

Getting Started with GNU/Linux

Jon Beck

Don Bindner

Caleb Jordan

Getting Started with GNU/Linux

by Jon Beck, Don Bindner, and Caleb Jordan

Copyright © 2003, 2005 Jon Beck, Donald J. Bindner, Caleb Jordan

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. A copy of the license is included in the section entitled "GNU General Public License."

Table of Contents

1. Introduction.....	1
The Linux Kernel	1
The Unix Filesystem	1
The Shell	2
2. Getting Started.....	5
Logging in	5
Logging out	5
Starting the X Window System.....	5
3. Working with Files and Directories.....	7
Getting information about files	7
Displaying information in a file	8
Creating and using directories.....	9
File security	9
Users and groups	9
File types.....	9
Permissions for regular files	10
Permissions for directories	11
Default permissions	11
4. Managing Your Working Environment	13
Environment variables.....	13
Common environment variables.....	13
Changing .bashrc.....	14
5. Editing Files with Vi	15
Modal editing	15
Starting Vi	15
Stopping Vi.....	15
Common tasks.....	16
Gaining proficiency	16
Endorsement.....	16
6. Editing Files with Emacs.....	17
Important Emacs commands	17
7. Working Within the Truman Network	19
Retrieving and saving files with Samba	19
Accessing Windows shares with smbclient	19
Accessing Windows shares using smbmount	20
Connecting to other computers using SSH	21
Transferring files with SSH tools	22
Transferring files with sftp	22
Transferring files with scp	23
Using the Truman VPN from Linux.....	23
Basic VPN setup.....	23
Advanced routing with VPN	24

8. Finding Things on a Linux system	27
Directory structure on a Linux system	27
Programs for finding files	27
Finding files with grep	27
Finding files with locate	28
Finding files with whereis and which	28
Finding files with find	29
Programs for finding people	29
Listing who is currently logged on	30
Finding information about an individual	31
9. Managing Jobs and Processes	33
Processes	33
The ps command	33
Job control	34
Foreground and background	34
Working with jobs	34
Putting a foreground job into the background	36
10. The X11 Graphical Environment	37
The X Window System as a network application	37
Simple X11 display forwarding	37
X display forwarding with SSH	37
Customizing the keyboard map	38
Mapping single keys with xmodmap	38
Mapping the entire keyboard with xmodmap	39
Other customizations and information about using xmodmap	39
11. Documentation and Help	41
Man pages	41
The Texinfo system	42
Other sources of help	43
Getting help from programs themselves	43
Miscellaneous documentation	44
The power of Google	45
12. Getting GNU/Linux for Yourself	47
Distributions	47
Installation	47
Loading a kernel	47
Partitioning a hard disk	47
Installing packages and setting up the system	48
Making the computer boot	48
Keeping current	48
13. Summary of useful commands	49
A. Advanced: Customizing file associations with <code>~/.mailcap</code>	51
Motivation	51
Helper scripts	51
Editing <code>~/.mailcap</code>	52
Lesson 1 - PDF files	52

Lesson 2 - HTML files.....	52
Finishing up	53
Other resources.....	53
B. Advanced: Password-less Authentication With SSH	55
Generating SSH keys	55
Creating an authorized_keys2 file.....	55
Using ssh-agent	56
Killing ssh-agent	56
C. Advanced: Off-site Email Using SSH.....	57
Forwarding ports	57
Configuring your email client	57
Disconnecting the tunnel.....	58
D. GNU General Public License	59
Preamble.....	59
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION	59
Section 0	60
Section 1	60
Section 2	60
Section 3	61
Section 4	62
Section 5	62
Section 6	62
Section 7	62
Section 8	63
Section 9	63
Section 10	63
NO WARRANTY Section 11	63
Section 12	64
How to Apply These Terms to Your New Programs	64
Index.....	67

List of Tables

3-1. File type designations	10
6-1. Important Emacs Commands:	17
7-1. smbclient commands	19
7-2. sftp commands.....	22
13-1. Commands:.....	49

Chapter 1. Introduction

“Operating system” (OS) is the name given to the set of programs that controls a computer’s hardware and software, and that interacts with the user. This Guide is about using a computer running the GNU/Linux OS. It assumes a fairly new PC and an up-to-date OS installation. Typical GNU/Linux distributions (distros) are Debian Woody and RedHat 7.x, 8, or 9.

In this guide, “unix” is used as a generic term to apply to all UNIX-type operating systems. All examples are based on GNU/Linux systems. While most of these commands will work on other unix systems such as Sun Solaris, BSD, HP-UX, Mac OS/X, and IBM’s AIX, there are often differences between them and GNU/Linux in the syntax of switches and the appearance of the output.

The unix operating system has three main components:

- The Linux kernel
- The filesystem
- The shell

The Linux Kernel

The kernel is the core part of the operating system. It is the part that loads first, and it stays resident in memory as long as the computer is powered on and running.

The key functions of the kernel include:

- Process Management - creating, suspending, and terminating processes, and maintaining their states.
- Interprocess Communication (IPC) - managing the facilities for processes to communicate with each other. The two main types are:
 - Pipes: temporary channels to allow processes to communicate on the same computer.
 - Sockets: Network (including Internet) channels to allow processes to communicate across the network.
- File Management - the kernel manages the filesystem and the hardware such as disks that the filesystem uses.
- Memory Management - the kernel allocates RAM for itself and all the other processes.

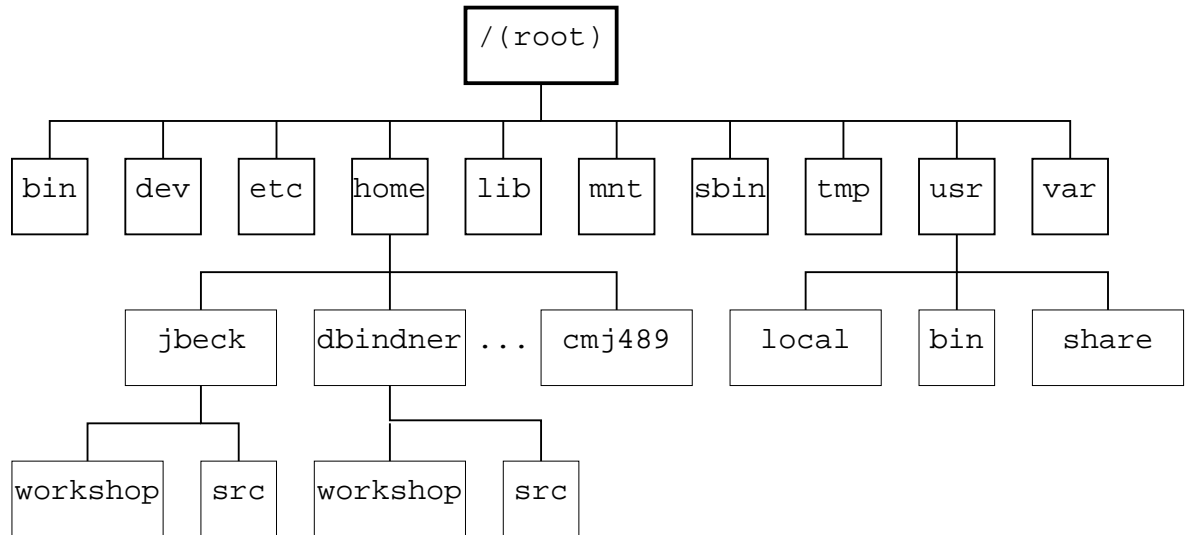
On a typical GNU/Linux system, the kernel is stored in a file named `/boot/vmlinuz` or `/boot/vmlinuz`. This file is loaded into memory when the computer boots.

The Unix Filesystem

A filesystem is the way that the operating system organizes stored information into files and keeps track of those files. The file is the basic unit of disk storage on a unix system. A file can hold any type of

information including text, a program, or digitally encoded music.

On a unix system, all files are organized into directories. A directory is just a special file that stores information about other files. You may find it intuitive to think of a directory as a kind of container that holds files or other directories. Some other operating systems call a directory a folder. The unix filesystem has a hierarchical structure.



At the root of the filesystem is the root directory, written as a slash (/). In this directory are some files (not shown above) and a number of subdirectories such as bin, dev, etc. Each of these subdirectories of / are themselves directories which contain some files and some further subdirectories.

Every file (including directories, which are also files) can be specified in at least two ways. First, there is an absolute path name, which starts with the root and specifies every directory name along the way to the file. An example of this is `/home/jbeck/workshop/index.html`. All absolute pathnames begin with a leading slash.

Alternatively, you can use a name for a file that is relative to your current working directory. If your current working directory is `/home/jbeck` and you wish to refer to a file named `index.html` which is in the subdirectory `workshop`, you can refer to it as `workshop/index.html`. Finally, if your current working directory is `/home/jbeck/src`, you can refer to the file as `../workshop/index.html`. The `..` means the parent directory of where you are, one level up in the hierarchy. Notice that relative pathnames do not begin with a leading slash.

If most of your computer experience comes from the MSDOS/Windows world, you may notice the lack of drive letters (like `C:` or `A:`). In unix operating systems, there are no drive letters. Instead, different drives have different directory names. So `/cdrom` might refer to the CD-ROM drive and `/floppy` might refer to what you might normally consider `A:`. The **mount** command can show you what “drives” you can currently access.

The Shell

When you log into a unix system in text mode, or open a terminal window in GUI mode, you are running a program called a shell. The shell is a command interpreter. You type commands, and the shell carries out those commands (by running other programs, etc.). You primarily interact with the unix OS by entering commands at the shell's command prompt. If you have ever used MS-DOS, or a DOS Window in Microsoft Windows, then you have experienced a shell.

While there are many shells available, the standard shell for a GNU/Linux system is **bash**. The normal command prompt for bash is a dollar sign, often preceded by various information such as the current working directory and your username.

When you enter a command at the shell prompt, the shell interprets the command, passes it to the kernel, then displays the results of the command back to you.

Most commands are a combination of the command name, switches, and arguments. For example, in the command `ls -l *.html`, `ls` is the command name (the command for listing files). Minus ell, `-l`, is a switch meaning give a detailed (long) listing instead of the default brief listing. Finally, `*.html` is the argument, in this case containing a wildcard (the star matches any string of letters).

The net effect of this command is to request a long-format directory listing of all files whose names end with `.html`. The result might look like:

Example 1-1. Typing a command in the shell

```
$ ls -l *.html
-rw-r--r--  1 dbindner dbindner  5997 Mar 23 15:01 commands.html
-rw-r--r--  1 dbindner dbindner  1030 Mar 23 15:01 edit.html
-rw-r--r--  1 dbindner dbindner  1175 Mar 23 15:01 environment.html
-rw-r--r--  1 dbindner dbindner   780 Mar 23 15:01 files.html
-rw-r--r--  1 dbindner dbindner  2996 Mar 23 15:11 fs.html
-rw-r--r--  1 dbindner dbindner  1758 Mar 21 23:58 index.html
-rw-r--r--  1 dbindner dbindner   428 Mar 23 15:01 indx.html
-rw-r--r--  1 dbindner dbindner   627 Mar 23 15:01 install.html
-rw-r--r--  1 dbindner dbindner  1538 Mar 23 15:01 intro.html
-rw-r--r--  1 dbindner dbindner   550 Mar 23 15:01 jobs.html
-rw-r--r--  1 dbindner dbindner   687 Mar 22 21:33 kernel.html
-rw-r--r--  1 dbindner dbindner   473 Mar 23 15:01 links.html
-rw-r--r--  1 dbindner dbindner   511 Mar 23 15:01 network.html
-rw-r--r--  1 dbindner dbindner  1726 Mar 23 15:14 shell.html
-rw-r--r--  1 dbindner dbindner  1339 Mar 23 15:01 start.html
-rw-r--r--  1 dbindner dbindner   555 Mar 22 16:31 template.html
```


Chapter 2. Getting Started

Logging in

Unix is a multi-user operating system. Thus, before you begin, you must login to authenticate yourself as an authorized user. On a Truman computer, your username is your TRUMAN domain network ID, and your password is your TRUMAN domain network password. On your own system, you may choose your username and password to be whatever you wish.

You may login using a completely text environment. This is common when connecting to a remote machine using a program like telnet or ssh. As soon as you authenticate, you are presented with a shell and can type commands.

