The GNU/Hurd User's Guide

Thomas Bushnell BSG
Gordon Matzigkeit
Matthew S. Grant
Thomas Hart

The GNU/Hurd User's Guide

This file documents the usage of GNU/Hurd. This edition of the documentation was last updated for version 0.3 of the Hurd.

1 Introduction

The *GNU Hurd* is the GNU Project's replacement for the Unix kernel. The Hurd is a collection of servers that run on the Mach microkernel to implement file systems, network protocols, file access control, and other features that are normally implemented by the Unix kernel or similar kernels such as Linux.

1.1 Audience

This manual is designed to be useful to everybody who is interested in using or administering the a GNU/Hurd system.

If you are an end-user and you are looking for help on running the Hurd, the first few chapters of this manual describe the essential parts of installing, starting up, and shutting down a *GNU/Hurd* workstation. Subsequent chapters describe day-to-day use of the system, setting up and using networking, and the use of translators.

If you need help with a specific program, you can get a short explanation of the program's use by typing `program --help` at the command prompt; for example, to get help on *grep*, type `grep --help`. More complete documentation is available if you type `info program`; continuing with our example of *grep*, type `info grep`.

This manual attempts to provide an introduction to the essential topics with which a user of GNU/Hurd, or any free UNIX-like system. New users of GNU/Hurd and GNU/Linux must often spend time learning what skills they must learn; for example, users must learn what tools are important, and what sources of information are most useful. We introduce the reader to the skills and concepts that must be learnt, and tell the reader where to find further information.

1.2 Overview

An *operating system kernel* provides a framework for programs to share a computer's hardware resources securely and efficiently. This framework includes mechanisms for programs to communicate safely, even if they do not trust one another.

The GNU Hurd breaks up the work of the traditional kernel, and implements it in separate programs. The Hurd formally defines the communication protocols that each of the servers understands, so that it is possible for different servers to implement the same interface; for instance, NFS and FTP use the same TCP/IP communication protocol. NFS, FTP, and the TCP/IP servers are all separate user-space programs (don't worry if these programs are not familiar to you, you can learn about them if and when you need to).

The *GNU C Library* provides a *POSIX* environment on the Hurd, by translating standard POSIX system calls into interactions with the appropriate Hurd server.

POSIX stands for Portable Operating System Interface. The term POSIX we hear so much about is a set standards set by the IEEE. Unix-like operating systems aim to be compliant or partially-compliant. GNU/Hurd is as compliant as any other Unix-like operating system. That is why ported applications, implementations of protocols and other essential services have been available since the introduction of GNU/Hurd. But always remember: GNU's Not Unix. GNU/Hurd complies with POSIX, but is constantly growing to address the limitations of Unix-like operating systems.

1.3 History

Richard Stallman (RMS) started *GNU*in 1983, as a project to create a complete free operating system. In the text of the GNU Manifesto, he mentioned that there is a primitive kernel. In the first GNUsletter, Feb. 1986, he says that GNU's kernel is TRIX, which was developed at the Massachusetts Institute of Technology.

By December of 1986, the Free Software Foundation (FSF) had "started working on the changes needed to TRIX" [Gnusletter, Jan. 1987]. Shortly thereafter, the FSF began "negotiating with Professor Rashid of Carnegie-Mellon University about working with them on the development of the Mach kernel" [Gnusletter, June, 1987]. The text implies that the FSF wanted to use someone else's work, rather than have to fix TRIX.

In [Gnusletter, Feb. 1988], RMS was talking about taking Mach and putting the Berkeley Sprite filesystem on top of it, "after the parts of Berkeley Unix... have been replaced."

Six months later, the FSF is saying that "if we can't get Mach, we'll use TRIX or Berkeley's Sprite." Here, they present Sprite as a full-kernel option, rather than just a filesystem.

In January, 1990, they say "we aren't doing any kernel work. It does not make sense for us to start a kernel project now, when we still hope to use Mach" [Gnusletter, Jan. 1990]. Nothing significant occurs until 1991, when a more detailed plan is announced:

> "We are still interested in a multi-process kernel running on top of *Mach*. The CMU lawyers are currently deciding if they can release Mach with distribution conditions that will enable us to distribute it. If they decide to do so, then we will probably start work. CMU has available under the same terms as Mach a single-server partial Unix emulator named Poe; it is rather slow and provides minimal functionality. We would probably begin by extending Poe to provide full functionality. Later we hope to have a modular emulator divided into multiple processes." [Gnusletter, Jan. 1991].

RMS explains the relationship between the Hurd and Linux in `http://www.gnu.org/software/hurd/hurd-and-linux.html`, where he mentions that the FSF started developing the Hurd in 1990. As of [Gnusletter, Nov. 1991], the Hurd (running on Mach) is GNU's official kernel.

These announcements made it clear that the GNU Project was getting a Mach microkernel as a component of the GNU System. Once Lites, a single-server 4.4BSD user land environment, had been implemented on top of Mach, Mach-based systems became usable. Members of the GNU Project then began hacking Mach for use with the GNU Project's multi-server kernel replacement. The individuals involved with making Mach work with the GNU Project's multi-server kernel replacement were Thomas Bushnell, BSG., and Roland McGrath. The Mach microkernel was originally developed at Carnegie Melon University (CMU); after development subsided at CMU, the University of Utah took over and continued adding better drivers and fixing other critical deficiencies.

In the latest GNUmach release notes, kernel and glibc maintainer Roland McGrath clarifies the development history of the GNUmach microkernel.

> "When maintenance of Mach 3.0 at CMU waned, the University of Utah's Flux Group took over in the form of the Mach4 project, and revamped much of the i386 machine support code in the course of their research. While at Columbia University, Shantanu Goel worked on using Linux device drivers in Mach, and later continued this work at the University of Utah. Utah Mach4 became the seat of Mach development on the 3.0-compatible line, and the microkernel underlying the GNU/Hurd multiserver operating system. When the Flux Group's research moved on from Mach to other systems, they wanted to reuse the work they had done in hardware support and device drivers; this work (and a whole lot more) eventually evolved into the OSKit. Meanwhile, when the Flux Group stopped maintaining Mach4, the Free Software Foundation's GNU Hurd Project had taken it up and produced the GNUmach release to go with the Hurd. Since then, the Hurd has become an all-volunteer project whose developers are not paid by the FSF, and later GNUmach releases incorporating bug fixes and updating the Goel/Utah encapsulation of Linux device drivers have been made by volunteers including Okuji Yoshinori and Thomas Bushnell.
>
> This bit of history explains some of why it was so easy to replace a lot of the GNUmach/Mach4 hardware support code with OSKit calls–in many cases the OSKit code is a direct evolution of the code originally in Mach4 that I was replacing, with the names changed and improvements Utah has made since the Mach4 days."

1.4 Who Should Use the Hurd?

Firstly, a note for end-users: if you do not consider yourself "computer literate," then you *definitely* will not want to use the Hurd. No official release of the Hurd has yet been made, and the system is currently unstable. If you run the Hurd, you will encounter many bugs. For those people who use their computers for web-surfing, email, word processing, etc., and just want the infernal machine to work, the Hurd's bugs would prove extremely annoying.

For such people, a more stable system is preferable. Fortunately, the GNU system currently exists in a very stable state with Linux substituted for the Hurd as the kernel, so it is possible for end-users to use a powerful, stable, and Free Unix-like operating system. *Debian GNU/Linux*, a very high-quality GNU/Linux distribution, is available; in addition, many commercial companies sell boxed GNU/Linux distributions with printed documentation, and, to various extents, take measures to hide the complexity of the system from the user. Keep in mind that the GNU/Hurd is not the system you use for web-surfing, email, word processing, and other such tasks *now* . . . it is the system that you will use for these tasks in the *future*.

Those who consider themselves computer-literate and are interested in the Hurd, but do not have experience with Unix-like systems, may also wish to learn the ropes using a more stable GNU/Linux system. Unix-like systems are quite different from other systems you may have used, and they take some getting used to. If you plan to use the GNU/Hurd in the future, we recommend you use Debian GNU/Linux.

If you consider yourself computer-literate, but are not a programmer, you can still contribute to the Hurd project. Tasks for non-programmers include running GNU/Hurd systems and testing them for bugs, writing documentation, and translating existing documentation into other languages. At present, most GNU/Hurd documentation is available only in English and/or French.

Anyone who might be interested in the Hurd: a student studying the system, a programmer helping to develop the Hurd servers, or an end-user finding bugs or writing documentation, will be interested in how GNU/Hurd is similar to, and different from, Unix-like kernels.

For all intents and purposes, the Hurd is a modern Unix-like kernel, like Linux and the *BSDs*. GNU/Hurd uses the GNU C Library, whose development closely tracks standards such as ANSI/ISO, BSD, POSIX, Single Unix, SVID, and X/Open. Hence, most programs available on GNU/Linux and BSD systems will eventually be ported to run on GNU/Hurd systems.

An advantage of all these systems - GNU/Hurd, GNU/Linux, and the BSDs - is that, unlike many popular operating systems, is that they are Free Software. Anybody can use, modify, and redistribute these systems under the terms of the GNU General Public License (see Section 12.1 [GNU General Public License], page 62) in the case of GNU/Hurd and GNU/Linux, and the BSD license in the case of the BSDs. In fact, the entire GNU System is a complete Unix-like operating system licensed under the GNU *GPL*.

Although it is similar to other Free Unix-like kernel projects, the Hurd has the potential to be much more. Unlike these other projects, the Hurd has an object-oriented structure that allows it to evolve without compromising its design. This structure will help the Hurd undergo major redesign and modifications without having to be entirely rewritten. This extensibility makes the Hurd an attractive platform for learning how to become a kernel hacker or for implementing new ideas in kernel technology, as every part of the system is designed to be modified and extended. For example, the MS-DOS FAT filesysetem was not supported by GNU/Hurd until a developer wrote a translator that allows us to access this filesystem. In a standard Unix-like environment, such a feature would be put into the kernel. In GNU/Hurd, this is done in a different manner and recompiling the kernel is not necessary, since the filesystem is implemented as a user-space program.

Scalability has traditionally been very difficult to achieve in Unix-like systems. Many computer applications in both science and business require support for symmetric multiprocessing (SMP). At the time of this writing, Linux could scale to a maximum of 8 processors. By contrast, the Hurd implementation is aggressively multi-threaded so that it runs efficiently on both single processors and symmetric multiprocessors. The Hurd interfaces are designed to allow transparent network clusters (*collectives*), although this feature has not yet been implemented.

Of course, the Hurd (currently) has its limitations. Many of these limitations are due to GNU Mach, the microkernel on which the Hurd runs. For example, although the Hurd has the potential to be a great platform for SMP, no such multiprocessing is currently possible, since GNU Mach has no support for SMP. The Hurd has supports less hardware than current versions of Linux, since GNU Mach uses the hardware drivers from version 2.0 of the Linux kernel. Finally, GNU Mach is a very slow microkernel, and contributes to the overall slowness of GNU/Hurd systems.

Mach is known as a 1st-generation *microkernel* - much work has gone into the much-improved 2nd-generation microkernels currently being developed. A long-term goal is to port the Hurd to L4, a very fast 2nd-generation microkernel. This port will offer substantial improvements to the system. In the short term, the Hurd developers plan to move the Hurd to OSKit Mach, an improved version of Mach being developed at the University of Utah.

The Hurd is still under active development, and no stable release has been made. This means that the Hurd's code base is much less mature than that of Linux or the BSDs. There are bugs in system that are still being found and fixed. Also, many features, such as a DHCP client, and support for several filesystem types, is currently missing.

The deficiencies in the Hurd are constantly being addressed; for example, until recently, *pthreads* (POSIX threads), were missing. This meant that several major applications, including GNOME, KDE, and Mozilla, could not run on GNU/Hurd. Now that the Hurd has a preliminary *pthreads* implementation, we may soon see these applications running on GNU/Hurd systems.

The Hurd is a very modern design. It is more modern than Linux or the BSDs, because it uses a microkernel instead of a monolithic kernel. It is also more modern than Apple's Darwin or Microsoft's NT microkernel, since the it has a multi-sever design, as opposed to the single-server design of Darwin and NT. This makes the Hurd an ideal platform for students interested in operating systems, since its design closely matches the recommendations of current operating system theory. In addition, the Hurd's modular nature makes it much easier to understand than many other kernel projects.

GNU/Hurd is also an excellent system for developers interested in kernel hacking. Whereas Linux and the BSDs are quite stable, there is still much work to be done on the Hurd; for example, among all the filesystem types in use today, the Hurd only supports ext2fs (the Linux filesystem), ufs (the BSD filesystem), and iso9660fs (the CD filesystem). GNU/Hurd offers a developer the opportunity to make a substantial contribution to a system at a relatively-early point in its development.

1.5 Hurd Today

The GNU operating system is alive and well today. In 1998 Marcus Brinkmann brought the Hurd to the *Debian Project*. This allowed GNU/Hurd to have a software management system(apt and dpkg), world wide access to the latest free software, and a extreme increase of popularity.

The original architect, Thomas Bushnell, BSG, is involved on the mailing lists helping the young hackers design and help with Hurd development. There is now a team of individuals from around the world. They are constantly giving talks at conferences about the importance of design and implementation of the Hurd's unique multiserver microkernel environment. There is also work on porting the Hurd to two other micro-kernels and another architecture.

Debian GNU/Hurd has been Debian's most active non-Linux port. The user applications that are ported to GNU/Hurd are very current and are actively being improved.

Development of GNU/Hurd within the Debian project has also contributed to incremental development releases on sets of CDs. Also, new developers are allowed to get the source code to the Hurd from anywhere on the planet via Debian's packaging tool apt, and can get an even-more recent snapshot release from the the GNU project's alpha.gnu.org (`ftp://alpha.gnu.org`) server.

As far as a multi-server microkernel based operating systems are concerned: GNU/Hurd has had it's share of critisim. Many people have criticized the Hurd for its lack of a stable release, its design, which is unfamiliar to those used to monolithic kernels such as Linux and the BSD kernels, and its connection to the strong ideals of the GNU Project; however, GNU/Hurd is a cutting-edge operating system. Other operating systems with designs similar to GNU/Hurd are QNX and Sawmill. QNX is meant for embedded systems and many of the GNU Projects most famous programs are available for it . Sawmill was a research project set up by IBM. It was never released outside the academic community. Sawmill is a version of the Linux kernel running as multi-server on top of the L4 microkernel. It used some of the same ideas as GNU/Hurd but with members of the L4 (`http://www.l4ka.org`) Project influencing some of there research in microkernel based technology.

1.6 Copying

The Hurd is distributed under the terms of the GNU General Public License (see Section 12.1 [GNU General Public License], page 62). The GNU *GPL* protects your right to use, modify, and distribute all parts of the GNU system.

1.7 Conventions used in this manual

We often show command prompts in this manual. When doing so, the command prompt for the *root user* (System Administrator) includes a '#' character, and the command prompt for a regular user includes a '$' character. For example, if we were discussing how to shut down the system, which only the root user may do, the prompt may look like this:

`# halt` ⟨ENTER⟩

When discussing how to list the contents of a directory, which regular users can do, provided they have read permission for the directory (more on this later), the prompt may appear as:

`$ ls` ⟨ENTER⟩

Prompts may include additional characters, for example:

`bash-2.05# mke2fs -o hurd /dev/hda3` ⟨ENTER⟩

Hyperlinks in this manual look like this: `http://www.gnu.org/`.

