1. BEFORE WE BEGIN

This handout is in two basic sections. The first (§2) is for the first-time user, who has either no computer experience whatsoever or who has worked only with Windows (which is almost equivalent to having no computer experience whatsoever). The second (§3) contains information that many people might find handy. Finally, §4 is a list of basic UNIX commands that everybody ought to know for working efficiently.

2. FOR THE FIRST-TIME USER ONLY

The best way, by far, to learn Unix is to sit in front of a computer for an hour or so and to try it out yourself. Once you master some basic commands, you’ll be amazed at the many useful things you can do.

2.1. Logging In

Our Sun computers use an operating system called Solaris. Under Solaris, you have a choice of desktop environments: *Open Windows* and the *Common Desktop Environment (CDE)*, which is the default. *Open Windows* is easier to use and takes up less memory. To use *Open Windows*, you must, before you log in, change the session to *Open Windows* under the Options menu.

2.2. Customizing the DeskTop

You can customize details of your desktop such as colors and fonts. To accomplish this, first place the cursor on the background and right-click the desktop; then click *properties*. A window should pop up allowing you to change the colors for your viewing pleasure. By clicking *Category*, you can change all sorts of things other than color, too.

As you can see by now, by right-clicking the desktop, a menu with a lot of useful commands will appear. Experiment!

---

\(^1\)Based on earlier versions by Anthony Truong and Jennefer King.
2.3. Directory Structure

The directory structure of Unix is organized in a hierarchical system, or by levels. Each user’s top level is the home directory in which you have full read/write/execute access. When you move throughout the directory system, you are moving from level to level. This directory is called /home/loginname. Figure 1 shows what a sample directory tree structure might look like. Here, home, loginname, and lab1 are directories. Notice that these directories are nested within one another; thus loginname is a subdirectory of home, and lab1 is a subdirectory of loginname.

Fig. 1.—A sample Unix directory tree.

Organizing your files in subdirectories is very important because over the semester you will be generating hundreds, if not thousands, of files. It makes sense to keep all files relevant to the first lab in a special subdirectory for that lab, called say lab1, and perhaps you might wish to have separate subdirectories under lab1 for the data, the writeup, and the analysis.

And specifying the permissions of accessibility is also very important. You might want the data to be accessible by everybody, both for reading and writing, but you certainly wouldn’t want the writeup to be accessible by everybody, especially for writing. And it makes sense to keep all your love letters in a separate file that isn’t accessible in any way by anybody else—including their recipient(s)!

2.4. Working with Directories

First, check to see where you are. At the Unix prompt, type the following:
ttauri% pwd
The command *pwd* stands for *present working directory*. The computer should return something along these lines: */home/loginame*. From this you know that you are in your *loginame* directory, which is located beneath the *home* directory.

To see a listing on what already exists in your directory, use the *ls* command. This gives you a listing of the subdirectories and files in the directory that you are currently in.

ttauri% ls
Depending on what’s in your directory, you’ll see different things. In general, directories are indicated with a “/”, as in *dirname/*, while files usually are just listed as *filename*. If you’ve just gotten your account, chances are there’s not much in your home directory. As you work with Unix more and more, it will balloon to monstrous proportions unless you create subdirectories and place the file appropriately.

Let’s create a subdirectory called *lab1* that resides just under the *home* directory. To accomplish this:

```
cd ~                           Make sure you are in your home directory (see below).
mkdir lab1                     create the directory */home/loginame/lab1*
```

Now check to see the contents of your directory, using the *ls* command. Note that *lab1* is listed as a directory, denoted as *lab1/*. Also notice something else—the directory that you’ve created is located in the directory that you were in when you typed the *mkdir* command. Make sure you know where you are when you create a new directory or file. (That’s why *pwd* can be so useful.)

To move into and out of directories, use the *cd* command. The command used to move into a directory below your current location is *cd dirname*. To move up from your present directory one level, enter at the prompt *cd ..* instead. So if you were in your *loginame* directory, to move up into the *home* directory you would enter *cd ..* at the prompt, and to move down into your *lab1* directory you would enter *cd lab1*. You can also move into directories by specifying the entire path—so if you wanted to get to your *lab1* directory you could type *cd */home/loginame/lab1*.

The ~ character is a shortcut to your home directory. Suppose you are at some arbitrary subdirectory, deep down in the structure; to get to your home directory, you’d type *cd */home/loginame*. But the shortcut allows the much easier *cd ~*. You can also use it as a shortcut to accessing your subdirectories. Suppose you want to get into your newly created *lab1* subdirectory, then you can type either *cd */home/loginame/lab1* or *cd ~/lab1*. Of course, if you know that you’re already in your home directory, you can just type *cd lab1*.