You may also login to a graphical environment. This is common when sitting at a physical workstation. A graphical login prompt is presented which leads to a full graphical environment after you have authenticated. In this case, it is necessary to open a “shell window,” to type commands, and you’ll generally find options to do that under a menu.. Sometimes a shell window is referred to as an xterm or rxvt window after the names of popular programs that provide a graphical means of running a shell.

Logging out

To logout of a text environment, you must be at a command prompt. Type the command **exit** and press the **Enter** key, or press **^D (Ctrl-D)**.

To logout of a graphical environment you will need to find a logout or exit option on one of the menus. This will usually be found under a button on the lower left part of the screen (in Microsoft “Start” button fashion), or by clicking one of the mouse buttons while the pointer is on the desktop (a place where no windows are opened). Some graphical environments will also prompt you to logout when you press **Ctrl-Alt-Del**. A last-resort method of terminating a graphical environment is to press **Ctrl-Alt-Backspace**, which is sometimes referred to as “zapping” the server. It will generally close everything down, but programs may not get a chance to shut down nicely.

Starting the X Window System

If the workstation you login to presents a text environment, you can manually invoke the X Windows graphical environment with the command **startx**. In this case, logging out will involve a two step process. First you will exit the graphical environment, and then you will logout from the text environment.

Chapter 3. Working with Files and Directories

Since “everything” on a unix system is a file, there are huge numbers of commands that affect files. Here we talk about some of the most important of these commands.

When you log in and start a command shell, your current working directory is your home directory. This special directory contains files that are used to customize your environment when you log in (see the Section called *Changing .bashrc* in Chapter 4 for more information on these startup files). It also contains any directories that you create and your own working files.

Getting information about files

The most fundamental file command is **ls** which is a mnemonic for “list.” This command displays the contents of a directory, i.e., the names of all the files and directories in it. Using the command without arguments or switches gives a brief multi-column listing of all the files in the current working directory:

Example 3-1. Basic ls command

```
$ ls
commands.html  environment.html  fs.png           jobs.html        start.html
CVS            #files.html#     index.html       kernel.html      style.css
edit           files.html       indx.html        links.html       template.html
edit.html      fs.dia           install.html     network.html     vi.html
emacs.html     fs.html          intro.html       shell.html
```

Here we see a listing of all the files in this directory, in case-insensitive alphabetical order, with regular files and subdirectories (CVS and edit) intermingled.

The most common switches used with the **ls** command are **-l** and **-a**. The **-l** switch (for long), and the **-a** switch (for all), expand the information shown. As with many commands, the switches may be combined:

Example 3-2. A long file listing

```
$ ls -al
total 132
drwxrwxr-x  4 jbeck  jbeck      4096 Mar 25 12:55 .
drwxrwxr-x  4 jbeck  jbeck      4096 Mar 19 16:34 ..
-rw-rw-r--  1 jbeck  jbeck      5691 Mar 25 05:18 commands.html
drwxrwxr-x  2 jbeck  jbeck      4096 Mar 25 12:43 CVS
drwxrwxr-x  2 jbeck  jbeck      4096 Mar 23 17:36 edit
-rw-rw-r--  1 jbeck  jbeck       972 Mar 23 17:39 edit.html
-rw-rw-r--  1 jbeck  jbeck     1324 Mar 24 21:13 emacs.html
-rw-rw-r--  1 jbeck  jbeck     1119 Mar 23 17:39 environment.html
-rw-rw-r--  1 jbeck  jbeck     1819 Mar 25 12:54 #files.html#
-rw-rw-r--  1 jbeck  jbeck       724 Mar 23 17:39 files.html
-rw-rw-r--  1 jbeck  jbeck     2627 Mar 23 08:06 fs.dia
-rw-rw-r--  1 jbeck  jbeck     3042 Mar 24 17:47 fs.html
-rw-rw-r--  1 jbeck  jbeck     6563 Mar 23 08:06 fs.png
-rw-rw-r--  1 jbeck  jbeck        0 Mar 25 12:55 .htaccess
```

```
-rw-rw-r-- 1 jbeck jbeck 1700 Mar 23 17:39 index.html
-rw-rw-r-- 1 jbeck jbeck 372 Mar 23 17:39 indx.html
-rw-rw-r-- 1 jbeck jbeck 5722 Mar 25 05:18 install.html
-rw-rw-r-- 1 jbeck jbeck 1519 Mar 24 17:48 intro.html
-rw-rw-r-- 1 jbeck jbeck 8993 Mar 25 05:18 jobs.html
-rw-rw-r-- 1 jbeck jbeck 1653 Mar 24 17:44 kernel.html
-rw-rw-r-- 1 jbeck jbeck 1220 Mar 25 05:18 links.html
-rw-rw-r-- 1 jbeck jbeck 5797 Mar 25 12:43 network.html
-rw-rw-r-- 1 jbeck jbeck 2891 Mar 24 17:36 shell.html
-rw-rw-r-- 1 jbeck jbeck 1281 Mar 23 17:39 start.html
-rw-rw-r-- 1 jbeck jbeck 206 Mar 23 15:55 style.css
-rw-rw-r-- 1 jbeck jbeck 499 Mar 23 17:39 template.html
-rw-rw-r-- 1 jbeck jbeck 6334 Mar 25 05:18 vi.html
```

The `-l` switch is what causes the output to be listed in the columns. In order, the columns list the file type (first character), permissions (9 characters), number of hard links, owner name, group name, size in bytes, modification time, and file name. The `-a` switch means show all files, even the “hidden” ones whose names begin with the period character.

The output of `ls` may be sorted in many ways. For example, `ls -ltr` specifies a long-format listing, sorted by timestamp, reversed (i.e. oldest to most recent). It is also very common to specify arguments to `ls` using wildcards. The asterisk is used to specify 0 or more characters, and the question mark is used to specify exactly one character. For example:

Example 3-3. A file listing with wildcards

```
$ ls -l f?.*
-rw-rw-r-- 1 jbeck jbeck 2627 Mar 23 08:06 fs.dia
-rw-rw-r-- 1 jbeck jbeck 3042 Mar 24 17:47 fs.html
-rw-rw-r-- 1 jbeck jbeck 6563 Mar 23 08:06 fs.png
```

While the `ls` command can tell you a lot about files, it does not “look inside them,” so it can’t really tell you what a file is for. Luckily, the `file` command can often help you figure out what kind of information is contained in a file:

Example 3-4. Using the file command

```
$ file fs.png edit
fs.png: PNG image data, 625 x 280, 8-bit/color RGB, non-interlaced
edit: directory
```

This command was used with no switches and two arguments. The output says that the file `fs.png` is Portable Network Graphics image data, while `edit` is a subdirectory.

Displaying information in a file

There are two main commands for displaying information in a text file, `cat` and `less`. The first, `cat`, is short for “concatenate.” Its primary function is for combining multiple files together, but when only one file is specified it writes the entire contents of the file to the screen. By contrast, `less` writes the file to the

screen one page at a time interactively. When in **less**, press **h** to get help with its commands. Press **q** to quit out of less.

Creating and using directories

The **mkdir** command creates new directories. For example, **mkdir coursework** creates a directory named `coursework` in the current working directory. Once this directory is created, the command **cd coursework** changes the working directory one level down, to the newly created directory. To navigate back up one level, to the parent directory of the current working directory, use **cd ..** which uses the `..` indicator for the parent of the current working directory. To remove an empty directory, employ the **rmdir** command. The **pwd** command prints the current working directory name, to let you know where you are.

File security

Because of its multiuser nature, it is expected that many different people will store their files on the same computer. It is understood that some files may be shared while others are private, and the operating system has a mechanism to prevent others from reading your private files.

Users and groups

The standard security paradigm on unix filesystems is based on two ideas. The first idea is that there is a single administrative user, called root, that is all-powerful. The root user can arbitrarily create and destroy files (even files owned by others). Root can read every file on a unix system.

The second idea of unix file security is group membership. Every user account is a member of one or more groups, which can be displayed by typing the **groups** command. Each file on a unix system has an associated owner and group, and these are used to determine how different users can access the file.

Example 3-5. Displaying an account's group memberships

```
$ groups
faculty
```

File types

The specific mechanism that provides file security (in the context of group membership) is called “file permissions.” Sometimes this is also referred to as the “mode” of the file. The **ls** command will report the current permissions on a file when passed the `-l` (minus ell) option. The first column of the output contains a set of letters that detail the type of file along with three classes of permissions: the permissions for the owner of the file, for the members of the group that the file belongs to, and for anyone.

Example 3-6. Long listing with permissions

```
$ ls -l
total 8
drwxr-xr-x    2 dbindner faculty    4096 Jul  8 20:38 stuff
-rw-r-----    1 dbindner faculty    186 Jul  8 20:38 test.pl
```

The example `ls -l` command above shows one directory named `stuff` and one regular file named `test.pl`. The owner of both is the `dbindner` account, and both belong to the group `faculty`.

Notice that the first letter on each output line shows the type of file, ‘d’ for directories and ‘-’ for regular files. Other less common designations include:

Table 3-1. File type designations

l	a symbolic link (something like the shortcuts in Windows)
b	block device (like a hard drive partition)
c	a character device (like a sound card)
p	a pipe
s	a socket

It is not necessary to understand all of the file types immediately. A beginner can expect to encounter mostly regular files, directories, and (sometimes) symbolic links.

Permissions for regular files

Immediately following the file type designator are listed the permissions for owner, group, and anyone. Each set of permissions is marked by three consecutive letters: r for read, w for write, and x for execute. The permission is granted if the letter is listed, and it is denied if the letter is missing. For `test.pl` the owner permissions are ‘rw-’ which indicate that the owner, `dbindner`, can read and write (or delete) the file. The group permissions are ‘r--’ which designate that any member of the group `faculty` can read (but not change or delete) the file. The permissions for anyone else would be ‘---’. No one else can read or modify the file in any way.

Notice that no one can execute the file as a program. This mechanism differs from the way an operating system like Windows would behave. In Windows, whether a file can be executed or not depends on its filename extension. Under unix, the form of the filename is irrelevant. It is the permissions that differentiate executable files (programs) from other files.

The permissions of a file may be changed by the file’s owner (or the system administrator) using the **chmod** (change mode) command. The simplest syntax accepts a number representing the new permissions/mode, and a list of files to apply the mode to. The mode is most commonly a three digit number. The first digit specifies the owner permissions, the second digit specifies the group permissions, and the third digit specifies the anyone permissions (this is the same order as in the listing).

Each individual permission has a value associated with it. Execute permission, x, is given the value 1. Write permission, w, is 2. Read permission, r, is 4. Notice how the values double as they read from right to left in the long file listing. Using these values, the current mode of the `test.pl` file could be correctly

described as 640 (6=4+2 because the owner has rw-, 4 because the group has r--, and 0 because anyone has ---). To mark this file as an executable program available to everyone, the owner could type:

Example 3-7. Using `chmod` to change the mode of a file

```
$ chmod 755 test.pl
$ ls -l test.pl
-rwxr-xr-x    1 dbindner faculty      186 Jul  8 20:38 test.pl
```

Permissions for directories

The read, write, and execute permissions take on slightly different meanings for directories. The most intuitive is write permission, which allows you to create new files and delete existing files. The only nonintuitive part of write permission on directories is that it allows you to delete files you do not own (or have write permission for) since you are effectively removing them from the directory and not actually modifying the files themselves.

The read and execute permissions on a directory limit how you can list and use a directory, and are generally granted together. You need execute permission to ‘cd’ to a directory, to use files in it, and (ironically) to produce a proper listing of its contents. You need read permission to get any listing at all of the directory’s contents. Generally, if you wish someone to be able to use a directory, you give them read and execute permission. If you wish to restrict access, you remove both.

Example 3-8. Using `chmod` to change the mode of a directory

```
$ chmod 700 stuff
$ ls -ld stuff
drwx-----    2 dbindner faculty    4096 Jul  8 20:38 stuff
```

Default permissions

The permissions that a newly created file has are a combination of default settings that can be customized by the user and settings of the program that creates the file. Mail programs, for example, universally create files with mode 600 (read and write for the owner only) because there is an assumption that email is private. A user’s default preference is called their umask. All programs take into account the current umask setting when creating a new file, although both the user and programs can modify the umask.

The umask setting, like a mode, is a number with one digit each for the owner, group, and anyone masks. Sometimes it will contain four digits, but you may ignore the first digit which is usually a leading 0. Only the last three digits are important to the discussion here. Although one might expect the umask to contain the default permissions allowed, the umask value actually indicates the permissions that are *disallowed* by default. Common umask values are 022 (writing for group and anyone is disallowed), 002 (writing for anyone is disallowed), and 077 (all permissions are disallowed for both group and anyone). The current umask value can be displayed and modified via the **umask** command.

