

# Beginner's Guide to Workstations, UNIX, and the Sun Java Desktop Environment

Copyright © 20G% Department of Computer Science, Rochester Institute of Technology  
All Rights Reserved

## Beginner's Guide to Workstations, UNIX, and the Sun Java Desktop Environment

Document Version %I% %G%

Faculty of the Department of Computer Science  
Rochester Institute of Technology

Printed August 31, 2006

## Acknowledgement

This document is based on similar guides covering  
(in order they were created) CDE,  
the SunView windowing system and the X Window  
System using the Open Look Window Manager, which  
were in turn based on a guide prepared by  
Professor Daniel Hyde of Bucknell University.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X Window System is a trademark of X Consortium, Inc.

Motif is a trademark of Open Software Foundation, Inc.

Open Look is a trademark of AT&T.

Sun and SunView are trademarks of Sun Microsystems, Inc.

MS-DOS and MS-Windows are trademarks of Microsoft, Inc.

VMS, VAX, and DEC are trademarks of Digital Equipment Corporation.

# Beginner's Guide

## 1. Introduction

Welcome to the Department of Computer Science workstations! Each workstation contains a powerful microcomputer, a monitor or screen, a keyboard and mouse, and network connections to other workstations and computers across campus. Programs running on each workstation provide the tools you'll need to complete many of the assignments in your courses. In addition, other software exists to help you in ways not directly related to programming, such as submitting your assignments, sending and receiving electronic mail, creating drawings or reports, and many other tasks.

### 1.1. What is a Workstation?

A *workstation* is a combination of *hardware* and *software* which together provide a powerful and flexible computing environment. In particular, this environment is well-suited to software development – the focus of many Computer Science courses.

The workstation hardware consists of a powerful *microcomputer*, a high-resolution *monitor* for displaying text and graphics, a *keyboard* for entering text, and a three-button *mouse* for graphical interaction. All the workstations are connected to a *network* which is used to send information among the workstations and other computers on campus. Also on the network are *file servers* each connected to large *disk drives* that hold user programs and data. By keeping all files in a central location, we eliminate the need for students to use the same workstation for each session, as well as the need to carry floppy disks around. In effect, we have most of the advantages of a personal computer with few of the disadvantages.

The workstation software consists of programs and associated data files that make the workstation useful. Indeed, it is the ability to write new programs to solve new problems that makes computers so useful. While you will develop lots of software during your time at RIT, it's nice to know that a great deal of useful software already exists to help you. First and foremost is *UNIX*, a powerful *operating system* originally developed at AT&T Bell Laboratories. It is the job of the operating system to keep track of the files being manipulated and the programs being executed so that the user can concentrate on the task at hand. There are other operating systems (such MS-Windows for personal computers, and OSX for Apple's MAC series of computers); we have chosen to use UNIX because of its portability (i.e., it is available on computers from many different manufacturers). In addition, UNIX also provides tools that are especially valuable during software development.

Built on top of UNIX is a *windowing system* which divides up the image on the monitor into different regions, or *windows*, each interacting with a different program. The advantage of such systems is that they let you work on several related tasks at the same time. There are many such systems available, even for a single computer. We have chosen to use The *Sun Java Desktop* (which is based off of *GNOME*, the GNU Network Object Model Environment). We'll postpone detailed discussion of the windowing system for the moment, until we've had time to present more information on the workstation hardware.

#### 1.1.1. The Keyboard

The workstation keyboard has the same basic configuration as a standard typewriter or personal computer keyboard. Unfortunately, while the positions of letters and digits are well-defined, computer manufacturers have a hard time agreeing on where to place punctuation characters and various special keys. So you'll have to adjust to the different keyboards in use. Whatever their location, the following special keys are heavily used in computer applications:

## Beginner's Guide

### Return

This key is on the right side of the keyboard, but its size and shape can differ from workstation to workstation; it is sometimes labeled **Enter** rather than **Return**. As with a typewriter, use this key to signal the end of a line of text. In particular, when typing UNIX commands (requests to UNIX asking it to perform some service for you), pressing **Return** signals the end of the command and requests that the appropriate action be performed. Whenever you are waiting for the system to do something after you have entered a command, make sure you have pressed the **Return** key.

### Delete

This key, often located just above and to the right of the **Return** key, sends a character which is used to *interrupt* the execution of a program or command you are running; usually, this *terminates* the program, although some are able to “catch” this key and take appropriate action.

### Backspace

The **Backspace** key, near the **Return** key, is used to erase the most recent character you have typed into a window.

### Shift

The **Shift** keys act like their counterparts on typewriters. If you hold down **Shift** while typing letters you get capitals; for other keys, you get the character on the top of the key. Thus, holding down **Shift** and typing **2** will produce an **@** character on many keyboards.

### Control

This key is similar to the **Shift** key, in that it is used in combination with other keys to generate different characters. For instance, holding down the **Control** key and pressing **h** will generate the same character as the **Backspace** key. The use of **Control** is so common that we have a special notation: when you see **Ctrl-x** it means hold down **Control** and type **x**. Thus, the **Control** and **h** combination described above would be written as **Ctrl-h**<sup>1</sup>. *Important note:* Your account has been configured to recognize a special control character, **Ctrl-z**. This is called the *suspend* character. Suspending a program pauses its execution, and a suspended program may be continued later. Note that this is *not* the same as the **Delete** key, which *terminates* the program. A terminated program cannot be continued, so any partially completed computations are lost.

### Caps

Sometimes labeled **Caps Lock**, this key is similar to a caps lock on a typewriter, although it does not physically lock down on many keyboards. When the caps lock is turned on, all letter characters you type are sent to the system as capitals. On some workstations, there is a small light inside the key which is turned on when the caps lock is on; however, on others, the only way to tell if you are using capitals or not is to type a few characters. To switch back to lower case letters, press the **Caps** key again.

---

<sup>1</sup>Note that this notation is not universal. Some books, for instance, use **^h** or **C-h** where we would write **Ctrl-h**. You must read books and documentation carefully to determine the notation in use.

## Beginner's Guide

### **Esc**

This is an abbreviation for “escape”; the key is used just like a letter or number – it sends a special, non-printing character to the computer. In particular, **Esc** is *not* held down like the **Shift** or **Control** keys.

The remaining keys are *function keys* with various labels on them. Some of these have predefined uses to speed up common operations; we will describe some of these operations later on.

### **1.2. Facilities**

All Computer Science labs are in the Golisano Building (#70), in the central area of the third floor. There is the Computer Science Lab (CSL), which is available for use by all students in Computer Science courses.

The building also houses special purpose laboratories, for things like Distributed Systems, Artificial Intelligence, Graphics, and Security, among others. In addition, six labs of 16 workstations each have been set up as Instructional Computing Labs (ICLs) for use in conjunction with particular Computer Science courses. If you are taking one of these specific courses, you will be enrolled in both a lecture section and a lab section; the lab will be held in one of these rooms. When not reserved for instruction, one or more of the instructional laboratories may be open for general use.

