USING UNIX

Introduction: UNIX is the name of the operating system used by our Sun computers and many other computers elsewhere. It acts as a kind of master of ceremonies, running programs, allotting space in memory, etc. Most of what is described in this section can be used on any system running UNIX.

The program that interacts with users is called a shell. The most common shell is called the C shell because parts of it work a lot like the C programming language. Some of the things described here are actually features of the C shell and will not work in other shells such as the Bourne shell. If you have never thought about shells, then you are almost certainly using a C shell. If you are sophisticated enough to run a shell other than the C Shell, you should know what kind of features are shell-dependent.

UNIX Commands: You use UNIX by typing commands. These are typed after the prompt. If you are dialing in from somewhere or are not using a window system, you just type. If you are using a window system, you type UNIX commands in a terminal-like window. These windows have names like Command Tool, Shell Tool or xterm.

There are basically two types of UNIX commands. In one type you ask the computer for information. For example, if you type

```
ls
```

the computer will respond by printing a list of your files on the screen.

In the other type of command, you ask the computer to do something. It doesn't necessarily need to print any response. These types of commands usually are asking for something to be done to a file. The basic form of these commands is `command` *filename*. These tells the machine to perform the action dictated by `command` on the file that is called *filename*. An example is:

```
rm paper.tex
```

which deletes the file named `paper.tex` from the system.

Many UNIX commands have options. These are preceded on the command line by a minus sign `-`. The basic form is: `command` `-`*option filename*, or just `command` `-`*option*. Two examples are:

```
rm -i paper.tex
ls -l
```

Getting Help: There is on-line documentation for most UNIX commands and programs. These are called manual pages or man pages. For example, to find out about the `ls` command type:

```
man ls
```

The man pages will be displayed using the program `more` described below.

Man pages contain lots of information. They tend to be rather technical. If you want a less technical introduction there are numerous introductory books on UNIX available at bookstores and libraries.

The UNIX File System: A lot of what you want do using UNIX involves the manipulation of files. In UNIX, groups of files are collected together in directories. The

directories are arranged in a tree. Directories that are contained in other directories are considered to be lower down in the tree. (This tree grows with its branches down!)

The very top of the tree is the root directory. When you login you find yourself in what is known as your home directory. It has the same name as your login name. For example, if you type `smith` to login, then your home directory is named `smith`. Your home directory is actually located fairly far down the tree. The list of directories from a file to the top of the tree is called its full path. An example of a full path is `/usr/local/maple`. In UNIX one writes a path using the "slash" `/` to separate a directory from the one above it in the tree. The directory that you are in is called the working or current directory.

What the Prompt is Telling You: Most people's accounts are set up so that the prompt is giving you information about where you are in the file system and what machine you are connected to. For example, the prompt

```
eastegg [Course] 138)
```

says that you are working on a machine called `eastegg`, and you are in a directory called `Course`. The `138` is giving you a label for this command line. This is useful for repeating the same command. See the section titled Repeating and Changing Commands below.

Ways to Refer to Files (`*, ~ , .., .`): The asterisk or wildcard `*` is very useful in referring to files. It matches any string of text. For example, `*.tex` refers to any files that end in `*.tex`. All the files in the current directory can be referred to by `*`. Thus

```
cp *.tex Papers
```

will copy all the files with the `.tex` extension into the directory called `Papers`.

Watch out! Doing stuff like `rm s*.tex` is good for removing a list of files called `stuff1.tex, stuff2.tex, ...` but it will also remove `special.tex` that you forgot was there. To be safe you can always do `ls s*.tex` to check what your wildcard is matching before you take action.

One dot `.` and two dots `..` refer to the current directory and the directory one level up, respectively. Thus, `cd ..` moves you one level up the directory tree. The tilde `~` is used to refer to home directories. For example, `cd ~` moves you to your home directory and `cd ~smith` moves you to the home directory of the user `smith`.

When you give a command that refers to a file or directory you can always give its full path from your home directory. For directories and files that are contained in your current directory you can just give the name. For example, if you are in a directory called *Papers* that is contained in your home directory and you want to move to the directory `Mail` that is also in your home directory, you can do `cd ../Mail` or `cd /Mail`.