A *definiton* is typeset as shown in this sentence.

There is a footnote at the end of this sentence.[1]

The names of commands are appear like this: `info`. Files are indicated like this: '`README.txt`'.

[1] This is a footnote.

This document includes cross-references, such as this one to the See Section 12.1 [GNU General Public License], page 62, and this parenthetical one to the see Section 12.1 [GNU General Public License], page 62.

2 Installing

Before you can use *GNU/Hurd* on your favorite machine, you'll need to install its base software components. Currently, the Hurd only runs on Intel i386-compatible architectures (such as the Pentium), using the GNU Mach microkernel.

If you have unsupported hardware or a different microkernel, you will not be able to run the Hurd until all the required software has been *ported* to your architecture. *Porting* is an involved process which requires considerable programming skills, and is not recommended for the faint-of-heart. If you have the talent and desire to do a port, contact `bug-hurd@gnu.org` in order to coordinate the effort.

2.1 Binary Distributions

By far the easiest and best way to install GNU/Hurd is to obtain a Debian GNU/Hurd *binary distribution*. Even if you plan on recompiling the Hurd itself, it is best to start off with an already-working GNU system so that you can avoid having to reboot every time you want to test a program.

You can get GNU from a friend under the conditions allowed by the GNU GPL (see Section 12.1 [GNU General Public License], page 62). Please consider sending a donation to the *Free Software Foundation* so that we can continue to improve GNU software.

You can order Debian GNU/Hurd on a CD-ROM from CopyLeft (`http://www.copyleft.co.nz/`) or you can also FTP the complete GNU system in iso format from your closest GNU mirror,or `ftp://ftp.gnu.org/iso`. Again, please consider donating to the Free Software Foundation.

The format of the binary distribution is prone to change, so this manual does not describe the details of how to install GNU. The '`README`' file distributed with the binary distribution gives you complete instructions.

After you follow all the appropriate instructions, you will have a working GNU/Hurd system. If you have used GNU/Linux (`http://www.gnu.org/gnu/linux-and-gnu.html`) systems or other Unix-like systems before, GNU/Hurd will look quite familiar. You should play with it for a while, referring to this manual only when you want to learn more about GNU/Hurd.

If GNU/Hurd is your first introduction to the GNU operating system, then you will need to learn more about GNU in order to be able to use it. This manual will provide the basics for you to expand on and for you to productively use your GNU/Hurd machine. You should talk to your friends who are familiar with GNU, in order to find out about classes, online tutorials, or books which can help you learn more about GNU.

If you have no friends who are already using GNU, you can find some useful starting points at the GNU web site, `http://www.gnu.org/`. You can also send e-mail to `help-hurd@gnu.org`, to contact fellow Hurd users. You can join this mailing list by sending a request to `help-hurd-request@gnu.org`. You may even want to chat with the developers yourself. You can reach some on irc.freenode.net in the #hurd channel.

2.2 Internet Install

A great way to install the Hurd is to use an existing GNU/Linux operating system. This is the easiest method I have ever seen at installing a working system. What you need is:

1. A functional Debian GNU/Linux system (or any GNU/Linux system with the dpkg/apt programs).
2. Connection to the Internet (and access to `ftp.debian.org` and `alpha.gnu.org`).
3. The crosshurd package installed(should be '`crosshurd-#.##.deb`')
4. 2 GB or less of free unpartitioned drive space.
5. A GRUB bootdisk (check out grub.gnu.org (`http://grub.gnu.org`) for help with this).

Using your GNU/Linux system, you need to make a 1 gb partition with either fdisk or cfdisk. Then you have to format it for the Hurd. Let's say I made a partition called /dev/hda2 I would format it this way:

```
bash-2.05# mke2fs -o hurd /dev/hda2 <ENTER>
```

After this is done you can mount the partition. You then need to install the crosshurd package from your nearest Debian mirror. When you run the crosshurd program it will ask you where your GNU/Hurd partition is mounted. In this example, we mount the partition on '`/gnu`', but you can mount your partition anywhere you choose. You will need to run crosshurd after you mount your filesystem. The install program will ask you if you'd like to make '`/usr`' a link to '`.`' (the single period is a shortcut for the current directory, which is the root directory on GNU/Hurd). It is a good idea to say yes to this option. GNU/Hurd does not require a '`/usr`' partition like traditional Unix-like systems.

you must be root user for this to work

```
bash-2.05# apt-get update <ENTER>
bash-2.05# apt-get install crosshurd <ENTER>
bash-2.05# mount /dev/hda2 /gnu <ENTER>
bash-2.05# crosshurd <ENTER>
```

After your done you should be able to reboot. You must use the *GRUB* boot disk to boot the Hurd. Once you've booted, you can finish the installation by running the `native-install` shell script.

Note: *When you are booting the Hurd for the first time, you must give GRUB the single-user option (-s), as in the following example. Also note that the first long module entry is all on one line.*

```
title GNU/Hurd
root (hd0,1)
kernel /boot/gnumach.gz -s root=device:hd0s2
module /hurd/ext2fs.static –multiboot-command-line=${kernel-command-line} –host-priv-port=${host-port} –device-master-port=${device-port} –exec-server-task=${exec-task} -T typed ${root} $(task-create) $(task-resume)
module /lib/ld.so.1 /hurd/exec $(exec-task=task-create)
```

When you have booted the Hurd you will be at a single user shell prompt. You should be able to type `native-install` and see it set up your translators and install some essential system software.

```
sh-2.05#./native-install <ENTER>
```

When this is done, you will be prompted to reboot and run `native-install` again. The second round of `native-install` should configure your system for use. After this, you must do some important things.

You will need to make your swap, home (optional), and cdrom device translators. You will need to `cd` into your '`/dev`' directory and run the `MAKEDEV` script. After these devices are created, you will need to edit the file '`/etc/fstab`' and add your swap device and optionally your cdrom. Before you can add the devices to that file, you need to set up your temporary terminal type. GNU/Hurd's default editor is called `nano`; it is a small lightweight simple editor.

```
sh-2.05# cd /dev <ENTER>
sh-2.05# ./MAKEDEV hd0s3 hd0s4 hd2 <ENTER>
sh-2.05# export TERM=mach <ENTER>
sh-2.05# nano /etc/fstab <ENTER>
```

The last command will open up the file '`fstab`' to be edited. You should see an entry for your root ('`/`') filesystem. A entry for your swap partition should be added to this file like in this example:

```
device          mount point    filesystem type  options  dump pass
/dev/hd0s2      /              ext2fs           rw       1    1
/dev/hd0s3      none           swap             sw       0    0
/dev/hd0s4      /home          ext2fs           rw       0    0
/dev/hd2        /cdrom         iso9660fs        ro,noauto 0    0
```

Note: You can use the same swap partition from your GNU/Linux system.

After editing this file and creating your devices, you should be able to reboot into a multi-user GNU/Hurd system.

3 Bootstrap

Bootstrapping[1] is the procedure by which your machine loads the microkernel and transfers control to the Hurd servers.

3.1 Bootloader

The *bootloader* is the first piece of software that runs on your machine. Many hardware architectures have a very simple startup routine which reads a very simple bootloader from the beginning of the internal hard disk, then transfers control to it. Other architectures have startup routines which are able to understand more of the contents of the hard disk, and directly start a more advanced bootloader.

Currently, *GRUB*[2] is the GNU bootloader. GNU GRUB provides advanced functionality, and is capable of loading several different kernels (such as Linux, the *BSD family, and DOS).

From the standpoint of the Hurd, the bootloader is just a mechanism to get the *microkernel* running and transfer control to the Hurd servers. You will need to refer to your bootloader and microkernel documentation for more information about the details of this process. However, you don't need to know how this all works in order to use GNU.

3.2 Installing GRUB

At the moment, you are booting the Hurd with a floppy disk. Doing so regularly is not a good idea, since floppy disks are prone to failure. This section will show you how to install GRUB into the master boot record of your hard-disk.

What you need are:

1. A GRUB boot floppy.
2. A copy of GRUB installed on your hard disk.
3. Knowledge of which partition contains GNU/Hurd.

First you will need to make a directory for GRUB. Do this now (you must be logged in as root).

```
bash-2.05# mkdir /boot/grub
```

Now you must copy the contents of the GRUB installed loader files. These are usually located in /lib/grub/i386-unknown-pc/. All the files in this directory need to be copied to the boot directory.

```
bash-2.05# cp /lib/grub/i386-unknown-pc/* /boot/grub/
```

Now, you have two more things to do. You must construct a '`menu.lst`' file to put in your '`/boot/grub`' directory. This file is the file that displays a menu at boot time. Before you tackle this, you should read the GRUB info pages to get a better understanding of

---

[1] The term *bootstrapping* refers to a Dutch legend about a boy who was able to fly by pulling himself up by his bootstraps. In computers, this term refers to any process where a simple system activates a more complicated system.

[2] The GRand Unified Bootloader, available from `http://www.gnu.org/software/grub/`.

the way GRUB works. The reason I ask this of you is that the Hurd is constantly being worked on to boot more effiecently and the syntax may change with the Hurd's development releases.

This is my sample '`menu.lst`' file

```
timeout 10

color light-grey/black red/light-grey

title Debian GNU/Hurd
root (hd0,0)
kernel /boot/gnumach.gz root=device:hd0s1
module /hurd/ext2fs.static –multiboot-command-line=${kernel-command-line} –host-
priv-port=${host-port} –device-master-port=${device-port} –exec-server-task=${exec-
task} -T typed ${root} $(task-create) $(task-resume)
module /lib/ld.so.1 /hurd/exec $(exec-task=task-create)

title Debian GNU/Linux
root (hd0,1)
kernel /vmlinuz root=/dev/hda2 ro
```

Now that you have your '`menu.lst`' file in '`/boot/grub`', you are ready to install GRUB and make your machine bootable from the hard disk. You should write down what you wrote in your '`menu.lst`' file. You must type this information into the GRUB command line. If you forget this information, your GRUB boot floppy should be able to boot your system so that you can write the relevant data down and continue with the installation of GRUB.

When the boot menu of your GRUB floppy appears, press ⓒ on your keyboard. This will put you at the GRUB command prompt. You need to start with the `root` command:

```
:grub> root (hd0,0)
```

The `root` command in the above example tells GRUB that my GNU/Hurd system is installed on the first partition of the first drive. After you press the ⟨Enter⟩ key, GRUB should print out something along the lines as `<ext2 and a bunch of numbers>`. This means that GRUB found your root filesytem. Next you need GRUB to load the kernel. This is the tricky part. Heres what I have:

```
:grub> kernel /boot/gnumach.gz root=device:hd0s1
```

Once you press ⟨Enter⟩ again, you will see some more info. If you get an error, that just means you made a typing mistake, or picked the wrong root device for GRUB. There is no need to work: this will not make your machine explode.

Next, you need to enter the modules for GRUB to use. This is probably the trickiest part of booting the system.

```
:grub> module /hurd/ext2fs.static –multiboot-command-line=${kernel-command-line} –
host-priv-port=${host-port} –device-master-port=${device-port} –exec-server-task=${exec-
task} -T typed ${root} $(task-create) $(task-resume)
```

Note this should be all on one line, do not press ⟨Enter⟩ *until the end.*

Next, type:

```
:grub> module /lib/ld.so.1 /hurd/exec $(exec-task=task-create)
```

That first monster module line is a lot to type each time you want to boot GNU/Hurd. We can save ourselves a lot of time and effort by putting this information, along with all other relevant information, in the 'menu.lst' file, as shown in the example above.

So, we open nano, type something similar to the example 'menu.lst' file shown above, reboot the system, and . . . . the system freezes on boot. What went wrong?

The answer is that nano automatically inserts line breaks at the end of long lines, so our big monster module line is broken up into a form that GRUB cannot understand.

There are two easy ways to solve this problem. First, you could create your 'menu.lst' file using another editor, such as GNU Emacs, that does not automatically insert line breaks. Another example is to use the ⟨\⟩ character, which tells GRUB to read the next line as if it were part of the current line; you could type, for example:

module /hurd/ext2fs.static \
–multiboot-command-line=${kernel-command-line} \
–host-priv-port=${host-port} –device-master-port=${device-port} \
–exec-server-task=${exec-task} \
-T typed ${root} $(task-create) $(task-resume)

GRUB would then interpret these five lines as one long line.

To fully understand all our module commands, you can check the Hurd Reference Manual, which should explain every detail. These are very important for developers, as these commands can do many interesting things with the GRUB bootloader and the Hurd. For instance the newest gnumach microkernel can use GRUB to debug it through the serial port on your computer. GRUB can also boot your kernel and then fetch the rest of your system over the network. This is partially implemented with GNU/Hurd but older Unix-like operating systems can do this easily.

If you're still with me, then you should be able to safely install GRUB on your hard disk. If you're confused, then you should consult the GRUB documentation. You should be able to install GRUB by typing the following command at your GRUB command prompt:

```
:grub> setup (hd0)
```

If this is successful, you should be able to press the reset button on your machine's case and remove the GRUB floppy. You just installed GRUB to the master boot record of the hard disk! That's one task many people are scared to attempt.

3.3 Server Bootstrap

The `serverboot` program has been deprecated. Newer GNU Mach kernels support processing the bootscript parameters and boot the Hurd directly.

The `serverboot` program is responsible for loading and executing the rest of the Hurd servers. Rather than containing specific instructions for starting the Hurd, it follows general steps given in a user-supplied boot script.

To boot the Hurd using `serverboot`, the microkernel must start `serverboot` as its first task, and pass it appropriate arguments. `serverboot` has a counterpart, called `boot`, which can be invoked while the Hurd is already running, and allows users to start their own complete sub-Hurds (see Section 3.3.3 [The Sub-Hurd], page 15).

3.3.1 Invoking `serverboot`

The `serverboot` program has the following synopsis:

```
serverboot -switch... [[host-port device-port] root-name]
```

Each *switch* is a single character, out of the following set:

'`a`' Prompt the user for the *root-name*, even if it was already supplied on the command line.

'`d`' Prompt the user to strike a key after the boot script has been read.

'`q`' Prompt the user for the name of the boot script. By default, use '`root-name:/boot/servers.boot`'.

All the *switches* are put into the `${boot-args}` script variable. *host-port* and *device-port* are integers which represent the microkernel host and device ports, respectively (and are used to initialize the `${host-port}` and `${device-port}` boot script variables). If these ports are not specified, then `serverboot` assumes that the Hurd is already running, and fetches the current ports from the procserver, which is documented in the GNU Hurd Reference Manual.

root-name is the name of the microkernel device that should be used as the Hurd bootstrap filesystem. `serverboot` uses this name to locate the boot script (described above), and to initialize the `${root-device}` script variable.

3.3.2 Boot Scripts

Boot Scripts are used to boot further Hurd systems in parallel to the first, and are parsed by `serverboot` to boot the Hurd. See '`/boot/servers.boot`' for an example of a Hurd boot script.

3.3.3 The Sub-Hurd

The `boot` program can be used to start a set of core Hurd servers while another Hurd is already running. You will rarely need to do this, and it requires superuser privileges to control the new Hurd (or allow it to access certain devices), but it is interesting to note that it can be done.

Usually, you would make changes to only one server, and simply tell your programs to use it in order to test out your changes. This process can be applied even to the core servers. However, some changes have far-reaching effects, and so it is nice to be able to test those effects without having to reboot the machine.