### **1.3. Seeking Assistance**

As you learn to work with the UNIX system, it is quite possible that at some point you will find yourself confused. Before you become too frustrated, realize there are many sources of help. While you are in lab class, your instructor is there to help you. If you are in the CSL, there is always a lab assistant on duty who can answer questions about general topics. If your question relates to a particular assignment, there are teaching assistants in the work area next to ICL3. These teaching assistants and your instructor have office hours (usually posted outside their doors). During these hours, you are free to drop by without an appointment. Note, there is sometimes a line of students waiting for help. Finally, there is a variety of printed material available from the lab assistants, as well as a great deal of on-line documentation.

If you have major problems using a computer in the lab, first ask the lab assistant to help you solve the problem. If this doesn't work, contact your instructor or the teaching assistant.

## Beginner's Guide

### 2. Gearing Up

This section has three goals:

- To show you how to be able to use ( *log in* to ) a workstation using the account name and password you've been given. The windowing system will be started on your behalf as a part of the *login process*; this will take several seconds.
- To teach you how to change your password. This is critical to ensuring that no one else can gain unauthorized access to your account and either destroy your work or copy it for their own use.
- To show you how to get off ( *log off* ) the system at the end of a working session. You should always log off when you're done so that others cannot accidentally or maliciously use your account.

#### 2.1. Before Logging In

Our workstations are *ALWAYS* left with the power on to extend the life of the electronic components. However, to prevent burnout, the screen goes dark if the system is idle for more than a few minutes. If the power light (found on right bottom of monitor) is green or amber, you can get a prompt, by simply pressing the **Return** key. If the power light is not on, press the power switch. After a time the screen will display a prompt screen. Before logging in, be sure the **Caps** key is not on. UNIX does *not* understand commands in upper case (that is, capital letters). If your output consists of all capital letters, sometimes preceded by a backslash (\), either log off and try again, or seek help immediately.

#### 2.2. Logging In

In order to use a workstation, you must first identify yourself ("log in") to the computer. You should see a prompt requesting your account name in the middle of the screen.

**EXERCISE 2.1:** At this time, type your account name into the window and press **Return**.

Your account name is your three initials and four digits chosen by RIT. This is the same name RIT assigns to you for your institute-wide computer account.

You will now be asked to give a password. Type in the password *exactly* as shown on the account sheet, and press **Return**.

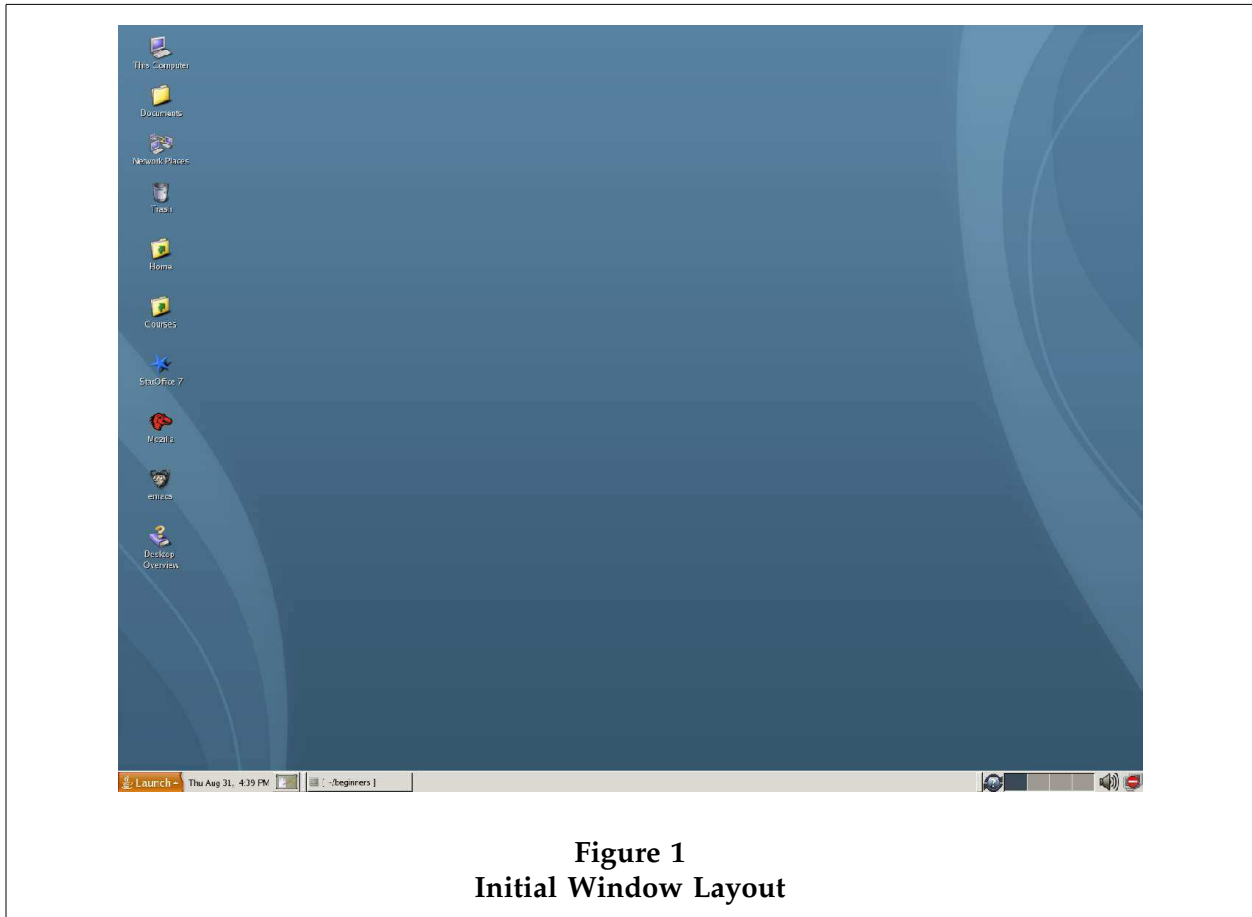
□

For security reasons, the characters you type here will not be displayed, so type carefully! If you know you made a mistake while typing in the password and you haven't pressed the **Return** key yet, you can type a **Ctrl-u** character to "wipe out" all the password characters you've typed in, and then start over. If you made a mistake but you've already pressed **Return**, you can't go back and fix the error; you'll see the error message `Login incorrect`, and you will be asked to log in again from the beginning. If you cannot log in after several tries, get help from the instructor or lab assistant.

The initial password on your account form was selected at random. Later in this section, you will learn how to change your password to something of your own choosing.

When you have successfully logged in, it may take a few moments to set up the windowing system and the initial window. Eventually you should see a display that bears some resemblance to the one shown in Figure 1. (Each year small changes are made to the desktop, browser, home page, etc., but this figure may not be up to date).

## Beginner's Guide



**Figure 1**  
**Initial Window Layout**

### 2.3. Mousing Around

Somewhere on the screen is a small glyph, which may be either a heavy “X” shape, an arrow pointing up and to the left, or a shape which looks like a thin “I”. This glyph is the *mouse pointer*. Move the mouse around on the pad, and note that the pointer on the screen follows this movement. This apparent connection between the position of the mouse on the pad and the position of the pointer on the screen is something of an illusion. The pointer on the screen follows the mouse’s movement only while the mouse is in contact with a surface, if you pick up the mouse and place it in a different position, the pointer on the screen doesn’t follow it (although the pointer may jiggle around a bit as you lift and place the mouse). This is because moving the mouse really only transmits movement coordinates to the computer, which translates them into pointer movements; the mouse can’t detect movement while it’s in the air.