Example 3-9. Using umask to set the file creation mask

```
$ umask
0022
$ touch file1.txt
$ ls -l file1.txt
-rw-r--r--    1 dbindner faculty          0 Jul  9 11:49 file1.txt
$ umask 0077
$ touch file2.txt
$ ls -l file2.txt
-rw-----    1 dbindner faculty          0 Jul  9 11:50 file2.txt
```

Here the **touch** command, which is commonly used to update the timestamp of a file, is used to create two new (empty) files, the second of which is private to the owner.

The current umask setting is remembered as long as your session is active. If you exit the shell or logout, it is forgotten. To customize the umask setting for every login, place a umask command in your startup scripts. If you use the bash shell, this would be the `~/ .bashrc` file.

Chapter 4. Managing Your Working Environment

Environment variables

The unix shells are programming environments, and because of this they have ways to define variables (names that can store data). Simple assignments can be made using the equals sign (=), and variables can be printed by preceding the name with a dollar sign (\$). Variable assignments can be erased with unset. For example:

Example 4-1. Shell variable assignment

```
$ stuff=1
$ echo $stuff
1
$ unset stuff
$ echo $stuff

$
```

In addition to the variables that the shell supports for programming, there are variables that the user can use to customize the working environment. These are called environment variables. In bash, environment variables begin life as regular variables and are “exported” into the environment with the export command. Unlike regular variables, exported variables are visible to every program you run from the shell, so anything you customize has “persistence” as you run programs.

Common environment variables

Any program can make use of environment variables (check its man page or documentation to determine what variables it supports). However, there are some environment variables that are understood by many programs. They include:

EDITOR

specifies the editor that will be used by other programs, such as a mail program.

PAGER

specifies the pager that will be used by programs, such as man, to display long files “one page at a time.” Common pagers are **more** and **less**.

PATH

specifies the directories that the shell will look in to find commands. The directories listed in the PATH are searched in the order in which they appear.

PRINTER

specifies the default printer to which the **lpr** command (the default printing command) will send output.

TERM

specifies the terminal (display) you are using. Common values are `xterm`, `vt100`, `rxvt`, and `linux`. Only occasionally will users find it necessary to set this by hand.

In bash, you can use the **echo** command to display the current settings of these variables, and the **export** command to set new values.

Example 4-2. Exporting variables in the shell

```
$ echo $EDITOR
nano
$ export EDITOR=vim
$ echo $PAGER

$ PAGER=less
$ export PAGER
```

These settings would change the default editor from **nano** to **vim**, and define **less** as a default pager. Notice that the **export** command can be separate or combined with a variable assignment on one line.

Changing `.bashrc`

Changes to environment variables survive as long as the shell runs. When you exit the shell (or logout) these settings are lost. If you want to customize the environment and have those changes take effect every time you login, you need to record those settings in your startup scripts. If you are using bash, that means you want to edit the `.bashrc` file in your home directory. Simply add the relevant variable assignments (and the export commands) to this file. Since `.bashrc` is read every time you login to the computer, your customizations will become permanent.

Chapter 5. Editing Files with Vi

Modal editing

The Vi editor is the quintessential unix text editor. It is relatively small in terms of program size, and is generally the one editor guaranteed present on any unix system. Anyone wishing to use a unix system seriously should obtain at least minimal skills with Vi.

The most difficult aspect for beginners to understand about Vi is that it is a modal editor. What that means is that there is a distinct mode the editor must be in for typing text, and a different mode for editing, pasting, and cursor movement. The typing mode is called insert mode, and the other mode is called command mode.

Other editors have modes, but not in the obvious way that Vi does. If **Alt-F** brings down a **File** menu in your favorite editor, you have implicitly switched from insert mode to command mode by pressing the **Alt** key. The same is true when you click a menu selection with the mouse. Historically, **Alt** keys and **Ctrl** keys have not worked well in all unix environments, or have worked differently from computer to computer. Because of this, Vi does not make use of them the same way that other editors do.

With Vi, changing modes requires an explicit act, and somewhat unintuitively the editor begins in command mode. In command mode, the letters of the alphabet (that you would normally use to type data into a document) perform editing functions instead. Some letters move the cursor around (you can quickly move by characters, words, sentences, lines and paragraphs). Other letters delete text (you can easily delete characters, words, sentences, and so on). Since every computer has alphabet keys that work the same way, the Vi editor is guaranteed to work well everywhere.

Starting Vi

Since Vi begins in command mode, generally the first thing a person learns is how to enter insert mode. Press **i** (lower case I), and begin typing. For example: **iThis is my first line of text**. You can type as much as you like, pressing **Enter** to break lines, and **Backspace** to correct mistakes. During insert mode, the letter keys work exactly as in any other editor. If you typed the text above, your editor should show:

```
This is my first line of text.
```

To return to command mode, press **Esc**. All of the keys return to their “editing” functions. For example, you can check that the letter **b** moves the cursor back one word at a time. Similarly the letter **w** moves the cursor forward by words. The arrow keys should work (which is not always true during insert mode for many Vi editors).

General practice is to enter command mode for cursor movement or editing and then return to insert mode. For example, we could use **b** and **w** to move the cursor to the beginning of the word “text” and type **iplain Esc** to insert the word “plain” before returning to command mode.

Stopping Vi

Quitting the editor is accomplished via command mode (press **Esc** if you aren't sure what mode you're in). From command mode, you can always quit by typing the "colon" command **:q!** (and pressing **Enter**). Many of the commands that operate on files begin with colon this way. For example **:w file.txt** writes everything to a file named `file.txt`, and **:e file.txt** loads `file.txt` and "edits" it.

Common tasks

Here are some of the more common things you might wish to do with an editor, with the commands to accomplish them in Vi. Naturally, all of these assume command mode.

- Search for a word using slash: **/word** or backward using question: **?word**
- Search again for the same word: **n**
- Search again, but reverse direction: **N**
- Go to particular line. Type the line number and hit capital G: **20G**
- Go to the end of the file. Use G without the line number: **G**
- Delete some lines of text. Type the number of lines to delete followed by **dd**: **3dd**
- Paste some previously deleted text: capital P to paste in front of the cursor or little p to paste after.
- Search for word1 and replace with word2 (note: this is a colon command): **:%s/word1/word2/g**
- Undo an edit: **u**

Gaining proficiency

Becoming really comfortable and proficient with Vi requires an investment. The more editing commands that you have memorized, the more you will enjoy Vi and the faster you will be at accomplishing edits, particularly complex ones. Watching a skilled Vi user is like watching a magician do sleight of hand. Text disappears and reappears as if by magic. Like magic, skill with Vi comes from both practice and work.

To get started learning commands, a popular and extensive Vi reference card is available online in both pdf (<http://limestone.truman.edu/~dbindner/mirror/vi-ref.pdf>) and postscript (<http://limestone.truman.edu/~dbindner/mirror/vi-ref.ps>) formats.

Endorsement

It is the author's personal opinion that the Vim editor is the superior Vi editor. There are text and graphical versions for various platforms (including Windows) available from the website: www.vim.org (<http://www.vim.org/>). If you are using Linux, Vim is undoubtedly packaged for your system, and you may already have it installed.

Chapter 6. Editing Files with Emacs

GNU/Emacs is a large, complex text editor. It has a very large number of custom modes for editing specific types of files. For example, when you edit a file of Ada source code, emacs goes into Ada mode, with proper indenting rules, color highlighting, and hotkeys for Ada. Emacs uses **Ctrl** and **Alt** key combinations for commands. In emacs, the **Alt** key is referred to as the **Meta** key. (On some systems, the **Alt** key is not mapped to **Meta**, and you instead use the **Esc** key for **Meta**.) Most key combinations are at least somewhat mnemonic. For example, the end-of-line function is invoked by **C-e** (which means **Ctrl-e**, holding down the **Ctrl** key while pressing **e**). Emacs is used so widely that some of the better-known key combinations are used in other applications. For example, **C-e** is also end-of-line in the Netscape html editor, adopted from emacs. Often, if a control-key command invokes some action, then the alt-key combination of the same key moves similarly but more so. For example **M-e** (i.e., **Meta-e**, meaning **Alt-e**) is end-of-sentence.

Important Emacs commands

Table 6-1. Important Emacs Commands:

C-x C-c	Exit emacs (prompts for save)
C-g	Cancel the current command which is in progress
C-x C-w	Writes the current buffer to a new file (save as)
C-h	Online help
C-x l	Hides all emacs windows except the current one
C-x i	Insert text from a file at the current location
M-<	Move cursor to the beginning of buffer
M->	Move cursor to the end of buffer
M-{	Move cursor to the beginning of paragraph
M-}	Move cursor to the end of paragraph
C-a (M-a)	Move cursor to beginning of line (sentence)
C-e (M-e)	Move cursor to end of line (sentence)
C-k (M-k)	Kill (delete) to end of line (sentence)
C-Space	Set the mark at the current cursor location
C-w	Kill the marked (highlighted) text and copy it to the kill ring buffer
M-w	Copy the marked text to the kill ring buffer without deleting it
C-x <	Start recording a keyboard macro
C-x)	Finish recording a keyboard macro
C-x e	Execute the most recently recorded keyboard macro
C-s	Begin interactive search
M-%	Interactive search and replace

Chapter 7. Working Within the Truman Network

Retrieving and saving files with Samba

Most of the files that students use on the Truman network are stored on Windows shares, sometimes called SMB shares or Samba shares. SMB is the name of the protocol used to transfer files to and from these shares, and Samba is the name of the program for Linux that speaks the SMB protocol.

Using Samba, Linux machines can create shares that can be read by the Windows machines on the network. More important for most users, however, is that Samba is the program that allows access to the Truman U: and Y: drives.

To use the Samba shares, it is necessary to know their “real” name. The drive letter assignments we generally refer to are just arbitrary names, but the real name specifies where on the network the data is stored. Under Windows, you can learn these names in My Computer. For example, the U: drive is described as Student_files on 'hydrogen'. This tells you the real name of the U: drive in Windows terms is \\hydrogen\Student_files. In a unix environment, we would normally reverse the slashes. The available shares currently at Truman are:

- The S: drive: //Ss1/Win_apps
- The T: drive: //xenon/user
- The U: drive: //hydrogen/Student_files
- The Y: drive: //hydrogen/user

Accessing Windows shares with smbclient

The easiest way to access one of these shares is using the **smbclient** program. It takes the name of a share, followed by various switches. The most important switches are `-U` to specify the username for connecting to the share and `-w` to specify the workgroup (domain).

Example 7-1. Connecting to the U: drive with smbclient

```
$ smbclient //hydrogen/Student_Files -U dbindner -w truman
added interface ip=150.243.160.42 bcast=150.243.191.255 nmask=255.255.224.0
Password:
Domain=[TRUMAN] OS=[Windows 5.0] Server=[Windows 2000 LAN Manager]
smb: \>
```

Once connected, **smbclient** accepts various commands. Among them are:

Table 7-1. smbclient commands

help	lists the commands understood by the program
help <i>command</i>	prints help for a specific command
cd	change the remote directory

dir (or ls)	list the contents of the remote directory
get <i>remotefile</i>	get a file from the remote machine
put <i>localfile</i>	put a file onto the remote machine
mget or mput	get or put multiple files at a time

You can use these commands to navigate the system and transfer files:

Example 7-2. Navigating in smbclient

```
smb: \> dir
.                DR          0  Fri Aug 30 15:08:00 2002
..               DR          0  Fri Aug 30 15:08:00 2002
_BU Student File Area  D          0  Mon Mar 17 17:26:41 2003
_ED Student File Area  D          0  Wed Feb 26 07:42:55 2003
_FN Student File Area  D          0  Tue Dec 10 13:12:32 2002
_HES Student File Area D          0  Wed Feb  5 16:28:11 2003
_ITS Student File Area D          0  Fri Feb 28 14:20:15 2003
_LL Student File Area  D          0  Tue Dec  3 17:08:22 2002
_MT Student File Area  D          0  Mon Feb  3 18:39:54 2003
_NU Student File Area  D          0  Fri Feb 21 11:53:03 2003
_PL Student File Area  D          0  Wed Mar 19 18:17:48 2003
_SC Student File Area  D          0  Fri Mar 21 16:42:10 2003
_SS Student File Area  D          0  Mon Nov 11 15:39:28 2002
_UB Student File Area  D          0  Tue Dec 17 09:51:32 2002

19200 blocks of size 4096. 19200 blocks available
smb: \> cd "_MT Student File Area"
smb: \_MT Student File Area\> cd dbindner
smb: \_MT Student File Area\dbindner\> dir
.                D          0  Tue Apr  9 14:05:05 2002
..               D          0  Tue Apr  9 14:05:05 2002
brian            D          0  Tue May  1 05:45:30 2001
CalcIII         D          0  Tue May  1 05:45:30 2001
csl20           D          0  Tue Apr  9 13:50:21 2002
gvim.lnk        509      Tue Apr  9 14:04:44 2002
Office2000      D          0  Tue Apr  9 13:49:38 2002
Sound           D          0  Tue May  1 05:44:57 2001
vim             D          0  Wed Oct 16 16:04:40 2002

19200 blocks of size 4096. 19200 blocks available
smb: \_MT Student File Area\dbindner\> get gvim.lnk
getting file gvim.lnk of size 509 as gvim.lnk (99.4 kb/s) (average 99.4 kb/s)
smb: \_MT Student File Area\dbindner\> quit
```

Accessing Windows shares using smbmount

If you prefer to have Windows shares appear as regular parts of the unix filesystem, you can do that with the **smbmount** command. Smbmount has two required arguments: the name of a share, and a directory to attach it to. As with the regular **mount** command, this directory is often referred to as a mount point.

