An Introduction to Linux/Unix at Denison

Jessen T. Havill

Last revised on January 6, 2006

Table of Contents

1 Introduction

The original UNIX operating system was invented by Dennis Ritchie and Ken Thompson in 1969 on a DEC PDP-7 at Bell Labs. In the years since, development has splintered into several UNIX variants, some proprietary and some open source.[1] A summary of UNIX development is shown in Figure 1 below. The variants that we use in the department are circled. All of these UNIX operating systems are bound by a set of standards called POSIX which ensures that code developed on one POSIX compliant operating system can be recompiled on another.

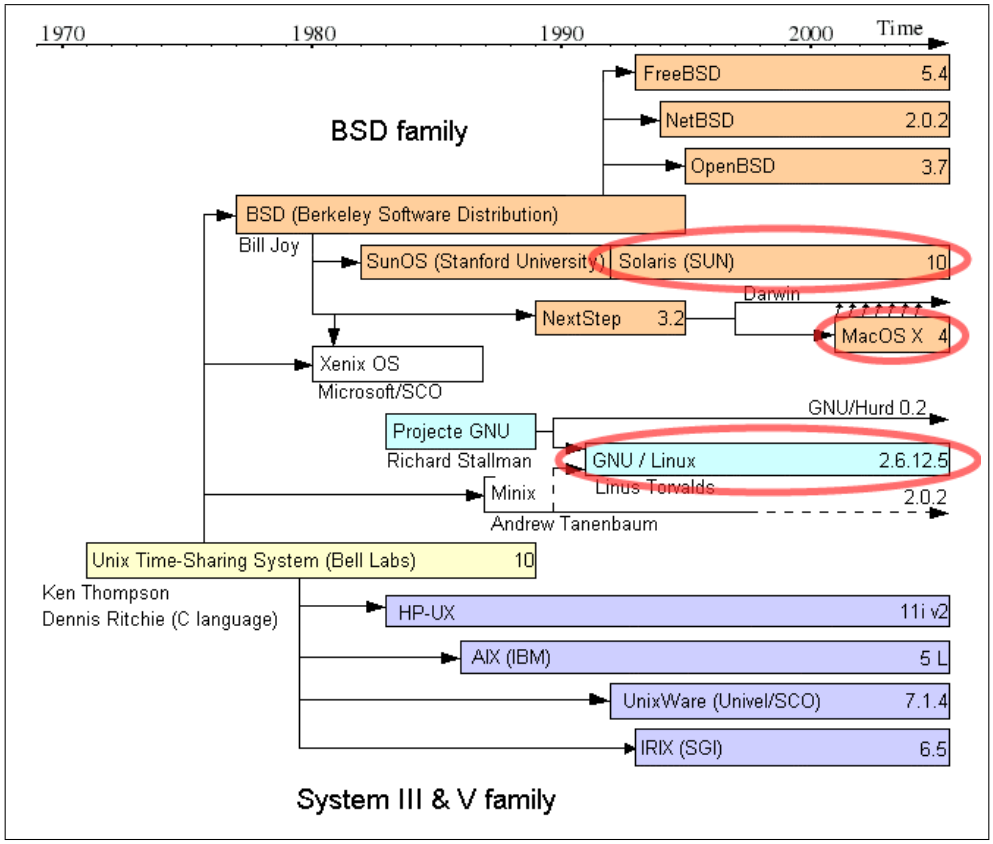

Figure 1: The Unix family.

Linux (pronounced "LĬ-nuks", with a short i) is a UNIX operating system that was born in 1991 as the hobby of a young student, Linus Torvalds, at the University of Helsinki in Finland. Thousands of developers have since joined the Linux project, and Linux has grown into a mature and powerful operating system. Linux is now widely used around the world by individuals and companies as diverse as L.L. Bean, Home Depot, Amazon.com, and Dreamworks SKG. It is supported by big computer companies like IBM, Dell, HP, and Novell.

Linux is developed and distributed under the *GNU General Public License* (GPL). The GNU (pronounced "guh-NEW") project (`www.gnu.org`), from which the GNU General Public License originates, was founded in 1984 with the goal of developing high quality, free software. Most of the software bundled with Linux comes from the GNU project. GNU is a recursive acronym for "GNU's Not UNIX". The name is a reaction to the original proprietary implementations of UNIX, to which the founders objected. In a nutshell, the GPL dictates that the operating system is free of cost and that its source code is freely available for anyone to examine or change.

The purpose of this document is to acquaint you with Linux (and UNIX operating systems in general) and the associated software installed on the machines in our labs. Appendix B contains a lengthy command reference that you can use once you are comfortable with the basics. However, this introduction is not, by any means, meant to be a complete reference for Linux. Rather, this document is meant as a starting point. Several pointers to additional references are given in the text for those who wish to learn more.

[1] The history of UNIX is very interesting. You can find it many places on the web.

Most of the machines in the department run some variant of UNIX.[2] In Olin 219, there are 16 machines running Linux. In Olin 216, there are three iMac G5 computers running Mac OS X and seven computers running Linux (or perhaps Windows XP). In Olin 217, there are 25 dual processor Power Mac G4 machines running Mac OS X. Our file server is a Sun Enterprise 250 running Solaris. Although you will find that the information in this document generally applies to all these machines, it is written specifically for Linux. We will try to maintain a relatively consistent suite of core software on all the UNIX machines in the department. We encourage you to use these new computers and experience these different operating systems.

2 Logging In and Out

If you do not already have a Linux account, your instructor should be arranging for one for you.[3] After you receive an account, you will be able to log into any of the UNIX machines with the same username and password. Each of the machines reads password information from our file server (named `sunshine`). Your personal home directory, in which you can store your personal files, will also be accessible from every machine. Your home directory has the pathname `/home/username`, where `username` is your username. A disk quota of 30 MB has been assigned to your account.

2.1 Logging In

If you sit down at a machine and the screen is black, wiggle the mouse a little. The screen saver will disappear and a login screen will appear. Just type your username in the appropriate box, hit Enter, enter your password, and hit Enter again.

2.2 Logging Out

Later, when you are ready to log out, you can do so from the Panel menu in the bottom left corner of the screen. You might want to check the button in the logout window to remember the current state of your session. Most of the programs you leave running will be started up automatically when you next log in.

Never shut any of the Linux machines off. In the unlikely event that a machine seems to hang for a very long time, tell Tony Silveira (Olin 225; `silveira@denison.edu`) and then move to another machine, rather than reboot. It can be very bad for the machine to be shut down improperly. Since these are networked, multiuser workstations, it is also entirely possible that someone is logged into one remotely. If you shut it off, that person will be logged out suddenly, will probably lose their work, and will not be very happy with you!

3 The Graphical User Interface

After logging in, a graphical user interface or GUI similar to Windows or Mac OS will appear. Basic GUI functionality in Linux is provided by a program called *X Windows*. The actual "look and feel" of the GUI you are using is specified by a program called a *window manager*. In your case, you are probably running a window manager in either the *Gnome* or *KDE* desktop environment. Gnome and KDE essentially run "beneath" the window manager to provide some useful tools and a uniform "look and feel" across different window managers. You can change the "look and feel" by using a different window manager and/or by changing your preferences.

The *Panel* contains several buttons that allow you to quickly perform common tasks. The leftmost button on the panel launches a menu that will allow you to access several useful programs. You will visit this menu often. Most of the other buttons start up specific programs. To learn what a button does, run the mouse pointer over it. Take some time later to browse these buttons and menus. You will discover a wide range of applications that you might find useful. You can also add your own buttons to the panel with the Panel menu. You can launch the Panel menu by pressing the right mouse button on a Panel edge.

X Windows expects a 3 button mouse.[4] The left, middle, and right mouse buttons are sometimes called buttons 1, 2, and 3, respectively. Mouse buttons do different things in different applications. For example, clicked on a Panel edge, the left mouse button collapses the Panel, the middle mouse button allows you to resize the Panel, and the right mouse button launches a Panel menu. If you press the right mouse button on the desktop (also called the *root window*), you will get a menu called a *root menu*. Depending on the window manager, the root menu(s) may give access to programs or utilities, or allow you to focus

[2] You can find more detailed information about these machines in Appendix A.

[3] Please note that this account is different from your Novell account.

[4] On a two button mouse, the middle button is simulated by pressing the left and right buttons simultaneously.

