Linux−From−Scratch−HOWTO

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Linux From Scratch

Gerard Beekmans

Version 2.2, April 3rd, 2000

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

2.Conventions used in this document

3.Packages you need to download

4.Preparing the new system

5.Making the LFS system bootable

6.Installing a kernel

7.Installing basic system software

8.Creating system boot scripts

9.Setting up basic networking

10.Installing Network Daemons

11.Installing Network Clients

12.Installing X Window System

13.Resources

14.The End

15.Copyright & Licensing Information

1.Introduction

1.1 What's this all about?

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

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

1.2 New versions

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

1.3 Version history

2.1.5 − March 26th, 2000

This is not a full list of modified things. Because v2.0 is a major release, only the major changes are mentioned and not the minor ones.

- Directory structure modified − LFS is FHS compliant now. Perhaps not 100%, but getting there.

* New Glibc installation method
* New GCC installation method
* Eliminated the need for the pre−compiled Debian packages.
* Totally revised software installation method − eliminated the need of all the statically linked packages in former chapter 6.1.
* Various bugs fixed in software installation
* Installed a few more programs from the util−linux package
* Added the installation of the Bzip2 program
* Explained in greater detail what the $LFS is all about − how to and how not to use it.
* Simplified installation procedures for all packages in chapters 5 through 9.1
* Moved the installation of Glibc and GCC to chapter 7 in stead of having their own chapters which isn't necessary.
* Modified Internet servers chapter: separated into Network Daemons and Network Clients chapter. Internet chapter has merged with these two new chapters.
* Switched chapters 13 and 14 (X and Internet) and merged the chapters about X and Window Maker into chapter 14.
* We're using a new Man program. This one is easier to use and configure than the man−db one we previously used. Both versions perform nearly the same jobs.
* Added new chapter 13: Resources. This chapter contains a number of books and HOWTOs you'll find useful to read during or after building an LFS system.
* Chapter 3: Fixed bzip2 link
* Chapter 7.2.42: Simplified Util−Linux installation method
* Chapter 3.1: Changed procps location
* Chapter 7.2: Switched installation of Vim and Util−Linux (as we need an editor to install Util−Linux)
* Chapter 7.3.33: Fixed procps installation.
* Chapter 5.2: stripped inittab file so it won't complain about missing files at boot time.
* Chapter 6: Rewrote kernel installation
* Chapter 10.3: Fixed Apache bootscript
* Chapter 10.3.2: Removed section about modifying the httpd.conf file. No longer necessary. Added the addition of /usr/apache/man to the /usr/share/misc/man.conf file
* Chapter 11.1: Provided a fixed mailx package with a working Makefile file to simplify the installation procedure
* Chapter 11.3.1: Added the −−shared switch to configure so that Zlib is installed as a dynamic library rather than a static one.
* Chapter 11.6: Have Lynx link against the Ncurses library in stead of the Slang.
* Chapter 12: The newer man−db already has the X11/man directory in it's man_db.config file

2.2 − April 3rd, 2000

* The linuxfromscratch.org and linuxfromscratch.com domains are now operational. All former links to huizen.dds.nl/~glb and tts.ookhoi.dds.nl have been replaced by the appropriate links on www.linuxfromscratch.org
* After the reboot in chapter 7.3, the swap partition is made active before we start compiling software.

1.4 Mailinglists

There are two mailing lists 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 document. The latter is an open moderated list. Anybody can subscribe to it, but you cannot post messages to it (only the moderator(s) can do this). This list is primarily used for announcements of new versions of this document.

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 mailing list. If you need to reach me personally, send an email to

---

2.Conventions used in this document

2.1 About $LFS

Please read the following carefully: throughout this document you will frequently see the variable name $LFS. $LFS must at all times be replaced by the directory where the partition that contains the LFS system is mounted. How to create and where to mount the partition will be explained later on in full detail in chapter 4. In my case the LFS partition is mounted on /mnt/hda5. If I read this document myself and I see $LFS somewhere, I will pretend that I read /mnt/hda5. If I read that I have to run this command: `cp inittab $LFS/etc` I actually will run this: `cp inittab /mnt/hda5/etc`

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

If you want, you can set the environment variable LFS. This way you can literally enter $LFS in stead of replacing it by something like /mnt/hda5. This is accomplished by running: export LFS=/mnt/hda5

If I read cp inittab $LFS/etc, I literally can type cp inittab $LFS/etc and the shell will replace this command by cp inittab /mnt/hda5/etc automatically.