You can delete (remove) a directory (say, *lab1*) by first, deleting (removing) all the files in the directory (see below), and then typing

```
rmdir lab1
```
2.5. Creating and Viewing Files

Files can be created by using a text editor; it allows you to create a file and type in whatever you want. You should learn emacs. Suppose you wish to compose a love letter and call the file love1.

```
ttauri% emacs love1
```

A new x window will pop up with a bar on the top that has menu-driven commands. You can experiment with these. But you should also learn the basic keystroke commands because they work not only in emacs but also on the unix command line. We summarize them below in §3.5.

Suppose you want to quickly view this file without editing it; type either `more love1` or `less love1` [which would you rather have? in this case, less is more! (because the arrow keys work as you’d expect and you can more not only downwards but upwards)]. The first page of your file will appear, and you can either view the rest of the file a page at a time by hitting the space bar, or line by line using the Return button. To stop looking at the file, hit q or Ctrl-c and you will be returned to the Unix prompt.

2.6. Moving and Copying Files

Two other commands that you’ll find useful will be the copy and move commands. The copy command takes an existing file and copies its contents into a new file. The move command takes an existing file and moves it to a new file\(^2\).

```
ttauri% cp love1 love2
```

Here the contents of love1 are copied into a new file named love2; this is handy if you want to send the same letter to a different person, for example, because you can edit love2 to change the name. If you want to copy it to a different directory, just include the directory name as part of the filename, like

```
ttauri% cp love1 ~/lab1/love2
```

You can always specify the filename with a different directory name in this way. Similarly, you can move a file with

```
ttauri% mv ~/lab1/love2 ~/lab1/tex/love3
```

2.7. Removing Stuff... Yikes!

So far you’ve learned only how to create files and directories, but what about removing them? This can be done just as easily. The command to remove a file is simply `rm filename`. The `rm` command has options, one of which is to force you to confirm that you really want to remove the file; we include this option when we set up your default login files for new accounts. Be sure that you want to remove a file when you do this! Once it’s been removed, it’s gone for good. The same

\(^2\)Note the subtle difference here—using copy, you are left with two files, with move only one file remains.
goes for removing a directory. The command here is **rmdir dirame.** Always exercise caution when removing anything. You really should define an aliaseses for the following commands in your `.aliasfile` by entering the lines

```bash
alias rm "rm -i"
alias mv "mv -i"
alias cp "cp -ip"
```

these will force the system to ask you permission before overwriting anything.

Example: if you wanted to remove the file `love1`, first you’d get into the directory that contains that file, and then you would enter:

```
tauri% rm love1
```

Suppose that directory was `lab1` and you also wish to remove the whole directory. First, remove all the files from that directory; then to remove the `lab1` directory, move into the directory just above it and then type:

```
tauri% rmdir lab1
```

### 3. FOR EXPERIENCED USERS

#### 3.1. More About Listing Files, or the `ls` Command

If you want to see a listing of the files in your directory, an `ls` will give you all you need to know. But sometimes you have other questions too, such as: How big are my files? What’s the last file I worked on? This is where options can come in handy. Options can be included in an `ls` command to enable your files to be listed in a way that’s the most useful to you. Let’s say you wanted to know what the last file is that you worked on. You can use the `-t` option, and enter at the prompt `ls -lt`. This should give you a listing of your files from most recently accessed to least recently accessed; the command `ls -lrt` gives the same, but in reverse order. The procedure for using other options is the same. The following is a list of some of the options you can use with the `ls` command:

- `-a` Lists all of your files, including the dot, or hidden ones.
- `-l` Your files are listed in a single column output.
- `-s` The sizes of your files are listed.
- `-t` Lists your files from most recently accessed to least recently accessed.
- `-l` Long format; meaning that info such as permissions, owner, size, and date are included.

These are just a few of the options available. To get more info, use `man ls`. You can get more info on any command by typing in `man commandname`. Usually there’ll be more info than

---

3But in UNIX some commands still have the power to overwrite inadvertently. You’ll learn this much to your chagrin some day. Be sure to keep current copies of anything important in a different directory.
you’ll ever need on that specific command; but it’s definitely a good resource to keep in mind.

3.2. Remote Logging In

You can log in from home if your computer has the secure login software, called ssh2.

```
ssh2 rrylrae
```

If you’re logging in from home on an (ugh!) windows machine, you need to be able to use X windows; for this, run the program exceed before logging in. Better yet, use Linux.

3.3. Aliases and Shortcuts

Aliases are shortcuts that you can use in place of typing out a long command over and over again. You can define an alias on the command line; alternatively, if you want to define it permanently, you can define it by editing your .cshrc file (which resides in your home directory).

Here’s an example: suppose you want to force UNIX to check whether it will overwrite a file when you use the `mv` command and, also, to ask you about it. To do this, you use the `-i` option, and you redefine the command `mv` by typing

```
alias mv "mv -i"
```

If you type it in a window, it will apply henceforth to that window alone. If you include it as a line in your .cshrc file, it will always apply (after you login again). You can check the definition of an alias by typing

```
which mv
```

3.4. Permissions

Permissions are important for security and privacy reasons and are set with `chmod` and its options. Among the users, there are three groups: `user` (yourself), `group` (undefined here), and `world` ("other") (u, g, o). Among the permissions, there are three important ones: read, write, execute (r, w, x). `chmod` allows you to add, take away, and set exactly permissions for different users with the operators (+, -, =).