Here are the steps you can follow to test out a new set of servers:

1. Create a pseudo-root device. Usually, you would do this by creating a new partition under your old Hurd, and initializing it with your favorite filesystem format. `boot` understands the regular `libstore` options (FIXME xref), so you may use a file or other store instead of a partition.

```
$ dd if=/dev/zero of=my-partition bs=1024k count=400
400+0 records in
400+0 records out
$ mke2fs ./my-partition
mke2fs 1.18, 11-Nov-1999 for EXT2 FS 0.5b, 95/08/09
my-partition is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: GNU/Hurd
Block size=1024 (log=0)
Fragment size=1024 (log=0)
102400 inodes, 409600 blocks
20480 blocks (5.00%) reserved for the super user
First data block=1
50 block groups
8192 blocks per group, 8192 fragments per group
2048 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729, 204801, 221185, 401409
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
$
```

2. Copy the core servers, C library, your modified programs, and anything else you need onto the pseudo-root.

```
$ settrans -c ./my-root /hurd/ext2fs -r `pwd`/my-partition
$ fsysopts ./my-root --writable
$ cd my-root
$ tar -zxpf /pub/debian/FIXME/gnu-20000929.tar.gz
$ cd ..
$ fsysopts ./my-root --readonly
$
```

3. Create a new boot script (FIXME xref).
4. Run `boot`.

```
-$ boot -D ./my-boot ./my-boot/boot/servers.boot ./my-partition
[...]
```

5. Here is an example using a hard drive that already has a GNU/Hurd system installed on an ext2 filesystem on '`/dev/hd2s1`':

```
$ settrans /mnt /hurd/ex2fs --readonly /dev/hd2s1
$ boot -d -D /mnt -I /mnt/boot/servers.boot /dev/hd2s1
```

6. See see Section 3.3.4 [Invoking boot], page 17 for help with boot.

Note that it is impossible to share microkernel devices between the two running Hurds, so don't confuse your sub-Hurd with your main GNU/Hurd system. When you're finished testing your new Hurd, then you can run the `halt` or `reboot` programs to return control to the parent Hurd.

If you're satisfied with your new Hurd, you can arrange for your bootloader to start it, and then reboot your machine. You will then be in a safe place to overwrite your old Hurd with the new one, and reboot back to your old configuration (with the new Hurd servers).

3.3.4 Invoking boot

Usage: boot [*option*...] *boot-script device*...

`--kernel-command-line=`*`command line`*
`-c` Simulated multiboot command line to supply.

`--pause`
`-d` Pause for user confirmation at various times during booting.

`--boot-root=`*`dir`*
`-D` Root of a directory tree in which to find the files specified in *boot-script*.

`--interleave=`*`blocks`*
Interleave in runs of length *blocks*.

`--isig`
`-I` Do not disable terminal signals, so you can suspend and interrupt the boot program itself, rather than the programs running in the booted system.

`--layer`
`-L` Layer multiple devices for redundancy.

`--single-user`
`-s` Boot into single user mode.

`--store-type=`*`type`*
`-T` Each *device* names a store of type *type*.

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options. If neither '`--interleave`' or '`--layer`' is specified, multiple *device*s are concatenated.

3.4 Shutdown

You can shut down or reboot your GNU/Hurd machine by typing these commands:

```
bash-2.05# reboot <ENTER>
bash-2.05# halt <ENTER>
```

In a matter of seconds it should tell you that it is in a type loop and pressing ⟨ctrl⟩, ⟨alt⟩, and ⟨Delete⟩ at the same time will make the system reboot. It is now safe to shut the computer off.

4 Using

Now that you have GNU/Hurd installed and booting, you're probably wondering "What can I do with it?" Well, GNU/Hurd is capable of being a console based workstation. The X Window System is known to work, but is not required in order to learn how to use GNU/Hurd. This chapter is intended to get you acquainted to the *command line* of GNU/Hurd. You should read the Hurd FAQ and also have a working knowledge of *Bash*, the GNU shell. If you are not familiar with this material, you can read see Section 4.1 [The Shell], page 18 for a brief introduction.

4.1 The Shell

If you successfully installed GNU/Hurd and you are in multi-user mode, you should have a login shell ready for you, which should look like this:

```
GNU 0.3 (hurd)

Most of the programs included with the Debian GNU/Hurd system are
freely redistributable; the exact distribution terms for each program
are described in the individual files in /usr/share/doc/*/copyright

Debian GNU/Hurd comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
login>

Type Login:USER or type HELP
```

You can go ahead and type `login root`<ENTER> to get started.

What you need to know for using GNU/Hurd is your basic Unix-like commands. The GNU Project has written a replacement for almost every Unix command. You need to learn how to copy, move, view, and modify files. It also is important to learn one of the many editors that are available for GNU/Hurd. The default editor is Nano, which is very simple and self explanatory yet not as powerful as Emacs or vi. It is there because it's small, and easy on the memory of your computer. I am not going to cover it here because it is quite self-explanatory. This section hopes to get you started with these necessity commands. If you have knowledge of these already, you can probably skip ahead. Also, you might want to get comfortable with the default editor first, then come back to this section to learn your basic GNU/Hurd shell commands.

The shell is the the program that interacts with the filesystem and the data on the filesystem. This is called bash (Bourne again shell)[1]. Once you get comfortable with the shell, you can happily sit at any Unix-like workstation and get some work done. It is very important that you get comfortable with the shell. It is also called the terminal, console, etc.

[1] This is a pun on the name of the original Unix shell, the Bourne shell, and the Christian idea of being "born again".

At this prompt you should be ready to begin. (Note: Shell variables cause your prompt to display your current working directory. Your prompt may look like this: "user@hurd:~/$".)

```
bash-2.05$
```

To make a directory for yourself, you would use the `mkdir` (make directory) command. You can make as many as you want as long as they are in your home directory.

You may notice that many commands are abbreviations. The tradition of Unix-like systems dictates that commands take this form. Since the user is expected to type many commands into the shell, the commands are kept succinct so that fewer keystrokes are required. The side effect, of course, is many command names that are confusing at first sight.

```
bash-2.05$ mkdir docs tmp sources <ENTER>
```

To change directories you would use the `cd` (change directory) command. One thing to remember is that wherever you are in the filesytem, typing `cd` alone will put you back in your home directory. (Note:*The bash shell features command line completion, by pressing the first letter of the file or directory plus* <TAB> *will do this.*)

```
bash-2.05$ cd /usr/local <ENTER> #this puts you in the /usr/local directory.
bash-2.05$ cd <ENTER>            #this puts you back in the home directory.
```

There is another short cut, `cd ../` This puts you in the directory above the one you're currently in.

There are many ways to view a file, but we'll use the most common command called `cat`[2].

```
bash-2.05$ cat README | less <ENTER>
```

You may wonder what `| less` means. We use the `|` (pipe symbol) to take the output of '`README`' and feed it to the `less` command. `less` allows you to view the documents one screen at a time and it allows you to go back to the beginning of the document. You have to type q to quit `less`.

To copy a file from one place to another you use the `cp` (copy) command. The `cp` command has many options such as `cp -f -v -r`. Luckily for us, the `info` command will clarify many of the confusing commands and all three options.

Typing,

```
bash-2.05$ cp README chapter1.txt foo1/ <ENTER>
```

will copy '`README`' and '`chapter1.txt`' to the '`foo1`' directory. Here are a few good options to remember:

```
bash-2.05$ cp -rfv foo1/ file2.txt tmp/ <ENTER>
bash-2.05$ cp -rf /cdrom/* ~/ <ENTER>
```

The `-r` means "recursive", so '`file2.txt`', the '`foo1`' directory, and all the contents of '`foo1`' are copied to the '`tmp`' directory. The `-f` means "force", I have a habit of forcing things. The `-v` means "verbose", It will show you on the screen what files are being copied. In the second example, we have this `~/`, which is a short cut representing your home directory. So that command would copy the contents of '`/cdrom`' to your home directory.(Note:*The asterik symbol '*' is called a wildcard character which means "anything".*)

---

[2] An abbreviation for *concatenate*, since `cat` can be used to concatenate files.

To rename or move a file or directory you would use the `mv` (move) command. Be careful, because `mv` is a very powerful command. You can lose data if you're not careful about what you do. To rename a file or directory you would type, for example:

```
bash-2.05$ mv foo1/ foo2 <ENTER>
```

Now you no longer have a '`foo1`' directory, it's called '`foo2`' now. The same thing can be done with plain files, too.

I'll bet you're saying, "I'm doing all this work, but how do I view my directories?" You answer would be the `ls` (list)command. The `ls` command also has many options.

```
bash-2.05$ ls -al <ENTER>
```

The two options I have here are used more often than any others. The `-a` will show us all hidden files in the current directory.[3] The `-l` option tells us to display the directory in long format. This means it shows us the date modified, file size, file attributes, etc. of all files.

A user will also notice that there are files with dots in front of some files in thier home directory. The dotted files are typically hidden configuration files. The user will also notice a dot, followed by two dots. In the unix file system a dot represents the current directory. The two dots represent the directory above your current working directory. Examples of using these dots are common in building software.

```
bash-2.05# ../configure # This runs configure from the directory above.
bash-2.05# ./program    # This runs program from current directory.
```

Now that you have made all this mess in your home directory, I'll bet you're wondering how you can get rid of things. There are two important deleting commands. The `rm` (remove) command is convienent for removing files. `rm` command can also remove directorys that the `rmdir` command can't. I'll show the you a basic `rm` example, then my favorite with the extra options. The `rmdir` command will only delete directories that are empty.

```
bash-2.05$ rm README <ENTER>
```

Now, `rm` with some options:

```
bash-2.05$ rm -rf foo2/ <ENTER>
```

The second example is very powerful. The `-r` means "recursive" and the -f means "force." This removes all the contents of '`foo2`', without prompting the user for confirmation.

These are only the basic commands to get you started. . . there are many, many two- and three-letter commands. The more you use GNU/Hurd, the more options and commands you'll learn. The more you learn, the more freedom you'll have.

And that is a very valuable thing.

Now that you have been cruising around your filesystem, you might be wondering how much space you have left on your drive. There is a simple command to check this:

```
bash-2.05$ df / <ENTER>
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hd0s1             1920748   1203996    619180  67% /
```

If you don't put a directory after the command such as '`/`' or '`/home`', the `df` (diskfree) command will spit out some errors.

[3] Important files are often hidden, to prevent the user from accidentally deleting them.

If your prompt is not telling you what directory you are in, you can use the `pwd` (present working directory) command:

```
bash-2.05$ pwd <ENTER>
/usr/src
```

As a final note, always remember: `info` *is your friend on GNU/Hurd.*

4.2 Standard Input and Output

Computers borrow many concepts from mathematics. One very important concept that will be familiar to those with a background in programming is the idea of a *function*. Informally, the term *function* encapsulates the following idea:

When you put something in, you get something out.

Let's look at some examples of functions:

1. In math classes, students draw graphs of functions, such as $y = x^2$. In this case, you put in x, and you get x^2; for example, if you put in 3, you get 9.
2. A drink machine: you input a quantity of money and a drink selection, and the machine outputs a drink and your change.
3. Programming functions, such as *C*'s `isdigit()`: you give it a character, and it returns 1 if the character is a digit, and 0 otherwise.

There are many more examples of functions. Notice from our examples that a function may have multiple inputs an outputs; however, a function must always produce the same output for any given set of inputs. It would not do for a drink machine to sometimes give root beer when asked for iced tea!

In GNU systems and other systems that follow the UNIX tradition, programs are functions. Every program in a GNU system has a *standard input* and a *standard output*.

By default, a program's standard input comes from the command line, and its output goes to the terminal. For example:

```
$ echo "Echo prints whatever you type."
Echo prints whatever you type.
```

We may *redirect* a program's standard output using the ⟨>⟩ key:

```
$ echo "Here's a quick way to write one-line files." > quickie.txt

$ ls
quickie.txt

$ cat quickie.txt
Here's a quick way to write one-line files.

$ ls quickie.txt > dir.txt

$ ls
quickie.txt   dir.txt
```

```
$ cat dir.txt
quickie.txt
```

Similarly, we can use the ⟨<⟩ key to redirect a program's standard input:

```
$ cat < dir.txt
quickie.txt
```

We can use ⟨>⟩ and ⟨<⟩ together:

```
$ cat < dir.txt > copy.txt

$ cat copy.txt
quickie.txt
```

Another property of functions that is useful to us is *composition*: we can make the output of one function the input of another. When we use ⟨|⟩, the pipe symbol, we are composing two functions; see See Section 4.1 [The Shell], page 18. For example, we can pipe the output of the `info` program into `less`, and scroll through a page:

```
$ info gcc | less
```

Understanding standard input and output, redirection, and pipes is fundamental to intermediate and advanced use of the GNU command shell. Keep this section in mind while reading the rest of this manual.

4.3 Searching your Computer

Your computer is full of files. Some of these files are your personal files; others, such as '`/etc/passwd`', are used by the system. You may often want to find certain files on your computer, or find specific information in one or more files. This searching could be done manually through many invocations of `cd`, `ls`, and `less`; however, repetitive tasks such as searching are the sort of jobs for which computers were invented, and your GNU system has several tools to make searching easier.

The executables on your system are easy to find; simply use the `which` command:

```
$ which ls
/usr/bin/ls
```

The `which` command is especially useful when you have multiple copies of a program, each at a different location in your filesystem, since it allows you to find out which is executed by default. For example, if I install one version of Mozilla using Debian's `apt-get` command, and another version by downloading the source from `http://www.mozilla.org` and compiling it, I can type `which mozilla` to find out which is executed by default.

Your GNU system's more general searching tools are:

- `locate`
- `find`
- `grep`
- `xargs`

Of all these programs, `locate` is the simplest. `locate` searches through a database for files whose names match the pattern passed to it on the command line. For example, typing `locate bash` would print out a list of all files for which "bash" is part of the file's name.

Since `locate` searches through a database instead of an actual filesystem, it runs very quickly. Building this database, however, takes a relatively-long time. The superuser (root) can rebuild `locate`'s database by typing `updatedb`; if you try to use the `locate`, and it gives an error message mentioning the databse, you probably have to run `updatedb`.

The `find` command is significantly more flexible than `locate`, and therefore more complicated to use. Since `find` searches through the filesystem instead of a database, it finds files more slowly than `locate` does. `find` can find files based on such properties as file name, size, type, and owner, or any combination of these attributes.

When invoking `find`, you must tell it where to begin searching, and what criteria to use. For example, to find all files whose names begin will the letter “b” in the current directory and all its subdirectories, you would type:

```
$ find . -name 'b*'
```

If my username is “tom” and I want to find all files in '`/tmp`' that belong to me, I would type:

```
$ find /tmp -user tom
/tmp/.X11-unix
/tmp/cvs610c01f0.1
/tmp/cvs610c0d70.1
/tmp/cvs610c8570.1
/tmp/XWinrl.log
```

I can also use conditions in combination; for example, I can search for files in '`/tmp`' that belong to me and begin with the string “cvs”:

```
$ find /tmp -user tom -name 'cvs*'
/tmp/cvs610c01f0.1
/tmp/cvs610c0d70.1
/tmp/cvs610c8570.1
```

We used the `find` command to find files matching given criteria. To search the contents of one or more files, we can use the `grep` command.[4] `grep` is like a flexible, command line version of the Search or Find dialogue boxes common in modern GUIs.

