`

5.3 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.

5.4 Copying the new kernel source tree to $LFS

Copy the entire source tree of the new kernel to $LFS/usr/src. This can easily be accomplished by running `cp -av <kernel directory> $LFS/usr/src`

5.5 Updating symlinks

Often the /usr/local/include/linux directory is a symlink to /usr/src/linux and /usr/src/linux is often a symlink to /usr/src/<kernel version>. Make sure that /usr/src/linux now points to directory of the kernel source that you have unpacked before.

It's possible that on your system /usr/include/linux points to /usr/src/linux − this depends on your distribution.

Execute the following commands to create the proper symlinks on the LFS system.

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

Please note that if you need to compile software that's going to be used on your normal Linux system and it needs the kernel headers, it might be a better idea to restore the symlinks back into their original position if you decide not to load the new kernel for the normal Linux system. It is perfectly safe to load the lfskernel for both the LFS system and the normal system.

5.6 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 (like crashing).

6. Installing the GNU C Library

6.1 Preparing the system for the GNU C Library installation

In this section we're going to install Glibc. But before we'll be able to install these libraries, we need to have a bunch of other software installed on the LFS system. Therefore all these programs need to be linked statically. This means quite a bit of extra work, because after Glibc and the GNU CC compilers are installed, we're going to re−install all these programs so they'll be linked dynamically. If somebody knows of a better way to accomplish this, without first building all the software statically and then rebuild them dynamically, please let me know.

I know of one other way and that's by installing Glibc using pre−compiled binaries. But that would be directly against what we're doing here. So that's not an option.

All software that is being installed in this section will be compiled on our normal working Linux system and then copied to the LFS system.

You'll notice that the installation of this software is very straightforward is most cases. I also won't explain what this software does, since it's all trivial software and if you don't know what some program does, you can always read the README file and other documentation.

Installing Make

- Unpack the Make archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=−static`

- Copy the `make` binary to $LFS/usr/bin

Installing Sed

- Unpack the Sed archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`
- Copy the `sed/sed` binary to $LFS/usr/bin

Installing Shell Utils

- Unpack the Shell Utils archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`
- Copy the following binaries from the src directory to $LFS/bin: `date echo false pwd sleep stty su true uname`
- Copy the following binary from the src directory to $LFS/sbin: `chroot`

Installing File Utils

- Unpack the File Utils archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`
- Copy the following binaries to $LFS/bin: `chgrp chmod chown cp dd df dir dircolors du ln ls mkdir mkfifo mknod rm rmdir sync touch vdir`
- Copy the following binary to $LFS/usr/bin: `ginstall`
- Rename the `$LFS/usr/bin/ginstall` file to `$LFS/usr/bin/install`

Installing Util Linux

- Unpack the Util Linux archive
- Configure the package by running `configure`
- Go to the lib directory and compile the files there by running `make`
- Go to the mount directory and compile the programs there by running `make LDFLAGS=-static`
- Copy the following binaries to $LFS/sbin: `losetup mount swapon umount`
- Create the symlink that links `$LFS/sbin/swapoff` to `$LFS/sbin/swapon`

Installing Text Utils

- Unpack the Text Utils archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`
- Copy the following binary from the src directory to $LFS/bin: `cat`
- Copy the following binaries from the src directory to $LFS/usr/bin: `cksum comm csplit cut`

expand fmt fold head join md5sum nl od paste pr sort split sum tac tail tr unexpand uniq wc

Installing Tar

- Unpack the Tar archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`

After compiling the programs in the src directory you will have two programs: tar and rmt. Tar is obvious. Rmt stands for ReMote Tapeserver. If you don't need this program (ie; you don't have a tapestreamer in your network or on your machine) then you don't need to copy this program.

- Copy the `src/tar` binary to $LFS/bin
- Copy the `src/rmt` binary to $LFS/bin (if you need it)

Installing Gzip

- Unpack the Gzip archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`

I'm using version 1.2.4 and during the compilation process I'm getting this error: conflicting types for basename. If you're also being troubled by this error, here's how to fix it:

- Edit the gzip.h file and find the line: *extern char *basename OF((char *fname));*
- Replace this line with: *extern char *basename2 OF((char *fname));*
- Edit the util.c file and find the line: *char *basename(fname)*
- Replace this line with: *char *basename2(fname)*

Recompile the package now (by running make LDFLAGS=−static again) and the error should be fixed.