Most window operations are performed using a combination of mouse movement and mouse button presses. Generally speaking, the pointer must be on or inside a window’s border in order for you to use the window. What is more, the effect of pressing a mouse button is determined by the current pointer position.

First, we need to understand how the mouse buttons are used in different situations. Your account has been pre-configured as follows:

## Beginner's Guide

- The *LEFT* mouse button is generally used to *select* an item from the screen in order to use it somehow. In the case of text, selecting the text will *highlight* the text by displaying it in reversed video. We will use the name **LEFT** to refer to this button.
- The *MIDDLE* button is used in conjunction with **LEFT** to copy text from one window to another. We will use the name **MIDDLE** to refer to this button.
- The *RIGHT* button serves several purposes. Within a window, it can be used to extend a selection made by **LEFT**. Within a title bar, border, or the screen background area, this button will bring up whatever *menu* is associated with that area. We will use the name **RIGHT** to refer to this button.

Note that these conventions are not universal. Many programs that can be run under this environment do not use the mouse buttons in exactly this manner; some applications associate menus with **LEFT** or **MIDDLE** instead of (or in addition to) **RIGHT**. It's a good idea to read the documentation for a new program (or just try it out) to see how it assigns responsibilities to each of the buttons.

Having given names to the buttons, let's define terms for various mouse operations:

- *Press* means push down on the button and hold it. Thus, "press **RIGHT**" means "press and hold down the **RIGHT** button".
- *Release* is the opposite of press: simply remove pressure from the button. Obviously, this assumes that the named button is being pressed.
- *Click* means "press-and-release" quickly (within 1/10 second or so).
- *Double click* means "click twice rapidly" (within 1/4 second or so).
- *Move* means use the mouse to reposition the pointer when no buttons are pressed.
- *Drag* means use the mouse to reposition the pointer while you hold down one or more buttons.

NOTE: Sometimes the mouse pointer will change shape to indicate some special action is taking place. For instance, many applications change the pointer to an hourglass or a stopwatch when they are busy with a long computation.

It is also possible to copy and paste using the keys on the extreme left of your keyboard. After highlighting the text you wish to copy, press the **Copy** key. Then move the mouse to the position you wish to place a copy of the text and press the **Paste** key.

### 2.4. Logging Out

Logging out is just as important as logging in. *If you forget to log out, anyone can access your files, or use your account for illegal activities for which you will be held accountable!*

The way to log out is to use the **Log Out** option of the **Launch Menu**. To access this menu, press the **LEFT** mouse button on the Launch menu item in the lower left corner of the screen. You will see a menu like Figure 2 appear. Click on **Log Out** item to log out of the system.

Note that if you log out you may lose any work in progress. It is usually a good idea to make sure all applications are closed before logging out.

**EXERCISE 2.2:** Log out of the workstation using one of the instructions above. After you have successfully logged out, you will see the login prompt screen again. Now log back in so you can continue these exercises.

□

## Beginner's Guide



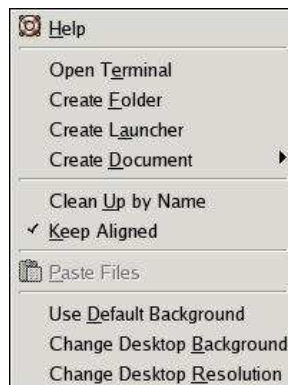
**Figure 2**  
**Launch Menu**

### 2.5. Terminal Windows

Unix is a system that, at its core, uses commands typed into a *terminal window* to perform tasks. Windowing environments (like the *Sun Java Desktop* have been developed to allow a user to perform tasks in a simpler, more intuitive manor. The problem is that not everything can be accomplished from the desktop, and so you will often need a terminal window.

You can open a terminal by first right clicking on the desktop, this will bring up the Desktop Menu. Figure 3 From this menu, the "Open Terminal" option will open up a UNIX terminal where you can type in commands.

You will want to open up at least one terminal window at this point so that you can perform the next step.



**Figure 3**  
**Desktop Menu**

## Beginner's Guide

### 2.6. Setting Your Password

Your password is a security device which ensures that only you have access to your files. Therefore, when you type your password it will not appear on the screen. To safeguard against unauthorized use of your account, be sure that you are the *only person* who knows your password. **NEVER** tell your password to anyone. Doing so is a violation of the rules for using the Computer Science systems.

You will need to change the password given to you with your account to something that will successfully protect your account. While the password should be easy for you to remember, it should not be too easy to guess.

A valid password should be a mix of upper and lower case letters, digits, and other characters.

- You can use an 8 character long password with characters from at least 3 of the previous 4 classes
- You can have a 7 character long password containing characters from all the 4 classes.

Note that an upper case letter that begins the password and a digit that ends it do not count towards the number of character classes used.

You may also use a passphrase as opposed to a password for your account. The passphrase should be of at least 3 words, 12 to 40 characters long and contain enough different characters.

Whatever you do, *do not* choose a simple English word, the name of your account, your license plate number, your best friend's name, etc. These are all easily guessed, and make your account insecure. To help you select a "good" password, the `cspasswd` command used on our systems to change passwords, checks your choice and makes sure it isn't too easy to guess. Be aware that other systems may not do this for you, so you should get in the habit of "thinking safe" when you think up new passwords.

**EXERCISE 2.3:** At this time decide on a new password.

□

Move the pointer into the terminal window, and note that there is a prompt looking something like:

```
abc1234[11]%
```

The prompt gives your account name and the sequence number of the next command to be run. Right after the prompt you will find a small rectangle; note that the rectangle is solid when the mouse pointer is outside the window, but "blinks" when the window is ready to accept input.

To change your password, type the following and press the **Return** key:

```
cspasswd
```

This will bring up a message explaining the guidelines for legal passwords. After the guidelines, you will now be see the following message (or something very similar):

```
Enter existing login password:
```

This is checking to make sure that the person authorized to change the password is the person actually changing it. At this point, type in the password you were given originally, then press **Return**. The password will not be printed, so type carefully!

## Beginner's Guide

Then you will be asked for the new password. After typing the new password once, you will be asked to type it again. Since you can't see what you typed, this helps catch typing mistakes. If the second password matches the first, and if the old password is correct, you'll see a message like:

```
passwd: password successfully changed for user-id
```

where **user-id** is your user ID for the Computer Science systems. This indicates that you have successfully changed your password on the Computer Science machines. However, because of the way that our computers are networked together, **it may take 20 minutes or more for all the Computer Science systems to know about your new password.** Also, remember that this *only* affects your Computer Science account. Your password on other computer systems (e.g., DCE, grace.rit.edu) is *separate* from your Computer Science account's password, so you will need to change those separately.

Messages such as

```
Weak password: not enough different characters or classes for this length.  
Weak password: is based on the old one.  
Weak password: is the same as the old one.  
Weak password: too short.  
cssswd: Sorry, wrong passwd  
Permission denied
```

means that you made a mistake somewhere. Most likely you typed your old password incorrectly, you weren't able to type in the new password the same way twice in a row, or your new password choice wasn't "safe" enough to please the program. If you can't change your password after several tries, get help from the lab assistant or your instructor.