The most basic way to use `grep` is to pass it a string, and a file in which to search for the string:

```
$ grep 'microkernel' faq.en.html
<p>When we are referring to the microkernel, we say ``Mach'' and use it
microkernel'' instead of just ``Mach''.
<p>{NHW} If you are using the GNU Mach microkernel, you can set your
```

We can see that `grep` outputs the lines of the file '`faq.en.html`' that contained the string “microkernel”.

We can also use a pipe to make `grep` search for a string in the output of another program:

```
$ echo "Mach is a microkernel" | grep 'microkernel'
Mach is a microkernel
```

The `xargs` command executes a command on each file in a list of files read from its standard input. Most often, we redirect a list of files to `xargs`. A very simple example would be:

[4] `grep` is an acronym; see section “Usage” in *Grep*.

```
$ ls | xargs cat
```

This command would print the contents of all files in the current directory to the screen (do not do this on a directory containing many large files!).

A more useful application of the `xargs` command comes from combining it with `grep` to search for files based on their contents. For example, we can type:

```
$ ls | xargs grep 'microkernel'
```

to search the current directory for files that contain the word “microkernel”. We can do more sophisticated searches by combining `find` with `xargs` and `grep`; for details, see section “Contents” in *Finding Files*.

Note that in modern desktop environments such as GNOME, GUIs exist that let you search for files much more easily. For the majority of users, these tools are sufficient. The advantage of the command line-based tools discussed in this section is that their output can be redirected, and they can be used in shell scripts (see Section 4.4 [Introduction to Scripting with Bash], page 24).

For more information on searching your computer, see section “Overview” in *Finding Files*.

4.4 Introduction to Scripting with Bash

These next few paragraphs are to aquaint the user of GNU/Hurd with extended shell capabilities. The bash shell is a very powerful program. You may have seen the term "Shell Programming", well that is what Bash can be used for. The shell can be used to write scripts to automate a handful of commands into one file. The start of a typical shell script will look like this:

```
#!/bin/bash
# The above line has to be the very first line of your
# script. If it's not it is taken as a comment.
#
# This is a comment
uname -a ; df /

echo This is a shell script

uptime &&

cat /etc/fstab
```

The very first line is a interesting one. The `#!/bin/bash` is to notify the shell of what program to use to run the script. The next two lines are comments, the '#' character is used to allow us to type important information that the script won't execute. Finally we get to the command `uname -a` which will tell us the operating system we are using, with the -a option it tells us everything we need. Following the first command is the ';' character which tells bash there is a compound command. The semicolon will tell bash to execute the command on the left first, then the right. The '&&' symbols after "uptime" in the above script tell bash not to execute the next command until the uptime command is complete.

The difference between the ";" and the "&&" is the first character will spit out a error and then continue to the next command. In order to execute the above script the file attributes need to be changed to an executable format. To change file attributes you use `chmod 755`. The `chmod` command is pretty tricky, so read the info and/or man pages to fully understand the capabilities of `chmod`.

Bash has other time saving features, for instance the `export`. You can export a couple of letters to represent a full path to a directory. The export command is typically used for this, here is a example:

```
bash-2.05$export SRC=/home/src
bash-2.05#echo $SRC
 /home/src
```

In the above example we use the `echo` to tell us what the value of 'SRC' is. Without the '$' character the `echo` would just print out 'SRC'. The `echo` is used extensively in shell scripts to place text on our screen either for debugging purposes or to prompt the user for input.

The great thing about shell scripts is that they have the capability of doing things just as quickly as a compiled program. A compiled program is written, compiled, and then run. The benefit of using a script is it is written then interpeted by the shell. When scripts get too large or complicated they are sometimes re-written in a compiled language. The reasoning behind this is that the shell script can slow the machine down the larger it gets, and take away resources that could be better used elsewhere on the computer. To get a better understanding of "Shell Programming" there are numerous books and on-line documents to be used.

Your GNU/Hurd workstations is comprised of many shell scripts. For instance when you login to the shell a couple scripts set up your enviroment for you. The scripts .profile and .bashrc set things such as

1. PATH : This is typically ":/bin:/sbin:/usr/bin:/usr/sbin:/usr/X11R6/bin" and can be modified to suit your needs. Generally you programs will reside in "bin" or "sbin" directories. Instead of typing /usr/bin/gcc you can just type gcc and the computer knows where it is.
2. PS1 : This is a bash variable that sets the look of your prompt. When the PS1 variable is not set you just see bash-2.05$ as your prompt.
3. MANPATH: The MANPATH variable is used so that when man is evoked It knows where all the man pages live.
4. alias : The alias command is used you disguise a command with anything your heart desires except another command. A user can alias the 'ls -a' to 'l' so at a bash prompt typing 'l <ENTER>' will evoke 'ls -a'

As you get more comfortable with you GNU/Hurd workstation you will end up customizing some of these values to suit your needs.

4.5 File Archivers and Compression

In this section, I want you to learn some extra things that will make life interesting. You will need to learn about archives. Archives come in many different formats. Archive is a general term for a software package. You may have heard of a *tarball*: this is a form of archive. The file extensions tell us what type of archive the file is. Generally they come in forms of .tgz, .tar.gz, .tar.bz2, .gz, etc.

Archiving is a method of packaging files. It has been around since the early days of computing. The `tar` command that we use when working with archives stands for “tape archiver”. It was originally used to create archived files for backup on large tape drives.

Closely associated with archiving is the concept of *compressing*. Various algorithms can be used to pack data into a smaller format. Archives are often compressed before being distributed; for example, a tarball may have the extension ‘`.tar.gz`’, which means that the tarball is a `tar` archive that has been compressed using `gzip`. Most software that you try and install from source-code will be packaged this way. I will try to give you some examples; if you’re confused, always consult the info pages.

Let’s say we get a `tar` file that has some documents you want to read. Make sure that you are in your home directory, then type:

```
bash-2.05$ tar xv docs.tar <ENTER>
```

After running this command, you would see the contents of the tar file extracted to your home directory. The `x` stands for “extract” and the `v` means “verbose”, which shows us the contents of the archive as it’s being decompressed. Compressing a file or directory is very similar. You would use type:

```
bash-2.05$ tar -cv new.tar foo1/ foo2/ <ENTER>
```

The `-c` is to create the tar file. We add the name of the tar file, then the contents that we want. In the above example, we created a file called ‘`new.tar`’ and we added the directories ‘`foo1/`’ and ‘`foo2/`’ to the tar file. With the `-v` option we see what is being added.

The `gzip` compression format is commonly used in the GNU system. Files with the extionsion ‘`.gz`’ are associated with `gzip`. The utilities `tar` and `gzip` work so well together that people have patched the tar command so that it can compress and decompress `gzip`ed files. Note, however, that `tar` cannot be used to decompress `gzip`ed files that are not `tar` archives.

```
bash-2.05$ tar -zxv file.tar.gz <ENTER> #This file was compressed once with
tar
                                         then with gzip
bash-2.05$ gzip -d file.gz <ENTER>      #notice no tar in filename means you
use
                                         gzip
```

In the first example, the `z` after the `tar` command is the option that tells `tar` that the file has been `gzip`ed. The second example is the `gzip` command, using the `d` option to decompress ‘`file.gz`’. `gunzip` is a command that is equivalent to `gzip -d`, but more intuitive.

```
bash-2.05$ gunzip file.gz <ENTER>
```

To compress a `tar` files, or any file, using `gzip`, you would type:

```
bash-2.05$ gzip -9 file.tar <ENTER>
```

The `-9` means best compression. There are many options available for `gzip`. A quick browse through the info pages will tell you everything you need to know.

Another common form of compression is the `bzip2` format. `tar` archives compressed using `bzip2` usually have the extension '`.tar.bz2`'. This format is known to be one of the best compression utilities. To extract a bzip2 file you would type:

```
bash-2.05$ bunzip2 file.bz2 <ENTER>
```

Like the `gzip` format, the `bzip2` format has many options. You will probably see `bzip2` compressing `tar` files. The `bzip2` utility many other commands linked to it. To extract a file that has been archived with `tar` and compressed with `bzip2`, you would do something similar to the following:

```
bash-2.05$ bzcat file.tar.bz2 |tar -xv <ENTER>
```

The `bzcat` command is a combination of `bunzip2` and `cat`. We `bzcat` the file, then pipe(`|`) the file to the `tar` command. Once again, I use `-x` to extract the file, and `-v` to see its contents. To compress a file using `bzip2` you would type:

```
bash-2.05$ bzip2 -z file1.txt <ENTER>
```

This command compresses '`file1.txt`' to '`file1.txt.bz2`'. You can also use the `'-9'` option for best compression.

Hopefully, this little chapter has gotten you more interested in using GNU/Hurd. If you are still confused, please read the info pages for the command that's giving you trouble. The command `info` will give you a list of all the programs on your Hurd machine that have documentation with them. The best way to learn GNU utilities is to read and practice. GNU/Hurd is just like music or sports: you can never learn enough. This chapter is meant to give you a stepping stone to freedom.

4.6 Administration

After the last chapter, you're probably wondering about this "root user" that everyone is so fond of. The Hurd is trying to get all things root out of the picture and allow you the freedom of doing things only root can do on legacy Unix systems. The designers and developers want you, the user, to be able to do things that you cannot do on a traditional Unix-like operating system.

For the time being, though, you will have to deal with the root user.

One reasonable request of the Unix-type gurus is making a normal user account. A normal user can only write to his or her home directory, and has limited access to system configuration. The reasoning behind this is that a normal user can not mess up the computer. This is a decent compromise because the more we explore and play, the more likely we'll lose a important file or make some other fatal mistake. So what we'll do is set up a user account for you. You must be root to do this.

```
bash-2.05# adduser <ENTER>
```

That's it! The computer will prompt you for a username and password, then you can just answer yes or whatever you require to the rest of the requests. To change the password on your account you just created you would type:

```
bash-2.05# passwd <USER_NAME> <ENTER>
Please Enter a Password:
```

There is a command that helps you become root while logged in as a normal user. This command is called `su`. When you type `su`, you are required to enter a password (if you have one) for the root account. If a `ssh` (Secure Shell) server doesn't allow root logins, as root you can `su` to a another user that has an access to that server. When you are done being another user you type `exit` and your back as root.

```
bash-2.05$ su <ENTER>
password:
```

There are many commands related to the Administering Unix-like operating systems. As a normal user, much is unneeded. Making a new user and giving him/her a password is enough for now. If you are interested in becoming a Administrator there are plenty of books and classes out there.

4.7 Accessing the cdrom

Traditionally, Unix-like systems have used a command called `mount` to merge removable storage devices, such as CDs and floppy disks, into the file system. On GNU/Hurd, we don't `mount` anything. The Hurd has a similar mechanism for accessing devices called "setting a translator".

We use the `settrans` command to do this. `settrans` is a program that aligns a type of Hurd server called a *translator* to a device supported by the kernel. For instance, on my GNU/Hurd system my cdrom device was detected as hd2. To get to my data on a iso9660 CD, I first had to make the device. Then I had to run the `settrans` command. This had to be done as the root user (root is the only user that has these privileges).

The term iso9660 is a standard set by the Industry Standards Organization.The ISO has set the universal access to CDROM media as standard number 9660. Some systems use an extension to this standard called Joliet. The Joliet extension is typical of Microsoft Windows. It is also supported on GNU/Hurd, GNU/Linux, and the BSDs.

```
bash-2.05# cd /dev <ENTER>
bash-2.05# ./MAKEDEV hd2 <ENTER>
bash-2.05# settrans -ac /cdrom /hurd/isofs /dev/hd2 <ENTER>
```

This isn't too hard, is it? Those options after `settrans` are very important. The `-a` makes the `/hurd/isofs` translator active. The `-c` creates the translator, and is only needed the first time.

Now you can type `cd /cdrom` and see the data on your CD.

4.8 Accessing the floppy

This section will help you get some data off of your floppy disk. I note that the `vfat (fat32)` and `msdos (fat16)` filesystems are currently not supported. The support will come eventually, but for now you can only access GNU/Linux floppies. You must be root to get to your floppy.

```
joe@hurd# settrans -a /floppy /hurd/ext2fs /dev/fd0 <ENTER>
```

Now you can head to the `/floppy` directory and see your data.

5 PC Hardware Basics

On Unix-like operating systems, it has always been a struggle for new users to get their hardware configured. Unix-like systems have a command called `dmesg` that allows users to view what hardware the kernel has detected. The command `dmesg` spits out all that data that we see at boot to your terminal. Unfortunately, the `dmesg` command has not been ported to GNU/Hurd. It should be ported in the near future. For now we have to use a couple of GNU programs to get equivalent information:

bash-2.05# cat /dev/klog > dmesg & <ENTER>
bash-2.05# less dmesg <ENTER>

As a user gets used to Unix-like systems, they should have a decent idea what they have in their systems. This section should help new users discover some things, and hopefully educate to some extent.

A typical computer has many common pieces of hardware regardless of vendor. A computer has these things in common:

Some systems have some of these pieces missing for certain reasons. Typically the missing pieces are omitted because of cost, the hardware being unneccesary, or other specialized reasons.

5.1 Motherboard

If your computer came with a printed manual, that manual will be very helpful; however, it will probably not tell you everything that you need to know. A person with their very own screwdriver can open up a computer and view some of the hardware. Just by looking at a certain card, you can get a lot of important information. On modern computer hardware, vendors are doing a better job at labeling their products. This is a great thing now that we have the Internet and search engines. Now, a person can write down some things that they see on a certain computer card, type that information into a search engine, and, in a matter of seconds, find out everything they need to know.

The *motherboard* lies at the heart of your computer's hardware. The name "motherboard" comes from the fact that all other pieces of hardware are connected to this board.

The motherboard is a large circuit board. It will have slots in which you may add other chips, such as a video card or a network card. When you see a computer advertised, and the advertisement mentions the number of expansion slots the motherboard has, it is referring to these slots.

The motherboard also has slots for RAM (memory) chips and CPUs. The number of slots for RAM chips determines the extent to which the computer's memory is upgradeable. As for CPUs, if your friend tells you that her computer is a dual-processor machine, that means that the machine's motherboard has slots for two CPUs, and that there is a CPU in each slot.

Other pieces of hardware that connect to the motherboard are power supplies, cooling fans, and disk drives. Together, your computer's hardware provides you with a platform on which you can run a complete operating system.

The motherboard has many chips all connected by SMT(surface mount technology) wiring. SMT is a technology that is used to connect tiny wires, transistors, diodes, capacitors, etc. The components collectively communicate with the CPU, Devices, and the BIOS. The Motherboard uses a IRQ(Interrupt ReQuest) controller to know which devices can interrupt the CPU to send of receive data. For instance a Harddisk will have lower IRQ number than a keyboard. The reasoning behind this is the keyboard is constantly being used whereas the harddisk only gets written to or read from periodically. A typical motherboard has 14 IRQ's, newer ones are giving the user more.

5.2 Floppy Drive and CDROM

Users need not be afraid of working on their computer. In the modern computer age manufacturers have made it very easy for us. Everything is like a puzzle piece. Almost no piece of hardware can be put in upside down or reversed. The hardest part of it all is the cabling that goes from the motherboard to the disks(Hard-drive, Floppy, Cdrom).

