UNIX At Smith

CONTENTS

Introduction 1
Logging Into grendel........... 1
Commands 3
Files 7
Pipes and Redirection 9
Job Control 10

Smith College
Department of Computer Science
August, 1993
Revised January, 1994

Introduction

This document introduces basic commands that can be used in a UNIX operating system. It focuses on commands helpful to Computer Science students using UNIX on "grendel," a Sun SPARCstation owned by the Department of Computer Science. Some of these commands have been specially designed for course accounts; however, most of the commands described below are applicable to any system running UNIX.

UNIX is an *operating system* which is a collection of programs that acts as an interface between a computer and a person using that computer.

An operating system does several jobs:

- It controls access to the computer by asking for a **login name** and a **password**.
- It keeps the contents of each user's account separate from those of others.
- It automatically gives each user an equal share of the computer's time (this makes UNIX a time-sharing system).
- It gives each user access to programs—text editors, language processors, printers, games and so on.
- It manages communications with other computers at Smith and elsewhere.

Before you start using UNIX, take a moment to look at the keyboard. A special key that is located near your left pinky is the one labeled CTRL. This is the **control** key. The control key lets us type **control characters**. To use the CTRL key, hold it down while you type a single letter. Control characters are interpreted by UNIX as commands rather than as text.

CTRL-U deletes the line you're currently typing in.

CTRL-C kills almost any program that is running (Emacs is one exception)

CTRL-D acts as an "end-of-file" character in many applications, including mail.

CTRL-S freezes the screen.

CTRL-Q unfreezes the screen.

Note: A key to avoid is the LOCK (or CAPS LOCK), the capital letter lock key. A light goes on in the upper right corner of most keyboards to indicate that CAPS LOCK is on. To turn it off press CAPS LOCK again. This is important because commands in UNIX are case sensitive; for instance, the command "more" is not the same as the command "More".

Logging Into grendel

To use grendel, you must first reach it through the campus LAN (local area network) and then log in. If you are working from a networked PC in a lab, you must first log in to the PC network and use Telnet to connect to the SPARCstation. If you are working from your own PC via a dataphone or modem, you should follow directions included with your dataphone or Kermit documentation for reaching the campus LAN.

In both cases, be sure to specify **grendel.csc** as the node to which you wish to connect. (The ".csc" at the end is necessary.) In lab networks, if you do not specify a node, you will be attached to Smith, the main academic VAX. When connecting from your room or off-campus, type **connect grendel.csc** when you reach the LAN prompt.

Once you reach grendel, follow the directions below to log in.

Step 1. At the `login:` prompt, type your username exactly as it appears on your account form, then press ENTER.

If you make a mistake while typing your login name, type CTRL-U to erase the entire line or press DELETE to backspace over single characters.

Step 2. At the `password:` prompt, type in your password, then press ENTER.

Nothing appears on the screen as you type your password; this is to keep it confidential.

If you have typed your username or password incorrectly, you receive an error message and another `login:` prompt. Repeat the steps above carefully to login successfully.

Step 3. Once you are successfully logged in, the computer will stop at the following line:

```
TERM = (vt100)
```

This is a question. UNIX wants to know what kind of terminal you are using so that it can customize its operations to take best advantage of your terminal. Pressing Enter or typing "vt100" will almost always work and you will see the following lines:

```
Erase is Delete

Kill Line is Ctrl-U
```

The two lines tell you what the backspace and erase characters are.

Once you have the "1 grendel(111a-bz) %" prompt, you are truly in your UNIX account. The number tells you how many commands UNIX has gotten from you; the "%" is the UNIX prompt character.

For simplicity, only % will be used in the examples that follow.

Changing Your Password

This step describes how to change your password. You need to do this the first time you log in. This is because the password you used for your first login is on your account sheet. You need to change it to make sure that it is entirely confidential.

Step 1. Type:

```
% passwd
```

Changing yp password for 111a-bz

Step 2. Type in your current password and Enter at

```
% Old password:
```

Step 3. Type in a new password and Enter at

```
% New password:
```

Step 4. Type it in again to verify it when you see

```
% Retype new password:
```

Note: The new password should be six to eight characters long. If you forget your password, talk to your instructor or to a consultant at the Computer Information Center.