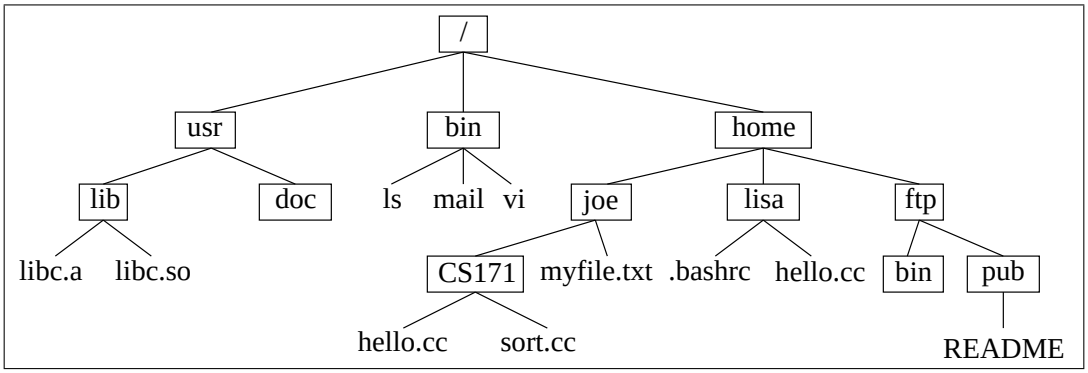

Figure 2: A simple example of a hierarchical file system.

existing windows. Of course, you can also focus, move, resize, iconify, and close windows using the mouse. These operations are generally accomplished the same way as in Windows or Mac OS X. The functionality of specific mouse buttons, however, depends on your Window Manager. Some brief experimentation should clarify things.

Once you are logged in, you will probably see several windows on your desktop; one may look a little like the Windows Explorer and another may be the Help Browser. You can use the former to find and open files much as you might in Windows or Mac OS X. You can use the latter to learn more about the user interface and the many programs bundled with Linux. (If these windows do not appear, they may opened later as needed by clicking on the folder icon labeled "Home Directory" and the button on the Panel with a question mark on it, respectively.)

When you are ready, click the right mouse button on the desktop and select Terminal Window from the root menu. This will bring up a *terminal* window into which you can type Linux commands. The program running in the terminal window that accepts these commands is called a *shell*. The default shell is called `bash` (the GNU Bourne-Again SHell). When you start the shell, you only see a *prompt* like:

```
joe@219f>
```

The prompt indicates that the shell is ready to accept your commands. In this example, the prompt indicates that user `joe` is logged into the computer named `219f`. As you learn more about Linux, you will find that many things can be accomplished either by "pointing and clicking" with the mouse or by typing commands in a shell. While using the mouse is often more convenient, shell commands can give you greater control over an operation. They are also often more convenient for performing some quick file operations. We will cover many shell commands later in this tutorial.

4 The Linux Filesystem

Linux maintains a *hierarchical* file system, similar to what you are used to with Windows or Mac OS. The file system is called hierarchical because directories (or folders) can be created inside other directories to create a hierarchical tree-like structure. Unlike Windows and Mac OS, however, this tree has only one root, called the *root directory*, represented by `/` (a forward slash). A simple example of a hierarchical file system is shown in Figure 2. Each boxed name represents a directory, while the unboxed names are files.

Linux file names are *case sensitive* and may contain almost any character. File names may or may not be followed by an extension like `.txt` or `.cc` or `html`. In fact, the period in a file name is not given any special significance by a shell, and extensions are rarely required for a file to be opened by a particular application. However, it is usually a good idea to include an extension for a file so it is easier for you to figure out what kind of file it is. By convention, executable programs in Linux usually have no extension.

Any directory that is not the root is usually called a *subdirectory*. For example, in Figure 2, `usr` is a subdirectory of `/` and `doc` is a subdirectory of `usr`. The directory `usr` is also called the *parent directory* of `doc` and `/` is the parent directory of `usr`. The root directory is the only directory without a parent; by convention, the root directory is its own parent.

In a Linux filesystem, the `bin` subdirectory contains programs that correspond to core Linux commands. The `usr` subdirectory contains many other parts of the basic Linux system. The `home` subdirectory contains the home directories of all the users with accounts on the system. If your username were `joe`, you could store your files in the `joe` subdirectory of `home`.

The *pathname* of a file contains a sequence of directories to follow to reach the file. For example, the pathname of the `joe`

subdirectory is `/home/joe`. The pathname of the file `myfile.txt` in the `joe` subdirectory is `/home/joe/myfile.txt`. The pathnames above are called *absolute pathnames* because they contain all the information needed to find a file. On the other hand, a *relative pathname* gives the information necessary to find a file from a particular point in the tree. For example, from the directory `/home`, the relative pathname of `myfile.txt` is just `joe/myfile.txt`. Notice that you can tell the difference between an absolute and a relative pathname by looking for the leading forward slash.

4.1 The Working Directory

Relative pathnames are always specified with respect to the *working directory*, which is a special directory remembered by the shell, and is interpreted to be the directory you are currently "in". To see your current working directory in the shell, type

```
pwd Enter
```

in the terminal window. (From now on, we will omit the Enter after commands; you will need to hit this key after every command you enter.) You should see something like

```
joe@219c> pwd
/home/joe
```

to indicate that your are currently in your home directory. (The subdirectory `joe` will be, of course, replaced by your user name.) Whenever you create or refer to a file using a relative pathname, the pathname of the working directory is simply prepended to get the correct absolute pathname.

You can also specify pathnames that go "up" in the tree by using the special symbol "`..`" which refers to the parent directory of the current directory. For example, in Figure 2, if the current working directory were `/bin`, then the relative pathname `../home/lisa/hello.cc` would refer to the file `hello.cc` in the subdirectory `lisa`.

You can use the `cd` command to change your working directory. For example, to change your working directory to the parent directory, type

```
cd ..
```

in the the terminal window. To confirm the change, type `pwd`. You will now see:

```
joe@219c> pwd
/home
```

To change back to your home directory, type

```
cd joe
```

(replacing `joe` with your username, of course). Here is a shortcut: typing `cd` by itself will always return you to your working directory from anywhere.

4.2 Listing Directory Contents

The `ls` command is used to list the contents of a directory. To list the contents of the working directory, type

```
ls
```

by itself. (If this is your first time using Linux, there may be no files to see quite yet.) To list the contents of another directory, either use `cd` to change to that directory and then type `ls`, or specify the name of the directory after the `ls` command. For example,

```
ls /bin
```

will list all the files and subdirectories in the `/bin` directory.

Linux commands can be augmented with *options* that follow a command and are usually prefixed by a - (a dash). For example, type

```
ls -a
```

from your home directory. Now you will notice that more files show up. These are called *dot files* since they all start with a dot (a period). These files contain configuration information for various programs (including the shell; more on this in Section 13) and are normally hidden. The `-a` option (short for *all*) displays these files in addition to the other files in the directory. The first two files that you see (`.` and `..`) are special characters that refer to the directory itself and the parent directory, respectively. While we are on the topic of special characters, it is worthwhile to mention that a tilde character (`~`) always refers to your home directory. So, for example, the command

```
ls ~
```

lists the contents of your home directory.

You may have noticed that some file names were followed by special symbols like `/`, `*`, or `@` in the directory listing. The `/` indicates that the file is actually a subdirectory. The `*` indicates that the file is an executable program. The `@` indicates that the file is actually a symbolic link (or alias) to another file somewhere else in the filesystem. (If you did not see these characters, use the `-F` option with `ls`.)

Now try typing

```
ls -al
```

in the shell. In this case, you are combining two options — the `-a` option and the `-l` option (for *long*). You will see an expanded file listing, something similar to (but not exactly like) this:

```
total 27
drwx------   70 joe      joe          6144 Jul 30 13:02 ./
drwxr-xr-x  102 root     root         2048 Mar 26 09:23 ../
-rw-r--r--    1 joe      joe          1067 Aug 16  1999 .Xdefaults
-rw-r--r--    1 joe      joe          1100 Mar 27 16:27 .bash_profile
-rw-r--r--    1 joe      joe          1625 Jan 12  2001 .bashrc
-rw-r--r--    1 joe      joe          4581 Feb  1 16:17 .emacs
drwxr-xr-x   12 joe      joe          1536 Jul 23 10:31 .gnome/
drwxrwxr-x    2 joe      joe           512 Mar 30 13:04 .gnome-desktop/
drwxr-xr-x    7 joe      joe          1024 Jul 27 17:02 .netscape/
drwxrwxr-x    3 joe      joe           512 Jan  3  2001 .sawfish/
-rwxr--r--    1 joe      joe          1606 Aug 18  1998 .xinitrc*
lrwxrwxrwx    1 joe      root            8 Aug 13  1999 .xsession -> .xinitrc*
```

The first line gives the total amount of disk storage used by the files in this directory, in disk blocks. Each additional line in the display gives information about one file or subdirectory in the directory. The first character in the first column in the file listing tells you what type of file you are looking at. A - means that this is a normal file, a `d` means that this is actually a subdirectory, and a `l` means that this is a symbolic link (more on links in Section 7.6). The rest of the characters in the first column give information about file permissions. We will talk about this more in Section 8.2. The second column gives the number of hard links to this file or directory (again, more in Section 7.6). The third columns tells who the *owner* of the file or directory is. Every file in a Linux filesystem has an owner who can decide who has access to that file. In this case, `joe` owns every file in his home directory except the parent directory. The fourth column indicates the *group* of each file or directory. Groups can be used to give access to a file to a group of specific users. The fifth column gives the size of the file or directory in bytes. The sixth column gives the time the file was last modified. Finally, the seventh column gives the name of the file or directory.