**EXERCISE 2.4:** Change your password using the instructions above.

□

## Beginner's Guide

### 3. Introduction to The Sun Java Desktop.

The Sun Java Desktop provides a configurable graphic interface to the Unix operating system. The next few pages will introduce you to some of the basic functionality of Sun Java Desktop.



**Figure 4**  
**Default Environment**

When you log in you will see the default Sun Java Desktop setting we have provided to get you started, which should look like the figure above. As you become more experienced with environment, you may wish to change some of the settings. For now, let's look at what's provided.

#### 3.1. Desktop Overview

Using the Sun Java Desktop windowing environment is very similar to working with a Microsoft Windows machine. for information about how to use the desktop you should look over the "Desktop Overview" help provided on the desktop.

## Beginner's Guide



**Figure 5**  
**Desktop Overview help**

### 3.2. The use of the Web in CS1

Your desktop contains an icon for launching a web browser. The one provided for you is Mozilla. This course has a website for most of the course administration information that you might need. You can find the CS1 webpage at:

<http://www.cs.rit.edu/~vcss231>

On this page you will find links to the syllabus, schedule, labs, in addition to a number of useful documents (like this one).

#### 3.2.1. Useful URLs

There are a number of useful websites at RIT. Most computer science courses have a home page with the URL of <http://www.cs.rit.edu/~vcss###> where ### is the course number<sup>2</sup>.

Various Department of Computer Science documents, including the Code of Conduct and Policy on Academic Dishonesty can be found at

<http://www.cs.rit.edu/usr/local/pub/doc/>.

RIT's home page is found at <http://www.rit.edu/>.

The home page for any Computer Science student can be found at <http://www.cs.rit.edu/~abc1234> where abc1234 is the user's login.

**EXERCISE 3.1:** Look at the address for your home page. Note the source for this file is stored in your public\_html directory.

□

---

<sup>2</sup>Note that some courses use the old **icss###** format instead of the **vcss###** format. The reason for this is that **vcss** is the GCCIS code, but previously Computer Science was part of CAST, whose code was **icss**).

## Beginner's Guide

### 4. Introduction to the Filesystem

In order for a computer to be of any use, it must be able to manipulate and store information. To provide this service, most computers have some type of mechanism for “filing” information called a *filesystem*. In this section, you will be introduced to some of the details of the UNIX filesystem.

#### 4.1. What are Files?

A *file* is the basic container for information. It has a *name* and *contents*. For instance, a file named `letter-to-mom` might contain the text of a letter home. Many files contain text (such as your letter home), while others contain programs, commands, screen pictures, etc.

On our UNIX systems, file names can be made up of any characters (letters, digits, and punctuation) except the slash (/) character. Names can be up to 255 characters long, though it's unusual to see one over 20 characters. We suggest that you start your file names with a letter, and use only letters, digits, periods, underscores, and dashes for the other characters in the name. While it's *legal* to use other characters in file names, it's often *unwise* to do so, as many UNIX commands cannot handle all possible names, and some legal characters can be tricky to type in as part of a file name.

It is best to pick a name that reflects the contents of the file. Although UNIX itself doesn't infer anything about the contents of a file from its name, many of the *commands* we execute under UNIX do. Some commands expect that the last few characters of a file's name, called a *suffix*, will indicate the contents of the file<sup>3</sup>. Common suffixes used with UNIX include: `.java` (files containing Java source code), `.C` (C++ source code), `.e` (Eiffel source code) and `.mod` (Modula-2 source code). While most UNIX suffixes are one character long, there is no actual limit on the suffix length, although the regular 255-character name length limit includes the suffix.

#### 4.2. What are Directories?

To simplify the task of organizing individual files, most filesystems contain a special type of file called a *directory*. A directory contains the names of other files along with the location of their contents on the disk. Directories can contain other directories and the “contained” directory is called a *subdirectory*. The subdirectory can contain other directories (subsubdirectories), etc. To identify a file in the filesystem, you specify its location by giving the *pathname* of the file, which is the list of directories in the proper sequence, which the operating system must examine to reach the file you want. Components of a file's pathname are separated with slash characters (/) so that the operating system can tell them apart<sup>4</sup>. Typically, the last component of a pathname is the thing (file or directory) you're trying to access.

Suppose we have a very small filesystem that contains the following things:

- a directory named `home` at the topmost level;
- within `home`, two subdirectories named `faculty` and `student`;
- within the `student` directory, a subdirectory named `jesse`;

---

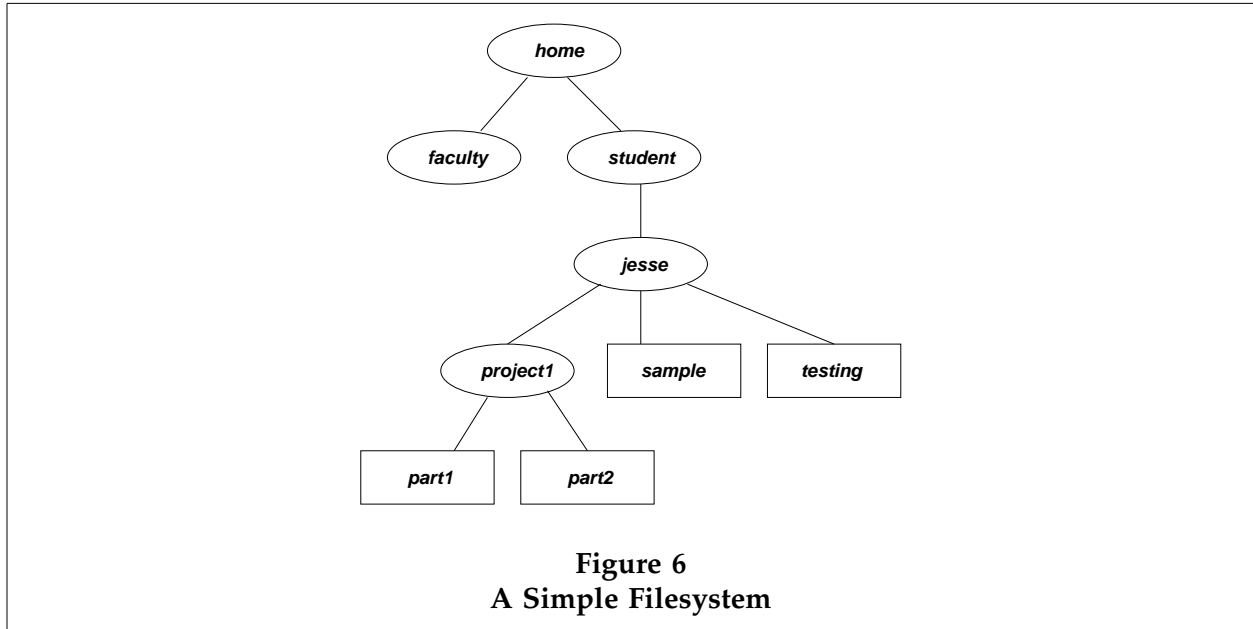
<sup>3</sup>Note that this is *not* the same thing as a file *extension*, such as is found in MS-DOS and VMS filesystems; in particular, the suffix is actually part of the file's name, there is no limit its length (other than the regular limit on the length of a file name), and the operating system takes absolutely no notice of the suffix.