Chfn

Chfn lets you input information about yourself for use by the Finger command (discussed below in section Commands). The command prompts you for your real name, office number, office phone and home phone. The default values are provided in brackets; simply pressing Enter without typing anything else causes the default value to be used.

```
% chfn
Name:[111a Student]     <type your name and press enter>
Office number:[]        <type your office number and press enter>
Office phone:[]         <type your office phone number and press enter>
Home phone:[]           <type your home number and press enter>
```

After you are done, type **finger** ***yourusername*** to see the results.

Logging Out

To leave UNIX, simply type:

```
% logout
```

Commands

In this section, some of the basic UNIX commands will be explored. Log in again as before with your account name and <u>new</u> password.

```
login: 111a-bz
password:
Last login: Sat Aug 20 22:36:11 on ttyp1
...(other information)...
******* Welcome to Computer Science at Smith!  *******
TERM = (vt100) <return>
Erase is DELETE
```

```
Kill line is ctrl-U
12 grendel(111a-bz) %
```

Notice that

- You've started to get the Last login message.
- You may also find that the number at the beginning of the line is greater than 1. For convenience, most accounts are set up to 'remember' the prior session's last 32 commands.

When you type a command, UNIX looks for a program with that name. If the program is found, UNIX runs it. If it doesn't, UNIX prints the error message

```
command not found
```

Following are some commands to try out.

Date

Date tells you what the date is.

```
% date
Tue Sep  4 10:27:33 EDT 1993
%
```

Whoami

Whoami tells you who you are, i.e., your account name.

```
% whoami
111a-bz
%
```

Mail

Mail is one of the more useful and entertaining commands. Try sending yourself a letter:

```
% mail 111a-bz
Subject: lunch
We simply must do lunch sometime...
--Mary
ctrl-D
%
```

The ctrl-D (typed by holding down the ctrl key and D at the same time) ends the letter and sends it. UNIX delivers mail to yourself very quickly. To read mail, just type mail without an address:

```
% mail
Mail version 2.18 5/19/83. Type ? for help
"/usr/spool/mail/111a-bz": 1 message 1 new
>N   1 111a-bz   Fri Sep 9 09:55   6/186   "Lunch"
&
```

The ampersand (&) is the mail program's prompt so that you won't confuse it with the UNIX prompt. To read a letter, type its number. In fact, just typing return will always print the next letter.

Some of the commands you can give at the ampersand prompt are:

& ?	ask for a list of commands
& d	delete the current letter
& 1\	read message number one
& R	reply to the current letter's sender [NOTE: upper case "R", not "r"]
& q	quit out of mail
& s *num file*	save message number *num* in a file named *file*

When you quit out of mail, the following message is displayed:

```
Held 1 message in /usr/spool/mail/111a-bz
%
```

If you had typed the command "**s 1 mbox**" at the "&" prompt, your message would have been saved in a file named mbox. Mail stays in the mailbox unless you save it to a file, but deleted mail is lost forever. Separate documentation is available for the mail command.

Who

Who or just plain w gives you a list of other people who are logged in and what they are doing.

```
% w
10:34am up 15 days, 23:18, 4 users,   load average:      4.12, 4.10, 4.15
user      tty from              login@    idle   JCPU   PCPU     what
orourke   04 UB1158             8:39am    8      57     3        -csh
spauster  p0 kate.smith.edu     10:34                            w
ssami     p1 131.229.26.36      5:56pm    1:17   1:17   2        -csh
```

The command **w** also gives you some information about the state of the machine itself. The first line tells you how long the system has been up and what the **load average** is, that is, how heavily loaded the system is.

Finger

Finger gives you more specific information about a user. It can be a login name or a real life name.

```
% finger thiebaut
Login name: thiebaut                        In real life: Dominique Thiebaut
Office: McConnell, x3854                    Home phone: xxxxxxx
Directory: /usr/faculty/thiebaut            Shell: /bin/csh
Last login Wed Aug 18 08:38 on ttyp7 from 131.229.65.161
Project:
Plan:
```

You can also use finger to learn about users on other systems using "@systemname". For example,

```
% finger @kate
```

tells you who is logged onto the academic VAX computer named "kate".

Chmod

Chmod allows you set permission for groups or individual users to read, write to and/or execute files. (If this is a course account, your "group" is the set of all student accounts in the course, together with the instructor.) For example, if you want to grant reading rights of a file, say fileA, to the user (u), owner(o) and group(g), type

```
% chmod guo=r fileA
```

Now if you do an ls (more on ls in the section on Files) on fileA, you'll see the following:

```
% ls -l fileA
--r--r--r-  1  111a-bz  143    Aug 18 9:45:32   fileA
```

The r's indicate that reading permission has been granted to the user, owner and group. To find out more about the other options possible, refer to the man pages, described below.