Do not forget to set the LFS variable at all times. If you haven't set the variable and you use it in a command, $LFS will be ignored and whatever is left will be executed. The command cp inittab $LFS/etc without the LFS variable set, will result in copying the inittab file to the /etc directory which will overwrite your system's inittab. A file like inittab isn't that big a problem as it can easily be restored, but if you would make this mistake during the installation of the C Library, you can break your system badly and might have to reinstall it if you don't know how to repair it. So that's why I strongly advise against using the LFS variable. You better replace $LFS yourself by something like /mnt/hda5. If you make a typo while entering /mnt/hda5, the worst thing that can happen is that you'll get an error saying "no such file or directory" but it won't break your system. Don't say I didn't warn you ;)

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, 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 package 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 document assumes that you unpack the archives under the $LFS/usr/src directory). You have to enter that new directory before you continue with the installation instructions. All the above will be summarized as 'Unpack the xxx archive'. So, when you read 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.

After you have installed a package you can do two things with it. You can either delete the directory that contains the sources or you can keep it. If you decide to keep it, that's fine by me. But if you need the same package again in a later chapter (all software up to chapter 7.2 will be re−installed in chapter 7.3) you need to delete the directory first before using it again. If you don't do this, you might end up in trouble because old settings will be used (settings that apply to your normal Linux system but which don't apply anymore when you have restarted your computer into the LFS system). Doing a `simple make` clean does not always guarantee a totally clean source tree. The configure script also has files lying around in various subdirectories which are rarely removed by the make clean process.

3.Packages you need to download

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

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/pub/linux/kernel/

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

Bzip2 (0.9.5d) : http://sourceware.cygnus.com/bzip2/

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

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

GCC (2.95.2) : ftp://ftp.gnu.org/gnu/gcc/

Glibc (2.1.3) : ftp://ftp.gnu.org/gnu/glibc/

Glibc−crypt (2.1.2) : ftp://ftp.gwdg.de/pub/linux/glibc/

Glibc−linuxthreads (2.1.3) : ftp://ftp.gnu.org/gnu/glibc/

Grep (2.4) : ftp://ftp.gnu.org/gnu/grep/

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

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

Sed (3.02) : ftp://ftp.gnu.org/gnu/sed/

Shell Utils (2.0) : ftp://ftp.gnu.org/gnu/sh−utils/

Tar (1.13) : ftp://ftp.gnu.org/gnu/tar/

Text Utils (2.0) : ftp://ftp.gnu.org/gnu/textutils/

Util Linux (2.10f) : ftp://ftp.win.tue.nl/pub/linux/utils/util−linux/

Bison (1.28) : ftp://ftp.gnu.org/gnu/bison/

Mawk (1.3.3) : ftp://ftp.whidbey.net/pub/brennan/

Find Utils (4.1) : ftp://ftp.gnu.org/gnu/findutils/

Ncurses (5.0) : ftp://ftp.gnu.org/gnu/ncurses/

Less (340) : ftp://ftp.gnu.org/gnu/less/

Perl (5.005_03) : ftp://ftp.gnu.org/gnu/perl/

M4 (1.4) : ftp://ftp.gnu.org/gnu/m4/

Texinfo (4.0) : ftp://ftp.gnu.org/gnu/texinfo/

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Vim (5.6) : ftp://ftp.vim.org/pub/editors/vim/unix/

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 (0.17) : ftp://ftp.uk.linux.org/pub/linux/Networking/netkit−devel/