<sup>4</sup>Note that the slash character (/) and the backslash character (\) are *not* the same, and do *significantly* different things under UNIX.

## Beginner's Guide

- within the jesse directory, files named sample and testing, and a directory named project1;
- and within the project1 directory, files named part1 and part2.

We can make this more clear by drawing a picture; in Figure 6, files are shown as rectangles, directories as ovals, and the relationship between a directory and its contents as a line leading from one to the other.



**Figure 6**  
**A Simple Filesystem**

Each file in the filesystem can be uniquely located through its *full pathname* (sometimes called an *absolute pathname*) which begins with a slash and includes all of the subdirectories between the topmost level of the filesystem and the desired file. In Figure 6, the full pathname of the file named part1 is /home/student/jesse/project1/part1. To further describe the relationship between a directory and its contents, we often refer to the directory that contains a file as the *parent* of that file, and the file as a *child* of the parent directory. In this example, student is the parent of jesse, and part1 is one child of project1. While a directory may have many children, directories and files have only one parent.

When your account was created, it was given a directory in the filesystem, which is called the *home directory* for your account. You have control over the files and directories in your home directory. With few exceptions, no one can access any of these files unless you explicitly permit it. As you use your account, you will create subdirectories to to organize your work. We suggest you have a directory for each course you're taking, with subdirectories for labs, projects, papers, etc.

It is not necessary to specify the full pathname of a file when you want to use it. The operating system keeps track of the *working directory* which is where you are in the system. A working directory is simply a short-cut into the filesystem.

If the file you want to use is in your working directory or in a subdirectory of your working directory, you can specify a *relative pathname* to the file instead of the full pathname. A file *in* the working directory can be located simply by its name; a file in a subdirectory can be located by the name of the subdirectory and the name of the file. In the filesystem of Figure 6, suppose that our working directory is /home/student/jesse; the file testing

## Beginner's Guide

can be identified either by its full pathname (`/home/student/jesse/testing`), or by a pathname that is relative to our working directory (`testing`). Similarly, `part2` can be located as either `/home/student/jesse/project1/part2` or `project1/part2` from this working directory. Remember: *If a pathname begins with a slash, it is a full pathname; otherwise, it is a relative pathname*<sup>5</sup>.

**EXERCISE 4.1:** Imagine your working directory is `student` in the filesystem of Figure 6. List a full and a relative pathname to the file `part1`.

□

### 4.3. Using Directories

#### 4.3.1. Finding Out What's In a Directory

In order to see what the files are in a directory, the UNIX filesystem, provides a program named `ls` which provides information about a directory or file. In a shell window, type the command

```
ls
```

and press **Return**. If you haven't used your account before, it may not have any files or subdirectories in it, so the `ls` command won't report any names to you.

By default, `ls` reports the names of things in your working directory in alphabetic order. If you want to see what's in another directory, you can give `ls` one or more arguments specifying other directories to be listed; for example,

```
ls /usr
```

executes the `ls` command on a directory named `usr`, which is near the topmost level in the filesystem.

Another interesting "feature" of `ls` is that it won't list things whose names begin with a `.` (dot) character. This is why we suggest that you use names that begin with a letter (not a dot) for files you create. If you want to see a list of *all* the files in the directory, use the `-a` option to `ls`.

**EXERCISE 4.2:** In a shell window, execute the following commands and compare their output.

```
ls -a
ls
```

□

There are other options to `ls` which allow you to get additional information about the contents of a directory. The `-l` option requests what is called a *long* listing, and produces lines like the following:

```
-rwxr-x--x 1 wrc000    8108 Aug 22 16:42 stories
```

The first field contains ten characters giving the file type and protection information. If this file is really a directory, the first character in this field is a `d`; for an ordinary file, the first character is a `-` (as in this example).

The next nine characters specify the protection information for the file in the form of *permissions*. These determine which accounts can access this file and in what ways. The permissions are organized as three sets of three characters each. Within a group, the

---

<sup>5</sup>Actually, as we'll see later, some pathnames which begin with a tilde (`~`) are also full pathnames.

## Beginner's Guide

characters indicate, from left to right, permission to read (r), write (w), or execute (x), this file. (Execute permission is only meaningful for files which contain “runnable” programs and for directories, but even text files can be given execute permission.) If a particular permission has been granted, the corresponding character will be present; if the permission has been denied, the position will be filled with a dash character. The groups themselves, again from left to right, specify the permissions given to three different collections of accounts: the owner of the file; accounts that are in the same *group* as the owner; and all the other accounts on the computer. These permission sets are sometimes referred to as *user*, *group*, and *other* (also known as *world*) permissions. In this example, we can see that the owner of the file *stories* can read, write, and execute the file.

**EXERCISE 4.3:** What are the group permissions for this file? What are the world permissions?

□

The second field gives the number of *links* to this file in the filesystem. Links are *aliases* (alternate names) for things in the filesystem. Under some circumstances, there may be several links to a file or directory; in this case, the link count field will be greater than one. In particular, directories will typically have a link count greater than one.

The third field in this example shows the name of the account that owns this file. (On some versions of UNIX, the `ls` command adds a second ownership field after this one; containing the group ownership of the file.

The fourth field contains the size of the file in *bytes* (characters); the fifth field shows the date and time this file was last modified. The final field is the file name itself.

**EXERCISE 4.4:** We can combine `ls` options into a single argument, or specify them separately. Execute the following commands, and compare the results for the file whose name is `.` (“dot”).

```
ls -l -a
ls -a -l
ls -la
```

Identify the following information: file type and permissions for the owner, group, and world; the file owner and group; the size of the file; and the date it was last modified.

□

Note that giving more than one pathname argument to `ls` will cause it to list information for *all* the supplied paths. In addition, the pathname arguments you give it can be pathnames of either directories or files. Giving `ls` directory pathnames causes it to generate a listing of the entire contents of those directories; giving it file pathnames causes it to confine its listing to *those files alone*. So we can get information on just one or two files, or on the entire contents of a directory.

**EXERCISE 4.5:** Use `ls` to generate long listings of just the files `.login` and `.cshrc` in your home directory. Next, use it to generate “short” (i.e., non-long) listings of the directories `/usr` and `/usr/lib/tmac`.

□

### 4.3.2. Creating Your Own Directories

To create a new directory, use the UNIX `mkdir` (make directory) command. This command takes the names of the directories you want to create as arguments. These can be

## Beginner's Guide

specified as either full or relative pathnames.

**EXERCISE 4.6:** In a shell window, create directories named `first`, `.second`, and `third`. Next, execute the following commands to see what your working directory now contains.

```
ls
ls -a
```

Why is it a bad idea to start a directory name with a dot?

One problem with “short” directory listings is that it can be hard to tell which names are file names and which are directory names. Many versions of `ls` support the option `-F` (that’s a capital **F**, not a lowercase **f**), which causes `ls` to append a slash (`/`) character to the end of directory names in the listing. It also appends other special characters to names; the one you’ll see most often is an asterisk (`*`) character, which is appended to the names of files which have execute permissions.

**EXERCISE 4.7:** Execute the following commands and identify the names of files and directories.

```
ls -aF
ls -alF
```

What are the `.` and `..` entries?