In addition to the required commands, it is customary to pass some options to the smbmount command via the **-o** switch. The most common options are **username** which specifies your Windows account name, and **workgroup** which specifies your workgroup (sometimes this is called your domain). The command will prompt for the password of this account when it runs.

Note: Mounting and unmounting filesystems usually requires root privileges. On most systems, smbmount “runs as root,” but if it does not on yours, you will need to become root first (usually via the **su** command).

Example 7-3. Mounting a Windows share

```
$ mkdir u_drive
$ smbmount //Ss1/Student_files u_drive -o username=dbindner,workgroup=TRUMAN
Password:
$
```

To unmount the share, simply use the **smbumount** command (note the spelling, just like the regular **umount** command).

Example 7-4. Unmounting a Windows share

```
$ smbumount u_drive
$
```

Connecting to other computers using SSH

It is common practice to work with unix computers remotely, i.e. over a modem or from the Internet. Historically, **telnet** or **rlogin** would have been used to do this. Neither was ever secure, and today both have been supplanted by a secure alternative, the Secure Shell **ssh**.

The ssh program allows you to work interactively on a remote computer. You can type commands and see their output. In addition, all of the data transferred between the local and remote computer is encrypted. No one can eavesdrop on the text you type, nor see the output that is sent back to you. In its easiest form, you just type **ssh remotehost**.

Example 7-5. Basic ssh example

```
$ ssh kronos
Last login: Tue Mar 25 10:21:42 2003 from limestone.truman.edu on pts/0
Linux vh222005 2.4.20 #2 SMP Sun Jan 26 20:13:46 CST 2003 i686 unknown
```

You can access your home directory from a Windows computer on the Truman network by sharing `\\kronos\user` where user is

your account name.

Daily filesystem backups can be found in /snapshot.

A new graphical statistics program is installed, called "salstat".

Here the "message of the day" for kronos has been displayed, and you can see on the last line that the prompt indicates a hostname of vh222005 (the true name of kronos). Any commands typed from this point on, will be executed on kronos. When you are finished, simply type **exit** or press **Ctrl-D** to exit the shell and close the connection.

The ssh program is the preferred method for creating secure connections to the unix servers at Truman, including xenon, gold, kronos, and gaia.

Sometimes you may need to login as a different user on the remote host than the username you use on the local host. In that case, you may specify the username and hostname together as *user@remote*.

Example 7-6. Using ssh with a different remote username

```
$ ssh root@kronos
```

Transferring files with SSH tools

The most common file transfer program of old was a program called ftp, short for "file transfer program." Like telnet and rlogin, the ftp program is unsecure and has been replaced by ssh tools. There are two tools you might use to copy files between machines. They are sftp and scp.

Transferring files with sftp

The sftp program is an interactive client, much like smbclient (or an ftp client if you have used one). You can use it to transfer files with any computer that you can ssh to. The syntax is simple. You may simply type **sftp remotehost**, or if you need to specify a username, **sftp user@remotehost**. When running, it accepts many of the same commands as smbclient, including:

Table 7-2. sftp commands

help	lists the commands understood by the program
cd	change the remote directory
dir (or ls)	list the contents of the remote directory
get remotefile	get a file from the remote machine
put localfile	put a file onto the remote machine

For example:

Example 7-7. Get and put with sftp

```
$ sftp kronos
```

```

Connecting to kronos...
sftp> get hello.cc
Fetching /home/ldap/dbindner/hello.cc to hello.cc
sftp> put goodbye.cc
Uploading goodbye.cc to /home/ldap/dbindner/goodbye.cc
sftp> quit

```

Transferring files with scp

The scp program is the other (secure) means to transfer files between computers. It works very much like the regular copy command cp, except that either the source file or destination file may be prepended with a hostname. For example:

Example 7-8. Copying with scp

```

$ scp hello.cc kronos:hello.cc
hello.cc          100% |*****|          344          00:00
$ scp kronos:hello.o .
hello.o           100% |*****|         3528          00:00
$ scp dbindner@kronos:hello.cc /tmp/hello.cc
hello.cc          100% |*****|          344          00:00

```

Here the first command copies a local file, `hello.cc` to the remote host `kronos`, leaving it in the home directory there. The second command copies a remote file `hello.o` to the local current directory (dot always refers to the current directory). The third command copies a file `hello.cc` from `kronos`, specifying explicitly that the connection on `kronos` should be made as user `dbindner`.

Using the Truman VPN from Linux

Basic VPN setup

Faculty and students who live off campus cannot directly use some Truman resources from their home computers. Examples include the network drives, some library databases, and the Safari Bookshelf at `oreilly.com`.

To provide access to Truman-only resources, ITS maintains a virtual private network (VPN) server. When you use the VPN, an encrypted connection is created between your computer and Truman. Your non-Truman computer receives a Truman internet address and can access any resource that a Truman computer could.

There is a free software program called `vpnc` available for Linux that is compatible with Truman's VPN. Configuration consists of a small file called `/etc/vpnc/default.conf` (which you create as root). This file must contain at least the following 4 lines.

Example 7-9. /etc/vpnc/default.conf

```
IPSec gateway vpn.truman.edu
IPSec ID GroupName
IPSec secret GroupPwd
Xauth username Username
```

For *Username*, you should use your own user name on the Truman network. For the *GroupName* and *GroupPwd* values, check the VPN Client for Linux (<https://secure.truman.edu/its-s/vpn/linux/index.asp>) directions at the ITS web site.

You will be prompted for your Truman password each time you connect to the VPN.

Some versions of Linux come with a helper script called `vpnc-connect` which connects to the VPN and routes network traffic through it.

```
# vpnc-connect
Enter password for dbindner@vpn.truman.edu:
VPNC started in background (pid: 25872)...
```

On systems that do not have the `vpnc-connect` program, running `vpnc` directly will start the VPN and route network traffic.

To stop the VPN, run `vpnc-disconnect`.

Advanced routing with VPN

One negative effect of starting the VPN is that it interrupts already existing network connections. For example if you were connected to another system via `ssh`, or if you were running an instant messenger, the session will freeze when the VPN is connected. The cause of this is the change in network routing that occurs. With the VPN, your computer effectively has a new internet address, so network traffic to the old address breaks.

This behavior can be avoided with a little care using an advanced technique called policy routing. The goal is to introduce a routing rule that protects existing network traffic while allowing the VPN to work correctly for new connections.

To invoke the correct rule, it is necessary to know some information about the way traffic is routed on your system. Routes differ from system to system, and can be reviewed using the `ip` tool.

```
# ip route show
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.2
default via 192.168.1.1 dev eth0
```

There are two things to be looking for. The first is the set of routes that exist on our ethernet device (look for the lines containing `eth0`). The second is our source address, which is `192.168.1.2`.

Using this information, we create a new routing policy:

```
# ip route add 192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.2 table 1
# ip route add default via 192.168.1.1 dev eth0 table 1
# ip rule add from 192.168.1.2/32 table 1 priority 501
# ip route flush cache
```

The first two lines simply recreate the existing routes in a new policy table, table 1. The third line adds policy to that table: table 1 is to be used for internet traffic whose source address is 192.168.1.2. The last line ensures that the new policy takes effect immediately.

This only needs to be done once (each time the computer is booted). The policy is correct whether the VPN is connected or disconnected, so there should be no need to delete it. If you wish, however, it can be removed by typing the same three lines while substituting "del" for "add".

Chapter 8. Finding Things on a Linux system

Modern computing systems, including Linux, typically contain tens of thousands of files or more. It can be easy to lose something of value among all of these files, whether you are a beginner or an advanced user. Luckily, there are conventions for standard file locations that can give you hints about where to begin a search. There are also powerful programs for searching the filesystem.

Directory structure on a Linux system

Many of the files on a Linux system go in specific locations. For example, you will find most installed software in the `/usr` (pronounced “user”) directory. In particular, most Linux distributions install their software in this directory. Under `/usr` there are directories for various parts of applications. Executable programs (the part that you would run) are often referred to as binaries, and can be found in `/usr/bin`. The “man” pages, containing online manuals are found in `/usr/man`, and so on.

By convention, applications that were not pre-packaged with your Linux distribution are stored in `/usr/local`, so they do not conflict with other programs. Like `/usr`, it is subdivided into `/usr/local/bin`, `/usr/local/man`, and so forth.

Hint: Although Linux beginners are usually aware that there are hundreds of commands that they might type in their shell, they often don’t know the names of many commands. One very enlightening way to learn about the commands on a system is to peruse the `/bin`, `/usr/bin`, and `/usr/local/bin` directories (a la `ls /bin | less`). Type `man command` to read the manual page about each command and learn what it does.

Other directories that will be of interest are `/etc` (pronounced “et-see”) where configuration information is kept for many programs, `/tmp` (“temp”) where temporary files may be placed, and `/mnt` (“mount”) where new filesystems are attached (mounted). Unread email and files that are in the process of being printed are both stored in `/var`, as are the system logs and other files which change frequently.

The `/home` directory is where personal files are stored. The typical place for a user’s home directory is named after their login account, `/home/user`, although this may differ from system to system. You can learn the location of your home directory by printing the `$HOME` variable.

Example 8-1. Printing the home directory

```
$ echo $HOME
/home/ldap/dbindner
```

Programs for finding files

Finding files with grep

There are many powerful programs for finding information on a Linux system, depending on what you know. Say for example, you were working on your resume in your home directory, but can’t remember

the name of the file you saved it in. In this case, you likely know some of the text in the file, and you could use **grep** to find it (and the file).

Example 8-2. Finding text with **grep**

```
$ grep "Work experience" *  
...
```

The above example would search every file (indicated by the `*`) for the exact text “Work experience” and return any matching lines along with the names of the files where the match occurred. This is a simple use of **grep**, but one that is very common. **Grep** is also capable of very sophisticated searches using objects called regular expressions. Regular expressions are an advanced topic of their own; for a beginner it is probably best to search only for plain words without punctuation. In particular, the symbols for “`^$. *+?()[]{} \`” have special meaning in searches and should be avoided until you have learned something about regular expressions.

Finding files with **locate**

If you know the name of the file you are looking for, but do not know what directory it is in, the simplest and fastest command for finding it is usually **locate**. It is commonly paired with **grep** to make the search more selective.

Example 8-3. Finding with **locate**

```
$ locate printcap  
/etc/printcap  
/etc/printcap.dpkg-dist  
/usr/share/doc/lpr/examples/printcap  
/usr/share/man/man5/printcap.5.gz  
$ locate .bashrc | grep dbindner  
/home/ldap/dbindner/.bashrc  
/home/ldap/dbindner/.bashrc.bak
```

In the example above, there are 4 files that match the pattern “printcap” in the first search. In the second search, it is important to remember that there are usually many `.bashrc` files on a system; almost every user is guaranteed to have one which the **locate** command will happily report. The **grep** command is used to limit the files to those also having the pattern “dbindner” in them, a much more manageable set.