What was wrong here? On my system there was a function called 'basename' in one of the standard system header files (I think it was string.h but I'm not sure anymore). The Gzip program has a function of it's own, also called basename and those two caused a collision if you will. By rename the Gzip specific basename function to basename2, the problem was solved.

- Remove from the following files the *.in* extension: `gzexe.in zdiff.in zforce.in zgrep.in zmore.in znew.in`
- Copy the following files to $LFS/bin: `gunzip gzexe gzip zcat zdiff zforce zgrep zmore znew`

Installing Binutils

- Unpack the Binutils archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-all-static`
- Copy the following binaries from the gas directory to $LFS/usr/bin: `as-new gasp-new`
- Rename those files to `$LFS/usr/bin/as` and `$LFS/usr/bin/gasp`
- Copy the following binaries from the ld directory to $LFS/usr/bin:`ld-new`
- Rename that file to `$LFS/usr/bin/ld`
- Copy the following binaries from the binutils directory to $LFS/usr/bin: `addr2line ar c++filt nm-new objcopy objdump ranlib size strings strip-new`
- Rename $LFS/usr/bin/nm-new to `$LFS/usr/bin/nm`
- Rename $LFS/usr/bin/strip-new to `$LFS/usr/bin/strip`

Installing Grep

- Unpack the Grep archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`
- Copy the following binaries from the src directory to $LFS/usr/bin: `egrep fgrep grep`

Installing Bison

- Unpack the Bison archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`
- Copy the following binary to $LFS/usr/bin: `bison`
- Copy the following files to $LFS/usr/share: `bison.hairy bison.simple`

Installing Mawk

- Unpack the Mawk archive
- Configure the package by running `configure`
- Compile the package by running `make CFLAGS="-O -static"`
- Copy the following binary to $LFS/usr/bin: `mawk`
- Create the symlinks that links `$LFS/usr/bin/awk` to `$LFS/usr/bin/mawk`

Installing Find Utils

- Unpack the Find Utils archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`

I'm using version 4.1 and during the compilation I'm getting this error. Although it is a fatal error, the compilation process doesn't stop when the errors occurs, so you need to watch your compilation output closely to find out if you also get the following error: defs.h:304: conflicting types for `basename'

If you're also troubled by that error, here's how to fix it:

- Edit the `find/Makefile` file and find the variable: *CFLAGS*
- Add the value *−D_GNU_SOURCE* to it
- Edit the `find/defs.h` file and file this line: *char *basename P_((char *fname));*
- Replace that line with: *char *basename2 P_((char *fname));*
- Edit the `find/util.c` file and find this line: *char *basename (fname)*

This line is separated over two lines ("char *" is on the first line and "basename (fname)" on the second line).

- Replace that line with: *char *basename2 (fname)*

You don't need to keep it separated on two lines, but if you want that's perfectly OK. Do whatever you think looks best.

Recompile the package (by running make LDFLAGS=−static again) and it should compile correctly this time.

- Copy the following binary from the find directory to $LFS/usr/bin: `find`

Installing Diff Utils

- Unpack the Diff Utils archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`
- Copy the following binaries to $LFS/usr/bin: `diff diff3 sdiff`

Installing Ld.so

- Unpack the Ld.so archive
- Go to the util directory
- Compile ldd by running `make ldd`
- Compile ldconfig by running `make ldconfig`
- Copy the following binary to `$LFS/bin:ldd`
- Copy the following binary to `$LFS/sbin: ldconfig`

Installing Perl

- Unpack the Perl archive
- Configure the package by running `Configure`

You can stick to all the default questions, except to the following.

When asked *What is the file extension used for shared libraries? [so]*

Answer with: *none*

When asked *Any additional ld flags (NOT including libraries)? [−L/usr/local/lib]*

Answer with: *−L/usr/local/lib −static*

When asked *Do you wish to use dynamic loading? [y]*

Answer with: *n*

- Compile the package by running `make`
- Copy the following binary to $LFS/usr/bin: `perl`

Installing M4

- Unpack the M4 archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`
- Copy the following binary from the src directory to $LFS/usr/bin: `m4`

Installing Texinfo

- Unpack the Texinfo archive
- Configure the package by running `configure`
- Compile the package by running `make LDFLAGS=-static`
- Copy the following binary from the makeinfo directory to $LFS/usr/bin: `makeinfo`