When you create a directory, two special subdirectories are automatically put into the new directory. These entries are actually alternate names for other directories in the filesystem. The `.` (dot) entry in a directory is an alias for the directory itself; the `..` (dot-dot) entry is an alias for the *parent* of this directory.

These two entries allow you to move up and down the filesystem at will. Consider again the filesystem in Figure 6. With `jesse` as the working directory, we can use `./testing` as a relative pathname for the file `testing`. From `jesse`, the command

```
ls ..
```

will list the contents of `jesse`’s parent directory (`student`);

```
ls ../jesse/project1
```

will list the contents of the `project1` subdirectory. It’s often useful to be able to refer to things which are above or next to your working directory in the filesystem without having to resort to full pathnames.

**EXERCISE 4.8:** Given the structure of Figure 6, and `faculty` as the working directory, write `ls` commands to list the file `testing` with a relative pathname.

### 4.3.3. Removing Directories

To remove a directory from the filesystem, use the `rmdir` command. This command takes one or more directory pathnames as arguments, and tries to remove them from the filesystem. A directory can not be removed from the filesystem unless you have permission to write in its *parent* directory, and the directory you’re removing doesn’t have any files or subdirectories in it. If either of these requirements isn’t met, `rmdir` will complain, and won’t remove the directory.

## Beginner's Guide

**EXERCISE 4.9:** Create a directory named `testing`, and, under it, one named `another`. (The second one, thus, will have the relative pathname `testing/another`.) Next, use `rmdir` to try and remove the `testing` directory. What happens? Next, remove `testing/another` and then `testing`; what happened this time? Confirm the results with `ls`.

### 4.4. Using Files

#### 4.4.1. Filename Wildcard Characters

Most UNIX commands that accept file names as arguments can process a single file or many files. By following some type of systematic file naming convention, you can use file name *patterns* to help you select the files to process. These patterns are made up of “normal” characters (letters, digits, etc.) along with characters known as *wildcard* characters, sometimes called *metacharacters*.

The most common wildcard character is the asterisk (`*`), which matches a sequence of zero or more characters in a filename. This is typically used to build an argument that matches a specific set of file names; for instance, `B*.e` matches all names beginning with `B` and ending with the `.e` suffix. You can use more than one wildcard in a filename argument; the pattern `do*a*t*` will match all of the following names: `doat`, `doormat`, `dogcatcher`, and `documentation`, but will not match `dot` or `adoart`.

**EXERCISE 4.10:** Explain why the pattern above matches the given file names.

**EXERCISE 4.11:** Which of the following five names would the pattern `*c*a*te` match?

accurate cat charter crate create

Another wildcard characters is `?` which matches any single character. The pattern `A?` matches any two-character filename that begins with `A`. Collections and ranges of characters can be specified with the `[ ]` wildcard which matches any characters between the brackets. `[ABC]` matches any of those three characters and `[Aa][Nn][Tt]` matches the word “ant” regardless of capitalization. Ranges of characters are specified by giving the first and last characters of the range, separated by a dash; so `[A-Z]` matches any single uppercase alphabetic. Multiple ranges can be specified, as in `[A-Za-z]`, which matches *any* alphabetic, and `[A-Za-z0-9]`, which matches any alphanumeric character.

#### 4.4.2. Creating Files

The simplest way to create a file is by making a copy of an existing file. This can be done by using the `cp` command which takes two (or more) arguments: the name of the existing file (called the *source* file), and the name of the file you want to put the copy into (called the *destination* file).

**EXERCISE 4.12:** Execute the command

```
cp ~cs1/pub/humpty my-humpty
```

in a shell window.

In the exercise above, we made a copy of a file named `humpty`, found in subdirectory

## Beginner's Guide

pub of the home directory for account cs1<sup>6</sup>. The new file is named my-humpty, and is in your working directory. Note that the source and destination file names can be specified using either full or relative pathnames; we used both in this command.

**EXERCISE 4.13:** Execute the command

```
cp ~cs1/pub/jack-theory .
```

to create file `jack-theory` in your working directory. Verify that it was copied by executing an `ls` command.

□

Note that using dot as the destination will cause `cp` to assume you wanted a new file with the same name as the source file.

Because the source and destination file names are pathnames, we can use file name wildcards in them. For instance, you could have copied *all* the accessible files in the `pub` directory into your account with the

```
cp ~cs1/pub/* .
```

command, which says “Copy all files that I can read that are in the directory `~cs1/pub` into my working directory, using the same names.” If you supply *more than one* source file name, `cp` requires that the destination name be the name of a directory; it will copy all the source files into the destination directory, preserving their names.

There is one thing to be careful about when using `cp`. If the destination file already exists and you have permission to write to it, the old contents of the file are silently discarded. Of course, if you don't have write permission to the file, you'll see an error message because `cp` won't be able to write into it.

### 4.4.3. Displaying the Contents of a File

The `cat` command (the name is derived from the word *concatenate*) is used to list the contents of files. It takes one or more file names as arguments on the command line, and simply writes their contents out in the window. So you would see the contents of all the named files displayed as if all the text came from one file. If there are more lines of text than lines in your shell window, they will disappear off the top of the window faster than you can read them!

The `more` command also takes the names of one or more files as its arguments, and displays their contents, in sequence. However, it doesn't run the file contents together in one big output stream; instead, it pauses between files. Furthermore, it also pauses after each screenful of text is displayed and prints a status line on the last line in the window, which looks like this:

```
--More--( NN %)
```

The **NN** percentage is how much of the file's contents have already been displayed. The `more` command pauses at this point, waiting for you to tell it to display the next screenful. Pressing the **space bar** once will advance the display by one screenful; typing a lowercase **d** or a **Ctrl-d** advances by half a screenful, and pressing **Return** advances the display by one line. If you've seen all you want to see, type a **q** to cause `more` to terminate.

---

<sup>6</sup>The notation `~cs1` is a shell shorthand notation for “the home directory of the account `cs1`”, which is actually `/home/course/cs1` on our systems. This is yet another way to write the full pathname of a home directory, and can be used with any account on the system.

## Beginner's Guide

### 4.4.4. Removing Files

To avoid cluttering up your account and wasting filesystem resources, you should remove files you are no longer using. The UNIX `rm` command will remove (delete) files that you no longer need. If you're not careful, however, you may also end up removing files you still want!

**EXERCISE 4.14:** Use `cp` to make two new copies of the file `my-humpty`; name one of them `junk`, and the other one `baloney`. Use `ls` to verify that these were created. Next, execute the command

```
rm junk baloney
```

to remove these files, and again use `ls` to verify the results.

□

It is important to remember that when you remove a file from a UNIX filesystem, the file is *gone forever!* The staff can *sometimes* retrieve an old version from a filesystem backup tape, but this takes time, and there is no guarantee that anything useful can be recovered. Therefore, to avoid inadvertently deleting the wrong file, type carefully when you use `rm`. In particular, think twice before using wildcards with `rm`, as you may accidentally delete many more files than you intend.

**EXERCISE 4.15:** *This is a mental exercise only – these commands may be dangerous!* What is the difference between these commands?

```
rm x*
rm x *
```

□

If you want to be especially careful when removing files, you can use the `-i` option of `rm`, which causes it to remove the specified files *interactively*. The command

```
rm -i *
```

causes `rm` to print out each file's name, followed by a question mark, *before it removes the file*. It waits for you to tell it whether or not to actually remove the file. If you tell it `y`, the file is removed; typing `q` will leave the file alone and cause `rm` to exit without doing anything else; and typing anything else tells `rm` to leave this file alone and go on to the next one.