Note: Rather than search the entire system for a file, **locate** uses a pre-compiled database of files. The database is typically updated once a day (by searching the entire system sometime during the night). This makes **locate** very fast, but it also means that **locate** cannot be used to find a file you created (and lost) 15 minutes ago.

Finding files with **whereis** and **which**

On many unix systems, there will be multiple sets of the same programs and tools installed. For example, there may be two **ps** programs, one conforming to AT&T Unix conventions and another BSD-style version. Although this is less common on Linux systems, it can still sometimes be helpful to

know exactly which program is being run when you type a command. The commands for learning this information are **whereis** and **which**.

Example 8-4. Using whereis and which

```
$ whereis chfn
chfn: /usr/bin/chfn /usr/local/bin/chfn /usr/share/man/man1/chfn.1.gz
$ which chfn
/usr/local/bin/chfn
```

Here there are two **chfn** programs. The one that came with the system is `/usr/bin/chfn`, and another one has been installed in `/usr/local/bin/chfn`. The **which** command identifies the “local” one as the default. That is, if you type **chfn** (the command to change your finger information), it will use the copy in `/usr/local/bin`. The other **chfn** program can still be used, but you must type the entire pathname of the program to run it.

Finding files with find

The heavyweight champion of finding programs is undoubtedly **find**. It is a very flexible program that few people (even advanced people) fully take advantage of. For the beginner however, basic use of **find** is still quite straightforward.

The syntax of a **find** command is **find path expression**. The *path* indicates where the search should begin (the search will recursively include all of the subdirectories of the starting spot) and is commonly either `/` for searching the whole filesystem or `.` for searching under the current directory. The *expression* indicates how the search is to be performed and the action to take on matches. Typical use would be:

Example 8-5. Finding a file with find

```
$ find . -name .bashrc -print
./bashrc
$ find . -iname "**Hello*" -print
./octave/hello.cc
./octave/hello.o
./octave/hello.oct
$ find / -name .bashrc -print 2>/dev/null | grep dbindner
/home/ldap/dbindner/.bashrc
```

The first search is straightforward. It prints the names of files that match `.bashrc` exactly. The second performs a case insensitive name match (`-iname`) for filenames containing “Hello” (the stars mean match anything at the beginning and end) and prints the results.

The final search looks through the whole filesystem for `.bashrc` files and then “greps” for those containing the string `dbindner`. This command takes some time because the whole filesystem has to be searched. The “`2>/dev/null`” part of the command tells the shell to ignore any error output that the **find** command might generate. **find** will print warnings when encountering directories we do not have the ability to search, and this will effectively ignore those.

Programs for finding people

Listing who is currently logged on

On a typical Linux system, there are numerous programs for finding out who is logged on, and what they are doing. Among these are **who**, **w**, and **finger**.

Example 8-6. Finding out who is logged on

```
$ who
thammond pts/0          May 28 10:14 (vh225002.truman.edu)
agarvey  pts/1          May 30 11:44 (vh216601.truman.edu)
hammond  vh225002.truman.edu:1 Jun  5 17:23
thammond pts/3          May 28 12:26 (vh225002.truman.edu)
dbindner pts/7          Jun 11 13:13 (modem65.ipv6.truman.edu)
hammond  pts/9          Jun 11 11:08 (vh225002.truman.edu)

$ w
 13:14:20 up 14 days, 19:31,  6 users,  load average: 2.07, 2.00, 1.98
USER      TTY      FROM          LOGIN@      IDLE   JCPU   PCPU   WHAT
thammond  pts/0    vh225002.truman. 28May03    2:06m   0.33s  0.33s  -bash
agarvey   pts/1    vh216601.truman. 30May03    9days   6.51s  6.51s  -bash
hammond   vh225002 -          Thu17     0.00s 13days  0.13s  sshd
thammond  pts/3    vh225002.truman. 28May03    2:06m   0.55s  0.55s  -bash
dbindner  pts/7    modem65.ipv6.tru 13:13     0.00s   0.03s  0.02s  w
hammond   pts/9    vh225002.truman. 11:08     1:58m   0.01s  0.01s  -bash

$ finger
Login      Name                Tty      Idle  Login Time  Office          Office Phone
agarvey    Alan Garvey         pts/1     9d    May 30 11:44 (vh216601.truman.edu)
dbindner   Donald J Bindner    pts/7     Jun 11 13:13 (modem65.ipv6.truman.edu)
hammond    Todd Hammond       *vh22500 Jun  5 17:23
hammond    Todd Hammond       pts/9     1:59  Jun 11 11:08 (vh225002.truman.edu)
thammond   Todd Hammond       pts/0     2:07  May 28 10:14 (vh225002.truman.edu)
thammond   Todd Hammond       pts/3     2:06  May 28 12:26 (vh225002.truman.edu)
```

The **who**, **w**, and **finger** commands return many of the same pieces of information, including the account name, the amount of idle time (i.e. how long since a key has been typed), and where the person is logged in from. Which of the commands you choose to use is mostly a matter of preference.

If you need to know who has been on recently, rather than who is on at the moment, the **last** command can list that. It takes a single argument **-n** where **n** indicates you would like a listing of the **n** most recent logins.

Example 8-7. Using the last command

```
$ last -10
dknudson  pts/8          0-1pool245-222.n Wed Jun 11 13:14 - 13:14 (00:00)
dbindner  pts/7          modem65.ipv6.tru Wed Jun 11 13:13 - 13:24 (00:10)
dgarth    vh222005.tru   Wed Jun 11 12:23 - 12:26 (00:02)
dknudson  pts/7          0-1pool245-222.n Wed Jun 11 12:20 - 12:46 (00:25)
hammond   vh222005.tru   Wed Jun 11 11:53 - 12:14 (00:21)
cmj489    pts/7          :0.0        Wed Jun 11 11:34 - 11:35 (00:00)
```

```
cmj489      :0                               Wed Jun 11 11:34 - 11:35 (00:00)
hammond    vh222005.tru                      Wed Jun 11 11:09 - 11:15 (00:05)
hammond    pts/9                          vh225002.truman. Wed Jun 11 11:08 still logged in
cmj489     pts/9                          :0.0                               Wed Jun 11 09:41 - 09:41 (00:00)
```

```
wtmp begins Sun Jun  1 18:51:35 2003
```

Finding information about an individual

There are some situations where you would like more specific information about a particular account. For example, you may have happened across a terminal where the previous user has forgotten to logout. You can identify the user easily with `who`:

Example 8-8. Who am I?

```
$ who am i
dbindner pts/7          Jun 11 13:13 (modem65.ipv6.truman.edu)
```

Perhaps you have found a file belonging to an account (say `dbindner`), or noticed that a particular user is using a lot of processor time. The `finger` command can conveniently place a name to the account. It can also tell you if that person has new email waiting for them on the system, or the last time they read it. Some systems can be fingered remotely as well, using a `user@host` syntax.

Example 8-9. Fingering a single account

```
$ finger dbindner
Login: dbindner                Name: Donald J Bindner
Directory: /home/ldap/dbindner Shell: /bin/bash
Office: Violette Hall 2244, 785-4258
On since Wed Jun 11 13:13 (CDT) on pts/7 from modem65.ipv6.truman.edu
No mail.
No Plan.
$ finger dbindner@limestone
[limestone.truman.edu]
Login: dbindner                Name: Donald J Bindner
Directory: /home/dbindner      Shell: /bin/bash
Office: 2244 Violette Hall, 785-4258
On since Tue May 13 10:41 (CDT) on tty1    29 days 2 hours idle
      (messages off)
Last login Wed Jun 11 12:15 (CDT) on pts/1 from modem65.ipv6.truman.edu
Mail last read Wed Jun 11 13:07 2003 (CDT)
Plan:

      Public key: http://limestone.truman.edu/~dbindner/public.txt
      Key fingerprint = 95B9 6D7A 3457 E0C7 4636 A9F9 2EE7 86CE 1B23 F31F
```

On some systems, the `finger` command has been disabled, but often much of the same information can be obtained from the `/etc/passwd` file using `grep`. For example:

Example 8-10. Grepping for finger information

```
$ grep "dbindner" /etc/passwd  
dbindner:x:1000:1000:Donald J Bindner,2244 Violette Hall,785-4258,:/home/dbindner:/bin/bash
```

Chapter 9. Managing Jobs and Processes

Processes

Unix is a multiuser, multitasking operating system, which means that many users can run many programs, all at the same time. It is the responsibility of the kernel to allocate CPU time to each of the running programs of a system.

Processes are what the kernel uses to handle this accounting. You can usually think of each process as a separate program that is running on the system. However, some large programs, like a web browser, may actually be comprised of many separate tasks (processes) that run simultaneously.

The ps command

The **ps** command is used to list the processes running on the computer. It can be used to list both your processes and the processes of other users. In its simplest form, you just type **ps**.

Example 9-1. Basic ps invocation

```
$ ps
  PID TTY          TIME CMD
 3155 tty2      00:00:00 -bash
 3190 tty2      00:00:00 ps
```

Here two processes are listed. The first process, whose process ID number (PID) is 3155, is bash. The second process refers to the ps program itself, which also uses a process when it is running.

The **ps** command can accept many options to alter the way it lists processes and to alter which processes are listed. Among these are **a** which tells ps to report processes for all users, **u** which tells ps to report the user that owns each process, and **x** which indicates to list all processes, even those you “might normally not be interested in.” For example:

Example 9-2. Running ps with x option

```
$ ps x
  PID TTY          STAT TIME COMMAND
   545 ?           SN    0:00 ssh-agent -s
 2660 ?           SN    0:00 ssh-agent -s
 2875 ?           SN    0:00 ssh-agent -s
 2961 ?           S     0:00 ssh-agent -s
 2989 tty1          S     0:00 -bash
 3117 tty1          S     0:01 vim jobs.html
 3155 tty2          S     0:00 -bash
 3206 tty2          R     0:00 ps -x
```

Here, ps has listed the processes that are running on terminal 2 (tty2), but has also listed the processes running on tty1. It has also listed several ssh-agent processes that aren't associated with any terminal at

all. On a graphical system, this command would have shown even more processes, including every graphical program being run by the user (at the very least). You can get an idea of all the processes running on a computer with **ps aux**.

Job control

Before graphical environments were common-place, job control was essential for working with multiple programs at the same time. Even in a graphical environment, job control can be a huge convenience for working with multiple programs without the need for a large number of open windows.

Foreground and background

To understand job control, it is useful to begin with foreground and background tasks. Generally, every program that runs is a foreground task. I might type **ls**, then edit a file, then **rm *.bak**. Each program, as a foreground task, writes its output to the screen and accepts input from the keyboard.

Sometimes we might want to run a program that we know is going to take a long time to complete. For example, we may have lost a file somewhere on the system called `needle.txt`. To find it, we could issue a find command, **find / -name needle.txt -print**, but searching the entire filesystem may take several minutes. Rather than wait, it makes sense to run this as a background task.

Example 9-3. Running a background task

```
$ find / -name needle.txt -print >list.txt 2>/dev/null &
```

Here, the find command is seen at the beginning of the line. The remaining parts of the line, in order, are:

`>list.txt`

Tells the shell to put the output of this command into a file. Since we are running the command in the background, we don't want the output to just go to the terminal where it would be interleaved with the output from other programs we are running. We can look in this file for the results when the program finishes.

`2>/dev/null`

Tells the shell that any error output should be put into `/dev/null` (effectively ignored).

`&`

Tells the shell to run this program in the background. A new prompt will immediately appear without waiting for the program to finish. The **find** program will continue running in the background until completion.

Working with jobs

Job control can be used for more than running programs in the background. It can also be used to conveniently switch back and forth between programs. For example, imagine you are making a file

called `calc.txt` to put various calculations into. A very crude (but simple) way to create such a file is using the `cat` command.

Example 9-4. Suspending a task

```
$ cat > calc.txt
The sum 1+1 is equal to 2.
Zero plus any number is itself.
^Z
[1]+  Stopped                  cat >calc.txt
```

Where the `^Z` is shown, the user pressed **Ctrl-Z**. This will suspend the execution of most programs and present another shell prompt that can be used to run more programs. Perhaps we need to calculate the value for `10!`. At the shell prompt, we could type:

Example 9-5. Running and suspending a second task

```
$ calc
C-style arbitrary precision calculator (version 2.11.5t4.5)
Calc is open software. For license details type:  help copyright
[Type "exit" to exit, or "help" for help.]

> 10!
          3628800
> ^Z
[2]+  Stopped                  calc
```

Now there are two stopped jobs. We can list the current jobs using the `jobs` command.