Net−tools (1.54) : http://www.tazenda.demon.co.uk/phil/net−tools/

Procmail (3.14) : ftp://ftp.procmail.org/pub/procmail/

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

Mailx (8.1.1) : http://www.linuxfromscratch.org/download/mailx−8.1.1−fixed.tar.gz

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

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

Netkit−telnet (0.17) : ftp://ftp.uk.linux.org/pub/linux/Networking/netkit−devel/

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

Netkit−ftp (0.17) : ftp://ftp.uk.linux.org/pub/linux/Networking/netkit−devel/

Apache (1.3.11) : http://www.apache.org/dist/

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

Lynx (2.8.2) : http://www.slcc.edu/lynx/release/

PPP (2.3.11) : ftp://cs.anu.edu.au/pub/software/ppp/

Xfree86 (3.3.5) : ftp://ftp.xfree86.org/pub/XFree86/

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

libXpm (4.7) : ftp://sunsite.unc.edu/pub/Linux/libs/X/

libpng (1.0.3) : http://www.cdrom.com/pub/png/

libtiff (3.4) : ftp://ftp.sgi.com/graphics/tiff/

libjpeg (6b) : http://www.ijg.org/

libungif (4.1.0) : ftp://prtr−13.ucsc.edu/pub/libungif/

WindowMaker (0.61.1) :

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 document to use right away. I suggest you first install a Linux distribution. It really doesn't matter which one you install. It also doesn't need to be the latest version, though it shouldn't be a too old one. If it is about a year old or newer it'll do just fine. You will safe yourself a lot of trouble if your normal system uses glibc−2.0 or newer. Libc5 can cause some problems and is not supported in this document as I don't have access to such a machine anymore.

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

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 document will assume that you have mounted the partition on a subdirectory under /mnt. It doesn't matter which subdirectory you choose (or you can use just the /mnt directory as the mounting point), but a good practice is to create a directory with the same name as the partition's designation. In my case the LFS partition is called hda5 and therefore I mount it on /mnt/hda5

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

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

4.5 Creating directories

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

```
cd $LFS
mkdir bin boot dev etc home lib mnt proc root sbin tmp usr var
cd $LFS/usr
mkdir bin include lib sbin share src
ln −s share/man man
ln −s share/doc doc
ln −s . local
ln −s ../etc etc
ln −s ../var var
cd $LFS/usr/share
mkdir dict doc info locale man nls misc terminfo zoneinfo
```

```
cd $LFS/usr/share/man
mkdir man1 man2 man3 man4 man5 man6 man7 man8
cd $LFS/var
mkdir lock log run spool tmp
```

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

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
chown root.root $LFS/dev/*
```

5.Making the LFS system bootable

5.1 Installing Sysvinit

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

- Unpack the Sysvinit archive
- Enter the src directory
- 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 −e 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:

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 the Bash shell

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

- Unpack the Bash archive
- Install Bash by running:

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

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. Then enter root's password and you should be on the bash−prompt now. You won't be able to shutdown the system with a program like shutdown. Although the program is present, it will give you the following error: "You don't exist. Go away." when you try to use the program. The meaning of this error is that the system isn't able to locate the password file. Although the shutdown program is statically linked against the libraries it needs, it still depends on the NSS Library (Name Server Switch) which is part of the GNU C Library, which will be installed in a later chapter. This NSS library passes on information where (in this case) the passwd file can be found.

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

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

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

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

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

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

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

7.1 About debugging symbols

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

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

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

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

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

7.2 Preparing LFS system for installing basic system software

Installing Binutils

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

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

Installing Bzip2

- Unpack the Bzip2 archive
- Edit the Makefile file in a text editor
- Find the lines that start with : $(CC) $(CFLAGS) −o
- Replace those parts with: $(CC) $(CFLAGS) $(LDFLAGS) −o
- Install the package by running:

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

Install Diffutils

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

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

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

Installing Fileutils

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

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

Installing GCC on the normal system if necessary

In order to compile Glibc−2.1.3 you need to have gcc−2.95.2 installed. Any version from 2.8 and up would do, but 2.95.2 is recommended. Many glibc−2.0 based systems have gcc−2.7.2.3 installed and you can't compile glibc−2.1 with that compiler. Therefore we will install gcc−2.95.2. also on the normal system, but without overwriting the existing compiler. Before you install gcc on your normal system, make sure whether you need it or not. Run `gcc --version` and check if the version number it reports equals or is higher than 2.8. If not, you need to install gcc−2.95.2. If you experience difficulties compiling glibc later on, you might want to install gcc−2.95.2 anyways.

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

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

Installing GCC on the LFS system

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