Installing Automake

- Unpack the automake archive
- Configure the package by running `configure`
- Copy the following scripts to $LFS/usr/bin: `automake aclocal`
- Create the following directory: `$LFS/usr/share/automake`
- Copy the following files to $LFS/usr/share/automake: `config.guess config.sub install-sh mdate-sh missing mkinstalldirs elisp-comp ylwrap acinstall`
- Copy the following files to $LFS/usr/share/automake: All `*.am` files
- Create the following directory: `$LFS/usr/share/aclocal`

- Copy the following files from the m4 directory to $LFS/usr/share/aclocal: all `*.m4` files

Installing Autoconf

- Unpack the Autoconf archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Copy the following files to $LFS/usr/bin: `autoconf autoheader autoreconf autoscan autoupdate ifnames`
- Create the following directory: `$LFS/usr/share/autoconf`
- Copy the following files to $LFS/usr/share/autoconf: All `*.m4*` files (in effect this means all *m4 and all *.m4f files)
- Also copy these following files to $LFS/usr/share/autoconf: `acconfig.h acfunctions acheaders acidentifiers acprograms acmakevars`

6.2 Installing the GNU C Library

We're not going to installed the latest Glibc version, 2.1.2, 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. And I also don't want to upgrade my working Linux system to gcc 2.95.2 (which is the latest version at the time of writing this document). Upgrading a compiler isn't as easy as it sounds and I don't want to break things on this working system.

So therefore I have to install glibc 2.0.7pre6. However, we are going to install the gcc 2.95.2 compiler. We also need to install the gcc 2.7.2.3 compiler because certain software can't be compiled with gcc 2.95.2 (due to bugs in the programs that aren't really bugs but the gcc 2.95.2 compiler defines them as bugs. This is not a bug in the compiler, but changes in the C standard (if I understood it correctly)).

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
- Move the glibc−crypt and glibc−linuxthreads archives into the unpacked glibc directory
- Unpack the glibc−crypt and glibc−linuxthreads there
- Configure the package by running (from the glibc top level directory) `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
# End configparms
```

- Compile the package by running `make`
- Reboot the computer into the LFS system
- Remount the LFS partition in read−write mode
- Mount the partition where the Glibc source files reside
- If this partition is different from the partition where your /usr directory is usually mounted on, also mount that partition
- Create a symlink that links `/usr/lib/gcc-lib` to the `usr/lib/gcc-lib` directory on the normal Linux system.
- Go to the Glibc source directory
- Install the package by running `make install`
- Remove the `/usr/lib/gcc-lib` symlink

You can check if the GNU C Library seems to be working by running a dynamically linked program that uses this library (like virtual every single program). A simple test is running the `ls` program that resides in the bin directory on your normal Linux partition. If you can run this program without getting any errors, than the GNU C Library seems to be installed correctly.

7. Installing the GNU CC compilers

You need to restart your system back into the normal Linux system to compile the gcc compilers.

7.1 Installing GCC 2.95.2

The GCC Installation notes recommend a separate directory for the object files. This means you have a directory where the gcc sources reside in and a different directory where compiled files are being created. Say you unpacked the gcc archive in /usr/src/gcc−2.95.2 , you could make a directory called /usr/src/gcc−install

- Unpack the GCC archive
- Go to the gcc−install directory
- Configure the package by running `../gcc-2.95.2/configure --prefix=/usr --enable-shared`
- Compile the package by running `make bootstrap`
- Create a file `$LFS/root/test.c` containing the following

```
// Begin test.c
#include <stdio.h>
int main() {
   printf("Hello World!\n");
   return 0;
}
// End test.c
```

* Create a file `$LFS/root/test2.c` containing the following:

```
// Begin test2.c
#include <iostream.h>
int main() {
   cout << "Hello World!" << endl;
   return 0;
}
// End test2.c
```

* Reboot the computer into the LFS system
* Remount the LFS partition in read−write mode
* Mount the partition that contains the gcc−2.95.2 and gcc−install directories
* Create a symlink that links `/usr/src/gcc-2.95.2` to the `usr/src/gcc-2.95.2` directory that contains the gcc source files.
* Go to the gcc−install directory
* Install the package by running `make install`
* Go to the /root directory
* Compile+link `test.c` by running gcc test.c −o test
* Compile+link `test2.c` by running g++ test2.c −o test2
* Run both programs
* If both programs run without errors or crashing than the compilers and linkers seem to be working
* Remove the `/usr/src/gcc-2.95.2` symlink

7.2 Installing GCC 2.7.2.3

* Unpack the gcc−2.7.2.3 archive
* Configure the package by running `configure --prefix=/usr/gcc2723 --enable-shared`
* If the configure script says it can't determine the system type, then configure the package by running `configure --host=<cpu>-linux-gnu --prefix=/usr/gcc2723 --enable-shared`

Enter the right value for <cpu>. If you have a Pentium, enter i586. If you have a PII or higher, enter i686. If you don't have an Intel based platform than you have to figure out yourself what to enter since the only experience I have are with Intel based platforms (sorry, but I just don't have the money to buy myself an alpha, SPARC or whatever system ;)

* Build the compiler by executing these commands in sequence:

```
make LANGUAGES=c
make stage1 (ignore errors about files not found)
make CC="stage1/xgcc -Bstage1/" CFLAGS="-g -O2"
make stage2
make CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O2"
make compare
```

- If make compare doesn't report any differences, the compiler is build successfully.
- Install the package by running `make install CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O2"`

8. Installing login utilities

8.1 Installing agetty + login

This parts needs to be done using the normal Linux system since we need a text editor. You need the Util Linux package again for this section. If you haven't deleted the Util Linux source directory, you can skip the first two steps.

- Unpack the Util Linux archive (if you have deleted it since last time we've used it)
- Configure the package by running `configure`
- Go to the login–utils directory
- Compile agetty and login by running `make agetty login`
- Copy the following binary to $LFS/sbin: `agetty`
- Copy the following binary to $LFS/bin: `login`

8.2 Modifying $LFS/etc/inittab

The next step is modifying the $LFS/etc/inittab file so that agetty is started on a virtual console every time we start the system. This is how it works on most if not every Linux system.

- Edit the `$LFS/etc/inittab` file
- Find this line and remove it: *1:2345:respawn:/sbin/sulogin*
- Where the previous line was, put the following lines:

```
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
```

8.3 Creating the UTMP record file

Every time you logon to a Linux system, the /var/run/utmp file is modified. When this file isn't present, a lot of programs will start complaining, including agetty and login. So we just create an empty $LFS/var/run/utmp file and those programs won't complain anymore.

- Create the `$LFS/var/run` directory
- Create an empty file by running `touch $LFS/var/run/utmp`

8.4 Testing the system

If you want you can test the system now. Restart the system and boot into the LFS system. After the kernel and sysvinit are done loading, agetty should start and prompt you with a username. Since the only user we currently have is 'root', you login as root.

9. Installing Vim

After Vim is installed, we don't need to use our normal Linux system anymore to "dress up" our LFS system. This means that you from now on no longer need to reboot your computer into the normal Linux system and back to the LFS system.

9.1 Preparing the system for the Vim installation

Installing Ncurses

In order to install Vim we need to have the ncurses libraries installed.

- Unpack the Ncurses archive
- Configure the package by running `configure --with-shared`

Because Ncurses isn't 100% correct according to the latest C standard, we need to compile it with gcc2723

- Compile the package by running `make CC=/usr/gcc2723/bin/gcc`
- Install the terminfo files by running `make install.data`
- Go to the test directory and run a few of the programs to verify that the libraries are working
- Install the libraries by running `make install`

9.2 Installing Vim

Vim comes in two separate parts: A 'src' package and a 'rt' (run–time) package. You need both in order to install Vim. If you put both archives in the same directory, the unpacked files of both archives will appear in the same directory that will be created when you unpack the first (it doesn't matter which one you unpack first).

- Unpack the Vim−src and the Vim−rt archives
- Configure the package by running `configure`

Also Vim doesn't compile with gcc 2.95.2, so we have to compile it with gcc 2.7.2.3 as well.

- Compile the package by running `make CC=/usr/gcc2723/bin/gcc`
- Install the package by running `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
   if [ ! -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. Reinstalling statically linked software

In this section we're going to reinstall all software that has been linked statically before dynamically. It's pretty straightforward like it was when we prepared our system for the Glibc installation.

It's important that you take a close look at this section. If you decide you can't be bothered reinstalling all the previously installed software, at least look at the new libraries and programs in this section. A few programs that are already installed depend on certain libraries when dynamically linked. But these libraries aren't only used by the already installed programs; other software might require it as well, so you want to install those. Also, a few programs recommend other programs to be installed. We didn't require those programs for the Glibc and GCC installation, but we might as well install them now to avoid problems later.

11.1 Installing the Termcap library

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