4.3 Wildcard Characters

Sometimes it is convenient to specify a group of files or directories as the argument to a command. If the names of these files are similar in some way, wildcard characters may be used. There are two main wildcard characters: a `*` can be substituted for any string of 0 or more characters and a `?` can be substituted for exactly one character. For example, if you wanted to list all files in the working directory that start with `prog`, you could type

```
ls prog*
```

in the shell. If you wanted all the files with a `.cc` extension, you could type

```
ls *.cc
```

in the shell. If you wanted all the files with names like `prog1.cc`, `prog2.cc`, etc., but not `prog.cc`, you could type

```
ls prog?.cc
```

in the shell.

Wildcard characters can be used anywhere you would normally use a file or directory name.

4.4 Multiple Disks in the Filesystem

In Windows and Mac OS, different disks maintain their own filesystems (directory trees) and each is named with a letter like `C:` (Windows/DOS) or a name like `Hard Drive` (Macintosh). So how are multiple disks accessed in the Linux filesystem with only one root? The answer is that each is *mounted* at a separate directory in the filesystem. So, for example, a separate disk may be mounted at the directory `/home`. In this case, every time you access something in directory `/home`, you are actually accessing a file on this separate disk. This is transparent to the user, but you can peek at which disks are mounted where on a computer by using the `df` command in the shell. You will see something like this:

```
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda8               497667     78511    393454  17% /
/dev/hda1                14607      2758     11095  20% /boot
/dev/hda7              5308207   1085340   3948063  22% /usr
sunshine:/export/home.linux
                       8357259   7024141    497393  94% /home
sunshine:/export/local.linux
                       8357259   7024141    497393  94% /usr/local
/dev/hdc                609750    609750         0 100% /mnt/cdrom
```

The first line indicates that the disk with symbolic name[5] `/dev/hda8` is mounted at the root directory. The second line indicates that another disk with symbolic name `/dev/hda1` is mounted at subdirectory `/boot`. The third line is similar. The disk in the fourth line is actually a subdirectory on a disk on another networked computer (`sunshine`). The `/home` directory on every computer in the lab points to this same directory on `sunshine`. This is how you are able to access your home directory (which is actually on `sunshine`) the same way from every computer! The protocol that allows this seamless mounting of remote disks is called the *Network File System* (NFS). The fifth line is similar to the fourth, and the sixth line represents the CD-ROM drive, which is mounted at `/mnt/cdrom`.

5 Viewing Files

5.1 Text Files

The easiest way to view the contents of a file is by using the `cat` command. The name of the command sounds odd at first, but it will make more sense when we revisit it in Section 7.7. For example, to view the file named `/etc/passwd`, simply type

```
cat /etc/passwd
```

in the shell. (This file contains some encrypted passwords for this machine.) The `cat` command is really only good for looking at small files, since the entire file is displayed without stopping. To view longer files, it is better to use the `more` command. For example, try to view the file again by typing

```
more /etc/passwd
```

in the shell. Notice that, this time, the command stops after one page is displayed. To view the next line, hit Enter. To view the next page, hit the space bar. To move backward one page, type b. To exit, type q. If you want to search for something while in `more`, type / followed by what you want to find and then hit Enter.

[5] This symbolic name looks a lot like a pathname because it actually is one! As you learn more about Linux, you will find that virtually everything is a file. In this case, the file `hda8` in the `/dev` directory is a virtual file that actually refers to a partition on a hard disk.

5.2 Other File Types

Files other than text files can viewed by using an appropriate program, as indicated in the table below. Follow the command with the name of the file to view it or omit the file name and select the file from the menu once the program is running.

File Type	Command
Postscript	`gs`
Portable Document Format (PDF)	`xpdf`
Image files	`gimp`
LaTeX ouput (DVI)	`kdvi`
Hypertext (HTML)	`mozilla`
Multimedia files	`xmms`

6 Online Help

The commands we have covered so far are just a tiny portion of those available. So how do you find out about these commands and remember all those you have learned? You don't; you use the online manuals instead!

6.1 Online Manual

The online manual can be invoked with the `man` command. For instance, to find more information about the `ls` command, type

```
man ls
```

from a shell. The manual page is displayed using `more`, so hit the space bar to see each page. If you want to find out more about `man` itself, type `man man`. There are manual pages on most Linux shell commands *and* on most standard C/C++ functions and Linux system calls.

6.2 Apropos

If you're looking for how to do something, but don't know the command yet, try the `apropos` command. For example, if you want to display a new clock window, try typing `apropos clock`. A list of matching manual sections will be listed. There is an stunning amount of software installed on these machines. Exploring is the only way to found out about it.

6.3 Info

The manual pages are slowly being replaced by a program called texinfo, so you might try this if a man page does not exist. The command to display these so-called "Info" pages on a particular command `c` is:

```
info c
```

Navigation through these info pages is confusing at first, but it basically works like Emacs (Section **??**), except that there are links you can follow in the text. Consult the info page on `info` for more "info" by typing `info info`. Typing h when you enter `info` will start an introduction.

6.4 The Gnome Help Browser

You can also access the manual pages and info pages through the Gnome Help Browser. If you are sitting at the console of a machine, this option is preferable to the command line alternatives since navigation is all via point-and-click. The Gnome Help Browser can be accessed by pressing the Panel button with a question mark on it (or with `gnome-help-browser` in the shell).

If the online help functions do not answer your question, you may want to consult one of the Linux books kept in the department library (Olin 213) or a book in the bibliography at the end of this document. If you are really stuck, ask Tony Silveira (`silveira@denision.edu`).

7 Manipulating and Processing Files

7.1 Copying Files

To make a copy of an existing file, use the `cp` command. For example, to make a copy of a file `myfile.txt` called `myfile2.txt` in the working directory, you would type:

```
cp myfile.txt myfile2.txt
```

If you want to create a copy of `myfile.txt` in a `cs100` subdirectory, type:

```
cp myfile.txt cs100
```

Notice that, if the destination file name is missing, the copy will have the same name as the original.

7.2 Moving and Renaming Files

The `mv` command is similar to the `cp` command except that the original file is not preserved. Instead, the file is "moved" to the destination location. For example, if you had typed

```
mv myfile.txt cs100
```

above instead of `cp myfile.txt cs100`, the file would have been moved to the subdirectory rather than copied.

The `mv` command is also used to rename files. For example, typing

```
mv myfile2.txt myfile3.txt
```

will rename `myfile2.txt` to `myfile3.txt`.

7.3 Removing Files

To remove (delete) a file or group of files, use the `rm` command. For example,

```
rm myfile3.txt
```

deletes the file `myfile3.txt`. The command

```
rm *
```

will remove all the files in the current directory. (*Be careful* using `rm` with wildcard characters!)

7.4 Creating Subdirectories

In order to organize all of your files, you will want to create subdirectories in your home directory. To create a new subdirectory, use the `mkdir` command. For example, to create a new subdirectory named `cs100` in your home directory, change to your home directory and type

```
mkdir cs100
```

in the shell. If you now type `ls`, you will see your new subdirectory.

7.5 Removing Subdirectories

To delete the subdirectory `cs100`, type:

```
rmdir cs100
```

To remove (delete) a subdirectory, the subdirectory must be empty. To delete files in a directory, you will need to use the `rm` command (Section 7.3).

The `rm -R` command can be used to remove a non-empty directory. The `-R` option is short for "recursive". This command deletes all the files in the directory and then the directory itself.

7.6 Symbolic Links

A symbolic (or soft) link is an indirect reference to a file, similar to an "alias" in Mac OS or a "shortcut" in Windows. Suppose you use a particular file often, but it is deeply nested in a subdirectory 4 levels down from your home directory. Rather than typing the full pathname of this file (or changing to that subdirectory) every time you want to access it, you can create a symbolic link to it in your home directory (or somewhere else handy). Then whenever you want to access that file, you can just access the symbolic link instead of the file itself. For example, suppose this commonly accessed file is called `~/personal/misc/letters/contacts`. To create a symbolic link to this file called `buddies` in your home directory, you would type:

```
ln -s ~/my/misc/stuff/friends ~/buddies
```

Typing `ls -l` in your home directory would now yield an entry like the following for your symbolic link:

```
srw-r--r--  1 joe   joe    40 Aug  4 11:01 buddies -> /home/joe/my/misc/stuff/friends
```

Referring to this symbolic link (with `cat`, for instance) will always refer to the real file `/home/joe/my/misc/stuff/friends`. You can also use symbolic links with directories.

Linux actually supports two different kinds of links: *hard* and *symbolic* (or *soft*). A hard link is a direct link to a lower level operating system maintained entry for a file on disk, and is less flexible than a symbolic link. Using the `ln` command without the `-s` option creates a hard link instead of a symbolic link.

7.7 I/O Redirection and Pipes

Many commands read their input from *standard input* and/or write their output to *standard output*. The former is the keyboard by default and the latter is the terminal window, by default. To redirect output to a file instead, you can use the > symbol. For example,

```
ls > ls-output
```

redirects the output of the `ls` command to the file `ls-output` instead of to the console. The symbol >> works like >, except that it appends the output to the end of an existing file. To redirect input from a file, use the < symbol. For example,

```
cat < input
```

reads the contents of the file named `input` are prints it to the terminal window. These symbols can be used together. For example,

```
cat < input > output
```

reads the contents of the file `input` and writes it to the file `output`.