```
mkdir $LFS/usr/src/gcc-build;cd $LFS/usr/src/gcc-build
../gcc-2.95.2/configure --enable-languages=c --disable-nls
make -e LDFLAGS=-static bootstrap
make -e prefix=$LFS/usr local_prefix=$LFS/usr install
```

Creating necessary symlinks

The system needs a few symlinks to ensure every program is able to find the compiler and the pre−processor. Some programs run the cc program, others run the gcc program, some programs expect the cpp program 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/lib; ln −s ../usr/lib/gcc−lib/<host>/2.95.2/cpp cpp
cd $LFS/usr/lib; ln −s gcc−lib/<host>/2.95.2/cpp cpp
cd $LFS/usr/bin; ln −s gcc cc
```

Replace <host> with the directory where the gcc−2.95.2 files were installed (i686−unknown−linux in my case). You will most likely find two different directories.

Installing Glibc

A note on the glibc−crypt package:

```
−*−*−*−*−*−
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/. 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.
- Create a new file `configparms` containing:

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

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

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

* If your normal didn't had a suitable gcc version and you installed gcc−2.95.2 on your normal system, install the package by running:

```
mkdir $LFS/usr/src/glibc-build; cd $LFS/usr/src/glibc-build
CC=/usr/gcc2952/bin/gcc ../glibc-2.1.3/configure --enable-add-ons
make; make install_root=$LFS install
```

Copying old NSS Library files

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

Your system uses glibc−2.0 if the output looks like: /lib/libc.so.6 −> libc−2.0.7.so

Your system uses glibc−2.1 is the output looks like: /lib/libc.so.6 −> libc−2.1.2.so

If your have a libc−2.0.x.so file (where x is the micro version number such as 7) copy the NSS Library files by running: `cp -av /lib/*nss* $LFS/lib`

Installing grep

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

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

This package is known to cause static linking problems on certain platforms. If you're having trouble compiling this package as well, you can download a fixed package from http://www.linuxfromscratch.org/download/grep−2.4−fixed.tar.gz

Installing gzip

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

```
./configure
make −e LDFLAGS=−static
make −e prefix=$LFS/usr install
cd $LFS/usr/bin
mv gunzip gzip $LFS/bin
```

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

Installing Make

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

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

Installing Sed

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

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

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

Installing Sh−utils

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

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

Installing Tar

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

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

Installing Textutils

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

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

Installing Util−linux

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

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

7.3 Installing basic systsem software

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

confusion and errors. Before you continue with this document you have to restart your system and boot into the LFS system. But before you do that, you need to determine which partition is used as your swap partition. This information can usually be found in the /etc/fstab file. Check this file for a line similar to this one:
/dev/hda6 none swap sw 0 0

The 4th field in a line must contain 'sw'. That line represents a swap partition. All you need to remember is it's designation (which is /dev/hda6 in my case but this will probably be different on your system). When you have determined which partition is the swap partition, you can reboot your computer now and continue from here.

Remounting partition and activating swap

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

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

Installing GCC

- Unpack the GCC archive and install it by running:

```
mkdir $LFS/usr/src/gcc-build;cd $LFS/usr/src/gcc-build
../gcc-2.95.2/configure --with-gxx-include-dir=/usr/include/g++ \
--enable-shared --enable-languages=c,c++
make bootstrap; make install
```

Installing Bison

- Unpack the bison archive and install it by running:

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

Installing Mawk

- Unpack the mawk archive and install it by running:

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

Installing Findutils

* Unpack the findutils archive and install it by running:

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

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

Installing Termcap

* Unpack the Termcap archive and install it by running:

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

Installing Ncurses

* Unpack the ncurses archive and install it by running:

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

Installing Less

* Unpack the Less archive and install it by running:

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

Installing Perl

* Unpack the Perl archive and install it by running:

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

Note that we skip the 'make test' step. This is because at 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
mv /usr/bin/bash /bin
```