### 4.5. Moving and Renaming Files and Directories

By definition, when you copy a file, you end up with *two* files: the original and the new copy. If you merely want to rename the file or move it to a new location in your account, you can use the UNIX command `mv`.

This command is a lot like `cp` in terms of how you use it:

- You can give `mv` two arguments which are the name of an existing file and the name of a new, possibly nonexistent file. As with `cp`, if the second file already exists and you have permission to write to it, its old contents are silently discarded.
- You can give `mv` two arguments which are the name of an existing file and the name of a directory. In this case, the file is moved to the directory using the same name.
- You can give `mv` *more* than two arguments. All but the last one must be existing files, and the last one must be an existing directory. Each of the named files is moved to the directory and keeps its names.

## Beginner's Guide

Because directories are files themselves, you can use `mv` to move and rename directories as well.

**EXERCISE 4.16:** Use `mv` to rename file `jack-theory` as `jack-knife` and confirm the change using `ls`. Use `mv` again to change the name back to `jack-theory` again, and again confirm that the change took place.

□

**EXERCISE 4.17:** Create a subdirectory named `beginning`, and move `jack-theory` into the new directory. Use `ls` to verify that `jack-theory` is no longer in your home directory, and *is* in the `beginning` directory.

□

### 4.6. Changing Directories

Recall that your working directory is the “starting point” UNIX uses when you specify a file or directory using a relative pathname. At login time, the working directory is set to your home directory; within a shell window, you can use the command `cd` (change directory) to select a different working directory *for that shell window*. This command takes a single argument, which is the name of the directory you want to “move to” in the filesystem. UNIX will change the working directory if the directory exists and is accessible.

**EXERCISE 4.18:** Use `cd` to change into the `beginning` directory, and run `ls`. If you did this correctly, you should see the `jack-theory` file you moved to `beginning` in the previous section.

□

To get back to your home directory, execute the `cd` command with no arguments. This is an easy way to get to a safe place if you become lost in the directory structure.

**EXERCISE 4.19:** Return to your home directory. Use `ls` to confirm that you really made it home.

□

As you move around among directories, you may sometimes get lost. The `pwd` command prints out the full name of your current working directory.

**EXERCISE 4.20:** Change to your home directory and execute `pwd`. What was printed? From the number of slashes in the name, it should be apparent that your home directory is actually a subdirectory several levels down in the system.

□

**EXERCISE 4.21:** In one terminal window, execute the command

```
cd /usr/tmp
```

In a second terminal window, execute the command

```
cd /home/course
```

Finally, execute the `pwd` command in both windows, and verify that the changes in working directory applied only to the particular window in which they were executed. (You may want to `cd` back to your home directory after you do this exercise, particularly if you're going to do more work now.)

□

## Beginner's Guide

### 4.7. Changing Permissions on a File or Directory

As mentioned previously, all files and directories have associated permissions. Read permission for a directory means you can see the names of the files and subdirectories in it. Write permission lets you add things to the directory and delete things from it (creating and removing files and directories). Execute permission allows you to access things in the directory according to the permissions on those things. Note the difference: read permission lets you see the names of things in a directory; execute permission is need to actually *access* the things themselves. If you can read a directory but cannot execute it, then you can see the file names, but you can't get at their contents! In practice, it is rare to allow reading or writing to a directory without also allowing execute permission.

To alter the permissions on a file or directory, use the `chmod` command which requires two or more arguments. The first argument is the desired new permissions and the remaining arguments are pathnames. Only the owner of a file or directory (and the system administrator) can change its permissions.

The permission specifier has three parts: `<who><op><permission>`, where `<who>` specifies which set of permissions are to be changed (owner, group, or world, or some combination thereof), `<op>` says how the permissions are to be changed (added or removed), and `<permission>` gives the permissions to be changed.

Valid values for `<who>` are `u` (user, equivalent to owner), `g` (group), `o` (other, equivalent to world), and `a` (all users); they may be combined in any way desired, such as `ug`, `go`, and `ugo`. Valid `<op>` values are `+`, `-`, and `=`, which add, remove, and set (make the permissions equal to) the specified permissions. Valid `<permission>` values are `r`, `w`, and `x`.

Here are some examples of `chmod` commands, and descriptions of what they do:

```
chmod a+rw file
    Anyone can read, write, and execute the file.
chmod go-w file
    Remove write permission for the group and world.
chmod u+w file
    Add write permission for the owner.
chmod a-rwx file
    Remove all permissions for all users.
```

**EXERCISE 4.22:** Change to directory `beginning` and execute the commands below to change the permissions on the file `jack-theory`. Use `ls -l` before you start and after each command, and explain why the permissions changed as they did.

```
chmod a-rwx jack-theory
chmod u+rx jack-theory
chmod go+rwx jack-theory
chmod a-w jack-theory
chmod o-x jack-theory
```

□

As with most UNIX commands, you can use file name patterns with `chmod` to modify a group of files. For instance, to remove all permissions except those for the owner on every file in the working directory, execute the command

```
chmod go-rwx *
```

## Beginner's Guide

You can achieve a similar effect by removing all permissions on the working directory itself with

```
chmod go-rwx .
```

This not only prevents other users from accessing the files, but also keeps them from even knowing what the file names are, and automatically provides that protection to new files you may add to the directory later.

### 4.8. Using the Filesystem with Commands

Many UNIX programs, including some of the ones you'll write, read their input data from the keyboard using *standard input* and write their output to the screen using *standard output* connections. Sometimes you'll want to provide input from a file, or capture a program's output in a file. UNIX shells provide *redirection* of the standard input and standard output to make this easy to do.

If you wanted to save a list of all the files and directories in your home directory, you can use the `ls` command to generate the list, and redirect its output to a file. To capture the output of the `ls` command in a file named `myfiles`, you would execute the command:

```
ls > myfiles
```

The `>` character tells causes the output of this command to be redirected into the file whose name follows the `>` (in this case, `myfiles`). The `ls` program itself is unaware that this redirection occurred; this makes it easy to write programs that work both interactively and with files.

**EXERCISE 4.23:** Use output redirection and `pwd` to get your working directory name into a file named `mydata`. Use `more` or `cat` to confirm that the working directory name is in the file.

□

When you use `>` with the name of an existing file, any data that was already in the file will be discarded (much like what happens with `cp` in this situation). If you want to append the new information to what was already there, use `>>` instead of `>`.

**EXERCISE 4.24:** Execute the command

```
echo NOTE >> mydata
```

in the directory where you did the previous exercise. The `echo` command just prints out a line containing all its arguments, so this should append the line `NOTE` to the previous data. Confirm this by listing the contents of the file.

□

As you might expect, UNIX also provides input redirection which allows you to have a program read data from a file instead of the keyboard. Input redirection is specified with the redirection character `<`.