Listing the Contents of a Directory (`ls`): The function `ls` gives a list of the names of the files in a directory. It can also be used to give the names of certain types of files. It has many options. Here are some examples.

`ls` List all files in the current directory
`ls *.tex` List all files that end in `.tex`
`ls -a` List all files including hidden ones that start with a dot.
`ls -s` Give the sizes of files in kilobytes.
`ls -l` Give the long form with permissions and modification date.

Moving, Copying and Removing Files (`mv, cp, rm`): The command `mv` moves (or renames) files. The simplest form of use is `mv` *oldfilename newfilename*. It will also move directories. If you use it in the form `mv` *filename directoryname* it will move the file into the named directory keeping its old name. To change its name you do `mv` *oldfilename directoryname/newfilename*

Copying files is similar to moving them but you use `cp` instead of `mv`.

Watch out! If you move a file and there already exists a file with your *newfilename*, that file will be destroyed. If you do not like this feature put the line

```
set noclobber
```

in your `.cshrc` file.

To remove a file you `rm` *filename*. On UNIX systems with shared file systems once you remove it it is gone. If you don't like this you can create a safer alias as instructed below. There are also regular backups made of files, but an ounce of prevention is worth a pound of cure.

Changing, Creating and Removing Directories (`cd, mkdir, rmdir`): You make a new directory via `mkdir` *newdirectoryname*. You can remove a directory using `rmdir` *directoryname*. To remove a directory, you must first remove all the files it contains.

To change directories to a directory that is contained in the current directory use `cd` *directoryname*. To move up one directory `cd ..` does the trick. To move to a directory on the same level, i.e. both the current directory and the desired one are both contained in the same directory, use `cd ../`*directoryname*.

Looking at the Contents of a File (`more, head, tail, cat`): The command `more` *filename* will display the named file page by page. To view the next page you hit the space bar. To see just the first few lines of a file use `head` *filename*. The last part of the file can be viewed via `tail` *filename*. The command `cat` *filename* will cause the entire file to be printed on the screen. This means that if the file is longer than a single screen full it will just scroll rapidly by. You can, of course, also look at a file in an editor like vi or emacs.

Printing Files (`lpr, lpq, lprm`): To print a text file you do `lpr` *filename*. To check the files waiting to be printed you type `lpq`. This will also tell you if something is wrong with the printer or if it is out of paper.

If you want to cancel a printing job before it is printed, first type `lpq`. Under the heading Job you will find a number for your printing job. If this number is 116 say, you remove the job by typing `lprm 116`.

Comparing and Concatenating Files (`diff, cat`): If you do `diff file1 file2` you will get a list of the differences between the two files.

To take two files and put them end to end in a new file you use `cat file1 file2 > newfile`. To put a file at the end of an existing file type `cat file1 >> existingfile`.

Checking Spelling (`spell, ispell`): The program `spell` will check the spelling of words in a file against a fairly large dictionary. For example, if you type

```
spell paper.tex
```

you will get a list of the words in the file `paper.tex` that could be misspelled. A problem with using this on a TEXfile is that TEXcommands come out as misspelled words.

One way out of this is to use the program `ispell` which is an interactive spelling checker. When the filename ends in a `.tex`, `ispell` ignores things that start with backslash. It will often give you suggestions about how to correct the spelling. The main things you need to know to use `ispell` are: `?` to get a list of commmands, `i` (= ignore) to keep a word as is, `r` to replace a word and the number of a selection to replace the word with that selection. When you tell `ispell` to ignore a word, the word will be placed in your personal dictionary (a file called `.ispell_words`). Words in this file will be treated as correctly spelled in all future spelling checks. If you want to ignore a word and *not* put it in your `.ispell_words`, you can use `a` instead of `i` .

Searching For and Inside Files (`find, grep`): The command `find` is used to find files with various attributes. It has many options. This makes it somewhat awkward for just searching for a file by name. For example, the command

```
find .  -name "*.tex" -print
```

finds all the files in the current directory and any directories below that have a name which ends in `.tex` and then prints the result of the search at the terminal.