Example 9-6. Listing jobs

```
$ jobs
[1]-  Stopped                  cat >calc.txt
[2]+  Stopped                  calc
```

The `fg` command will bring a job into the foreground. Without arguments, `fg` will attempt to guess what the “current job” is and bring that to the foreground. You can specify a particular job using the percent sign (%) and the job number. For example:

Example 9-7. Resuming a job

```
$ fg %1
cat >calc.txt
The value for 10! is 3628800.
^D
```

The `fg` command restarted the `cat` job, and it ran until we typed **Ctrl-D** (the symbol that marks the end of input under unix). If there were many calculations to complete, we could just as easily have used **Ctrl-Z** to suspend again and brought the calculator back to the foreground in the same fashion.

Since we are finished using the calculator, we can simply close the `calc` program. Typing `fg` will bring it to the foreground (it is now the current job, since the other program has finished), and we can exit

normally. If there were some problem with the `calc` program, we could also kill it with `kill %2` or `kill -9 %2` as necessary.

Putting a foreground job into the background

Starting a job in the foreground only to discover that it should go in the background is a fairly common occurrence. To move the task to the background, simply suspend it with **Ctrl-Z** and then use the **bg** command. For example:

Example 9-8. Moving a job to the background

```
$ xterm
^Z
[1]+  Stopped                  xterm
$ bg %1
[1]+  xterm &
$
```

This `xterm` was accidentally started in the foreground and then moved to the background. It will now show on the job list as *Running*, and it can be moved back into the foreground if necessary with the **fg** command.

Chapter 10. The X11 Graphical Environment

Some kind of introduction should go here.

The X Window System as a network application

Simple X11 display forwarding

One very powerful aspect of the X11 system is network transparency. With X11, it is very easy to run a program on one computer (say, kronos, in the math division), yet have the graphical output of the program be displayed on another computer (say, your personal workstation). You might do this to run a very resource-intensive program like Mathematica on a powerful computer from your wimpy little home machine.

One simple (but insecure) way to run graphical programs over a network connection is to start X with the `-query` option.

Example 10-1. Invoking X with the `-query` option

```
$ X -query kronos.truman.edu :0
```

This starts an X server on the workstation and tells it to contact kronos for output. When it works correctly, a kronos login screen is displayed. If your system is already running a copy of X, it may be necessary to specify a different display number; perhaps `:1` or `:2`.

If you login and begin working, every program you run will execute on kronos, but its interface will display on the machine in front of you. Part of the magic that makes this work is the `DISPLAY` environment variable. It contains the host (optional), display, and screen number of the current X server.

Example 10-2. Viewing the `DISPLAY` variable

```
$ echo $DISPLAY
limestone.truman.edu:0.0
```

This can be a very convenient way to work with graphical programs on another system, and it highlights the part of the power of the X11 system. However, it is also insecure. None of the data sent between the systems is encrypted. That means that anyone who can see the traffic on the network knows what you type (including your login password) and can see what you see. There are more secure ways of remotely using graphical programs.

X display forwarding with SSH

The secure shell program, SSH, can be used to forward X11 data. All of the data travels through an encrypted tunnel, which means that your keystrokes and visual data cannot be intercepted. Setting up the tunnel is as easy as using ssh's `-X` option.

Example 10-3. Forwarding X11 via SSH

```
$ ssh -X dbindner@kronos
dbindner@kronos's password:
kronos$ echo $DISPLAY
localhost:11.0
kronos$ xlogo
```

Although the DISPLAY variable shows the display as localhost (meaning kronos, because that is the machine we ran the command on) display number 11 screen 0, it is actually sent back through the SSH connection. That means the xlogo program displays (securely) on my workstation as would any other graphical program I invoke.

Customizing the keyboard map

It might at first seem a bit strange to think that you would want to change the keyboard map, i.e. the symbols that the various keys on the keyboard generate. However, the way key presses are converted into characters under X11 is completely customizable, and this can be surprisingly convenient.

For example, if you enroll in a foreign language class, you may find yourself typing papers that contain characters or diacritical marks not used in English. In French, it is common to need a ‘c’ with a cedilla or an ‘e’ with a grave accent. In German, the eszett is required. All of these characters can be conveniently mapped to keys on the keyboard so they are available in programs like OpenOffice.

Before creating new key definitions, it is important to realize that there are two kinds of keys on a keyboard. There are keys that “do something” and there are keys that modify the behavior of other keys.

Shift, **Ctrl**, **Alt**, **Caps Lock**, **Num Lock**, and **Scroll Lock** are all modifier keys. On some keyboards you may also find **Meta**, **Mode switch**, and **Multi** keys. In fact, the Meta key is so important, that on many keyboards without a separate **Meta** key, the key labeled **Alt** is actually mapped to Meta.

There is a program that can tell you all of the key bindings under X called **xev** which reports “X events.” When run from an xterm window, xev opens a new small window. Move the mouse into the window and numerous events will be reported back in the xterm, mouse events in this case. Hit a key, however, and you see events corresponding to the key press and key release. For our use, the keycode number and keysym are the important parts to note.

Mapping single keys with xmodmap

According to xev, the right-most “Windows key” on my IBM keyboard has keycode 117. I don’t have much use for such a key under X, so I have given it a new assignment with xmodmap, to **Mode switch**. The command to do this is:

Example 10-4. Assigning one key with xmodmap

```
$ xmodmap -e "keycode 117 = Mode_switch"
```

The **Mode switch** key is sort of like a super shift key, and we can use it to map foreign characters onto the keyboard. For example, to add a grave ‘e’ to the keyboard, we might use:

Example 10-5. Assigning an accented character to the keyboard

```
$ xmodmap -e "keysym e = e E egrave Egrave"
```

Notice here the assignment is made by key symbol and not by key code, but otherwise the syntax is similar. This line assigns 4 characters to the what is currently the E key (usually this is the key with the letter E painted on it, but hopefully you can see why this wouldn't have to be the case). Unshifted, the key produces a lowercase e. With shift, it produces an upper case E. With mode switched (when you hold the **Mode switch** and press **E**), it produces a lowercase e with a grave accent. Finally, with both shift and mode switch, it produces an uppercase E with a grave accent.

Mapping the entire keyboard with xmodmap

The commands above might be convenient for mapping one or two keys, but it would be a burden to make mappings for a whole keyboard that way. Fortunately it is easy to save a key map, edit it, and reload the modified map again. To get the current map from xmodmap, use the `-pke` option.

Example 10-6. Saving the current key map

```
$ xmodmap -pke > keyboard.orig
```

This creates a file of keycode mappings, one per line, describing the current keyboard layout. Edit a copy of this file to make any desired changes and reload it with xmodmap to make the new assignments effective.

Example 10-7. Changing the entire key map

```
$ cp keyboard.orig keyboard.new
$ vi keyboard.new
$ xmodmap keyboard.new
```

The keyboard map is reset every time you logout (you wouldn't want your friends stuck with your map), so this file also serves as a way to restore everything to your preferred configuration when you login again.

Other customizations and information about using xmodmap

To make the most effective use of foreign characters and diacritical marks, it is helpful to have a list of available key symbols. One authoritative source of symbol names is `keysymdef.h` which is often located in `/usr/X11R6/include/X11/keysymdef.h`. You can also search for it using the **locate** command. It contain key symbol definitions for programmer, which typically have names like `XK_egrave`. Remove the `XK_` to obtain names that can be used in xmodmap.

Note: `keysymdef.h` has symbols for many different character sets. You are probably using the Latin 1 set, so look for the section that begins `"#ifdef XK_LATIN1"`.

Xmodmap can be used to make changes to the pointer (mouse) buttons as well. The most common use is to reverse the order of the buttons for left-handed people. It can also make buttons sticky, the way the **Caps Lock** is. Check the man page for more details.

Finally, there is one other way to enter foreign characters from the keyboard, by using a **Multi** key. If you define a key to have the `Multi_key` symbol, when you press that key, the computer knows the next two keys are to be combined to form a special character. For example, **Multi** then `'` then **a** would enter a grave a.

Chapter 11. Documentation and Help

If you make very much use of any computer system, you will find yourself needing help. This may be even more true with unix systems, where much of the strength of the environment depends on typed commands. Luckily, unix systems typically have generous amounts of online documentation.

Man pages

The most traditional form of documentation on a unix system is the collection of man, short for manual, pages. The man page for a program can be accessed using the **man** command. To learn about a program, type “man” and the name of the program, and a browsable set of directions is displayed (shown below in the example). For example, to learn about the **ls** program, type **man ls**. Almost all commands have man pages. Even the man program itself has a man page (to read it, do **man man**).

Example 11-1. Viewing the ls man page

```
LS(1)                                FSF                                LS(1)

NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILES (the current directory by
    default). Sort entries alphabetically if none of -cftuSUX
    nor --sort.

    -a, --all
        do not hide entries starting with .

    -A, --almost-all
        do not list implied . and ..

    -b, --escape
        print octal escapes for nongraphic characters

    --block-size=SIZE
        use SIZE-byte blocks

    -B, --ignore-backups
        do not list implied entries ending with ~

    -c      with -lt: sort by, and show, ctime (time of last
            modification of file status information) with -l:
            show ctime and sort by name otherwise: sort by
            ctime

lines 1-34
```

Since the man environment is interactive, it is helpful to know some of the keystrokes for navigating through it. As with most unix pagers (programs that show data one page at a time), **Space** or **f** moves one page forward. Use **b** to move one page backward, **/** to search for a word, and **n** to repeat the last search. Finally, use **q** to quit. To learn other useful keys, see the man pages for **more** or **less**, two popular pagers. Also check out the Section called *Environment variables* in Chapter 4 to learn how to set a default pager.

Sometimes you will need help, but you will not know the name of the command to ask for. When this happens, the **apropos** command is very helpful. For example, to find help related to web browsers, try **apropos browser** for a list of possible topics. Apropos will do a keyword search through the description sections of man pages and list the results, in this case including the names of web browsers on your system. Sometimes a little creativity is required as well. For example, to learn how to delete a file, you might be tempted to try **apropos delete** without success. However, **apropos remove** lists, among other things, the desired **rm** command.

One note about man pages. Most beginners find them daunting at first. A good man page is very detailed, usually containing more detail than a beginner wants. Amazingly enough, one day you'll find yourself wishing man pages had more detail. When that happens you will know you are on your way to becoming a skilled unix user.

Another note is in order. More than just programs are documented by man pages. Many programming topics also have man pages, a real convenience for programmers. For example, there is a **mkdir** command for making a new directory, but there is also a **mkdir()** function that can be used in C programs to make a new directory. Both have man pages. You may, on some occasions, find man returning a page other than the one you seek. You can get man to return all of the pages in turn by passing the **-a** switch, as in **man -a mkdir**.

The Texinfo system

The Texinfo system is a more recent development for program documentation than man pages, and it is the standard documentation format for many GNU programs (the GNU project is an important source of free software). It was designed so that the same file could be used to generate printed documentation with indexes as well as hyperlinked online documentation (Texinfo predates the web but works similarly). The most common tool for browsing Texinfo documentation is the **info** command, and Texinfo documentation is often referred to as info pages.

To see, for example, the info page for **ls**, type **info ls**. Naturally, info itself has info pages which can be displayed with **info info**. There are pages both about writing info documentation as well as browsing it. To learn about the "stand-alone" info browser type **info info-stand**.

Example 11-2. Learning about the stand-alone info browser

```
File: info-stand.info, Node: Top, Next: What is Info, Up: (dir)
```

```
GNU Info
*****
```

```
    This file documents GNU Info, a program for viewing the on-line
    formatted versions of Texinfo files, version 4.1. This documentation
    is different from the documentation for the Info reader that is part of
    GNU Emacs.
```

This manual is for Info version 4.1, updated 2 March 2002.

```
* Menu:

* What is Info::          What is Info?
* Invoking Info::       Options you can pass on the command line.
* Cursor Commands::    Commands which move the cursor within a node.
* Scrolling Commands:: Commands for reading the text within a node.
* Node Commands::      Commands for selecting a new node.
* Searching Commands::  Commands for searching an Info file.
* Xref Commands::      Commands for selecting cross references.
* Window Commands::    Commands which manipulate multiple windows.
* Printing Nodes::     How to print out the contents of a node.
* Miscellaneous Commands:: A few commands that defy categories.
* Variables::          How to change the default behavior of Info.
* Custom Key Bindings:: How to define your own key-to-command
                       bindings.
* Index::              Global index containing keystrokes,
                       command names, variable names,
                       and general concepts.

--zz-Info: (info-stnd.info.gz)Top, 31 lines --All-- Subfile: info-stnd.info-1.gz
Welcome to Info version 4.1. Type C-h for help, m for menu item.
```

Info pages are browsed using different keys than man pages. As might be expected, **PgDn** and **PgUp** scroll by pages. **Tab** moves forward from one hyperlink to the next, and **Esc-Tab** moves backward through hyperlinks. The **Enter** key follows a hyperlink. Searching can be done with the **s** key and the last search is repeated with **Ctrl-xn**. Finally, pages are hierarchical, and **u** moves up one level of the hierarchy.