When taking apart your computer for the first time, it is always good to have a magic marker or some sort of writing utensil handy. It is a good idea to mark your cables as you pull them out just in case you have to stop working ,for example, to take out the dog. When you get back to to what you are doing you'll have some idea of what that cable belongs to. Another important thing to remember is on the cables that go to your internal devices always have a red line representing pin one. You should never try to do two things at the same time, this could make things more confusing.

5.3 PCI and EISA Slots

An internal device is usually a hard disk drive, floppy disk drive, CD drive, video card, network card or other important add-on. The disk drives are connected to the motherboard with ribbon cables. The video card, network card, and possibly a modem are inserted in slots on the motherboard. Slots are called PCI, EISA, and AGP.

5.4 Video Cards

The AGP (advance graphics port) slot is for video cards only. EISA (Enhanced Industry Standard Architecture) slots are for older 16-bit devices. You won't see that many EISA slots on todays motherboards. More favorable are PCI(Peripheral Component Interconnect) slots these are the most abundant on todays motherboards because they are for 32bit devices. The PCI slots are usually beige in color and are smaller than the longer black near obsolete EISA slots.

5.5 Hard Drive

The cabling connecting the disks to the motherboard are considered IDE(Intergrated Drive Electronics) ribbon cables. The motherboard also has a special cable for your floppy disk drive. Both the ide and floppy controllers should be labeled in very small print on your motherboard. The controllers should also have very small numbers on the ends to tell us where pin one is. The cables will have a red line to match pin one on the controller to pin one on the disk drives. If your lucky your disk drives and cables will be brand new so none of this will matter and they will only be allowed to connect the right way.

Each IDE controller will allow you to connect two devices. The two devices are considered master and slave. The master is always the first or only drive on a IDE controller. Master drives are usually bootable and contain the operating system's bootloader and kernel. A slave drive can be a CDROM or another Hard disk. Some operating systems are fussy about where they reside. It is generally a good idea to have a operating system booting off of a master harddrive. This isn't written in stone, Alot of software developers will have multiple operating systems on a single computer. Yet they usually put there bootloader on the master ide drive along with a couple of operating systems.

The motherboard usually has two IDE controllers; a primary and secondary. We are allowed a total of four IDE devices cabled off of our motherboards. Some of the newer motherboards are being manufactured with up to four ide controllers. A typical machine might have one cable connecting a hard disk off of the primary ide contoller. The machine could have the cdrom drive connected with a cable to the secondary ide controller. The machine might even have the cdrom drive connected as a slave on the primary ide controller. A very important way of connecting these devices this way are small jumpers on the disk drives. There should be clear instructions on the disks drives on how these jumpers should look for a certain configuration. The jumper selection area on the hardisk should look similar to the figure below.

```
MASTER SLAVE CABLE SELECT
  |———-|
  | o o o |
  | o o o |
  |———-|
```

If your new disk is going to be a master on the secondary ide controller **,** you need to make sure these jumpers are set correctly. If the jumpers are wrong and your cables are backwards you computer will not work the way you expect it too.

5.6 CPU

The devices connected through slots, and disks connected to the controllers all end up communicating with the CPU. These devices communicate via the "bus". The term "bus" comes from the fact that the data is sent down wires on the motherboard to the CPU all at once. You sometimes hear the term "I have a hundred megahertz bus"; this is the speed the data is traveling down the bus to the CPU. Megahertz(MHZ)is also the measurement of speed of a CPU. A CPU can be found in a slot on the motherboard. The CPU is either in slot form or in a ZIF(Zero Insertion Force)socket. Their are different types of ZIF configurations for certain CPUs. The ZIF sockets are usually labeled socket 8, socket 7, socket 3, etc. The slot based CPU are primarily for Pentium II and Pentium III CPUs. The very new CPUs use a new ZIF configuration called PGA(Primary Grid Array). The PGA sockets are usually for the Athlon and Pentium 4 processors. A ZIP socket makes it very easy to replace the CPU. With a little lever on the side of the socket a CPU can easily be replaced. All of these configurations regardless of CPU have heat sinks with fans built into them. The fan and heat sink play an important role. The heat sink keeps the CPU from burning up and overheating your computer. If you try and use a computer with out a working fan or without a heat sink your likely to run into sporadic errors. The errors can range from software errors to the computer restarting by itself.

5.7 RAM

While data is being worked on by the CPU the machine also uses RAM to store temporary data that the machine needs. RAM is one important component that allows us to use a computer that multi-tasks. The term multi-tasking is simply doing more than one thing at the same time. We should also know that if we don't have enough RAM for what we need to do, a computer will not multi-task efficiently. RAM is mounted on the motherboard in DIMM and/or SIMM slots. The RAM can only go in one way. A typical problem a person can have with RAM is not putting it in completely. Another problem is pushing the RAM too hard and cracking the motherboard. If either situation happens, when you start your computer you will get no picture, beeps, or any response at all. A DIMM stands for Dual Inline Memory Module. The DIMM is the most typical of newer mother- boards. The SIMM which stands for Single Inline Memory Module is not seen as much as the DIMM. The SIMM has a slower speed than the DIMM. The SIMM is seen on most older motherboards but they can both coexist together. The problem with the two different RAM Modules coexisting is the SIMM can only work with 66MHZ bus systems. If you have a DIMM that is rated for 100MHZ systems they will default to 66MHZ to live peacefully with the other RAM. When the bus speed is defaulted to 66MHZ, this can actually change the CPU speeds on some motherboards. Other differences between memory modules are the pins. A DIMM should have 168 pins while the SIMM will only have 72 pins.

5.8 BIOS

The BIOS is another good place to find information about the computer's hardware. The BIOS is the motherboard's little brain. It holds the data in the machine's CMOS (Complimetary Metal Oxide TranSistor) which is sort of like long term storage. The programming in the CMOS of the BIOS holds data such as hard disk information (Primary or Secondary, Master or Slave, Heads, Sectors,etc). It also holds the date, time, and some RAM-related information. A user can access the BIOS only at boot time. Usually there is a message telling us to hit F1,F2 or possibly Delete to enter setup. There are many confusing options in the BIOS. A good idea is to have the manual for your motherboard when dealing with any unknown features of the board. If a user changes some thing that causes havoc on the machine there is always a option to restore the settings to factory default. This can be a savior if we change too many things and forget what we did.

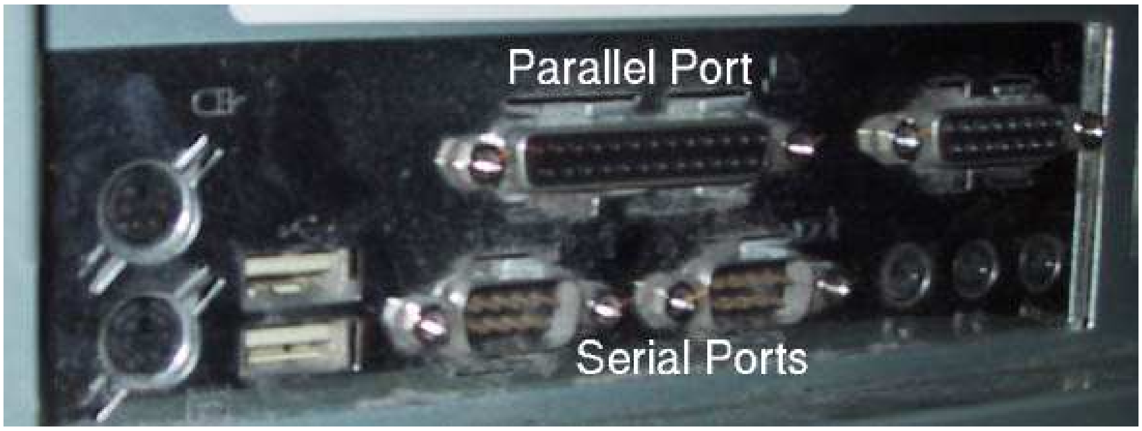
5.9 Power Supply

A computer's power supply has one connector that powers the mother- board. The motherboard's power connector is a large white block with yellow, black, and red wires.

The other connectors from the power supply are smaller D-shaped, with the same colored wires. The yellow wires are rated +-13 volts. The red wires are +-5 volts and the black ones are ground(0 volts). Both the motherboard and internal device connectors cannot be put in backwards or upside down. One common problem is not pressing the power connectors in completely.The floppy disk drive has it's very own power connector that is also keyed (think key and keyhole) to go in correctly. If your power supply is running hot or overheating, this can cause your machine to reboot and occasionally lock up. To figure out if this is the case one can simply touch the back of computer, right above the power supply. If it is hot to the touch this may be the clue you need. Power Supplies are also coming in different watt(240 thru 400) ratings. CPU's from different manufacturers require specific power requirements related to power supply and motherboard. These require- ments can sometimes add confusion when upgrading your system.

5.10 Serial and Parallel Ports

On the back of your computer there are D-shaped connectors. The D shaped connectors are for connecting external devices. Typically a motherboard could have two serial ports and one parallel port. The twenty-five pin parallel port is typically used for a printer or an iomega zip disk drive. The serial ports are the smaller fifteen pin ports. The serial ports are typically used for connecting external modems, palm devices, and possibly another computer. The uses of these ports are endless because they have been included with computers since the beginning of the computer age.



A few technologies that haven't been mentioned so far are USB (Universal Serial Bus), FireWire, and SCSI(Small Computer Systems Interface), which are beyond the scope of this text. The reason being is that they are technologies that are currently evolveing. The oldest of the three mentioned is SCSI. SCSI has undergone many revisions, we have SCSI-1, SCSI-2, Fast SCSI, Fast Ultra-Wide SCSI, and others. SCSI is a very versitile technology. We see SCSI in servers, high performance work stations and many other places. If you plan to make a profession out of hardware support, SCSI is a very important technology to learn.

When dealing with computer hardware it is always good to have as much documentation as possible. The previous text is by no means extensive, it is just an introduction. The more work you do with hardware, the easier troubleshooting becomes. Your natural senses help alot with dealing with computer hardware. Smell is a very good way of telling yourself that something is wrong. If you smell burning semi-conductor this is a sure sign that some part in your system has overheated to a point of failure. Vision is another good sense to use. If you see no picture on your monitor that could give a clue as to what maybe wrong, such as a unplugged monitor, unseated RAM, or other possibilities. Hearing is another sense that can be helpful. If you hear more than one or two beeps at boot, this can tell you that you computers BIOS is warning you that something is wrong. Again the more you get your hands dirty with computer hardware the more comfortable and educated you will be.

6 Cornerstone GNU Software

This section is to get a new user acquainted with software published by the GNU project. The GNU Project has applications for everything from desktop workstations to server-based solutions. The vast amount of free software available can keep the eager individual constantly learning and productive. In the next few sections we try to explain the most popular cornerstone GNU applications in no particular order. There are many free applications that are not listed here, so please do some research to find something you can use, learn, and enjoy.

A new GNU user should know some "lingo" relating to free software. *Source code* is software that can be built (compiled) using your computer's tools. Packages are *binary* (already compiled) distributions of a program. Packages can be made for easy install on many machines, since compiling the same program on many machines can take a long time.

Currently, Debian GNU/Hurd is the only GNU/Hurd distribution, so we use the Debian Project's *dpkg* package format. A Debian package has a '`.deb`' file extension. Other packages exist such as RedHat'ss RPM Package or a Unix package seen in FreeBSD or Slackware with a '`.tgz`' file extension.

6.1 GCC

The GCC (GNU Compiler Collection) software is set of computer language compilers and utilities. The computer programming languages the GCC supports is vast and grows frequently because of the extensive use of GCC. The GNU Compiler Collection has been ported to almost every computer operating system under the sun. GCC used to stand for GNU 'C' Compiler but was change to address all the added functionality that GCC has received. Richard Stallman originally wrote GCC as a component for his free operating system vision. Now there is hardly a UNIX-based operating system that can be developed without GCC. The GNU Project's compiler collection has been so influential that commercial companies use and contribute to it. This type of popularity has made GCC a industry standard and will keep it so for years to come.

6.2 GNU Emacs

GNU Emacs is the text editor component that Richard Stallman brought to his free operating system vision. RMS's GNU Emacs was far ahead of it's restrictive counterparts. Emacses were popular in the mid-eighties and there were many of them; however, GNU Emacs was the one that stood the test of time. Because of the license of distribution terms involved with GNU Emacs, people added functionality to the editor and as RMS applied those changes and made it available to anyone.

Today GNU Emacs can be considered more than a text editor; it can be considered a "one-stop shop", meaning that the features of GNU Emacs are endless. Consider the following features and try them for yourself:

- IDE(Integrated Development Environment)Support for many languages and debugging(GDB) of those languages.
- Send and Receive E-Mail

- Calendar
- CVS(Concurrent Version System)Front-end
- Many cool games(tetris,snake,et)
- Shell Access
- Access to Info pages

Judging by this list, there are plenty of stock utilities and fun things to enjoy in GNU Emacs. The above list is not complete and there are other goodies that can be added into GNU Emacs such as IRC (Internet Relay Chat) clients, web browser and many others.

6.3 GNOME

GNOME is a desktop environment that is integrated, fast, and feature-rich. GNOME stands for the Gnu Network Object Model Environment and consists of many applications to form the desktop on top of XFree86.

GNOME was written with a toolkit called GTK which stands for Gimp Tool Kit. GNOME gives us users a customizable launcher panel and menus which we can drag and drop icons of frequently used programs. We also have access to thousands of applications written with GTK that add endless possibilities to what can be integrated into our GNOME desktop.

Your *basic* GNOME Enviroment will have these following packages:

- gnome-core
- gnome-libs
- gnome-panel
- gnome-applets

A benefit of GNOME is that it can be used with many different window- managers. Window managers are programs that give us a user interface on UNIX-like systems. The GNOME Desktop environment is run on top of your favorite window manager. There are many, many window managers that are GNOME-compliant but the three that GNOME likes the most are called Enlightenment and Sawmill, and Metacity. They are all fast and nice to our computer's resources.

6.4 Windowmaker

Windowmaker is a great window manager for computers that are too slow or resource-deprived for GNOME. It features dockable icons and multiple desktops. Windowmaker is a GNOME-compliant window manager, but it also provides a great desktop on its own.

Windowmaker is capable of using themes and can be customized to no end. Many seasoned free software gurus use it because of speed, simplicity, and stability.

6.5 GIMP

The GIMP application is regarded as the best free photo editor around. GIMP has been developed to do more than edit digital photos, it is gaining functionallity to address the other common media that proprietary photo applications offer. The GIMP is widely used in digital movie studios to cut cost and deliver high quality digital media. GIMP is easy to pickup and learn but has layers and layers of features that can even keep a serious graphic designer busy. The GIMP was the cause of GTK (Gimp Tool Kit), which software developers can be helpful outside of the GIMP.

6.6 Binutils

The Binutils package is used with GCC to compile programs and is an essential piece of your development enviroment. Binutils containts multiple programs: an assembler, linker, and binary program utilities. It is common to use these programs to build programs aimed at different computer architectures and operating systems.

6.7 Coreutils

Coreutils is a combined package; it is essentially GNU's textutils, fileutils, and shell utilities. Coreutils is basically your core UNIX-like enviroment. The utilities are GNU replacements for traditional UNIX tools. The Coreutils package gives us the essentials, such as: `cat`, `wc`, `mknod`, `chgrp`, and many others.