Installing Flex

* Unpack the Flex archive and install it by running:

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

Installing Binutils

* Unpack the Binutils archive and install it by running:

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

Installing Bzip2

* Unpack the Bzip2 archive and install it by running:

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

Installing 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
mv /usr/sbin/mklost+found /sbin
```

Installing File

- Unpack the File archive and install it by running:

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

Installing Fileutils

- Unpack the Fileutils archive and install it by running:

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

Installing Grep

- Unpack the Grep archive and install it by running:

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

Installing Groff

- Unpack the Groff archive and install it by running:

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

Installing Gzip

- Unpack the Gzip archive and install it by running:

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

Installing Ld.so

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

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

Installing Libtool

- Unpack the Libtool archive and install it by running:

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

Installing Linux86

- Unpack the Linux86 archive and install it by running:

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

Installing Lilo

- Unpack the Lilo archive and install it by running:

```
make; make install
```

Installing Make

* Unpack the Make archive and install it by running:

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

Instaling Sh−Utils

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

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

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
cd /usr/sbin
mv chpasswd dpasswd groupadd groupdel groupmod logoutd mkpasswd \
newusers useradd userdel usermod grpck pwck vipw grpconv grpunconv \
pwconv pwunconv /sbin
```

Installing Man

* Unpack the Man archive and install it by running:

```
./configure −default
make all; make install
```

Installing Modutils

* Unpack the Modutils 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:

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

Installing Psmisc

* Unpack the Psmisc archive and install it by running:

```
make; make install
```

Installing Sed

* Unpack the Sed archive and install it by running:

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

Installing start−stop−daemon

* Unpack the start−stop−daemon archive and install it by running:

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

Installing Sysklogd

* Unpack the Sysklogd archive and install it by running:

```
make; make install
```

Installing Sysvinit

* Unpack the Sysvinit archive and install it by running:

```
cd src
make; make install
```

Install Tar

* Unpack the Tar archive and install it by running:

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

Installing Textutils

* Unpack the Textutils archive and install it by running:

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

Installing Vim

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

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

Installing Util−linux

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

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

- Install the package by running:

```
groupadd -g 5 tty
./configure
make; make install
```

7.4 Removing old NSS Library files

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

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

7.5 Configuring the software

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

Configuring Glib

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

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

```
# Begin /etc/nsswitch.conf
passwd: files
group: files
shadow: files

hosts: files dns
networks: files

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

netgroup: db files
# End /etc/nsswitch.conf
```

* Run the tzselect script and answer the questions regarding your timezone
* When you're done, the program will give you the file location you need.
* Create the localtime symlink by running: ln −s /usr/share/zoneinfo/<tzselect's output> /etc/localtime

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

Configuring LILO

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

* Create the /mnt/original directory
* Mount your normal Linux system on this mount point by running mount /dev/xxx /mnt/original (replace /dev/xxx with your normal partition's designation).
* Copy the lilo configuration file and kernel images that lilo uses by running:

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

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

Configuring Sysklogd

* Create the /var/log directory by running: mkdir /var/log
* Create the /etc/syslog.conf file containing the following:

```
# Begin /etc/syslog.conf

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

# End /etc/syslog.conf
```

Configuring Shadow Password Suite

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

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 (at least not under Linux), but you can avoid a hassle by testing properly and reading manuals ;)

Configuring Sysvinit

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

* Edit the /etc/inittab file and modify it so it contains the following:

```
# Begin /etc/inittab
```

```
id:2:initdefault:

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

su:S:wait:/sbin/sulogin

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

ft:6:respawn:/sbin/sulogin

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

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

# End /etc/inittab
```

Creating /var/run/utmp file

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

- Create the /var/run/utmp file by running: touch /var/run/utmp
- Give it the proper file permissions by running: chmod 644 /var/run/utmp

---

8.Creating system boot scripts

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

8.1 Preparing the directories and master files

You need the Sysvinit package again for this section.