Pipes are a more powerful way to redirect input and output. If you set up a pipe between two programs, the output of the first program is used as the input to the second program. The second program is called a *filter*. For example,

```
ls -l | grep foo
```

performs a directory listing and then pipes this listing to the `grep` command, which prints out only those lines that contain `foo`. This combination is useful if you are looking for the process ID of a particular process you need to kill (see Section 9). If you expect there to be quite a few lines containing `foo` from the last command, you may want to further pipe the results to the `more` command to display one page at a time. To do this, type:

```
ls -l | grep foo | more
```

7.8 Printing

You can print a file either by selecting a menu option in a running application, or by issuing the `lpr` command from a shell. To print a file named `file.cc` using `lpr`, type:

```
lpr file.cc
```

By default, you will print to the printer in Olin 219. If you want to print to another printer, say the one in Olin 216, use:

```
lpr -Ppr216 file.cc
```

The `-P` option, followed by the name of a printer, changes the printer to which to print. The printers in Olin 216, 217, and 219 are named `pr216`, `pr217`, and `pr219`, respectively.

To look at the current print queue, use the `lpq` command. The `-P` option can also be used with `lpq` to specify the printer. For example, to see the current entries in the queue for the printer in Olin 216, type

```
lpq -Ppr216
```

(or just `lpq` if `pr216` is your default printer). This command will display some information about the printer followed by a list of print requests like the following:

```
 Rank   Owner/ID                  Class Job Files                 Size Time
active lisa@219f+4                 A     4 proj4.cc              4289 14:39:38
2      joe@219c+7                  A     7 myfile.txt            1501 14:40:08
```

This display shows that `joe`'s print job is second in line behind one sent by the user `lisa` from the machine `219f`.

To remove a job from the print queue that you sent in error, use the `lprm` command followed by the print job number given by `lpq`. For example, to remove your print job above, you would type:

```
lprm -Ppr216 7
```

7.9 Searching for Files

7.9.1 Searching by File Name

If you need to access a specific file in the filesystem and you do not know where it is located, you can use the `find` command to find it. For example, the command

```
find .  -name paper.txt -print
```

will search for the file `paper.txt` starting in the current working directory (`.`). The command

```
find /bin -name ma* -print
```

will search for all files that begin with the characters `ma`, starting in the directory `/bin`. In each case, the absolute pathname of each matching file is printed. Although the syntax of the `find` command looks strange, it gets the job done. There are many other ways to search for files with `find`. Consult the online manual (Section 6) for more information.

7.9.2 Searching by Content

Sometimes it is useful to be able to search for a file based on the file's content instead of its name. The `grep` command can be used to do just this. For example, the command

```
grep example ./*
```

will search all files in the current directory for the string `example`. You can also use wildcards (and, in general, regular expressions) with `grep`. For example, the command

```
grep ex*le ./*
```

will search all files in the current directory for strings starting with `ex` and ending with `le`.

7.10 Working with Floppy Disks

You can read and write DOS/Windows formatted floppy disks in Linux. There is a suite of utilities for just this purpose called MTools. This utilities allow you to work with a floppy disk the same way you would from MSDOS. In these commands, you refer to the floppy disk as the `A:` drive. Here are the 4 most useful commands:

`mdir`	directory listing
`mcd [dir name]`	change directory or display current directory
`mcopy source dest`	copy a file from `source` to `dest`
`mdel file`	delete a file named `file`

For example, to copy a file named `A:\CS\PROGRAM.CPP` from a floppy disk to the current directory on the Linux machine, type:

```
mcopy A:/CS/PROGRAM.CPP .
```

Notice the use of the forward slashes instead of backslashes. To copy the same file from the current directory on the Linux machine to the current directory on the floppy disk, type

```
mcopy program.cpp A:
```

8 Security

8.1 Changing your Password

It is always a good idea to change your password periodically. To do so, use the `passwd` command. When you issue the `passwd` command, you will prompted for both your old password and your new password.

When you choose a new password, it is important to choose something that is easy for you to remember, yet hard for others (including automatic "cracker" software) to guess. Use a combination of lower and upper case letters, punctuation, and digits in your password.

8.2 File Permissions

Recall that when you issued the `ls -al` command earlier, you were presented with an expanded file listing with a line like the following for each file:

```
-rwxr--r--    1 joe      joe          1606 Aug 18  1998 .xinitrc*
```

Recall that the first character in the first column (−) indicates that the file is a normal file. The next 9 characters indicate the *access permissions* for the file. The 9 characters are divided into 3 groups of 3. Each group indicates whether read (r), write (w), and execute (x) permission is granted to a particular group of users. The first group, corresponding to the first 3 characters, is the "user" (or file owner), the second group is the file "group", and the third group is "other", which consists of all users on the system. So the line above indicates that the owner has read, write, and execute permission for the file and everyone else has only read permission.

When the file is a normal file, the meaning of read and write permissions should be obvious. Execute permission means that the file is executable (either a binary executable or a shell script).

When the file is actually a directory, read and write permissions refer to the ability to read the contents of the directory and create new entries in the directory. Executable permission means that one can search in the directory (or list its contents with `ls`), but not read from or write to it.

The `chmod` command is used to change the access permissions of a file or a directory. There are two ways to specify permissions for a file with `chmod`. The first is to use a *symbolic mode* representation of the permissions. In this mode, you use some combination of `u` for user, `g` for group, `o` for other, or `a` for all to specify which of these group(s) of users' permissions to modify. Then use + to add a permission or − to take away a permission. Lastly, specify the permission(s) to add/subtract: `r` for read, `w` for write, or `x` for execute. For example, suppose the current working directory contained the following files:

```
drwxr-xr-x    1 joe      joe           512 Jul 23 16:56 doc
-rwxr-xr-x    1 joe      joe         23034 Aug  4 11:03 prog1
-rw-r--r--    1 joe      joe          1501 Aug  4 11:01 prog1.cc
```

Then the command

```
chmod g+w *
```

gives write permission to the group for all files in the current directory. The permissions of the files would then look like:

```
drwxrwxr-x    1 joe      joe           512 Jul 23 16:56 doc
-rwxrwxr-x    1 joe      joe         23034 Aug  4 11:03 prog1
-rw-rw-r--    1 joe      joe          1501 Aug  4 11:01 prog1.cc
```

If you then issued the

```
chmod a-rw prog1*
```

command, this would prevent anyone from reading or writing `prog1` and `prog1.cc`. The resulting directory listing would look like:

```
drwxrwxr-x    1 joe      joe           512 Jul 23 16:56 doc
---x--x--x    1 joe      joe         23034 Aug  4 11:03 prog1
----------    1 joe      joe          1501 Aug  4 11:01 prog1.cc
```

It is important to restrict access to your home directory. The command

```
chmod go-rwx ~
```

takes all permissions for your home directory away from everyone but the user. This is how the permissions of your home directory should be set. If they are not, type this command now.

The second way to use the `chmod` command is to use *octal mode*. Think of the `rwx` permissions for a group (`u`, `g`, or `o`) as a representation of a 3 bit binary number (0 – 7 in decimal). If a permission is set, that bit is a 1; otherwise it is a 0. Use the 3 octal (or decimal) digits for the 3 groups as the permission. For example, `r-x` is equivalent to 4+1=5 and `-w-` is equivalent to 2. Taken as a group, `rw-r--r--` is equivalent to `644`. If you wanted to grant these permissions to the file `prog1.cc`, you would type:

```
chmod 644 prog1.cc
```

To give yourself all permissions to your home directory, but everyone else none, you would type:

```
chmod 700 ~
```

9 Processes

Linux, like most modern operating system, is *multitasking*, which means that it can execute many programs simultaneously. A program that is currently executing is called a *process*. A Linux system consists of several dozen active processes at any time. Some of these processes are *system processes* that perform important "behind the scenes" tasks and some are *user processes* corresponding to programs like Netscape or Emacs.

9.1 Listing Processes

You can view the processes that are currently running on the system with the `ps` command. Type

```
ps
```

now and you will see something like the following:

```
  PID TTY          TIME CMD
23985 pts/0    00:00:01 bash
24001 pts/0    00:00:00 ps
```

This tells you that you are running 2 processes and the commands (CMD) used to start them were bash (your shell) and ps (the process you just started when you entered ps). The first column in the list, headed by PID, gives the *process ID* for each process, an integer used to uniquely identify each process. The next column, TTY, gives the terminal name to which the process is attached. The third column, TIME, gives the amount of time the process has been running for (or the amount of time it has been running since it was last sleeping or stopped).