**EXERCISE 4.25:** Use the `echo` command four times to put the following lines of text into a file named `sortinput` (remember to use `>>` to append to the file):

```
Beth  
Adam  
Zeke  
Tom
```

Next, execute the command

## Beginner's Guide

```
sort < sortinput
```

The `sort` command usually reads its input from the terminal, sorts it alphabetically, and prints the results to the screen; in this case, however, we've redirected its input to come from the file `sortinput`.

□

A *pipe* is another form of redirection of standard input and standard output. Essentially, a pipe connects the standard output of one command to the standard input of another without having to create an intermediate file. Suppose we have two files, `part1` and `part2`, which contain collections of words, and we want to put all of them together in a file named `results` in sorted (alphabetical) order. We could do that with this series of commands:

```
cat part1 part2 > tempfile
sort < tempfile > results
```

but that creates an extra file, `tempfile`. We can use a pipe to connect the output of `cat` to the input of `sort`, and just save the resulting output:

```
cat part1 part2 | sort > results
```

This says "execute the `cat` command on the files `part1` and `part2`, send the output from `cat` as input to the program `sort`, and redirect the output from `sort` into a file named `results`."

### 4.9. Printing Files

Many times you will want to print a copy of a file on paper. With our versions of UNIX, the `lp` command expects one or more file names as its arguments, and sends the contents of those files to the printer. For example, suppose you wanted to print the file `my-humpty`. Executing the command

```
lp my-humpty
```

would send this file to a printer. You can verify that it's being sent to the printer with the command `lpstat -o`, which lists all the entries in the printer *queues*.

Student accounts are set up so that the output they produce is sent to the default printer for the machine in use. The `-dname` option to `lp` can be used to send the output to the printer named *name*. For example, `lp -d csl_lw1 filename` will send the data in the file `filename` to the printer named `csl_lw1` in the Computer Science Lab.

Printers that are available for your use are `csl_lw1`, `csl_lw2` and `csl_xpar`. The first two listed are LaserWriters, and the last one is for transparencies.

**EXERCISE 4.26:** Find the default printer for your current machine by typing

```
env | grep PRINTER
```

and pressing **RETURN**. This will look through all your current environment settings for the printer setting and return it to you.

□

#### 4.9.1. Printing with Headings

The `lp` command only prints what's in the file; it does *not* include the date, or even the name of the file, on the listing. To get labeled listings you can use the `pr` command. This command divides the file's contents into page-sized chunks, printing the date, time, file name, and page number at the top of each page. Note that `pr` does not send its output directly to the printer; you must either save its output in a file and print that file, or use a

## Beginner's Guide

pipe to connect `pr` to `lp`:

```
pr jack-theory | lp
```

takes the labeled output from `pr` and sends it to `lp`, printing it on your default printer.

### 4.9.2. Removing Requests From a Printer Queue

Sometimes you may want to remove a listing from a printer's queue, as you have requested a print job you really don't need.

To remove a print request, run `lpstat -o` and find the entry for the listing you want to remove. The entry will be the name of the printer, followed by a dash and the job number. To remove the listing (job) from the queue, you must execute the command `cancel`. For example, to remove job `csl_lw1-267`, type

```
cancel csl_lw1-267
```

### 4.10. How Much Disk Space Am I Using?

Each student account has a *disk quota* associated with it to help manage the usage of disk space. Different accounts may have different quotas associated with them, based on their needs. If you exceed your quota, you won't be able to create new files until you remove some old ones. If you don't clean out old files regularly, you will eventually pass this limit.

Our systems provide two commands you can use to figure out how much disk space you're using. Executing the `du` (disk usage) command from your home directory tells you how much space is in each of your subdirectories and in your entire account in *blocks*, which are units of storage in the filesystem. Unfortunately, different systems have different ideas about how big a "block" is; in our labs, some systems think it's 512 bytes, and others think it's 1024 bytes<sup>7</sup>. For example, here is the output of `du` in one student's home directory:

```
19      ./pub
6       ./labs
59      .
```

This says that the `pub` subdirectory currently contains 19 blocks; the `labs` directory, 6 blocks; and the total space within the working directory (which we said was the home directory, and therefore holds the entire account) is 59 blocks.

While the `du` command tells you how much space you're using in your account, it doesn't tell you what your quota is. To get this information, you must use the `quota` command. Execute it as `quota -v`; the output will be a table showing how much space you're using, what your quota is, and how much time you have before you can no longer create any files. The quota is reported as two limit values: *soft* and *hard*. The hard limit is the absolute "maximum usage" value for your account. The soft limit is the point at which the system begins watching your usage in earnest. When you exceed your soft limit, our system starts a timer; you have until that time runs out (usually a week) to reduce your usage below the soft limit. If you haven't done so, your account is locked and you can only get access with the assistance of a system administrator.

---

<sup>7</sup>On 512-byte block systems, `du` typically has an option, `-k`, which says "report in 1024-byte blocks instead of 512-byte blocks."

## Beginner's Guide

You should get in the habit of checking your quota and disk usage periodically to avoid going over your quota.

*EXERCISE 4.27:* Change to your home directory and execute the `du` command. How much space are you currently using?

*EXERCISE 4.28:* Execute the command

```
quota -v
```

What are your hard and soft disk limits?

## **Beginner's Guide**

## Beginner's Guide

### 5. Glossary

The glossary can be found in [www.cs.rit.edu/~f2y-grd/beg-guide.glossary.pdf](http://www.cs.rit.edu/~f2y-grd/beg-guide.glossary.pdf)

# Beginner's Guide

## Table of Contents

<b>1</b>	<b>Introduction</b> .....	1-1
1.1	What is a Workstation? .....	1-1
1.1.1	The Keyboard .....	1-1
1.2	Facilities .....	1-3
1.3	Seeking Assistance .....	1-3
<b>2</b>	<b>Gearing Up</b> .....	2-1
2.1	Before Logging In .....	2-1
2.2	Logging In .....	2-1
2.3	Mousing Around .....	2-2
2.4	Logging Out .....	2-3
2.5	Terminal Windows .....	2-4
2.6	Setting Your Password .....	2-5
<b>3</b>	<b>Introduction to The Sun Java Desktop.</b> .....	3-1
3.1	Desktop Overview .....	3-1
3.2	The use of the Web in CS1 .....	3-2
3.2.1	Useful URLs .....	3-2
<b>4</b>	<b>Introduction to the Filesystem</b> .....	4-1
4.1	What are Files? .....	4-1
4.2	What are Directories? .....	4-1
4.3	Using Directories .....	4-3
4.3.1	Finding Out What's In a Directory .....	4-3
4.3.2	Creating Your Own Directories .....	4-4
4.3.3	Removing Directories .....	4-5
4.4	Using Files .....	4-6
4.4.1	Filename Wildcard Characters .....	4-6
4.4.2	Creating Files .....	4-6
4.4.3	Displaying the Contents of a File .....	4-7
4.4.4	Removing Files .....	4-8
4.5	Moving and Renaming Files and Directories .....	4-8
4.6	Changing Directories .....	4-9
4.7	Changing Permissions on a File or Directory .....	4-10
4.8	Using the Filesystem with Commands .....	4-11
4.9	Printing Files .....	4-12
4.9.1	Printing with Headings .....	4-12
4.9.2	Removing Requests From a Printer Queue .....	4-13
4.10	How Much Disk Space Am I Using? .....	4-13
<b>5</b>	<b>Glossary</b> .....	G-1