Other sources of help

Getting help from programs themselves

Many programs can offer immediate help without the need for searching out man pages, info pages, or other documentation. It is a very widely-respected convention that programs offer help when invoked with an appropriate switch. Programs that support only what are called “short options,” options formed by a minus sign and a single letter, often print help when the **-h** switch is used. Programs that support “long options,” options formed with two minus signs and a word, often accept a **--help** switch. The long help option is particularly well-supported by GNU programs.

Example 11-3. Getting help from the gzip program

```
$ gzip --help
gzip 1.3.2
(2001-11-03)
usage: gzip [-cdfhlLnNrtvV19] [-S suffix] [file ...]
```

```
-c --stdout      write on standard output, keep original files unchanged
-d --decompress  decompress
-f --force       force overwrite of output file and compress links
-h --help        give this help
-l --list        list compressed file contents
-L --license     display software license
-n --no-name     do not save or restore the original name and time stamp
-N --name       save or restore the original name and time stamp
-q --quiet       suppress all warnings
-r --recursive   operate recursively on directories
-S .suf --suffix .suf      use suffix .suf on compressed files
-t --test        test compressed file integrity
-v --verbose     verbose mode
-V --version     display version number
-l --fast        compress faster
-9 --best        compress better
file...         files to (de)compress. If none given, use standard input.
Report bugs to <bug-gzip@gnu.org>.
```

Some programs also support a `-?` switch, which in most shells must be typed as `- \?`. Other programs will also print help whenever they are passed incorrect arguments.

Miscellaneous documentation

On many systems, miscellaneous documentation will be found under the `/usr/doc` or `/usr/share/doc` directory. For example, the mutt program, a popular email client has an extensive user manual that is neither man pages nor Texinfo documentation. On many systems, it can be found in `/usr/share/doc/mutt/manual.txt.gz`. In this case, the documentation is a compressed text file (as indicated by its extension). It can be viewed easily with **zless**, a version of less that automatically uncompresses compressed files.

Example 11-4. Viewing the mutt user manual.

```
The Mutt E-Mail Client
by Michael Elkins <me@cs.hmc.edu>
version 1.3.28
```

```
"All mail clients suck. This one just sucks less." -me, circa 1995
```

Table of Contents

1. Introduction
 - 1.1 Mutt Home Page
 - 1.2 Mailing Lists
 - 1.3 Software Distribution Sites
 - 1.4 IRC
 - 1.5 USENET
 - 1.6 Copyright
2. Getting Started

- 2.1 Moving Around in Menus
- 2.2 Editing Input Fields
- 2.3 Reading Mail - The Index and Pager
 - 2.3.1 The Message Index
 - 2.3.1.1 Status Flags
 - 2.3.2 The Pager
 - 2.3.3 Threaded Mode
 - 2.3.4 Miscellaneous Functions
- 2.4 Sending Mail
 - 2.4.1 Editing the message header
 - 2.4.2 Using Mutt with PGP
 - 2.4.3 Sending anonymous messages via mixmaster.
- 2.5 Forwarding and Bouncing Mail
- 2.6 Postponing Mail

lines 1-32

The power of Google

The Linux community developed in parallel with the Internet community. As a consequence, Linux information is often easily found on the Internet. Many times, help is as close as a search at google.com for your question and the word "linux." Google also has a "Linux topic search," available at www.google.com/linux.

Chapter 12. Getting GNU/Linux for Yourself

Distributions

Nearly all of the software that comes with a Linux system is free software (<http://www.gnu.org/philosophy/free-sw.html>). You are free to make copies of the programs for yourself. You are free to share copies with others. You are free to change the programs (if you have the skill) and give away the modified versions. Implicit in these statements is that you get the source code to the programs.

It is possible to build an entire Linux system from scratch using the source. However, it requires a great deal of work, more than most people want to do. To make Linux systems easier to install, various groups and companies have pre-built and tested complete systems which can be obtained on CD. These are called distributions.

Although the software is free, many people find it quite reasonable to pay real money for Linux distributions--to obtain the convenience they provide. Many distributions are also available for no cost. Some of the most popular Linux distributions are:

- Slackware Linux (<http://www.slackware.com/>): The grandfather of Linux distributions. If you meet someone who has been using Linux for a long time, they started with this.
- Red Hat Linux (<http://www.redhat.com/>): The most recognized commercial name in Linux. Several ITS servers run Red Hat; including the CourseInfo server, xenon, and gold.
- Debian GNU/Linux (<http://www.debian.org/>): Maintained by hundreds of volunteers, Debian is the “most free” of the Linux distributions. Many MTCS servers run this, including kronos.
- Mandrake Linux (<http://www.mandrakelinux.com/en/>): A commercially produced Linux; from France. It claims a very polished install and relative ease for beginners.
- SuSE Linux (http://www.suse.com/index_us.html): A commercially produced Linux; from Germany. SuSE is very popular in Europe.
- turbolinux (<http://www.turbolinux.com/>): A commercially produced Linux. The most popular Linux in Asia.

Installation

Each distribution of Linux has a custom install program. Some are text-based, some are graphical. All can install from a CD. Some can install from floppy disks or over a network. Despite the differences, all Linux installs must complete the same basic steps:

Loading a kernel

All Linux install programs run under a Linux kernel. Generally this kernel is loaded immediately from a boot CD. In some cases, it can be loaded from floppy disk instead.

Partitioning a hard disk

Every distribution needs space on a hard disk to store its files. Generally this space cannot be shared with other operating systems, like Windows (although often there are ways to read Linux files from Windows and Windows files from Linux). The process of creating this space is called partitioning. For most Linux installs, you will need a minimum of 2 gigabytes of space devoted to Linux partitions.

If your hard drive already has data on it, it will need to be moved to make room for new Linux partitions. Depending on the kinds of partitions already on your hard disk, a free software program like parted (<http://www.gnu.org/software/parted/>) may be able to move and resize the current partitions to make room for new ones. Some partitions are difficult to resize, and may require a commercial program like PartitionMagic (<http://www.powerquest.com/partitionmagic/>). No special software is needed if you intend to delete your old data and go “completely Linux” or if you are willing to re-install your other operating systems.

Installing packages and setting up the system

Linux systems come with lots of software. For example, the current Debian distribution ships with over 8000 packages (ready-to-install programs). Installers vary in the amount of information they require at this step. Some ask for your preferences in great detail, while others ask only a few simple questions.

The installer also takes care of detecting your hardware (like your sound card and video card) and setting up your network. This is another area where installers differ greatly. Some have elaborate automatic detection, and others prompt you to specify everything.

Making the computer boot

Once everything is installed and set up, the installer will re-write the master boot record of your hard disk so that Linux can boot automatically when you start the computer. The Linux bootloaders are smart enough to boot other operating systems as well, so you can still load Windows if it is on your hard drive (a setup like this is referred to as dual-boot).

Keeping current

Like any system on the Internet, Linux systems are subject to attack and exploitation if they are not maintained. Because of this, many distributions provide downloadable security updates. This may be at no cost or for a fee, depending on the distribution. It is probably worth investigating the update mechanism when choosing a distribution. After all, with luck you’ll only install once. You’ll be downloading security updates forever.

Chapter 13. Summary of useful commands

Table 13-1. Commands:

<code>a2ps filename</code>	send <i>filename</i> to the default printer, formatted in an intelligent way
<code>bg</code>	put a job in the background
<code>cal</code>	display a calendar
<code>cat filename</code>	display the contents of the file
<code>cd</code>	change to your home directory
<code>cd pathname</code>	change working directory to <i>pathname</i>
<code>chmod permissions filename</code>	change permissions on a file
<code>cp oldfile newfile</code>	copy <i>oldfile</i> to <i>newfile</i>
<code>cp filename pathname</code>	copy <i>filename</i> to a new location
<code>date</code>	display system's current date and time
<code>df</code>	display disk space information
<code>echo argument</code>	display the argument
<code>emacs filename</code>	edit <i>filename</i> with the emacs editor
<code>file filename</code>	display information about a file
<code>find</code>	search for files with various characteristics
<code>fg</code>	put a job in the foreground
<code>grep "pattern" files</code>	show the lines in <i>files</i> that match <i>pattern</i>
<code>info command</code>	shows the (hypertext) info page of <i>command</i>
<code>jobs</code>	show jobs currently suspended or running on the terminal
<code>kill PID</code>	send a "kill" signal to process with process number <i>PID</i>
<code>less filename</code>	display <i>filename</i> on the screen, one page at a time
<code>lpq filename</code>	examine the queue of the default printer
<code>lpr filename</code>	send <i>filename</i> to the default printer, as plain text
<code>lprm jobnumber</code>	cancel <i>jobnumber</i> , obtained with <code>lpq</code> , from the queue of the default printer
<code>ls</code>	list the names of files (many, many options)
<code>man command</code>	display the manual page of <i>command</i>
<code>mkdir directoryname</code>	make a new directory named <i>directoryname</i>
<code>mv oldfile newfile</code>	rename <i>oldfile</i> to <i>newfile</i>
<code>mv filename pathname</code>	move <i>filename</i> from here to a new location
<code>ps</code>	list processes
<code>pwd</code>	print working directory name

Chapter 13. Summary of useful commands

<code>rm filename</code>	remove (delete) <i>filename</i>
<code>rmdir pathname</code>	remove an empty directory named <i>pathname</i>
<code>su username</code>	assume the identity of <i>username</i>
<code>talk username</code>	instant messaging, 25 years before AOL was invented
<code>vi filename</code>	edit a file with the vi editor
<code>whereis filename</code>	search for a file in standard locations

Appendix A. Advanced: Customizing file associations with `~/mailcap`

In computing environments, we have become accustomed to having various file types automatically associated to programs that can open them. In a Windows environment, that customization is called “file associations” and is typically determined by file extension. We know, for example, when we click on a `.html` file that it will open in a web browser. The same kind of associations exist for unix environments, and are based on MIME types. This is much like the way that helper applications are assigned in a web browser. The file that contains your personal association preferences is called `~/mailcap`

Motivation

Here’s my situation; yours may be similar or different. I generally access my computer at least once a day from home over a modem connection and in my office from X. I read my email in Mutt, the world’s premier mail program. It sucks less than other mail programs.

Because the modem is so slow, I prefer my software to display attachments (say a Word document or PDF file) differently when reading my mail at home than when I am at my office working in a graphical environment. At my office, I want everything to work graphically (and automatically). At home I want “just the text”. Anything more would be way too slow.

Using my `~/mailcap` file, I can have both. I need only a few custom rules. And a few helper scripts.

Helper scripts

To that end, I have created a few helper shell scripts. These have to be placed somewhere in your `PATH` (use `echo $PATH` to see what directories this includes). A typical place to put personal scripts would be `~/bin`, and you may need to create the directory if it doesn’t already exist.

First, for reading Word files as text, I use a script I call **wordcat**:

```
#!/bin/bash

cat $* | strings | fold -s
```

For PDF files, I use a script I call **pdfcat**. It requires a program called **pdftotext** (you probably have it installed with **xpdf**, a common graphical PDF viewer).

```
#!/bin/bash

for i in $*; do
  pdftotext $i - | fold -s
done
```

Essentially, each of these scripts yank the text out of a document, break long lines on a whitespace, and dump the result to stdout. It’s not beautiful, but it is certainly acceptable in the text-only environment of a modem connection.

Editing ~/.mailcap

Lesson 1 - PDF files

To use these scripts with your favorite mail client, web browser, etc. you must edit your ~/.mailcap file. The ~/.mailcap file contains lines that describe how to handle different content-types with helper applications. For example, from my ~/.mailcap file come these lines:

```
application/pdf;gv '%s'; test=test -n "$DISPLAY"  
application/pdf;pdfcat '%s'; copiousoutput
```

The first line says that to view a document of type application/pdf (a PDF attachment) you should use the **gv** program. The %s is a place-marker that contains the name of the pdf-file to view (generally this is some kind of temporary file). The last part of the line gives a test to indicate when this rule applies: if the DISPLAY variable is non-empty (i.e. when working under X11). In plain English, this line means that in graphical environments, **gv** will open PDF files.