Using ps by itself only gives information about processes that you own, which, unless you are a system administrator, is usually all the information you need. If you want information about all the processes running on the system, you can issue the

```
ps aux
```

command. This command combines 3 options (the normal dash before options is not necessary) which, together, give information about all processes that have users have attached to them. Here is a (significantly trimmed) example of what this output might look like:

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1324    76 ?        S    Jul23   0:12 init [5]
root         2  0.0  0.0      0     0 ?        SW   Jul23   0:21 [kflushd]
root         3  0.0  0.0      0     0 ?        SW   Jul23   0:03 [kupdate]
root         4  0.0  0.0      0     0 ?        SW   Jul23   0:00 [kpiod]
root         5  0.0  0.0      0     0 ?        SW   Jul23   0:14 [kswapd]
root         6  0.0  0.0      0     0 ?        SW<  Jul23   0:00 [mdrecoveryd]
root       415  0.0  0.1   1548   160 ?        S    Jul23   0:02 syslogd -m 0
rpc        440  0.0  0.1   1472   252 ?        S    Jul23   0:00 portmap
root       456  0.0  0.0      0     0 ?        SW   Jul23   0:00 [lockd]
root       500  0.0  0.2   5744   272 ?        S    Jul23   0:00 [ypbind]
nobody     566  0.0  0.0   7616    24 ?        S    Jul23   0:00 [identd]
                                                .
                                                .
                                                .
root      1006 14.7 24.7 62796 31612 ?        R    Jul23  53:50 /etc/X11/X
joe       7892  0.0  1.2  7580 1564 ?        S    14:15   0:00 gnome-session
joe       7957  0.0  1.3  4700 1700 ?        S    14:15   1:39 sawmill
joe       7966  0.0  2.1  9336 2784 ?        S    14:15   3:51 panel
joe       7969  0.0  0.8  8404 1092 ?        S    14:15   0:00 gmc
joe       7972  0.0  0.7  3560  968 ?        S    14:15   0:11 xscreensaver
joe       7992  0.0  1.3  7676 1676 ?        S    14:26   0:21 gnome-terminal
joe       7994  0.0  0.6  2440  796 pts/0    S    14:26   0:01 bash
joe      12504  0.0  0.1  2004  156 pts/0    S    15:08   0:00 netscape
joe      24059  0.0  0.7  2728  900 pts/0    R    15:42   0:00 ps aux
```

This listing gives a lot of information about the 100 or so processes that are probably currently executing on the system. For example, you can gather that the computer was rebooted last on July 23. You can see that X windows (/etc/X11/X) is using almost 25% of the computer's memory at the moment and 15% of the CPU time. You can also view the current status (STAT) of each of the processes in the system. An R in this column means that the processing is currently "runnable" which, as you will learn when you take Operating Systems, means that the process is either currently running or in the ready queue, waiting to gain access to the processor. An S indicates that the process is "sleeping" (currently not waiting to use the processor). A W means that, in addition to sleeping, the process has been "swapped out" (not currently resident in memory but on the disk instead). Lastly, the < marks a process that is given higher priority than normal while waiting for the CPU.

9.2 Ending a Process

Sometimes you will come across a process that locks up and just won't quit, no matter what you do. In this case, the only alternative is to force the process to quit. If the process is one that you started from the shell in the foreground, you can end it by typing Ctrl-C. If the process is running in the background, you will need to issue the kill command followed by a process ID. For example, to kill the netscape process above, you would type:

```
kill 12504
```

If this does not work, use the `-9` (or `-s SIGKILL`) option to forcibly kill the process.

The `kill` command (or Ctrl-C) also comes in handy when you must kill a program that you wrote with an infinite loop!

9.3 Running a Process in the Background

When you run a program in Linux, it can either be run in the *foreground* or in the *background*. When you run a program in the foreground (which is the default), the resulting process must finish before you get another prompt in the shell. This is fine when you run short processes like `ls`, for example. However, if you want to run Netscape (for example) from the shell, you probably do not want to have to quit Netscape in order to get a new prompt to, say, copy a file. The alternative is to run Netscape in the background, which means that the new process will run concurrently with the shell prompt, and a prompt will be returned to you immediately. To run a program in the background, simply follow the command with an ampersand (`&`) character. For example, to run Netscape in the background, type:

```
netscape &
```

Generally speaking, any time you run a program that generates its own window on the desktop, it is best to run it in the background.

If you start a program in the foreground by mistake and want to turn it into a background process, hit Ctrl-Z, which suspends the process. Then type

```
bg
```

which restarts the currently suspended process in the background.

10 Creating and Editing Text Files

To write programs, you will need to use a *text editor*. A text editor is a program that edits text files and is different from a *word processor* in that the resulting files do not contain formatting information of any kind.

There is a wide range of text editors available. Below we list a few of these. See the `man` pages or the application's help function for more information on each one. (Many students currently prefer `J` or `TextWrangler` (on Mac OS X).)

- **Command line text editors** (work in a terminal window):

  `vi` and `vim`, `pico`, `emacs` (more information in Appendix C)

- **Basic text editors**:

  `gedit`, `kedit`

- **Code-oriented text editors** (with syntax highlighting, indentation, etc.):

  `J`, `TextWrangler` (Mac OS X only), `jedit`, `nedit`, `kwrite`, `xemacs` (more information in Appendix C)

Emacs is an old, but very powerful text editor that we used to make everyone learn. However, many newer, easier to use editors have arisen in recent years that students tend to prefer. We still think emacs is worth learning, and so we have retained the emacs tutorial in Appendix C. We encourage you to take go there if you are interested in learning more.

11 Compiling Programs

The standard UNIX compilers are the GNU compilers, distributed as part of the GNU project. The C++ compiler is called `g++` and the C compiler is called `gcc`. You can learn more about `g++` later by looking at the man pages. (There's a lot there!) For now, we'll just look at the basics.

11.1 Compiling From the Shell

To compile a C++ program named `sort.cc` in the shell, simply type

```
g++ -o sort sort.cc
```

in the terminal window. (The `.cc` extension is the one most commonly used for C++ programs in Linux. You can also use `.cpp` extension, if you prefer.) The `-o` option gives the name of the executable to create; if omitted, the executable will be called `a.out`.

In general you'll probably want to compile programs with more options specified. Here is a better way to compile a program:

```
g++ -g -Wall -o sort sort.cc
```

The `-g` option produces debugging information in the executable, for use with a debugger such as DDD (more in Section 11.3). The `-Wall` option instructs the compile to display "all warnings". Be sure to take warnings seriously, as they may point to a looming problem.

To compile multiple source files into one executable, just include them all on the command line. For example, to compile `sort.cc` and `swap.cc` into a single executable called `sort`, type:

```
g++ -g Wall -o sort sort.cc swap.cc
```

Once you have a working executable program, go to a terminal window to execute it. Make sure you are in the correct directory and then type the name of the executable program.

You can also compile and debug programs directly from within emacs. For more information, see Appendix C.

11.2 Using Make

When you are creating large software projects, it is usually a good idea to break the code into multiple source files. This way, errors are usually easier to find and edit, and when you recompile the program you only need to recompile the file you changed and those upon which it depends. Of course, this can become confusing and complicated when the system becomes large.

Linux has a powerful tool called `make` that allows you to efficiently manage a complex project and compile it efficiently. The `make` utility reads a file called a makefile that describes the files involved in the project and the dependencies between them. Each line in a makefile is of the form:

```
targets:  dependencies
<Tab> commands
```

Here, `targets` is a list of target files separated by spaces and `dependencies` is a list of files on which the `targets` depend. In other words, if any of the files in a dependency list is modified, `make` should recompile and/or relink the target. The `commands` are the commands used to recreate the targets. (Note that the `<Tab>` preceding `commands` really must be a tab character and not several spaces.)

For example, consider the following simple makefile:

```
# A simple makefile

sort: sort.o list.o
     g++ sort.o list.o -o sort

sort.o: sort.cc sort.h
     g++ -g -c sort.cc

list.o: list.cc list.h
     g++ -g -c list.cc
```

This file states that the executable `sort` is dependent on the object files `sort.o` and `list.o`. If either of these object files change, the command `g++ sort.o list.o -o sort` should be used to recreate (link) `sort` from the object files. The next two groups show how to create the object files. The first states that `sort.o` should be recreated (compiled using `g++`) if `sort.cc` or `sort.h` changes. (The `-c` option to `g++` tells the compiler to only compile, creating object files, and not to link

them together.) The third group does a similar thing for `list.o`. So, you can see that if, for example, `list.h` is modified, `list.o` and then `sort` will need to be recreated but `sort.o` will not.

To use this makefile, it should be placed in the same directory as the source files and named `Makefile`. Then, execute

```
make sort
```

to compile and link everything.

Here is a more complicated example of a makefile:

```
# Makefile for compiling a network sender project

CC = g++
LD = g++

INCDIR = -I../
CFLAGS = -g -c $(INCDIR)
LDFLAGS =

all: sender

sender: sender.o packet.o ack.o
	$(LD) $(LDFLAGS) sender.o packet.o ack.o -o sender

sender.o: sender.cc packet.cc ack.cc proj.h packet.h ack.h
	$(CC) $(CFLAGS) sender.cc

packet.o: packet.cc ack.cc packet.h ack.h
	$(CC) $(CFLAGS) packet.cc

ack.o: ack.cc ack.h
	$(CC) $(CFLAGS) ack.cc

clean:
	rm *.o
```

In this makefile, you see the use of *macros* like `CC` (the name of the compiler) and `LD` (the name of the linker). Macros can sometimes simplify a makefile and make it easier to modify. For example, if we wanted to change the compiler we could just change the definition of the `CC` macro instead of changing each and every command list. The `INCDIR`, `CFLAGS`, and `LDFLAGS` macros contain an option giving additional directories in which to look for header files, compiler options, and linker options, respectively.

The `all` target can be followed by the default target(s) you want to create if no target is specified on the command line. In this case, if `make` is given on the command line by itself, the target `sender` will be created. The `clean` target at the end can be used to remove unneeded object files from the directory.

11.3 Debugging

The standard Linux debugger is called GDB (GNU DeBugger). It is a command line debugger, which means that you can step through your program, add breakpoints, display variable values, etc. by typing in commands at a prompt. Normally, however, it is more convenient to be able to use a GUI debugger to debug your code. There are a couple of GUI debuggers available for Linux, both of which are graphical front ends to GDB. XXGDB (`xxgdb`) is a relatively simple front end, while GNU DDD (Data Display Debugger) has a more sophisticated interface that, among other things, incorporates an interactive graphical data display feature, where data structures are displayed as graphs.

You can start up DDD by typing `ddd` in the shell, followed by the name of your executable program. In order for the source program to be visible to the debugger, you had to have compiled the program with the `-g` option, which includes debugging information in the executable. To simply run your program in DDD, press the Run button in the small floating window that

appears.[6] The output of your program will be displayed in the window below your source code. The Interrupt button will stop a running program, the same way a Ctrl-C does in the shell. If you want to continue to run your program from where you left off, press the Cont button.

Usually, when you are debugging a program, you want the program to stop execution at some point so that you can examine the value of a variable or check some other part of your program. To stop your program, you set a *breakpoint*. To set a breakpoint, find the line of your source code where you wish to stop and click to the left of it with the left mouse button. Then press the Break button (with the stop sign) below the menu bar. A stop sign icon should appear to the left of the line. Now, when you run your program, execution will stop every time this line is reached. You can set as many breakpoints as needed. To clear a breakpoint, do the same thing, except this time the stop sign will be dimmed and the button will say Clear instead.

If you wish to have your program stop every time the value of some variable changes, you can set a *watchpoint*. To set a watchpoint, left-click on a variable name in your source code and press the Watch button below the menu bar.

Once your program has stopped, there are two options (buttons) for stepping through your program a line at a time — Step and Next. To step means to proceed to the next line of source code in your program, even if it is inside a function. The Next option advances to the next source line, but treats function calls as if they were single instructions. The Stepi and Nexti buttons behave similarly, but recognize individual machine language instructions, not lines of source code.

To view the value of a variable as your program runs, select that variable name by left-clicking on it and then press the Display button below the menu bar. A box with that variable's value will appear in a new window above your source code.

These are the basics of DDD. But there is quite a bit more to learn if you desire. The display function of DDD, in particular, is quite sophisticated and flexible. Under the Help menu, you can find context sensitive help and a detailed manual (also available at `www.gnu.org/manual/ddd/`). You will also notice that if you linger over a button with the cursor, a description of that button will appear.

12 Network Applications

Linux was designed from the start to be a net-centric operating system. For the most part, network resources can be used in Linux seamlessly. Many times, you do not even realize you are using the network. We saw two examples of this earlier: remotely stored passwords with NIS and accessing your home directory via NFS.

Linux is complete with all the standard network tools including Mozilla, sftp, ssh, etc.

12.1 Web

To use Mozilla, just type `mozilla` on the command line or push the mozilla button.

12.2 Remote Login and FTP

You can remotely log in to any Linux computer on our network by using `ssh` (secure shell). `Ssh` sets up a secure encrypted connection between two machines. For example, use

```
ssh 219a
```

to securely log in to `219a`. You may see a message like the following:

```
The authenticity of host '219a (140.141.132.78)' can't be established.
DSA key fingerprint is 70:fa:15:6f:9f:da:2a:a8:b3:5d:61:8b:56:3a:c0:a9.
Are you sure you want to continue connecting (yes/no)?
```

Simply answer `yes` and you will be prompted for your password. To end your `ssh` session, type `exit`.

To transfer files from one machine to another, use the similar `sftp` program. You can use `cd` and `ls` to change and view the contents of directories. Then use the `get` command to retrieve a file or `put` to send a file.

[6] If you need to supply command line arguments to your program, select the Run option from the Program menu (or press F2).

12.3 Remote X Windows

Beyond text-based remote login, X Windows actually allows you to see GUI applications that you are running on another computer on the display in front of you. This is very useful if you are working on a machine (say, in your room) that does not have a particular application installed, but it is installed on another computer to which you have access (say, in an Olin lab). It may also be convenient to use this capability even if you have the application locally but you need to access a file in your home directory which is only available from machines in the labs.

X Windows is an *event-driven* system, meaning that X Windows intercepts *events* like mouse clicks and key presses, and puts them in an *event queue* associated with the corresponding application. The X Windows system that is running on the machine in front of you (and intercepting events and displaying applications) is called an X Windows *server*. Application programs are called *clients*. X Windows clients may be either *local* or *remote*. A local client is an application that is running on the same machine as the server that is hosting it. A remote client is an application that is running on a different machine on the network, but being displayed on the screen managed by the server. When an event is detected on the local machine that corresponds to a remote client's window, that event is sent to the remote client over the network. The remote client then reacts to the event (by redrawing its window, for example) and this reaction is reflected on the local screen.

There are two ways to start up a remote client application. The first way is to simply issue a command like

```
ssh -X 219a mozilla
```

and type your password at the prompt that follows. In this command `219a` is the name of the remote host and `mozilla` is the name of the program you want to execute.

The second way is more involved but might be easier if you anticipate running a number of applications remotely over a period of time. The first step is to give permission to applications on the remote machine to use this server. To do this, issue the command

```
xhost +remote-host
```

in a local shell, where `remote-host` is the name of the remote host. Then log in to the remote host on which you want to start the application. Once you are logged in, you have to specify that applications you start should be displayed on your local machine instead of on the remote one. To do this, you need to set the `DISPLAY` environment variable:

```
DISPLAY=local-host:0
```

The value `local-host` is the name of the host running the X server (the one at which you are sitting). The `:0` is the display number. Now you are ready to execute any number of client applications. For example, executing `mozilla` now on the remote host will display the Mozilla application on your screen.

13 Customizing your Account with Shell Scripts

Besides a command interpreter, Linux shells also include a programming language of their own. Programs written in a shell programming language are usually called *shell scripts*. Shell scripts are executed by typing their name on the command line, just like binary executables. (A shell script must have executable permission to be executed.) When executed, the shell interprets the commands in the shell script as if they were being typed on the command line. We will specifically discuss the `bash` shell here. The shell scripts with which you are most likely to come into contact are called `.bash_profile`, `.bashrc`, and `.bash_logout`, all located in your home directory. The first script is executed automatically by the shell when you log in, the second is executed every time you start a new `bash` shell, and the third is executed when you log out. An in-depth discussion of `bash` shell programming is well beyond the scope of this tutorial, but we will very briefly discuss some of the basics here — just enough to get you familiar with these initialization files.

Here is an example of a simple `.bash_profile` file:

```
# User specific environment variables
PATH="$PATH:.:~/bin:/bin:/usr/bin:/usr/local/bin:/sbin:/usr/sbin:/usr/X11R6/bin"
CDPATH="$CDPATH:.:~/"
PS1="\u@\h> "
PRINTER="pr219"
USER=$(id -un)
HOSTNAME=$(/bin/hostname)
```

```
export PATH CDPATH PS1 USER MAIL PRINTER HOSTNAME

umask 077

# Get the aliases and functions from .bashrc
if [ -f ~/.bashrc ]; then
   . ~/.bashrc
fi
```

The first line in the file above is a comment, which is ignored by the shell. The next seven lines assign values to a variety of *environment variables*. There are two types of variables in `bash`: *environment variables* and *user-defined variables*. The values of environment variables are used by the shell and programs executed from the shell to customize their execution. User-defined variables are used as temporary locations for data during a shell script. You can enter the command

```
printenv
```

to see the values of all currently set environment variables. The value of an environment variable can be set by using an assignment statement like those above. For example, the statement

```
PS1="\u@\h> "
```

sets your shell prompt to the form `user@host>` . The special character `\u` stands for your user name and the special character `\h` stands for the machine's host name. Similarly, in the statement