6.8 Bash

The Bash shell is a standard on GNU-based systems. Its features include command line completion, history, and non-interactive usage (scripting and login).

Bash is a powerful program that has been ported, like many GNU programs, to many other computing platforms. A user who learns even minimal Bash skills can certainly be comfortable on other GNU-based systems. Please see (see Section 4.1 [The Shell], page 18) in this document for a brief tutorial.

6.9 Texinfo

Texinfo is GNU's official documentation system. Please see (see Section 11.4 [An Introduction to Texinfo], page 57) in this document for a detailed description.

6.10 Others

There is so much free software available for GNU and other free operating systems. The GNU Project has a searchable free software directory on `http://www.gnu.org`. Almost anyone willing to dig around the internet will be able to find something they can use. Sometimes a program can be found to replace your current favorite application at half the cost or no cost at all. Finding the software you need can sometimes be overwhelming, but asking a guru might lead you in the right direction.

Other programs available from the GNU Project that might be of use include:

- gawk: The gnu version of the awk language.
- gnupg: The GNU Privacy Gaurd is a replacement for PGP used for secure communications via e-mail or other means.
- gnumeric: A excellent full featured spreadsheet program.
- automake, autoconf, gmake : These three are a great addition to your developement toolbox. The three have been a tremendous help in getting free software's quality to the state it is today.

7 Networking

In this chapter, you'll learn about the Hurd's networking features. The Hurd's networking is a work in progress. The services that the Hurd now has should be sufficient for the average user. Available now are `ftp`, `nfs`, `telnet`, and, of course, web-surfing and e-mail.

7.1 Configuring

Before we get to the outside world, we need to configure the *network card*. This is done with the `settrans` command, yet it can be a little tricky. First you have to pay attention when you boot to see if the microkernel detected your card as `eth0`. Once you are sure that your network card has been detected, you can give it the translator that it needs.

```
bash-2.05# settrans -fgcap /servers/socket/2 /hurd/pfinet -i eth0 \
-a 192.168.1.3 -m 255.255.255.0 -g 192.168.1.5 <ENTER>
```

In this example, the `settrans` command uses the following options: `'f'`, `'g'`, `'c'`, `'a'`, and `'p'`. The `-fg` forces anything that is attached to '`/servers/socket/2`' and '`/hurd/pfinet`' to go away. Next, the `-c` tells `settrans` to create a translator. The `-ap` tells the '`/server/socket/2`' and '`/hurd/pfinet`' to be both active and passive. The passive option will make the configuration remain valid even after the reboots and shutdowns. Here are the rest of the options:

```
-i eth0 = The interface that we are configuring.
-a 192.168.1.3 = This is the ip address of my machine.
-m 255.255.255.0 = This is the netmask of my machine.
-g 192.168.1.5 = This is the gateway machine on the network.
```

Unfortunately, GNU/Hurd currently has no *DHCP* client (DHCP stands for Dynamic Host Configuration Protocol, its used by many computers to attach themselves to a network). There is, however, a workaround if you have another operating system, such as Debian GNU/Linux, installed on your GNU/Hurd machine. In describing this workaround, I'll assume that you are using a GNU/Linux system.

Boot GNU/Linux and log in as root. Type the following two commands, and observe their outputs:

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:E0:4C:E1:C0:6D
          inet addr:154.17.21.134  Bcast:154.17.31.255  Mask:255.255.240.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16187 errors:0 dropped:0 overruns:0 frame:0
          TX packets:438 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:2235054 (2.1 MiB)  TX bytes:59122 (57.7 KiB)
          Interrupt:11 Base address:0xe400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:3924  Metric:1
          RX packets:366 errors:0 dropped:0 overruns:0 frame:0
```

```
          TX packets:366 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:18916 (18.4 KiB)  TX bytes:18916 (18.4 KiB)

# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
154.17.16.0     *               255.255.240.0   U     0      0        0 eth0
default         MYrouter.Domain 0.0.0.0         UG    0      0        0 eth0
```

Note the `inet addr` and `Mask` from `ifconfig`, and the `Destination` field from `route`. In our example, the values are for the former two are 154.17.21.134 and 255.255.240.0; these are your IP address and netmask, respectively.

To get the value for your gateway machine, use the `ping` program on the host name corresponding to the *default* destination from the output of `route`; in our example, we would type

```
$ ping MYrouter.Domain.ca ⟨ENTER⟩
```

Note the IP address that `ping` reports; this is the IP address of your gateway machine. Reboot into GNU/Hurd, and use these three values to set your network translator as described above.

The next items to configure are our '`resolv.conf`' file and our '`hosts`' file. These are located in the '`/etc`' directory. In fact, all GNU/Hurd's configuration files are located in the '`/etc`' directory. The '`resolv.conf`' file is the first file to edit. It only consists of 2 to 3 lines but it is one of the most important networking files. Here is an example:

```
search mydomain            #This should be your networks domain name
nameserver xxx.xxx.xxx.xxx #These should be the primary and secondary
nameserver xxx.xxx.xxx.xxx #ip addresses of your isp's nameservers(aka DNS
servers)
```

If you used the DHCP workaround described above, just copy your '`/etc/resolv.conf`' (in GNU/Linux) to '`/etc/resolv.conf`' (in GNU/Hurd).

After this is all set up, you should be able to use the `ping`[1] program on any host on the Internet and recieve a response. The next file to setup is the '`/etc/hosts`' file. This file is used for machines you access frequently. Instead if typing ip addresses for computers on your network, you edit '`/etc/hosts`' to allow you to just type the name of the computer you want to access.

```
127.0.0.1      hurd localhost
192.168.1.3    hurd.mydomain.org hurd
192.168.1.5    jojo.mydomain.org jojo
192.168.1.6    mojo.mydomain.org mojo
```

Now I can access 192.168.1.5 by simply using its name, *jojo*. The same is true with *mojo* also. For instance if I wanted to `telnet` into *jojo*, I would type `telnet jojo`.[2]

[1] `ping` is a small program that we use to test whether or not we can talk to another machine.

[2] With the proliferation of mischief on the Internet it is recommended you use the OpenSSH (`http://www.openssh.org`) suite to access remote computers.

7.2 Accessing FTP

FTP stands for File Transfer Protocol. This protocol has been around since the beginnings of GNU and Unix. It is a easy and fast way of sending files to friends, family, and colleagues. The Hurd has two methods of using ftp. One is setting a translator with the Hurds `ftpfs` server. The other way is the traditional client-server method in which you login and use `get` and `put` commands.

The `ftpfs` translator is quite unique. It allows you to have a remote FTP server accessible on a local directory. You would then begin copying files to and from it as if it were a directory on your machine.

```
$bash-2.05#settrans -ac /mnt /hurd/ftpfs / alpha.gnu.org
```

This command will set a translator for `ftpfs`. The `ftpfs` translator places the root directory of alpha.gnu.org's anonymous ftp server on the '`/mnt`' directory. Now one can access the data as if it were a local filesystem.

The FTP protocol is traditionally accessed using a client program. GNU/Hurd currently can use the one that comes with the GNU projects `inet-utils` package. The traditional method of ftp is pretty much a standard for Unix-like operating systems. If you learn this method you should be able to comfortably sit at any Unix-like workstation and send and retrieve files. I will give you a start but, there are many more options, so read the info pages for some more complex operations.

```
bash-2.05:$ ftp <ENTER>
ftp> open ftp.gnu.org
```

You should then be prompted for a username and password. You can log into servers on the Internet who offer anonymous ftp access. A anonymous ftp server is a server that allows anyone access to its files. You would simply type anonymous at the user prompt then type your e-mail address as the password.

The standard commands that are used are `get`, `put`, `mget`, `mput`, `ls`, `pwd`, and `cd`. These will work with any standard ftp client. The commands themselves are fairly self-explanatory. The `mget` and `mput` are for multiple files. An example would be:

```
ftp> mget *.tar.gz
```

This would try to download any files in the current directory with a '`.tar.gz`' extension. Don't worry if you say `mget` instead of `get` by accident, because it will prompt you at every file. It will allow you to say `yes, no, all, abort` and `quit`. The ftp protocol and applications are fairly simple to learn, you just have to practice. Trying it a handful of times will get you very familiar with it, very quickly.

7.3 Acessing NFS

The NFS protocol is a standard method for sharing files between Unix-like workstations. The server allocates directories that it wants to make sharable through exporting. Traditionally, the workstation mounts the shared directory onto the local filesystem.

For instance, Server1's '`/etc/exports`' file it says:

`/usr/src mojo(rw,insecure)`

This would allow the workstation known as *mojo* read and write access to the '`/usr/src`' directory. The workstation would then type `mount Server1:/usr/src /mnt`. The workstation (*mojo*) can `ls /mnt` and see all the files in *Server1*'s '`/usr/src`' directory. (Note: The '`/etc/hosts`' file on both server and workstation must have appropriate ip address in order for things to work.)

On GNU/Hurd, we access a network share by setting the `nfs` translator. This is done like so:

```
bash-2.05# settrans -ac /mnt /hurd/nfs hostname:/shared/directory
```

The remote shared directory is now visible on our '`/mnt`' directory. We can then start copying, reading, and writing files as if the directory was on our own workstation.

This is just the basics of the NFS protocol. If you are very interested in NFS there is limitless amounts of documentation on the Internet. It is well known to be insecure, so be very careful what you do with it.

7.4 Web Surfing

On GNU/Hurd, we can surf the web on the console. The program that we use is a text-mode web browser called Lynx. The benefits of using a text-mode web browser are speed and simplicity. In text-mode, you don't have to load any graphics, and you don't get those annoying pop-ups and other annoyances that are involved with the World Wide Web. To use Lynx, you type the command `lynx`, then a URL like this example:

```
bash-2.05$ lynx www.google.com <ENTER>
```

This starts us out with '`www.google.com`'. To go to another URL, you type the letter, g, meaning "Go." Lynx then prompts you for a address. To access a link on a web page, press the -> key. To go back, press the <- key. Pressing the h key gives us a big section of help topics. You can even download information by placing the cursor over the item you want, and pressing the p key. Lynx then prompts us to "Save to File" or "Print". Lynx is very addictive, and once you get used to it you might not want any other browser.

8 Translators

The Hurd filesystem allows you to set translators on any file or directory that you own. A *translator* is any Hurd server which provides the basic filesystem interface. Translated nodes are somewhat like a cross between Unix symbolic links and mount points.

Whenever a program tries to access the contents of a translated node, the filesystem server redirects the request to the appropriate translator (starting it if necessary). Then, the new translator services the client's request. The GNU C library makes this behavior seamless from the client's perspective, so that standard Unix programs behave correctly under GNU/Hurd.

Translators run with the privileges of the translated node's *owner*, so they cannot be used to compromise the security of the system. This also means that *any* user can write their own translators, and provide other users with arbitrary filesystem-structured data, regardless of the data's actual source. Other chapters in this manual describe existing translators, and how you can modify them or write your own.

The standard Hurd filesystem servers are constantly evolving to provide innovative features that users want. Here are a few examples of existing translators:

- Disk-based filesystem formats, such as `ext2fs`, `ufs`, and `isofs`.
- Network filesystems, such as `nfs` and `ftpfs`.
- Single files with dynamic content, such as FIXME: we need a good example.
- Hurd servers which translate rendezvous filesystem nodes in standard locations, so that other programs can easily find them and use server-specific interfaces. For example, `pflocal` implements the filesystem interfaces, but it also provides a special Unix-domain socket RPC interface (FIXME xref). Programs can fetch a port to this translator simply by calling `file_name_lookup` (FIXME xref) on '`/servers/socket/1`'[1], then use Unix socket-specific RPCs on that port, rather than adhering to the file protocol.

This section focuses on the generic programs that must understand in order to use existing translators. The Hurd Hacking Guide and the GNU Hurd Reference Manual describe how you can write your own translators.

8.1 Invoking `settrans`

The `settrans` program allows you to set a translator on a file or directory. By default, the passive translator is set (see the '`--passive`' option).

The `settrans` program has the following synopsis:

```
settrans [option]... node [translator arg...]
```

where *translator* is the absolute filename of the new translator program. Each *arg* is passed to *translator* when it starts. If *translator* is not specified, then `settrans` clears the existing translator rather than setting a new one.

`settrans` accepts the following options:

[1] The number 1 corresponds to the `PF_LOCAL` C library socket domain constant.

'-a'
'--active'
: Set *node*'s active translator. *Active translators* are started immediately and are not persistent: if the system is rebooted then they are lost.

'-c'
'--create'
: Create *node* as a zero-length file if it doesn't already exist.

'-L'
'--dereference'
: If *node* is already translated, stack the new translator on top of it (rather than replacing the existing translator).

'--help' Display a brief usage message, then exit.

'-p'
'--passive'
: Set *node*'s passive translator. *Passive translators* are only activated by the underlying filesystem when clients try to use the *node*, and they shut down automatically after they are no longer active in order to conserve system resources.

  Passive translators are stored on the underlying filesystem media, and so they persist between system reboots. Not all filesystems support passive translators, due to limitations in their underlying media. Consult the filesystem-specific documentation to see if they are supported.

  If you are setting the passive translator, and *node* already has an active translator, then the following options apply:

  '-g'
  '--goaway'
  : Tell the active translator to go away. In this case, the following additional options apply:

    '-f'
    '--force' If the active translator doesn't go away, then force it.

    '-S'
    '--nosync'
    : Don't flush its contents to disk before terminating.

    '-R'
    '--recursive'
    : Shut down all of the active translator's children, too.

  '-k'
  '--keep-active'
  : Leave the existing active translator running. The new translator will not be started unless the active translator has stopped.

'-P'
'--pause' When starting an active translator, prompt and wait for a newline on standard input before completing the startup handshake. This is useful when debugging a translator, as it gives you time to start the debugger.

'-t *sec*'
'--timeout=*sec*'
If the translator does not start up in *sec* seconds (the default is 60), then return an error; if *sec* is 0, then never timeout.

'--version'
Output program version information and exit.

'-x'
'--exclusive'
Only set the translator if there is none already.

8.2 Invoking `showtrans`

The `showtrans` program allows you to show the passive translator setting on a file system node.

The `showtrans` program has the following synopsis:

```
showtrans [option]... file...
```

`showtrans` accepts the following options:

-p
--prefix Always display *filename*: before translators.

-P
--no-prefix
Never display *filename*: before translators.

-s
--silent No output; useful when checking error status.

-t
--translated
Only display files that have translators.

8.3 Invoking `mount`

8.4 Invoking `fsysopts`

The `fsysopts` program allows you to retrieve or set command line options for running translator *filesys*.

The `fsysopts` program has the following synopsis:

```
fsysopts [option...] filesys [fs_option...]
```

`fsysopts` accepts the following options:

`-L`
`--dereference`
: If *filesys* is a symbolic link, follow it.

`-R`
`--recursive`
: Pass these options to any child translators.

The legal values for *fs_option* depends on *filesys*, but some common ones are:

`--readonly`
`--writable`
`--remount`
`--sync[=`*`interval`*`]`
`--nosync`

If no options are supplied, *filesys*' current options are printed.

9 Troubleshooting

In this chapter, we'll describe some basic troubleshooting steps to recover from a system crash. These sections are only the basics to get you up and running in multi-user mode after a lock-up or some other minor catastrophe.

9.1 Fscking the filesystem

Occasionally, a mistake we make, or a bug in a program, causes the machine to lock up. Usually, you have to shut the machine down with the power button, or reset it. Your filesystem on which GNU/Hurd resides does not like this at all. GNU/Hurd uses the `e2fsck` tool to repair the damage done. When something happens, GNU/Hurd will invoke this at boot if the file system is marked unclean.

Occasionally, the system cannot do this automatically, and you must repair the filesystem manually. First of all, you cannot repair a filesystem that is read-only; you need the `fsysopt` command that we learned from the previous chapter.

To begin, you must be in single-user mode. If you boot your machine after a crash and the system cannot recover automatically, you will be in single-user mode by default after booting. At the prompt, enter the following commands (of course, substitute the device name of your root filesystem for '`/dev/hd0s1`'):

sh-2.05# fsysopts / –writable <ENTER>
sh-2.05# e2fsck /dev/hd0s1 <ENTER>

The first command makes root filesystem writable, and the second runs the *`e2fsck`* program on the device where the filesystem resides. As this command runs, it will ask the user to repair certain things, always defaulting to YES. You should be able to just sit there and press *`ENTER`* until it tells you that the filesystem is clean.

You can then either type *`reboot`* and have a normal, clean startup, or you can type *`exit`* to leave single user mode and boot into multi-user mode.

9.2 Booting and GRUB

If your having problems getting your system to boot using the pre-made Grub boot disk please take a look at this section.

1. You should try and remember(and write down) where you installed GNU/Hurd regardless of which method of installation you chose. For instance if you installed from the CDs and the Hurd was placed on /dev/hda2 you need to know that your Hurd install is on /dev/hd0s2
2. You need to remember that Grub's root is one down from your installation root whether your using GNU/Linux of GNU/Hurd. When I mention *one down* I mean if you have a Hurd install on hd0s1 your Grub's root should be "root (hd0,0)" and if your Hurd's install is on hd0s2 your Grub's root should be "root (hd0,1)".

3. You should remember that Grub ignores your cdrom drive. So if your second harddrive is the primary drive on the second ide bus the Hurd sees it as hd2s1 whereas Grub's root would be (hd1,0).. (Confusing for us LiLO converts isn't it?)
4. Most importantly you need to make sure the syntax is correct on your module lines; Brackets, Spaces, Etc. have thier meannings and cannot be interchanged. If your editor is a pain you can put a forward slash to seperate the commands on seperate lines. This will still be seen as one module line to Grub.For instance:

   module /hurd/ext2fs.static \
   –multiboot-command-line=${kernel-command-line} \
   –host-priv-port=${host-port} \
   –device-master-port=${device-port} \
   –exec-server-task=${exec-task} -T typed ${root} \
   $(task-create) $(task-resume)
5. If nothing is going your way you can try the out-dated serverboot method.

   root (hd0,0)
   kernel /boot/gnumach.gz root=hd0s1
   module /boot/serverboot.gz

   Makeing sure you replace Grub and the Hurd's root with the appropriate syntax.

10 Finding More Information

10.1 The info pages

The info pages contain complete manuals for GNU software. The GNU Project uses Texinfo (see Section 11.4 [An Introduction to Texinfo], page 57) to produce manuals as info pages, web pages, and printed documents.

An info file is simply a file on your machine. To access an info file, you will need an info reader. The default info reader is called `info`, others are `pinfo` and the GNOME Help Browser. In our examples, we will assume that you are using `info`.

Info pages are hierarchally-structured documents. Each info page consists of a set of nodes, which typically correspond to chapters and sections in a printed manual. Each node contains pointers to other nodes, which behave similarly to hyperlinks on web pages. Minimally, each node has three pointers: *Next*, *Prev* (Previous), and *Up*. Nodes may contain other pointers that are cross-references to other nodes.

If you are reading this manual as an info file, this node's name is "The info pages," its *Next* pointer points to "The man pages," and its *Prev* and *Up* pointers both point to "Finding More Information." In a printed version of this manual, "Finding More Information" is a chapter, and "The info pages" and "The man pages" are sections of that chapter. In addition, the first paragraph of this chapter contains a cross-reference to a node called "An Introduction to Texinfo."

Info pages show us how to use GNU software. As an example, suppose that we have GNU Shogi[1] installed on our GNU system, and we want to know how to play it. Typing `info gnushogi` at our command prompt, we are presented with the following:

```
File: gnushogi.info,  Node: Top,  Next: (dir),  Prev: (dir),  Up: (dir)

GNU Shogi (Japanese chess)
**************************

* Menu:

* Introduction::          What is GNU shogi?
* License::               The GNU General Public License.
* About shogi::           General information, rules, etc.
* gnushogi::              How to play GNU shogi (gnushogi).
* xshogi::                The X interface to GNU shogi.
* References and links:: Where to go for more information.
* Acknowledgements::
* Bugs::                  Where and how to report bugs.
* Index::
```

This menu presents us with a list of pointers to nodes in the GNU Shogi manual. If we move our cusor to, for example, the "gnushogi::" entry and press ⟨Enter⟩, we will move to a new node of the GNU Shogi manual.

[1] Shogi is the Japanese version of chess.

There are a few keys that you will use repeatedly when using `info`; for further information, type `info info` at your command prompt.

- ⟨n⟩ : Follow this node's *Next* pointer.
- ⟨p⟩ : Follow this node's *Prev* pointer.
- ⟨u⟩ : Follow this node's *Up* pointer.
- ⟨l⟩ : Return to the previously-visited node (like hitting "Back" in a web browser).
- ⟨q⟩ : Quit `info`.

10.2 The man pages

The man pages, short for manual pages, are the traditional source of documentation for UNIX and UNIX-like systems. In a UNIX system, each command, such as `cat` or `cp`, would have its own man page. Manual pages would also exist for the system calls used by *C* programmers, such as `mmap`.

As with the GNU system's info pages, man pages are files stored on your disk, and you need a man page reader in order to view them. On GNU systems, the man page reader is called `man`. To view, for example, the man page for `rm`, we would type `man rm` at our command prompt.

The man pages have several deficiencies when compared to the info pages; for example, they have no hyperlink-like mechanism for navigation, and often assume that the reader has a great deal of technical knowledge. These, and other problems, make the man pages a poor source of documentation for beginners. Nevertheless, many GNU users who come from a UNIX background are very used to the man pages, so the man pages are available on GNU systems.

Since the man pages do not provide any hyperlink-like mechanism, we must find another way to search a long manual page for information. The solution to this problem is the the ⟨/⟩ key. Pressing ⟨/⟩ while viewing a manual page causes the `man` program to prompt us to enter a string, followed by the ⟨Enter⟩ key. `man` will then jump to the first occurence of the string we enter, and highlight all occurences of the string. Pressing ⟨/⟩ again, followed by ⟨Enter⟩, will tell `man` to jump to the next occurence of the string.

The ⟨/⟩ key allows us to search within documents, but we still need a way to find out which man pages to consult on a particular topic. The command that helps us do this is `apropos`. For example, if we did not know the command for copying files, we would type `apropos copy`. The `apropos` program would then present us with a list of man pages for which "copy" is a keyword.

More information on the man pages can be obtained by typing `man man` or `man apropos` at your command prompt.

10.3 HOWTOs

A HOWTO provides help on a specific subject, such as:

- Using software RAID
- Printing

- Programming `bash`
- Using APT

You will find HOWTOs to be extremely useful. A good example HOWTO that should be of interest to you is the *APT HOWTO* at `http://www.debian.org/doc/manuals/apt-howto/index.en.html`.

10.4 Websites of Interest

There are several websites that you will find extremely useful as you learn how to use Free UNIX-like systems such as GNU:

- **LinuxNewbie.org - `http://www.linuxnewbie.org/`**: LinuxNewbie.org offers *Newbieized Help Files (NHFs)*, which are like HOWTOs, but aimed specifically at beginners. Despite the name, LinuxNewbie.org, most of the information provided is applicable to non-Linux-based GNU systems, and to *BSD systems also.
- **The Linux Documentation Project - `http://www.tldp.org/`**: *The Linux Documentation Project* makes HOWTOs, in-depth guides, FAQs, and man pages. In fact, many of the man pages available on GNU systems come from the Project. As with LinuxNewbie.org, most of the Project's documentation is applicable to non-Linux-based GNU systems, and to *BSD systems also.
- **Debian Documentation - `http://www.debian.org/doc/`**: Debian provides manuals, HOWTOs, FAQs, and other documents with information specific to Debians distribution of GNU systems.
- **Documentation of the GNU Project - `http://www.gnu.org/doc/doc.html`**: Provides online versions of the GNU info pages, along with other Free Documentation.
- **GNU's Hurd Page - `http://www.gnu.org/software/hurd/`**: The official web site of the GNU Hurd.
- **Debian GNU/Hurd - `http://www.debian.org/ports/hurd/`**: Debian's GNU/Hurd web pages.
- **The Hurd Wiki - `http://hurd.gnufans.org/`**: A wiki is a forumn for online discussion and collaboration that is browsable as a web page. To a user, it is essentially a web page that is editable by its users. The Hurd Wiki is a wiki for GNU/Hurd, and it contains a wealth of insights by other GNU/Hurd users; it is an excellent resource.

10.5 Searching the web

When we have a specific problem, and cannot easily find a solution in the documentation, searching the web will often help us solve our problem. Currently, the most popular search engine is *Google*, available at `http://www.google.com`.

We will present a real-life example. Suppose that I am using the Mozilla web browser on a Debian GNU system, and I visit a web site that contains a Java applet, but Mozilla does not launch the applet. To find out how to fix this problem, I visit Google:

sh-2.05$ lynx www.google.com ⟨Enter⟩

Google presents me with a text box into which I can enter *search terms*. Into this text box, I type *debian java mozilla*. Google will then search the web for pages containing these three terms. I look through the web pages that it gives me, and find one that tells me to make a symbolic link from Mozilla's plugin folder to the Java plugin provided by my Java installation:

cd /usr/lib/mozilla/plugins ⟨Enter⟩
ln -s /usr/j2sdk1.4.0_01/jre/plugin/i386/ns610/libjavaplugin_oji.so ⟨Enter⟩

I then re-start Mozilla, and am able to use Java applets.

10.6 Mailing lists

Many free software projects offer mailing lists for support, inter-project communications, and significant announcements. The mailing lists are essential sources of information.

Mailing lists for GNU/Hurd are served to us by the GNU Project. The Debian GNU/Hurd port also has a mailing list hosted by the Debian Project. New GNU/Hurd users have three mailing lists to help them on thier way: help-hurd@gnu.org, bug-hurd@gnu.org, and debian-hurd@lists.debian.org.

- **help-hurd@gnu.org** : This is the primary list for new users who need help installing, booting, and using GNU/Hurd. If you are a programmer, this list is also the appropriate place to for help developing programs that use Hurd-specific features.
- **debian-hurd@lists.debian.org** : This list is used to communicate about issues regarding the Debian distribution (currently the only distribution) of GNU/Hurd. Much information is replicated on the help-hurd list.
- **bug-hurd@gnu.org** : This is a list to report bugs and to see if you are experiencing some side effect of a bug. Hurd developers also use this list to discuss the design and implementation of the Hurd servers. If you subscribe to the Bug-Hurd list, you will receive many such emails. Do not worry, users are not expected to understand all the technical details discussed on this list.

There are two other GNU/Hurd lists that will be of lesser interest to a new user: l4-hurd@gnu.org, on which people discuss a planned port of the Hurd servers to the L4 micro-kernel, and web-hurd@gnu.org, on which peopole discuss the Hurd's web pages.

You can sign up for the Help-Hurd, Bug-Hurd, and Debian GNU/Hurd mailing lists at the following URLs:

- **help-hurd@gnu.org** : `http://mail.gnu.org/mailman/listinfo/help-hurd`.
- **debian-hurd@lists.debian.org** : `http://lists.debian.org/debian-hurd/`.
- **bug-hurd@gnu.org** : `http://mail.gnu.org/mailman/listinfo/bug-hurd`.

Mailing list archives are kept for many years. If you search for some information on a topic, you might get data from years ago. This old information might be of some help to you, or it may be outdated. Mailing list archives are so heavily-used that a search on the web will give you many hits from different lists. To narrow things down, you need to find out where the list in question resides and search the server hosting the list. The lists mentioned above are currently archived at the following locations:

- **help-hurd@gnu.org** : `http://mail.gnu.org/pipermail/help-hurd/`.
- **debian-hurd@lists.debian.org** : `http://lists.debian.org/debian-hurd/`.

- **bug-hurd@gnu.org** : `http://mail.gnu.org/pipermail/bug-hurd/`.

Most hackers hate repeating publicly-available information. Before asking for help with a certain issue on a mailing list, a user should search the GNU/Hurd documentation and the mailing list archives for an answer to the question. People who post to the Help-Hurd mailing list asking questions that are answered in the Hurd FAQ, for example, will usually be ignored, or at least told to consult the relevant documentation.

Before leaving the subject of mailing lists, we should discuss the topic of mailing list etiquette. We have already mentioned that it is impolite to ask questions that are answered in the documentation or the mailing list archives. There are a number of rules of thumb that, if followed, will make your life much easier:

- **Be polite and respectful, and refrain from insulting others.** : This is a good rule of thumb in all social interactions, but it can be easy to forget when the other person is half-way across the world, and unable to give you a dirty look. Now, if you follow the mailing lists for any free software project, you will notice people who violate this rule. This behaviour may be tolerated if it comes from a great hacker, but it does not make the person more popular.
- **Stay out of flame wars.** : A *flame war* is an emotional, and often irrational, argument between people who hold vastly different views on a subject. Flame wars on mailing lists usually only annoy the list's subscribers, who have joined the list in hope of sharing *useful* information. There are people in this world who will get into flame wars on just about everything, from choice of kernel, to text editor, to desktop environment. Ignore them.
- **Respect the terminology and values of the GNU Project and Debian.** : Users of Debian GNU/Hurd are not in any way required to share the views of GNU or Debian; however, it is polite to realize that these projects have social goals, and to respect those goals while participating in disucssions on the mailing lists. For example, promoting proprietary software on and GNU mailing lists is discouraged. Similarly, it is in good taste to use the terminology used by the GNU and Debian projects; for example, say *GNU/Linux* when referring to the GNU system running on the Linux kernel, instead of referring to the entire system as simply *Linux*, and say *free software*, not *open source*. Again, following this rule of thumb is not a requirement, but a courtesy which will make your life easier.
- **Be prepared to help solve problems.** : If you report a problem on a mailing list, it is likely that you are not the only user experiencing that problem. It is impolite to report your problem, but ask that no one reply to you. GNU/Hurd users and developers want to fix problems in the GNU/Hurd system. When reporting a new problem, you may be asked for more information, or asked to try different solutions to the problem. In the next chapter, we will provide you with an introduction to the skills you will need in order to help track down the sources of these problems, and otherwise help contribute to the development of GNU/Hurd.

11 Helping Out

In this chapter, we'll be describing the basic skills you need in order to help develop the GNU system. You do not need to be a hot-shot programmer capable of writing incredibly-obscure code to help with GNU's development; users willing to test the system for bugs and suggest improvements are valuable, as are writers of documentation, promoters, and others.

Although this chapter is intended to give you an introduction to the skills you will need in order to help develop GNU, much of this material will also be of interest to users who want to tweak their system, and compile bleeding-edge programs.[1]

11.1 Basic CVS Usage

Web pages for Free Software often state something along the lines of, "The source for this program is available through CVS." A reasonable question upon seeing this statement is, "What the heck is CVS?"

CVS stands for Concurrent Versions Systems. It is a very important tool for Free Software developers. CVS stores the files in a project in a centralized location called the *Repository*, which will usually be connected to a server.[2]

In the GNU system, we access CVS repositories using a program called `cvs`. Your first introduction to CVS will probably be getting files for a bleeding-edge program using *anonymous CVS*. This means that you will contact a CVS server, tell it that you are an anoymous user, and ask it for the files you want. When using anonymous CVS, you typically have permission to get files, but no permission to store files in the Repository.

As an example, let's look at some example CVS commands from the *Hurd Hacking Guide*:

```
#! /bin/sh

cd $HOME
mkdir hurd-cvs
cd hurd-cvs/

# Use the empty string password:
cvs -d:pserver:anoncvs@subversions.gnu.org:/cvsroot/hurd login

for module in hurd gnumach
do
  cvs -z3 -d:pserver:anoncvs@subversions.gnu.org:/cvsroot/hurd \
     co $module
```

[1] "Bleeding-edge" is a pun on "leading-edge"; leading-edge products often have had very little testing, and may contain serious bugs.

[2] In general, the term *server* refers to a program or a computer that provides a service to others; in this case, the server is a computer connected to the Internet, coupled with software that allows people store and retreive files through CVS.