11.2 Installing the Readline library

- Unpack the Readline archive
- Configure the package by running `configure`
- Compile the package by running `make CC=/usr/gcc2723/bin/gcc SHOBJ_CC=/usr/gcc2723/bin/gcc shared`
- Install the package by running `make CC=/usr/gcc2723/bin/gcc install`
- Install the shared libraries by running `make install-shared`

11.3 Reinstalling Bash

- Unpack the Bash archive
- Configure the package by running `configure --with-installed-readline`
- Compile the package by running `make CC=/usr/gcc2723/bin/gcc`
- Edit the `Makefile` file and find the variable: *bindir*
- Replace the current value with: */bin*
- Install the package by running `make install`

The just installed Bash version is compiled with the −g compiler flag, which means it's compiled with debugging information. This means that when you ever need to run bash through a debugger, the output is human readable, whereas a binary compiled without debugger information is very hard to debug. The downside is that the Bash executable is now about 1MB in size. If you remove the debug information, you'll have an executable of around 340KB in size. This is quite a difference and worth it if you don't debug programs at all.

You can edit the Makefile files whenever you compile a program so you can remove the −g compiler flags (often found in a CFLAGS variable), or you can run the strip program with one or more executables as the parameter(s). All debugging information will be deleted (this won't affect the program itself in any way whatsoever). The choice is yours.

11.4 Reinstalling Sysvinit

- Unpack the Sysvinit package
- Go to the src directory
- Compile the package by running `make`
- Install the package by running `make install`

11.5 Reinstalling Make

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

11.6 Reinstalling Sed

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

11.7 Reinstalling Shell Utils

- Unpack the Shell Utils archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Copy the following binaries from the src directory to /bin: `date echo false pwd sleep stty su true uname`
- Copy the following binary from the src directory to /sbin: `chroot`
- Copy the following binaries from the src directory to /usr/bin: `basename dirname env expr factor groups id logname nice nohup patchchk printenv printf seq tee test tty uptime users who whoami yes`

11.8 Reinstalling File Utils

- Unpack the File Utils archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Edit the `Makefile` file
- Find the following variables: *bindir sbindir sysconfdir localstatedir*
- Remove the *$(exec_prefix)* and *$(prefix)* parts so you'll be left with the values: */bin /sbin /etc* and */var*
- Install the package by running `make install`
- Move the `/bin/install` file to the `/usr/bin` directory

11.9 Reinstalling + Installing Util Linux

- Unpack the Util Linux archive
- Configure the package by running `configure`
- Go to the lib directory

* Compile the files there by running `make`
* Go to the fdisk directory
* Compile fdisk by running `make fdisk`
* Compile cfdisk by running `make cfdisk`
* Copy the following binaries to /sbin: `cfdisk fdisk`
* Copy the following files to /usr/man/man8: `cfdisk.8 fdisk.8`
* Go to the login–utils directory
* Copy the following file to /usr/man/man1: `login.1`
* Copy the following file to /usr/man/man8: `agetty.8`
* Go to the mount directory
* Compile the utilities by running `make`
* Copy the following binaries to /sbin: `mount umount swapon losetup`
* Copy the following files to /usr/man/man8: All `*.8` files
* Remove the /sbin/swapoff symlink and recreate the symlink that links `/sbin/swapoff` to `/sbin/swapon`
* Go to the sys–utils directory
* Compile dmesg by running `make dmesg`
* Compile rdev by running `make rdev`
* Copy the following binary to /bin: `dmesg`
* Copy the following binary to /sbin: `rdev`
* Copy the following files to /usr/man/man8: `dmesg.8 rdev.8 swapdev.8 ramsize.8 vidmode.8 rootflags.8`
* Create the symlinks that link `/sbin/rdev, /sbin/swapdev, /sbin/ramsize, /sbin/vidmode` and `/sbin/rootflags` to `/sbin/rdev`
* Go to the text–utils directory
* Compile more by running `make more MOREHELPDIR=/usr/share/more`
* Copy the following binary to /usr/bin: `more`
* Copy the following file to /usr/man/man1: `more.1`
* Create the `/usr/share/more` directory
* Copy the following file to /usr/share/more: `more.help`

11.10 Reinstalling Text Utils

* Unpack the Text Utils archive
* Configure the package by running `configure`
* Compile the package by running `make`
* Edit the `src/Makefile` file and find the variable: *bindir*
* Replace the current value with: */usr/bin*
* Install the package by running `make install`
* Move the /usr/bin/cat file to `/bin/cat`

11.11 Reinstalling Tar