```
PS1="\w$ "
```

the `\w` is a special character that stands for the current working directory. So this command changes your prompt to the name of the working directory followed by a `$` and a space.

The `PATH` variable contains a list of directories (separated by colons) in which the shell will look for programs whose names you type on the command line. If you need to execute a common program by typing its absolute pathname on the command line, add the directory to this list and you can simply type the program name from now on. Notice the mention of `$PATH` at the beginning of the list. When a variable name is prefixed with `$`, the value of the variable is returned. So, in this case, we are appending the list of directories to the current value of `PATH`. The `CDPATH` variable contains a list of directories in which the shell looks for relative pathnames specified with the `cd` command. The `PRINTER` variable contains the name of your default printer. The `USER` variable contains your user name. The syntax on the right hand side of the assignment is for *command substitution*. This means that the command inside the `$(command)` construct is executed and the result is assigned to the variable. In this case, the `id -un` command returns your user name. The `HOSTNAME` variable contains the host name of the machine. The `/bin/hostname` command gets this information and the result is stored in the variable.

After the variable assignments is the `export` command. This command causes the variables that follow to be copied by the shell to every program executed from the shell. For example, if we did not `export PRINTER` here, the default printer name would not be known to the `lpr` command later.

The `umask` command specifies the default access permissions for new files that you create. The *bit mask* argument of `umask` specifies which permissions should be turned off when a new file is created. The default permissions are obtained by subtracting the argument of `umask` from `666` for files and `777` for directories. In this case, the default permissions are set to `600` (`rw-------`) for files and `700` (`rwx------`) for directories.

The last part of the file is an example of a conditional construct. This particular statement tests to see if the file `~/.bashrc` exists and if so, executes it. See one of the references in the bibliography for more information about shell programming if you are curious about how this works.

In your `.bashrc` file, you will find a number of `alias` commands. Aliases are shorthand for commonly used commands. For example, the command

```
alias rm="rm -i"
```

makes `rm` (the remove file command) an alias for `rm -i`, which means that every time you delete a file, you are asked whether you are sure you want to delete it. (The `-i` option is short for "interactive". Without this option, `rm` would delete a file without asking first.)

If you want to print the value of a variable (or anything else) on the screen, you can use the simple `echo` command. For example,

```
echo Hello World!
```

prints `Hello World!` in the shell. To print the value of a variable, prefix the variable name with `$`. For example,

```
echo $PS1
```

prints the value of the `PS1` variable in the shell.

You are encouraged to make changes to these configuration files as you please. After you make a change, execute

```
source ~/.bash_profile
```

(or substitute another shell script) to execute the script and institute the changes.

14 How to Learn More

We encourage you to play around with these systems so that you begin to feel more comfortable with them. Linux provides a very powerful environment in which to work (and play!). There is tons more to learn, if you are so inclined. The menu in the Panel at the bottom of the screen is a good starting point. Also refer to the command reference in Appendix B, the bibliography below, and the books in the department library (Olin 213) for information on more applications and commands.

Bibliography

[1] Paul K. Anderson. *Just Enough UNIX*. McGraw Hill, 3rd edition, 2000.

[2] Bill Ball, David Pitts, and William Ball. *Red Hat Linux 7 Unleashed*. Sams, 2000.

[3] Debra Cameron, Bill Rosenblatt, and Eric Raymond. *Learning GNU Emacs*. O'Reilly and Associates, 1996.

[4] Sumitabha Das. *Your Unix: The Ultimate Guide*. McGraw Hill, 2001.

[5] Jessica Perry Hekman. *Linux in a Nutshell*. O'Reilly and Associates, 1997.

[6] Jerry Peek, Grace Todino, and John Strang. *Learning the UNIX Operating System*. O'Reilly & Associates, 1998.

[7] Syed Mansoor Sarwar, Robert Koretsky, and Syed Aqeel Sarwar. *Linux: The Textbook*. Addison Wesley Longman, 2002.

[8] Matt Welsh and Lar Kaufman. *Running Linux*. O'Reilly and Associates, 1996.

A UNIX Machine Information

Below you will find more detailed information about the computers in our department running some variant of UNIX.

The hostname of each machine is a symbolic name and is in the Internet domain `mathsci.denison.edu`. Hence, `216b` is really called `216b.mathsci.denison.edu`, for example.

You will be able to use the information below to log in to these machines remotely using ssh or remote X Windows (from the computer in your room, for example). Virtually any number of users can be logged in to one machine at the same time and not interfere with each others work.

A.1 Linux

Hostname	Computer	CPU	Location
`219a`, `219b`, ..., `219p`, and `219in`	Dell Optiplex GX150	1GHz Pentium III	Olin 219
`216a`, `216b`, ..., `216g`	Dell Optiplex GX280	3GHz Pentium 4	Olin 216*

A.2 Mac OS X

Hostname	Computer	CPU	Location
`217a`, `217b`, ..., `217x`, and `217in`	Power Mac G4	2 × 1.25GHz PowerPC G4	Olin 217
`olin216mac1`, ..., `olin216mac3`	iMac G5	2GHz PowerPC G5	Olin 216

A.3 Sun Solaris

Hostname	Computer	CPU	Location
`sunshine`	Sun Enterprise 250	2 × 400MHz UltraSPARC-II	Olin 218

B Linux Command Reference

The following is a (brief) summary of many of the commands available in Linux. Look at the `man` pages to learn more about them. In what follows, text parameters enclosed in square brackets (`[ ]`) are optional.

Special Symbols

- `.` the current directory
- `..` the parent directory
- `~` your home directory
- `/` separator between directories in a pathname
- `*` wildcard in a filename that matches any sequence of characters
- `?` wildcard in a filename that matches any single character
- `&` run a process in the background

  Example: `xemacs &` runs `xemacs` in the background, freeing up the terminal.

*Any of these machines may have Windows XP installed instead.

Account Management

`passwd`	Change your password.
`ypchfn`	Change the name associated with your account.
`ypchsh`	Change your default shell.

Applications

`emacs`, `xemacs`	a very well endowed text editor.
`gedit`	a simple text editor
`gimp`	the GNU image manipulation program (similar to PhotoShop)
`xpdf file`	display pdf file `file`
`j`	a text editor (written in Java)
`kedit`, `kwrite`	KDE text editors
`latex`	a typesetting program used by mathematicians and scientists (see references below)
`nedit`	a text editor
`mozilla`	the web browser
`soffice`	OpenOffice is an office application suite, complete with a word processor, spreadsheet, HTML editor, etc. It is Microsoft Office compatible. To use OpenOffice, access the Office menu from the menu in the bottom left corner of your desktop.

Disk and File System Manipulation

`cd [dir]` — Change the current working directory to `dir`.

`cp [-f][-i][-R] file1 file2` — Copy `file1` to file or directory `file2`.

Options: `-f` force copy without asking
`-i` interactive — ask before copying
`-R` recursively copy directory and its contents

Examples: `cp sort.cc sort/sort.cc` copies `sort.cc` from the current directory to the subdirectory `sort`.
`cp /home/173/example.cc .` copies `example.cc` from `/home/173/` to the current directory.

`df` — Display disk space available on each mounted volume.

`du [directories]` — Display the space used by files in the current directory or `directories` if specified.

`fdformat /dev/fd0` — Format a floppy disk. `/dev/fd0` is the name of the floppy disk drive.

`find dir -name file -print` — Find and print locations of `file` on the disk by descending the file system from directory `dir`.

Examples: `find . -name foo.bar -print` prints location of `foo.bar`, starting from current directory.
`find /home -name '*.txt' -print` prints location of files ending in `.txt`, starting from `/home`.

`ln -s source [dest]` — Make a symbolic link (alias) for the file `source` in the current directory or name the link `dest`.

`ls [-a][-F][-l][-R] [files]` — List contents of current directory or `files`, which may be files or directories.

Options: `-a` show all files, even those prefaced with a `.`
`-F` follow directory names with a `/`
`-l` show file sizes, dates, and permissions
`-R` list subdirectories recursively

`mkdir dirs` — Create directory(ies) with name(s) `dirs`.

`mtools` — Tools to read DOS disks (`mcopy`, `mdir`, etc.)

`mv [-f][-i] file1 file2` — Move `file1` to file or directory `file2`

Options: `-f` force move without asking
`-i` interactive — ask before moving

Examples: `mv sort/sort.cc .` moves `sort.cc` from subdirectory `sort` to the current directory.
`mv sort.cc sorting.cc` renames `sort.cc` to `sorting.cc`.

`pwd` — Print the name of the current working directory.

`rm [-R][-f] files` — Delete (remove) `files`.

Options: `-R` remove files and directories recursively
`-f` force delete – do not ask before deleting

`rmdir [dir]` — Delete (remove) directory `dir`. The directory must be empty.

File Manipulation

`cat [files]` — Print files to standard output. Reads from standard input by default.

Examples: `cat file.txt` displays `file.txt`.
`cat > file.txt` creates `file.txt` from standard input.
`cat f1.txt f2.txt > f3.txt` writes concatenation of `f1.txt` and `f2.txt` to `f3.txt`.

`diff file1 file2` — Display the differences between `file1` and `file2`. The output prefaces lines from `file1` with < and lines from `file2` with >.

`expand [-t num] files` — Convert the tabs in `files` to spaces. Specify the number of spaces with the `-t` option.

`grep [-i][-n] regexp files` — Find instances of regular expression `regexp` in `files`.

Options: `-i` ignore case
`-n` display line numbers with occurrences

`gzip [-r][-v] [files]` — Compress `files` (or standard input, by default) using Lempel-Ziv encoding.

Options: `-r` descend recursively into directories
`-v` verbose output

`gunzip [-r][-v] [files]` — Decompress `files` (or standard input by default) compressed with `gzip`.

`head [-n num] [files]` — Display first `num` (10 by default) lines of `files`.

`more [files]` — Display files one page at a time. During the display, type spacebar to see next page, return to show next line, = to show line number, `/pattern` to search for `pattern`, or `q` to quit.

`tail [-n num] [files]` — Display last `num` (10 by default) lines of `files`.

`tar cf tarfile files` — Create an archive called `tarfile` containing `files`.

`tar xf tarfile` — Extract files from an archive called `tarfile`.

File Permissions

`chmod mode files` — Set access `mode` (permissions) for `files`. There are two ways to specify permissions for a file.

1. First, use some combination of `u` for user, `g` for group, `o` for other, or `a` for all to specify which of these group(s) of users' permissions to modify. Then use + to add a permission or − to take away a permission. Lastly, specify the permission(s) to add/subtract: `r` for read, `w` for write, or `x` for execute.

Example: `chmod go-rw *` takes away read and write permission from group and other for all files in the current directory.

2. Think of the `rwx` for a group (`u`, `g`, or `o`) as a representation of a 3 bit binary number (0 – 7 in decimal). If a permission is set, that bit is a 1; otherwise it is a 0. Use the 3 decimal digits for the 3 groups as the permission.

Example: `chmod 744 *` gives all permissions to user, and only read permission to group and other.

Help

`apropos [-w wildcard] keyword`	Search manual pages for `keyword`.
`info [topic]`	Display help on `command`.
`man [section] [topic]`	Display manual page on `command` in `section`.

Miscellaneous

`cal [[month] year]`	Display a calendar for a month or year. By default, display the current month.
`clear`	Clear the terminal screen.
`date`	Display current date and time.
`xclock`	a graphical analog clock

Networking

`host [host]`	Convert host name to IP address, or vice versa, using default name server.
`hostname [-d]`	Display host name of this machine. The `-d` option displays domain instead.
`sftp host`	Initiate secure file transfer protocol with remote machine `host`.
`ssh [host]`	Remotely log in to the machine `host`.

Printing

`lpq [-Pprinter]`	Display contents of default printer queue or queue of `printer`.
`lpr [-Pprinter] [files]`	Print `files` to default printer or `printer`.
`lprm [-Pprinter] [job nums]`	Remove jobs with numbers `nums` from a printer queue.

Process Management

`kill [-9] pid`	End process with process id `pid`. The `-9` option kills the process unconditionally.
`nice command`	Execute `command` with lower than normal priority (to be "nice").
`ps [aux]`	Display information about currently executing processes. The `aux` options show a lot of information about all processes in the system.
`top`	Display the top processes with respect to CPU utilization. Hit `q` to quit.

Programming

`ddd`	Data Display Debugger (you must compile with the `-g` option to use the debugger on your program)
`gcc`	the GNU C compiler
`g++`	the GNU C++ compiler
`gdb`	the GNU command line debugger

System Resource Information

`free`	Show free memory and swap space.
`uptime`	Show amount of time passed since machine was last booted.
`who`	Show who is logged into the local host.
`whoami`	Show your login name.

C Emacs

Emacs is an extremely powerful and flexible program, capable of doing much more than just editing text. There exist fervent Emacs aficionados that prefer to do *all* of their work through Emacs; they rarely ever type anything into a terminal window! However, Emacs can also be used in a very intuitive manner, just like any other text editor you may have used in the past.

There are two versions of Emacs available. One version, simply called `emacs`, works either in text mode in a terminal window or with a GUI in its own window. Which mode is invoked depends upon whether you are logged in remotely or locally. The other version, `xemacs`, only has a GUI, but with more extensive menus and a nicer scroll bar.

C.1 XEmacs Features

Most of the basic features of `xemacs` are self-explanatory. You can manipulate text files in all the ways you are used to. To start up XEmacs, type