- Create the necessary directories by running:

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

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

8.2 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
```

8.3 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
```

8.4 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..."
/bin/mount -n -o remount,rw /
check_status

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

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

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

8.5 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 "Deactivating swap..."
/bin/swapoff -av
check_status

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

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

8.6 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
```

8.7 Creating the checkroot bootscript

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

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

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

    if [ $? -gt 1 ]
    then
      echo
      echo "fsck failed. Please repair your file system manually by"
      echo "running fsck without the -a option"

      echo "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
```

8.8 Creating the Sysklogd bootscript

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

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

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

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

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

```
    check_status
    ;;

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

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

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

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

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

    sleep 1

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

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

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

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

8.9 Setting up symlinks and permissions

- Set the proper file permissions and symlinks by running:

```
chmod 755 rcS reboot halt mountfs umountfs sendsignals checkroot sysklogd
cd ../rc0.d
ln -s ../init.d/sysklogd K90sysklogd
ln -s ../init.d/sendsignals S80sendsignals
ln -s ../init.d/umountfs S90umountfs
```

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

8.10 Creating the /etc/fstab file

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

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

9.Setting up basic networking

9.1 Installing Netkit−base

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

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

9.2 Installing Net−tools

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

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

Creating the /etc/init.d/localnet bootscript

* Create a new file `/etc/init.d/localnet` containing 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/localnet S03localnet`

Creating the /etc/hostname file

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

Creating the /etc/hosts file

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

```
<myip> myhost.mydomain.org aliases
```

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

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

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

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

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.

9.3 Testing the system

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

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

10.Installing Network Daemons

10.1 Setting up SMTP

Creating groups and user

Create the groups needed by Sendmail by running:

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

Creating directory

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

```
mkdir /var/spool
mkdir /var/mail
cd /var/spool; ln −s ../mail mail
chmod 700 /var/spool/mqueue
chmod 775 /var/mail
chgrp mail /var/mail
chmod 1777 /tmp
```

Installing Sendmail

- Unpack the Sendmail archive and install it by running:

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

Configuring Sendmail

Configuring Sendmail isn't as easily said as done. There are a lot of things you need to consider while configuring Sendmail and I can't take everything into account. That's why at this time we'll create a very basic and standard setup. If you want to tweak Sendmail to your own liking, go right ahead, 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.mc > 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`

Installing Procmail

- Unpack the Procmail archive and install it by running:

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

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 -o -x /usr/sbin/sendmail -- -bd
    check_status
    ;;

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

  reload)
    echo -n "Reloading Sendmail configuration file..."
```

```
    start-stop-daemon -K -q -s 1 -p /var/run/sendmail.pid
    check_status
    ;;

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

    sleep 1

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

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

esac

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

Setting up permissions and symlinks

- Set the proper permissions by running `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
```

10.2 Setting up FTP

Creating 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 −s /bin/sh −m ftp
```

Installing Proftpd

- Unpack the Proftpd archive and install it by running:

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

Creating the /etc/init.d/proftpd 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 −o −x /usr/sbin/proftpd
    check_status
    ;;

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

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

    sleep 1

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

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

esac

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

Setting up permissions and symlinks

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

10.3 Setting up HTTP

Installing Apache

- Unpack the Apache archive and install it by running:

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

Configuring Apache

There's not much that needs to be configured. The only thing we need to do is to add the /usr/apache/man path to /usr/share/misc/man.conf

- Edit the /usr/share/misc/man.conf file
- Add this line underneath the other lines that start with MANPATH: *MANPATH /usr/apache/man*

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/bin/apachectl start
    ;;

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

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

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

    sleep 1

    echo -n "Starting Apache HTTP daemon..."
    /usr/apache/bin/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
```