If the DISPLAY variable happens to be empty, the first rule fails, and the second rule is tried. The second rule indicates that the file should be processed by **pdfcat**. The final term on the line, copiousoutput, indicates that the output of pdfcat may be a large amount of text and may need to be "paged". For example, when reading a PDF in mutt, this tells mutt to use its built-in pager to view the text.

Lesson 2 - HTML files

The rules for html attachments that I use are:

```
text/html; mozilla '%s'; edit=gvim -f '%s'; test=test -n "$DISPLAY"  
text/html; w3m -T text/html '%s' ; needsterminal  
text/html; w3m -T text/html -dump '%s' ; copiousoutput
```

According to the first line, HTML attachments will be displayed using Mozilla when working in graphical environments. Failing that, the second line invokes the text web browser w3m to view the attachment. Notice the term needsterminal. This indicates that the program requires a console window, xterm, or similar to function (this is typically the case that applies when working over the phone).

The third rule has a special relationship to mutt. The -dump mode of w3m works like a filter to dump the text of a web page to the console (as opposed to browsing it interactively). In this mode, copiousoutput is appropriate. But more importantly, in your ~/.muttrc you can add a line to automatically view HTML attachments. This is particularly convenient for interacting with those who send HTML emails. When auto-viewing, mutt will always choose a copiousoutput rule, even when it is placed later in the ~/.mailcap file. To enable this, add to your ~/.muttrc:

```
auto_view text/html
```

As always in mutt, you have the option of pressing **v** to view the attachments and selecting an HTML attachment for interactive viewing (according to the first or second rule as appropriate). The effect is that you get automatic text rendering of the HTML in your mail (which can be quoted when replying, etc.) and the ability to have a more interactive or graphical rendering if you desire.

If you wish, you can set any mime-types to be auto-viewed. For example, if you turn it on for application/pdf, then the rule above will ensure that the text of PDF attachments is automatically extracted in the mails you read (and you can choose to view them specifically by pressing **v** and selecting the appropriate attachment).

Finishing up

To round out the ~/.mailcap file, I include a few Microsoft Word rules, and a few generic rules:

```
application/msword; abiword '%s'; test=test -n "$DISPLAY"
application/msword; wordcat '%s'; copiousoutput

text/*; gview -Mf '%s' ; edit=gvim -f '%s'; compose=gvim -f '%s'; test=test -n "$DISPLAY"
text/*; view -Mf '%s' ; edit=vim -f '%s'; compose=vim -f '%s'; needsterminal
```

The first two should require no further comment. The last are generic rules that match any text mime-type that doesn't match a more specific rule. The first line specifies that in graphical environments gview/gvim (a graphical Vi program) is used to view/edit/compose as appropriate. Otherwise, text versions of Vi are used.

Other resources

There is a system-wide mailcap file in /etc/mailcap. Your local settings override the settings found there, but it can be a good source of rule ideas. It also shows you what the default behavior of your system is.

There are files that associate filename extensions with mime-types. The system-wide file is /etc/mime.types and you may create your own in ~/.mime.types.

Appendix B. Advanced: Password-less Authentication With SSH

Generating SSH keys

In addition to password authentication, ssh can use an encryption authorization method. Instead of typing a password, you authenticate to the remote computer by means of a special file, that you might think of as a “key”. Done correctly, this can allow you to login to remote machines without requiring you to enter a password every time.

The first step for password authentication is generating a public/private key pair. This is a pair of files that together can be used to encrypt and decrypt data. One file is used for the encryption part of the process, and the other for decryption. Because of the way that these keys were designed, they can be used to authenticate a user to a system.

To begin, you must generate your key pair. The command to do this is **ssh-keygen**. By default, with the `-t dsa` option, this will create two files, `id_dsa` and `id_dsa.pub`. The `id_dsa.pub` file is referred to as the public half of your key, and `id_dsa` is the secret half of your key.

Example B-1. Generating a key pair.

```
$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/caleb/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):passphrase
Enter same passphrase again:passphrase
Your identification has been saved in /home/caleb/.ssh/id_dsa.
Your public key has been saved in /home/caleb/.ssh/id_dsa.pub.
The key fingerprint is:
00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00 caleb@athlonsmp
```

Although leaving the passphrase empty can make it easier to do password-less logins, that is not the best approach. Without a passphrase, anyone that might get a copy of your `id_dsa` file could use it to impersonate you.

Creating an `authorized_keys2` file.

In order to use your new key pair to login on another machine, it is necessary to tell the remote computer which keys should be accepted. You do this by creating an `authorized_keys2` file. In it, place the public half of any key pair that you want to use. Anyone with the secret half of the key will then be able to login to the account without a password.

Example B-2. Making `authorized_keys2`

```
$ sftp username@xenon.truman.edu
Connecting to xenon.truman.edu...
```

```
username@xenon.truman.edu's password:password
sftp> mkdir .ssh
sftp> cd .ssh
sftp> put .ssh/id_dsa.pub authorized_keys2
Uploading .ssh/id_dsa.pub to /home/username/.ssh/authorized_keys2
sftp> exit
```

Using ssh-agent

If the key you created had an empty passphrase (shame on you!) then you can use it now to login to xenon without a password. Simply type: **ssh xenon.truman.edu**.

If you created a key with a passphrase, then you need a mechanism to load the key and keep it ready for you. The program that does this is ssh-agent. To load it, type **eval 'ssh-agent'**. Once the agent is loaded, you can use **ssh-add** to load your keys. You should be prompted for the passphrase for each (secret) key you load. From that point on, your secret keys will be available for use in that window, without a password.

Example B-3. Invoking ssh-agent and loading a key

```
$ eval `ssh-agent`
Agent pid 1884
$ ssh-add
Enter passphrase for /home/2002/username/.ssh/id_dsa: passphrase
$ ssh xenon.truman.edu
```

Killing ssh-agent

When you no longer want your keys to be loaded, you can delete them using **ssh-del**, or you can kill your ssh-agent process. The way to do this is with **eval 'ssh-agent -k'**.

Appendix C. Advanced: Off-site Email Using SSH

The ability to use email from an off-site location, while using your Truman email address, may be somewhat difficult to accomplish if your ISP does not allow email relaying. If you have had trouble setting up email for your Truman account using your ISP's SMTP server, one solution may be to tunnel your email data into the Truman network using SSH and use their mail servers. The script shown below is one that I wrote which allows me to use email from my home over CableONE's cable internet service.

Forwarding ports

Example C-1. email-tunnel.sh

```
#!/bin/bash

# Set up a tunnel
ssh username@xenon.truman.edu \
  -L8143:studentpop3.truman.edu:143 \
  -L8025:mail.truman.edu:25 \
  'sleep 30;exit' &

# Run your favorite email client
evolution &
```

This script does two things. The first is that it forwards two ports on the local computer via xenon to two computers on the Truman network. The second is that it starts an email client to use those ports.

The `-L` option determines the port forwarding. The ports that are forwarded are:

- Local port 8143 is forwarded to port 143 on studentpop3.truman.edu. Port 143 is the standard port for the IMAP protocol. (If you intend to use POP3 to retrieve your email instead, you would likely want to map local port 8110 to port 110 on studentpop3.truman.edu instead.)
- Local port 8025 is forwarded to port 25 on mail.truman.edu. Port 25 is the standard port for delivering email.

Anytime a computer program tries to connect to port 8143 or 8025 on your computer, that data will be forwarded via the ssh tunnel to the appropriate computers on the Truman network.

One neat feature of ssh is that it allows the connection to stay open as long as an application is using the port forwarding, even if the interactive terminal session is over. So, when the script automatically runs the exit command after 30 seconds, the terminal session is over, but I can still transmit and receive data over the forwarded ports.

Configuring your email client

To take advantage of these forwarded ports, you must configure your email client to take advantage of them. In particular, you should customize your program to retrieve your email using IMAP from localhost on port 8143 (or if using POP3 from localhost:8110).

To be able to send mail, set your outgoing mail server (sometimes called your SMTP server) to localhost on port 8025.

Disconnecting the tunnel

As long as the email client is running and using the forwarded ports, the tunnels will remain in place. When you exit your mail client and the ports become idle, the tunnels will automatically be torn down.

Appendix D. GNU General Public License

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

Section 0

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.

Section 1

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.

Section 2

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:

1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
2. 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.
3. 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.

Section 3

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:

1. 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,
2. 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,
3. 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.

Section 4

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.

Section 5

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.

Section 6

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.

Section 7

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.

Section 8

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.

Section 9

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.

Section 10

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 Section 11

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.

Section 12

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 a brief idea of what it does.> Copyright (C) <year> <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) year 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

- & running a task in the background, 34
- .. parent directory, 9
- /etc directory, 27
- /etc/passwd, 31
- /home directory, 27
- /mnt directory, 27
- /tmp directory, 27
- /usr directory, 27
- /usr/doc, 44
- /usr/local directory, 27
- /usr/share/doc, 44
- /var directory, 27
- ~/.bashrc file, 14
- ~/.mailcap file, 51
- ~/.mime.types file, 53
- a2ps, 49
- Accented characters, 38
- authorized_keys2, 55
- bg, 36, 49
- cal, 49
- cat, 8, 49
- cd, 9, 49
- chmod, 11, 11, 49
- cp, 49
- Ctrl-Z, 35
- date, 49
- df, 49
- directories, 2
 - changing with cd, 9
 - creating with mkdir, 9
 - present working directory, 9
 - removing with rmdir, 9
- display number, 37
- drive letters (A:, C:), 2
- echo, 14, 49
- editors
 - emacs, 17
 - nano, 14
 - vi, 15
- emacs, 17, 49
- environment variables, 13
 - DISPLAY, 37, 52
 - EDITOR, 13
 - making permanent in .bashrc file, 14
 - PAGER, 13
 - PATH, 13
 - PRINTER, 13
 - TERM, 13
- export, 13
- fg, 35, 49
- file, 8, 49
- file types, 9
- filesystem, 1
- find, 29, 49
- finger, 30, 31
- fold, 51
- grep, 27, 49
- gv, 52
- id_dsa and id_dsa.pub, 55
- info, 42, 49
- jobs, 35, 49
- kernel, 1
- kill, 49
- last, 30
- less, 8, 49
- locate, 28
- lpq, 49
- lpr, 49
- lprm, 49
- ls, 3, 7, 49
 - common switches, 7
 - sorting output, 8
- man, 41, 49
- Meta key, 38
- mime-types, 52
- mkdir, 9, 49
- Mode switch, 38
- mount, 2
- Multi, 39
- mv, 49
- nano, 14
- partitioning a hard drive, 48
- path
 - absolute path to a file, 2
 - command search path, 13
 - relative path to a file, 2
- PDF files
 - viewing with gv, 52
 - viewing with xpdf, 51
- pdftotext, 51
- permissions
 - executable files, 10
 - file creation mask, 11

- for directories, 11
 - for regular files, 10
 - listed by ls, 9
 - numeric values, 10
 - setting with chmod, 11, 11
- processes, 33
- ps, 33, 49
 - common options, 33
- pwd, 9, 49
- remote use of the X Window System, 37
- rlogin, 21
- rm, 49
- rmdir, 9, 49
- samba, 19
- scp, 23
- sftp, 22
- shares
 - Windows shares, 19
- shell, 3, 13
- smbclient, 19
- smbmount, 21
- smbumount, 21
- ssh, 21
 - port forwarding, 57
 - X11 forwarding, 37
- ssh-add, 56
- ssh-agent, 56, 56
- ssh-del, 56
- ssh-keygen, 55
- startx, 5
- strings, 51
- su, 21, 49
- talk, 49
- telnet, 21
- Texinfo, 42
- umask, 11
- unset, 13
- vi, 15, 49
 - modal editing, 15
 - vim, 16
- w, 30
- web browsers
 - mozilla, 52
 - w3m, 52
- whereis, 28, 49
- which, 28
- who, 30
 - who am i, 31
- X Window System
 - starting with startx, 5
- X11
 - query option, 37
 - forwarding via ssh, 37
- xev, 38
- xmodmap, 38
- xpdf, 51
- zless, 44