```
xemacs &
```

in the shell. (Or you can type `emacs` to start up the other version. If you want to open a file on start up with either version, follow the command with a filename.) To get rid of the opening screen, just click somewhere in the window. If, for some reason, the program starts up with a split screen, select the Un-split (Keep This) option from the File menu to return to just one window.

Menus and Buttons You can accomplish most of what you need in `xemacs` using the mouse. There are menus at the top of the window that allow you to do a lot of basic things like opening and closing files, printing, editing, and compiling (more later). For instance, to open a new file, select the Open... option from the File menu. To select a file in the box that appears next, use the **middle** mouse button (or the left and right buttons together). There are also buttons below the menus that you can use to do many common things.

Editing Text You can move the cursor to a new location by clicking with the left mouse button just as you would in any other text editor. You can also select text by dragging with the left mouse button depressed. You can cut, copy, and paste text using the Edit menu. You can also copy and paste text using just the mouse. To do so, select the text with the left mouse button and then click with the middle mouse button on the location where you want it pasted. This process works in virtually every window on the desktop. For example, if you want to copy a line of text from a terminal window into an Emacs window, you can select it with the left mouse button in the terminal window and paste it into the Emacs window with the middle mouse button.

Multiple Buffers You can have multiple files open at the same time in Emacs. These files are stored in multiple "buffers". If you open a new file when one is already open, the buffer holding the existing file is put into the background while the new buffer is displayed in the foreground. To choose a buffer that is not currently in the foreground, select the name of the buffer from the Buffers menu in `xemacs`.

C.2 Emacs Keyboard Commands

Every command that is in a menu (and many, many more) have keyboard equivalents in Emacs. It is sometimes useful to know how to issue these commands from the keyboard, especially if you want to use `emacs` remotely in text mode. Most Emacs keyboard commands involve two special keyboard modifier keys, the Ctrl key and the META key. Key combinations

involving the Ctrl key in Emacs are prefixed with C-, both in this document and in Emacs documentation. For instance, C-x means you should hold down the Ctrl key while you press x. The META key is the key to either side of the space bar on most keyboards. (The META key may be labeled Alt on your keyboard.) Key combinations involving the META key are prefixed with M-. For example, M-x means you should hold down the META key while typing x.

Some commands in Emacs have a slightly more complicated form. For instance, C-x C-s means you should hold down the Ctrl key while typing x and then hold down the Ctrl key while typing s. C-h t means you should hold down the Ctrl key while typing h and then press t by itself.

When you issue a keyboard command, that command appears in the "minibuffer", the line at the far bottom of the window. For some commands (liking that for saving a file for the first time), you need to type something in the minibuffer (like a filename) after the command. When you are typing anything in the minibuffer window, hitting the Tab key will automatically complete the command or file name, if the completion is unique among all options available.

Working with files	
C-x C-f	Open (find) a file
C-x C-s	Save a file
C-x C-w	Save (write) a file as...
C-g	Cancel a minibuffer command
C-x b	Select an existing buffer to bring to the front
C-x C-c	Quit Emacs
Moving in a document	
C-v or Page Up	Scroll down one page
M-v or Page Down	Scroll up one page
M-<	Move to beginning of document
M->	Move to end of document
C-p or ↑	Go to previous line
C-n or ↓	Go to next line
C-f or →	Move forward one character
C-b or ←	Move backward one character
M-f	Move forward one word
M-b	Move backward one word
C-a	Move to beginning of line
C-e	Move to end of line
Cutting and Pasting	
C-w	Cut (kill) selected text
C-k	Cut (kill) a line
C-y	Paste (yank) killed text
C-_ or C-x u	Undo (may be used repeatedly to undo several operations)
Searching	
C-s	Search forward in text (repeat to search for next instance)

Figure 3: Brief Emacs command reference

In Figure 3, you will find a table of some of the more commonly used Emacs key combinations. If you wish to learn more advanced features, you are encouraged to browse the detailed, easy-to-use help facility available within the application. There is also an Emacs book in the department library that we recommend you consult, as appropriate.

C.3 Compiling From Within Emacs

You can compile programs from within Emacs by selecting the Compile... option from the Tools menu in either `xemacs` or `emacs`. (Alternatively, you can type `M-x compile`.) In `emacs`, this will bring up the compile command

```
g++ -g -Wall
```

at the bottom of the window. In `xemacs`, you may need to edit the compile command. You will need to complete this command so that it is appropriate to the program you are compiling. For instance, if you are compiling `sort.cc`, you should add

```
-o sort sort.cc
```

to the compile command. Then hit enter to compile your program. You'll notice that the window splits and the compiler output is displayed in the bottom half. If there are errors in your code, you can click on the error message with the middle mouse button to go directly to that line in the file. If you wish to hide the messages that appear in the lower half of the window, click in the top half (where your file is) and type `C-x 1`. You can also accomplish this by selecting one of the Un-Split options from the File menu in `xemacs`.

In `xemacs`, after you have compiled a program once, you can just hit the Compile button at the top of the window to compile the program again.

Once you have a working executable program, go to a terminal window to execute it.