The program `grep` looks inside files for specified text. For example,

```
grep Diophantine paper.tex
```

will search the file `paper.tex` for any occurrences of the text "Diophantine" and then then print those lines at the terminal. If you are searching in more than one file as in `grep Diophantine *.tex` it will print the name of the file in which the line was found.

Compressing and Encrypting Files (`compress, uncompress, zcat, crypt`): You can reduce the storage space needed for a file by compressing it. The command is `compress` *filename*. This takes the named file, compresses it and then puts the result in a file called *filename*`.Z`. To uncompress it you use `uncompress` *filename*`.Z`. You can look at a compressed file without uncompressing it via
`zcat filename | more` (the vertical bar `|` is called a pipe and will be discussed below).

You encrypt a file using something similar to :

```
crypt key < clear.file > encrypted.file
```

This takes the contents of the file named `clear.file`, encrypts it, and puts the result in the file named `encrypted.file`. (The `>` and `<` are redirects and will be discussed below.) The word you use in place of `key` is the key needed to decrypt the file, so remember it! When you use `crypt`, it *does not* remove the file `clear.file`. If you want to remove it, you use the standard `rm clear.file`. To decrypt the file you use:

```
crypt key < encrypted.file > clear.file
```

Where in place of `key` you use the same key word you used to encrypt the file.

Changing the Permissions on a File (`chmod`): The permissions on a file control who can read and change the file,. This is described in full detail in section on File Permissions.

Getting Information (`whoami, pwd, finger, setenv, jobs, ps, history`): The command `pwd` prints the name of your current (or working) directory on your screen. Typing `history` will give a list of your most recent commands. You can repeat a command using `!` For example, `!12` repeats the command that is numbered 12 in your history list.

Typing `whoami` will tell you whose account you are currently using. The command `finger` will get you information on all users currently logged onto your system. You can also find out about a single person. For example, `finger smith` will give you information about the user smith. Finger will make a guess for you, so you can use it to find out email addresses. Thus, `finger smyth` will probably give you information on smith whose name you had forgotten how to spell. You can also use finger to ask about people elsewhere on internet using their email address as in `finger boyland@symcom.math.uiuc.edu`. This doesn't work 100% of the time.

A list of your environmental variables can be obtained via `setenv`. The commands `ps` and `jobs` tell you the current processes and jobs that are owned by you. To get all processes try `ps -aux`.

Killing Jobs and Processes (`kill`): To kill or terminate a job you type `kill %`*number* where *number* is the job number as reported by `jobs`. To kill a process, use `kill` *processnumber* where the *processnumber* is reported by `ps`. Watch out! Don't kill a process unless you know what you are doing.

Repeating and Changing Commands (`!, ^`): You can repeat the command you just gave using `!!` The command !*number* and !*text* repeat the command labeled by *number* on your history list and the last command that stated with *text*, respectively. Thus, for example, `!23` repeats the command labeled 23 on the list reported by `history`. The command, `!r` repeats the last command that started with an "r", for example a `runtex` command.

There is a way to make substitutions in previous commands using the `^` operator. If you had typed `mw file1 file2` you will get a message that `mw` is unknown. What you really wanted was `mv`. If you then type `^w^v` it will take the `w` from your last command and change it to a `v` and thus execute `mv file1 file2`

Command Aliases (`alias`): You can make your own names for commands or combinations of commands using `alias`. The usual way to use it is in the form `alias` *name ´commands ´*. For example, if you want to be asked if you really want to delete a file you can use:

```
alias del ´rm -i´
```

After doing this, every time you type `del file`, it is the same as `rm -i file`. When you set an alias by typing it at a prompt it is only valid for that session and that window. If you want to have the alias available every time you can put the alias in your `.cshrc` file as described below.

To remove an alias you use `unalias` *name*. If you have changed a command name into an alias you can use its original form by preceding it with a backslash. Thus, if you have set `alias rm ´rm -i´`, you can still use the built-in remove by using `\rm`

If you type just `alias`, you get a list of all the current aliases. Because there are standard aliases included in your `.cshrc` file you will find that you have aliases even if you

never created them.