```
done
```

This simple shell script makes a directory in our home directory called '`hurd-cvs`', and places the source code for GNU Mach and the Hurd in that directory.

We'll translate these `cvs` commands into English to help you get started with CVS. As the CVS manual (available on the GNU system through `info cvs`) states, the structure of CVS commands is:

cvs [cvs_options] cvs_command [command_options] [command_args]

`login` and `co` (checkout) are both `cvs_commands`. So, our `cvs_options` in the above commands are `-d:pserver:anoncvs@subversions.gnu.org:/cvsroot/hurd`, for both commands, and `-z3` for the second command.

`-z3` is simple to understand: it specifies the level of compression of the files we're checking out. In this case, the files have been compressed at compression level three.

`-d:pserver:anoncvs@subversions.gnu.org:/cvsroot/hurd` is slightly more complicated. The `-d` stands for *directory*; it tells `cvs` that the remainder of this string specifies the location with which we are communicating. `:pserver:` stands for *password-authenticated server*; `subversions.gnu.org` is a server that uses passwords to authenticate its users. '`/cvsroot/hurd`' is a directory on `subversions.gnu.org`. So, the whole string says, "I want to interact with a password-authenticated server called `anoncvs@subversions.gnu.org`, and the files that interest me are in a directory on it called '`/cvsroot/hurd`'."

This should give you a basic idea of what CVS is. Don't be intimidated by it; although its syntax may seem cryptic, you will probably not need to use the full range of its abilities, and what you do need to know can be easily learned when you need it.

More advanced CVS usage is beyond the scope of this document; however, the CVS manual is quite good and can be accessed locally by typing `info cvs` in your shell on any GNU system. The manual is also available online at `http://www.gnu.org/manual/cvs/index.html`.

11.2 Using gcc

GCC stands for "GNU Compiler Collection." The acronym originally stood for "GNU C Compiler." GCC is one of the cornerstones of Free Software.

GCC is a *compiler*: a program that takes as input a file in a high-level computer language, and produces a set of machine-readable instructions that form an executable program. The input file is called the *source code*; the availablility of this code to all for modification and improvement is a hallmark of the Free Software and Open Source movements.

To see how compilers such as GCC work, we will use a common example program called *Hello World*. Any book that describles a computer language begins by showing a variation of this program written in that language, since it shows the minimum functionality required to let the user know that the program has been run. Our version of Hello World is written in *C*, the language in which GNU Mach and the Hurd servers are written.

Type the following text into your favourite text editor:

```
#include <stdio.h>
```

```
int
main()
{
  printf(``Hello, Welcome to the GNU/Hurd Community!\n'');
  return 0;
}
```

We will not delve into the meaning of this code, as a discussion of *C* is beyond the scope of this book. There are, however, many excellent books on *C* available; the official *GNU C Programming Tutorial* is available at `http://savannah.gnu.org/projects/ctut-mb-rwhe/`.

Save this file as '`hello.c`'. Then, type the following command at your command prompt:

sh-2.05$ gcc hello.c ⟨<ENTER>⟩

The command `gcc` invokes gcc. `gcc` will create an executable program called, by default, '`a.out`'. Now, at your command prompt, type:

sh-2.05$ ./a.out ⟨<ENTER>⟩

You should see the text "Hello, Welcome to the GNU/Hurd Community!" appear on your screen.

Congratulations! You have just compiled a program.

The `gcc` program has many options that are useful to programmers. Whether you are a programmer or not, this basic knowledge of compilers will help you understand your GNU System.

To learn more about GCC, type `info gcc` at your command prompt.

11.3 Makefiles

In the previous section, we describled the use of `gcc`, the GNU System's compiler. You may have guessed that building complex software packages such as the Hurd using only `gcc` is much too difficult a task for the average end-user. Acutally, even developers would find such a task extremely complex.

To alleviate this difficulty, we introduce *makefiles*. Makefiles are used by the GNU System, as well as many other systems, as a way of automating the process of building programs.

Recall the "Hello, World!" program from the previous section. Most programs are much more complicated than this small example, and are composed of multiple files. If one file, call it '`foo.c`', depends on code in a file called '`bar.c`', then we must build '`bar.c`' before we can build '`foo.c`'. Furthermore, if we make a change to '`bar.c`', then we must re-compile both '`bar.c`' and '`foo.c`'. For a large program consisting of tens or hundreds of files, these dependencies can become difficult, if not impossible, to manage manually.

The `make` program automates the process of rebuilding files through a system of *targets* and *prerequisites*. A makefile consists of a set of targets, and each target specified what prerequisites must be fulfilled in order for that target to be satisfied. Prerequisites may be files, or they may non-file targets. In addition to specifying prerequisites, targets may specify commands that must be executed. For our example of the files '`foo.c`' and '`bar.c`', the makefile entries could be as follows:

```
foo: foo.c bar.o
gcc foo.c bar.o

bar.o: bar.c
gcc -c bar.c
```

If we type `make foo`, then `make` will compile '`bar.c`', and then compile '`foo.c`' afterwards. In addition, `make` will check the timestamps on the files, and will only rebuild those whose source files have changed.

With make, we can also specify *phony targets*, which do not build any files, but cause other targets to be built. For example, GNU makefiles should include targets such as *install*, which will cause a program to be copied into your system directories, and *clean*, which will cause files created during the build process to be deleted.

All the gritty details of `make` are available via `info make`; however, this introduction should be sufficient to give a casual user enough of an idea of how `make` works to use `make` confidently.

11.4 An Introduction to Texinfo

The Texinfo documentation language was invented by Richard M. Stallman. He devised the system from a MIT project called Bolio (combined with TeX to form BoTeX) and a CMU project called Scribe. Robert J. Chassell, Brian Fox, and Karl Berry have also helped developed Texinfo into the mature, robust, and well-documented format we have today.

The purpose of Texinfo is to be able to create one document that can be transfered to another document format. A user can make a document that can be viewed in a terminal's info viewer, a web browser, or as a Postscript, PDF, or DVI file. The latter formats can be printed to product hard-copies of GNU manuals.

The Texinfo syntax has many `@` symbols. These `@` symbols precede Texinfo commands that manipulate the text. The greatest thing about the official GNU documentation format is that it is well-documented. The Texinfo manual can be invoked with the command `info texinfo`. The Texinfo manual has sample documents, tips, suggestions, and tutorials (the Emacs text editor works great with Texinfo and is covered in the manual).

Some readers may have had previous experience with TeX or LaTeX, the latter of which is used for most mathematical typesetting. If you have used either of these typesetting languages, you will notice that Texinfo has a similar flavour, which is different from the languages in the SGML/HTML/XML family. Texinfo is quite easy to learn, however, and is an excellent tool for producing documentation.

11.5 Debugging with GDB

A *bug* is any problem with a program that results in the program crashing, or otherwise behaving incorrectly. If you follow the mailing lists of any software project, you will see people discuss bugs, and how to fix them.

Finding and fixing bugs in programs is primarily the responsibility of the programs' developers. Bugs, however, can be devious, and may only show up under certain conditions. Sometimes, these conditions do not occur on any developer's system, and the users of a program are the first to find a bug. An advantage of Free Software is that when users with programming skills find bugs, they can often fix the bugs themselves, and send the fixes to the project's core developers. Even when users cannot fix the bug in question themselves, they can often find the cause of the bug, enabling the project developers to fix the bug more quickly.

A *debugger* is a program that helps you debug another program. The GNU debugger is called GDB, and is invoked using the `gdb` command.

We will introduce you to GDB with a small example; for more complete documentation, consult the GDB manual (available, of course, by typing `info gdb` at the command prompt of any GNU system).

In this section, we assume that you are familiar with the *C* programming language. Consider the following small *C* program, which accepts a list of numbers on the command line, and outputs their sum:

```
/* sum.c
 *
 * Outputs the sum of the numbers entered on the command line.
 */

#include<stdio.h>

int
sum(int n, int index, char** vector)
{
  if (index == n)
    return 0;
  else
    return atoi(vector[index]) + sum(n,index++,vector);
}

int
main(int argc, char* argv[])
{
  printf("sum is %d\n", sum(argc,1,argv));
  return 0;
}
```

Observe that we have implemented the `sum()` function as a recursive function. We could have implemented `sum()` iteritavely, but this implementation is useful for our example.

Also note that we add the contents of the argument vector (`argv`) starting at position 1; this is because `argv[0]` contains the name of the program, and `argv[1]` through `argv[argc-1]` are the actual command-line arguments.

We can compile this program using `gcc` and run it:

```
$ gcc -o sum sum.c ⟨ENTER⟩
$ ./sum 1 2 3 4 5 ⟨ENTER⟩
$
```

Running this program on my system produces no output (hence the prompt on the next line). The program should, however, print "sum is 15".

We can use GDB to find the cause of this problem. To help GDB do its job, we will recompile our program with *debugging symbols*, which will provide GDB with extra information. We do this by passing the `-g` option to `gcc`:

```
$ gcc -g -o sum sum.c ⟨ENTER⟩
```

To debug this program using GDB, we type:

```
$ gdb sum ⟨ENTER⟩
GNU gdb 2002-12-19-cvs (cygwin-special)
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and
you are welcome to change it and/or distribute copies of it under
certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty"
for details.
This GDB was configured as "i686-pc-cygwin"...
(gdb)
```

We are given a *(gdb)* prompt for running `gdb` commands. GDB includes a builting command, `help`, that provides help on GDB commands. Typing `help` on its own causes GDB to print a list of commands, while typing `help command` produces help on the command `command`.

The first command we will use is the `list` command, which shows us the part of the code for the program we are debugging (this is some of the information we get by compiling with debugging symbols):

```
(gdb) list ⟨ENTER⟩
11          if (index == n)
12            return 0;
13          else
14            return atoi(vector[index]) + sum(n,index++,vector);
15        }
16
17        int
18        main(int argc, char* argv[])
19        {
20          printf("sum is %d\n", sum(argc,1,argv));
(gdb)
```

Notice that each line of the program source starts with a number. These numbers are important, since they are one of the means by which we tell GDB to examine different parts of the program. For example, we can type `list 1` to have GDB list the program source around line 1. Line numbers, however, are not the only means by which we can refer to parts of our program; we can also type `list sum` to have GDB list the program source around the beginning of the `sum()` function.

Now, on to the actual debugging. We can examine the `main()` function, and observe that its contents are fairly trivial; we thus guess that the error must be in our `sum()` function. We can set a *breakpoint* at the beginning of the sum function like so:

```
(gdb) break sum ⟨ENTER⟩
Breakpoint 1 at 0x401097: file sum.c, line 11.
(gdb)
```

We can then insturct GDB to begin running our program with the inputs 1, 2, 3, 4, and 5:

```
(gdb) run 1 2 3 4 5 ⟨ENTER⟩
```

GDB stops when it hits the breakpoint we set:

```
Breakpoint 1, sum (n=6, index=1, vector=0xa041eb0) at sum.c:11
11        if (index == n)
(gdb)
```

This seems all right. We tell GDB to continue execution:

```
(gdb) continue ⟨ENTER⟩
```

GDB stops the next time we hit our breakpoint:

```
Breakpoint 1, sum (n=6, index=1, vector=0xa041eb0) at sum.c:11
11        if (index == n)
(gdb)
```

Now *that* is odd. The value of `index` is unchanged, but it should have been incremented. As a sanity check, we can tell GDB explicitly to print out the value of `index`:

```
(gdb) print index ⟨ENTER⟩
$1 = 1
(gdb)
```

As expected, `index` is still 1. We tell GDB to go to the next line of code:

```
(gdb) next ⟨ENTER⟩
14          return atoi(vector[index]) + sum(n,index++,vector);
(gdb)
```

That is normal. We continue to the next line of code:

```
(gdb) next ⟨ENTER⟩

Breakpoint 1, sum (n=6, index=0, vector=0xa041eb0) at sum.c:11
11        if (index == n)
```

We are back at our breakpoint, and, again, index is unchanged.

GDB has been very helpful, and we now must use another debugging tool - our minds - to complete the job. We know that `index` is not being incremented, and that the only line of code in our problem area that should modify `index` is:

```
14          return atoi(vector[index]) + sum(n,index++,vector);
```

We then see our little mistake. In our function call `sum(n,index++,vector)`, our use of `index++` causes index to be incremented *after* the call to `sum()`. Hence, we use the same value of `index` at each invocation of `sum()`.

In theory, this creates infinite recursion; however, since each call to `sum()` requires the use of a stack frame, we quickly run out of space on the stack, causing the program to crash. We can see the exact error message by clearing the breakpoint in our program, and telling GDB to continue execution:

```
(gdb) clear sum <ENTER>
Deleted breakpoint 1
(gdb) continue <ENTER>
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x610ac933 in _libkernel32_a_iname ()
(gdb)
```

To finish our debugging session, we use the `finish` command to stop the program being debugged, and the `quit` command to quit GDB:

```
(gdb) finish <ENTER>
Run till exit from #0  0x610ac933 in _libkernel32_a_iname ()
Warning:
Cannot insert breakpoint 0.
Error accessing memory address 0x0: I/O error.

(gdb) quit <ENTER>
```

We can fix our program by changing `index++` to `++index`, so that the value of `index` is incremented *before* the call to `sum()`. We then recompile the program, and run it again:

```
$ ./sum 1 2 3 4 5
sum is 15
$
```

The program now works correctly.

In our example, we have only scratched the surface of what GDB can do. We can also *attach* gdb to a process that is already running, using GDB's `attach` command; this is often necessary when debugging the Hurd servers. To learn all the details of using GDB, consult the GDB manual - again, this is accessible via the command `info gdb`.

11.6 Reporting Bugs

when to use bug-hurd versus Debian BTS, how to file a *good* bug report

12 Copying the Hurd and This Manual

The Hurd is covered under the GNU General Public License, as is this manual.

12.1 GNU General Public License

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you

indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

12. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
13. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea of what it does.
Copyright (C) 19yy  name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc.,
59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Index

M

N

O

P

R

S

T

U

W

X

Short Contents

Table of Contents