* Unpack the Tar archive
* Configure the package by running `configure`
* Compile the package by running `make`
* Edit the src/Makefile file and find the variables: *bindir* and *libexecdir*

- Give bindir the value: */bin*
- Give libexecdir the value: */usr/bin*
- Install the package by running `make install`
- If you don't need the ReMote Tapeserver program, you can delete the `/usr/bin/rmt` program

11.12 Reinstalling Gzip

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

I'm using version 1.2.4 and during the compilation process I'm getting this error: conflicting types for basename. If you're also being troubled by this error, here's how to fix it:

- Edit the `gzip.h` file and find this line: *extern char *basename OF((char *fname));*
- Replace this line with: *extern char *basename2 OF((char *fname));*
- Edit the `util.c` file and find the line: *char *basename(fname)*
- Replace this line with: *char *basename2(fname)*

Recompile the package now (with make) and the compilation process should finish properly this time

- Edit the `Makefile` file and find the variable: bindir
- Replace the current value with: /bin
- Install the package by running `make install`

11.13 Reinstalling Bison

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

11.14 Installing Flex

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

11.15 Reinstalling Binutils

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

11.16 Reinstalling Grep

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

11.17 Reinstalling Mawk

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

11.18 Reinstalling Find Utils

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

I'm using version 4.1 and during the compilation I'm getting this error. Although it is a fatal error, the compilation process doesn't stop when the errors occurs, so you need to watch your compilation output closely to find out if you also get the following error: defs.h:304: conflicting types for `basename'. If you're also troubled by that error, here's how to fix it:

- Edit the `find/Makefile` file and find the variable: *CFLAGS*
- Add the value: *−D_GNU_SOURCE*
- Edit the `find/defs.h` file and find this line: *char *basename P_ ((char *fname));*
- Replace this line with: *char *basename2 P_ ((char *fname));*
- Edit the `find/util.c` file and find this line: *char *basename (fname)*

This line is separated over two lines ("char *" is on the first line and "basename(fname)" on the second line).

- Replace this line with: *char *basename2(fname)*

You don't need to keep this line separated over two lines. It doesn't matter at all whether you keep it like that or not.

Recompile the package (with make) and the compilation process should finish properly this time.

- Install the package by running `make install`

11.19 Reinstalling Diff Utils

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

11.20 Installing Less

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

11.21 Reinstalling Perl

- Unpack the Perl archive
- Configure the package by running `Configure`

If you agree on all default values, you might want to configure the package by running `Configure -d`. This way you don't have to press enter all the time to accept the default values.

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

11.22 Reinstalling M4

- Unpack the M4 archive
- Configure the package by running `configure`
- Compile the package by running `make`
- Install the package by running `make install`

11.23 Reinstalling Texinfo

- Unpack the Texinfo archive
- Configure the package by running `configure`
- Compile the package by running `make CC=/usr/gcc2723/bin/gcc`
- Install the package by running `make install`

12. 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.

12.1 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`

12.2 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`

12.3 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`

12.4 Installing Modutils

* Unpack the Modutils archive
* Configure the package by running `configure`
* Compile the package by running `make CC=/usr/gcc2723/bin/gcc`
* Install the package by running make install

12.5 Installing Linux86

This package will only be used, as far as I can tell and know, for the installation of Lilo which will be installed next. So you could remove the two programs as86 and ld86 after you've installed Lilo.

* 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`

12.6 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

12.7 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`

12.8 Installing Sysklogd

Installing Sysklogd

- Unpack the Sysklogd archive
- Compile the package by running `make CC=/usr/gcc2723/bin/gcc`
- 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
```