Pipes and Redirects (`|`, `>`, `>>`, `<`): Many UNIX programs take input, transform it, and then give output. This is usually set up so that the input comes from a file. If no filename is given, input comes from what is called *standard input*. The output goes to *standard output*. Almost always, standard input is the terminal keyboard and standard output is the terminal screen.

Often you want to control the input and output of programs. This is done with pipes and redirects. What a pipe does is connect standard input to standard output. In other words, it takes the output of one command and makes it the input to the next command. For example,

```
ls | more
```

takes the output from `ls` and sends it to the program `more` which displays it page by page.

The redirects work similarly, but they connect standard input or standard output to a file. The greater than sign `>` sends the output of the previous command to a file. For example,

```
ls > file1
```

sends the result of `ls` (*i.e.* a list of the files in the directory) to a file called `file1`. The double greater sign `>>` is similar but it appends the result to the file.

The less than sign `<` takes the file that is after it and uses its contents for input to the command that precedes it. An example is

```
maple < commd.in > commd.out
```

which takes the contents of the file `commd.in` and sends it as input to the program `maple`. After `maple` processes the file, the result is put in the file `commd.out`.

The command `cat` has a special role in all this. It takes a file and sends it to standard output. Thus `cat file` sends the file to the terminal screen and `cat file1 >> file2` puts the contents of `file1` at the end of `file2`. You can put a group of files together in a single file using

```
cat file1 file2 file3 > newfile
```

You can combine many commands, pipes and redirects. For example,

```
ps -aux | grep idraw > idraw.pc
```

lists all the processes running and pipes this result to `grep idraw`. This looks for all the lines that contain the word `idraw` and then sends this result to a file named `idraw.pc`. One of the main organizing principles of UNIX is to have simple commands that can be easily linked together in various ways to get powerful results.

Startup Files (`.login, .cshrc, .mailrc, ...`): There are a number of files that control how various aspects of UNIX will work for you. You can put commands into these files and thereby "customize" your environment. The filenames of these files often start with a dot. These type of files have various names like, "dot rc" files, startup files, etc. Files that begin with a dot are not listed with the standard `ls` command, so they are sometimes also called "hidden" files. You can get a listing of all these files using `ls -a`

The `.login` file is read when you login. It controls the messages you get initially, sets the terminal type, etc. The `.logout` is analogous but is read at logout.

Most of the things you will probably want to do involve putting things into your `.cshrc` file. Commands in the `.cshrc` file are executed every time you start a C shell. If you want to have various aliases available whenever you are logged in, you put the `alias` commands in your `.cshrc` files.

The `.mailrc` file controls how your mailreader works. The most common use for this file is for mail aliases. These have the form:

```
alias mike michalm@math.princeton.edu
```

```
alias msgdyn msgs, bradley@adx.adelphi.edu, pkoch@ccmail
```

After you put these in your `.mailrc` file and then start your mail tool, you can send mail to `mike` and it will go to `michalm@math.princeton.edu`. The next example of `msgdyn` shows how you can include several people in an alias. Their addresses should be separated by commas and they should be on a single line. These aliases will work with any mail system you use from your account, eg. regular mail, emacs mail, Xmail, SunView mailtool, OpenWindows mailtool, etc.

Window systems also have startup files, as do programs like Maple and Mathematica. You need to look at the manuals for each program to learn how to work with these files.

Here are some examples of aliases. They illustrate many of the things covered in this section on UNIX. The one new thing is the `\!*`. It is used in an alias to refer to the stuff you type on the command line after the command.

```
alias lo        'logout'
alias h         'history'
alias j         'jobs -l'
alias m         'more'
alias ls        'ls -F'
alias rp        'runtex -preview'
alias rt        'runtex'
alias lls       'ls -sAl \!* | sort | more'
alias lsl       'ls -l | more'
alias px        'ps -aux \!* | sort +6 -7 +0 -2 | more'
alias e-math    'rlogin -l e-math e-math.ams.com'
alias ml        'grep < ~/.mailrc'
alias stars     'telnet 129.49.200.4 7002'
alias rm        'mv \!* ~/GARBAGE'
```

With the last one you could, for example, put the line

```
\rm ~/GARBAGE/*
```

in your `.logout` file.