For example, to restrict your love letter `love1` from being read, executed, or written on by everybody but yourself, type

```
chmod go-wxr love1
```

You can apply permissions to a whole subdirectory instead of single file. For example, your datafiles are located in `~/lab1/data` and you wish to allow the world to read all your datafiles but not to write on them, so you type

```
chmod go+r,go-wx ~/lab1/data
```

You can check permissions with the `ls -l` command, for which the permissions are listed in the first 9 columns in three groups of three columns. The first group of three columns is the individual
user, the second the group, and the third the world; and within each group, the first is execute,
the second read, the third write. Try it!

3.5. X Windows, Command-Line Editing, and Emacs

You should always use X windows in UNIX because you can then both scroll backwards and
use command-line editing. The editor of choice is emacs, for several reasons; among these is that
it incorporates the same editing commands that work for command-line editing in X windows.
The most important editing commands are:

- **arrow keys** move the cursor as you'd expect.
- **Ctrl-d** deletes the character under the cursor.
- **backspace** deletes the character behind the cursor.
- **Ctrl-e** moves the cursor to the end of the line.
- **Ctrl-a** moves the cursor to the beginning of the line.
- **Ctrl-k** deletes the rest of the line.

Sometimes, when command-line editing, you inadvertently hit **Ctrl-s**; this prevents the cursor
from responding to your keystrokes. If you encounter this condition, type **Ctrl-q**, after which
things will work normally again.

In Solaris, you can customize any X window to your desires (e.g., fontsize) by putting the
cursor on the window, holding down the CTRL key and, simultaneously, holding down a mouse
button; each one provides different options. If you want to customize them permanently, edit the
*.openwin-init* file in your home directory.

3.6. Piping, etc: |, >, <

Piping directs the output of a command to the next succeeding command. For example,

```
ls | grep /
```

directs the output of the listing command to *grep*, which here selects all names containing the
string "/"; those are directories, so this gives a list of directories just under the current directory.

Normally the result of a UNIX command is written to the terminal for you to see. However,
you can direct the output elsewhere. For example,

```
cat love1 love2
```

writes the concatenation of the files *love1* and *love2* onto the screen, while

```
cat love1 love2 > loveboth
```

writes the concatenation into a new file called *loveboth*.

Normally the input to a UNIX command is expected to be from the terminal. However, you
can get the input from elsewhere. For example,

```
mail heiles < complaint.txt
```
mails the file complaint.txt to heiles; try it! (Do you think it will do any good???)
4. COMMON COMMANDS

Following is a list of some useful UNIX commands, which can be used as a quick reference.

```
passwd
man commandname
pwd
cd dirname
cd ..
mkdir dirname
rmr dirname
rm filename
cp file1 file2
mv oldfile newfile
cat file1 file2 > fileboth
which cp
history
!!
find dirame -name filename
find dirame -name "*love*"
ls -lrt
ls -lrt | grep /
du -k dirame
df -k
grep -il text file
lp filename
less filename
tail -40 filename
lq
cancel jobnum
top
ps -u username
kill processnum
```

- Useful for changing your account password.
- Gets info for you on a specific command.
- Shows your “present working directory”.
- Moves you into the subdirectory, below your present directory.
- Moves you out of a subdirectory into the directory above it.
- Creates a subdirectory named `dirname`.
- Removes a subdirectory named `dirname`.
- Removes a file named `filename`.
- Copies the contents of `file1` into `file2`. You are left with two files.
- Moves (or renames) `oldfile` as `newfile`.
- Concatenates `file1` and `file2`, writing them into the new `fileboth`.
- Tells the current definition of `cp` (which works for any command);
  gives a numbered list of the previous commands you’ve typed;
  typing `!number` repeats that command.
- Repeats the previous UNIX command.
- Finds all files with `filename` in and under the directory with `dirname`.
- Finds all files whose names contain the string “love”.
- Lists the files and subdirectories in the present directory.
- The `-lrt` gives a long format in reverse time order.
- Pipes the output to `grep`, which
  selects only those names containing “/” (which are directories).
- Tells kilobytes used by everything in `dirname`.
  Also handy for giving the directory tree structure.
- Tells kilobytes used and available on all disks.
- Searches the file for occurrences of the string `text`
  The `-i` ignores capitalization and `l` lists only the filename.
- Prints the file `filename`.
- Shows you the contents of the file named `filename` on one screen at a time;
  more flexible than `more`.
- Shows you the last 40 lines of the file `filename`.
- Displays the print queue.
- Removes the `jobnum` in the print queue. You must own the job
- Shows CPU usage, etc, for jobs on your machine.
- List the programs that `username` is currently
  running on the machine you are logged onto.
- Kills the process listed with `processnum`. You must own the process