12.9 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`

12.10 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`

12.11 Installing Procps

* Compile the package by running make CC=/usr/gcc2723/bin/gcc
* Edit the Makefile file and comment out the variable: *XSCPT*
* Install the package by running `make install`

12.12 Installing Procinfo

* Compile the package by running `make CC=/usr/gcc2723/bin/gcc`
* Install the package by running `make install`

12.13 Installing Procmisc

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

12.14 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 ;)

12.15 Installing GNU C++ Library

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

The installation by running make install right now will fail because it can't find all the header files that need to be copied to /usr/include/g++−v3. The thing is, the installation script tries to find the files in the src/bits src/shadow src/ext and src/backwards directories. The files are actually in the bits, shadow, ext and backwards directories in the top−level directory. I don't know who to blame; the make program, or the Makefile file. Either way, by making a few symlinks and copying some extra header files to a different directory the installation will finish properly.

To setup up the directories and file in such a way that the Makefile script can find them, execute the following commands from within the src directory:

```
ln −s ../bits bits
ln −s ../backward backward
ln −s ../ext ext
ln −s ../shadow shadow
cp ../stl/bits/* bits
cp ../stl/backward/* backward
cp ../stl/ext/* ext
```

Now that the files are in a place where they can be found during make install, we can proceed with this step.

- Install the package by running `make install`

13. Setting up basic networking

13.1 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`

13.2 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. Below a quoted paragraph from O'Reilly's book "Linux Network Administrator's Guide"

−−− Begin quote −−−

If your network is not connected to the Internet and won't be in the near future, you are free to choose any legal network address. Just make sure no packets from your internal network escape to the real Internet. To make sure no harm is done, even when packets did escape, you should use one of the network numbers reserved for private use. The Internet Assigned Numbers Authority (IANA) has set aside several network numbers from classes A, B and C that you can use without registering. These addresses are only valid within your private network and are not routed between Internet sites.

The numbers 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
```

−−− End quote −−−

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.

14. Setting up Email sub system

14.1 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`

14.2 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`

14.3 Installing Sendmail

Installing Sendmail

* Unpack the Sendmail archive
* Go to the src directory
* Compile the package by running `Build CC=/usr/gcc2723/bin/gcc`
* 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`

14.4 Installing Mailx

* Unpack the Mailx archive
* Compile the package by running `make *.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`

14.5 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 "$i" 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
```

14.6 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
```

14.7 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`

14.8 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`

14.9 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

15. 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

15.1 Installing telnet daemon + client

- Unpack the Netkit−telnet archive
- Configure the package by running `configure --with-c-compiler=/usr/gcc2723/bin/gcc --with-c++-compiler=/usr/gcc2723/bin/c++`
- Compile the package by running `make`
- Install the package by running `make install`

15.2 Installing Proftpd

- Unpack the Proftpd archive
- Compile the package by running `make CC=/usr/gcc2723/bin/gcc`
- Install the package by running `make install`

15.3 Installing Netkit−ftp

- Unpack the Netkit−ftp archive
- Configure the package by running `configure`
- Compile the package by running `make CC=/usr/gcc2723/bin/gcc`
- Install the package by running `make install`

15.4 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
- Compile the package by running `make CC=/usr/gcc2723/bin/gcc`
- Install the package by running `make install`

15.5 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_CC=/usr/gcc2723/gcc elf`
- Install the package by running `make CC=/usr/gcc2723/bin/gcc install-elf`
- Create extra symlinks for the library by running `make install-links`

15.6 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`

15.7 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 CC=/usr/gcc2723/bin/gcc`
- Install the package by running `make install`
- Install the helpfile by running `make install-help`
- Install other documentation by running `make install-doc`

15.8 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.

15.9 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
```

15.10 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
```

15.11 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
```

15.12 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.

16. Installing X Window System

16.1 Creating missing symlink

On my system the symlink /lib/cpp that is supposed to point to /usr/bin/cpp was missing for some reason. Perhaps it never was there or I deleted it by mistake I don't know. Check if the link is in place on your system. If not, re–create it by running ln –s /usr/bin/cpp /lib/cpp

16.2 Installing X

- Unpack the X archive
- Compile the package by running `make CC=/usr/gcc2723/bin/gcc 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`

16.3 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`

16.4 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*

16.5 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`

16.6 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`

16.7 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`

16.8 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.

16.9 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.

17. 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.

17.1 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 en 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 CC=/usr/gcc2723/bin/gcc`
- Install the package by running `make install`

17.2 Updating dynamic loader cache

- Update the dynamic loader cache by running `ldconfig`

17.3 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`

17.4 Testing WindowMaker

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

18. Configuring system for Internet

18.1 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.

18.2 Creating groups and directories

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

18.3 Installing PPP

- Unpack the PPP archive
- Configure the package by running `configure`
- Compile the package by running `make CC=/usr/gcc2723/bin/gcc`
- Install the package by running `make install`

18.4 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
```

18.5 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
```

18.6 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
```

18.7 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
```

18.8 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.

18.9 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/

19. Copyright & Licensing Information