10.4 Setting up Telnet

Installing telnet daemon + client

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

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

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 -o -x /usr/sbin/inetd
    check_status
    ;;

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

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

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

```
    check_status

    sleep 1

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

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

esac

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

Setting up permissions and symlinks

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

10.5 Setting up PPP

Configuring the Kernel

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

Creating group

- Create the daemon group by running `groupadd -g7 daemon`

Installing PPP

- Unpack the PPP archive and install it by running:

```
./configure
```

```
make; make install
```

Creating /etc/resolv.conf

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

```
# Begin /etc/resolv.conf

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

# End /etc/resolv.conf
```

Creating /etc/ppp/peers/provider

- Create the `/etc/ppp/peers` directory
- 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
```

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
```

Note on password authentication

As you see from the sample scripts (these are the actual scripts I use when I'm not using X) 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 (with my provider it's mostly 'if') things like the username and password are sent.

11.Installing Network Clients

11.1 Installing Email clients

Installing Mailx

- Unpack the Mailx archive and install it by running

```
make; make install
```

Installing Mutt

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

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

- Unpack the Mutt archive and install it by running:

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

Installing Fetchmail

- Unpack the Fetchmail archive and install it by running:

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

Testing the email 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 running 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; passwd 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

11.2 Installing FTP client

Installing Netkit−ftp

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

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

Testing FTP system

- 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.

11.3 Installing HTTP client

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 and install it by running:

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

Installing Lynx

- Unpack the Lynx archive and install it by running:

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

Testing HTTP system

- 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.

11.4 Installing Telnet client

The Telnet client has already been installed when we installed the Telnet daemon in the previous chapter.

Testing Telnet system

- 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.

11.5 Installing PPP clients

Creating the connect script

* 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
```

Creating the disconnect script

* 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
```

Testing PPP system

* Connect to the Internet by running `pon`
* Try to connect to a site like http://www.linuxfromscratch.org
* Disconnect from the Internet by running `poff`

12.Installing X Window System

12.1 Installing X

* Unpack the X archive and install it by running:

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

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.

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 do it.

12.2 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`

12.3 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`

12.4 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`

12.5 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`

12.6 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.

12.7 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.

12.8 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.

12.9 Preparing the system for the Window Maker installation

Installing libPropList

- Unpack the libPropList archive and install it by running:

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

Installing libXpm

- Unpack the libXpm archive and install it by running:

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

This slightly different installation is necessary due to a bug in one of the Makefiles. It depends on files in the lib directory which aren't installed yet and it's not searching for them in the lib directory, so we have to install those files first before compiling the actual package.

Installing libpng

- Unpack the libpng archive and install it by running:

```
make -f scripts/makefile.lnx; make -f scripts/makefile.lnx install
```

Installing libtiff

- Unpack the libtiff archive and install it by running:

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

Installing libjpeg

* Unpack the libjpeg archive and install it by running:

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

Installing libungif

* Unpack the libungif archive and install it by running:

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

Installing WindowMaker

* Unpack the WindowMaker archive and install it by running:

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

12.10 Updating dynamic loader cache

* Update the dynamic loader cache by running: `ldconfig`

12.11 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`

12.12 Testing WindowMaker

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

13.Resources

A list of books, HOWTOs and other documents you might find useful to download or buy follows. This list is just a small list to start with. We hope to be able to expand this list in time as we come across more useful documents or books.

13.1 Books

- Sendmail published by O'Reilly. ISBN: 1−56592−222−0
- Linux Network Administrator's Guide published by O'Reilly. ISBN: 1−56592−087−2
- Running Linux published by O'Reilly. ISBN: 1−56592−151−8

13.2 HOWTOs

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

13.3 Other

- The various manual and info pages that come with packages

14.The End

You have reached the end of the Linux From Scratch HOWTO. I hope this experience helped you getting to know Linux better. If you have anything that you think needs to be mentioned in here (be it a bug fix, extra software which has been forgotten but which you consider important) let us know. Together with your help and suggestions this HOWTO can be improved significantly.

15.Copyright & Licensing Information