Man Pages

The online manual pages on UNIX can be an invaluable resource at your fingertips. You can look up references to any command by typing,

```
% man command
```

The -k option is very handy when you don't know the exact command. Say you are looking for a file related command. Type

```
% man -k file
```

It will list one-line references to all file-related commands. For other options, type **man man**

Files

Computers store information in **files**. By now, you probably have a few files in your account. Log in and try out some commands that help you manipulate files.

Ls

Ls gives you a listing of the files in your account.

```
% ls

.cshrc      .logout     .project    stuff

.login      .plan       mbox        things

%
```

Stuff and things are just made up names. Your files will probably have different names. File names that begin with a period like ".plan" are **hidden files** which you can use to customize your own environment. (Sometimes you must type "**ls -a**" in order to have the hidden files listed along with your other files; this depends on the default file display style defined on the system.)

More

More allows you to print the contents of your file to your terminal one screenful at a time. Suppose you had created the file mbox as described above in the description of the mail command. To view it, you could use:

```
% more mbox

From 111a-bz Fri Sep 9 09:55:54 1993

Subject: Lunch


We simply must do lunch sometime...

--Mary

%
```

File mbox is short, but if it were longer, it would print one screenful of the file at a time. The command more will print a brief command summary if you type **h** while using more.

Cp

Cp lets you copy a file.

```
% cp fromfile  tofile
```

Fromfile is the file that you are copying and *tofile* is the name of the copy. For example, to make a copy of mbox and call it message, type

```
% cp mbox message

%
```

Running the ls command again shows you that you have a new file. Some things to know about naming files:

- File names cannot have spaces in them
- Try to use names that are long enough to be meaningful but short enough to type. If a name really needs to be two words long, you can use a period to connect the two words, e.g., **old.homework**
- If you try to copy a file without supplying a file name, you get an error message along with a terse example of proper usage.
- If you try to give a new file a name that is already being used for another file, UNIX asks you if you really want to overwrite and, in the process, destroy the existing contents of the file.

```
% cp mbox stuff
overwrite stuff? n
%
```

Only the response **y** will overwrite the existing file.

Rm

The rm command removes files. In class accounts, the rm command has been *aliased* to a modified version of the command that always asks you to confirm that you really want to remove the file. For example,

```
% rm mbox
remove mbox? y
% ls
.cshrc      .logout     .project    stuff
.login      .plan       message     things
%
```

The file mbox is not listed anymore.

Mv

Mv moves a file, that is, it renames it. Like **cp**, it must be given two file names:

```
% mv oldname newname
```

For example, in making a copy of mbox named message and then removing mbox, you took a roundabout route to accomplish what could have been done with just one command:

```
% mv mbox message
```

Pipes and Redirection

At last, you reach the features that made UNIX famous. To understand them, though, you have to know what two words mean:

Input is the information or data that goes to a program, usually from your keyboard. For instance, the mail program's input is the letter that you type in.

Output is the information or data that a program produces, usually on your terminal screen. For instance, the output of the date command is the date printed to the screen.

Following are three special UNIX symbols that you also need to be familiar with, the two redirection arrows and the pipe.

Redirection arrows,

Redirection arrows, '<' and '>', redirect input and output to and from other places besides the default keyboard and terminal.

> means "command or program output should go to" a file that you specify.

< means "command or program input should come from" a file that you specify.

Generally, we'll use files to be the alternative source of input or output. Here are examples.

```
% date > today
% more today
Tue Sep  4 14:07:15 EDT 1990
```

As you can see, nothing printed on the screen when we typed the date command. Instead, the output from date was redirected to the file *today*, as the more command revealed.

An example of input redirection would be:

```
% mail 111a-bz < today
%
```

This says "mail 111a-bz a letter but get its contents from file *today*." So you don't get a chance to type the letter in yourself.

Pipes

Pipes, '|', are used to channel the ouput of one program as the input of another. **a** | **b** means 'the output of command **a** is the input of command **b**'. For example, to send 111a-bz a listing of the files in our home directory, we could use:

```
% ls | mail 111a-bz
%
```

Pipes and redirection can be used to link together a sequence of special-purpose tools into a meaningful command. For example, suppose you want to have a file called Wake that just happens to have the complete text of *Finnegan's Wake*. The command

```
% deroff -w Wake | sort | uniq | wc -w > WakeWords
```

first breaks the file into one-word lines with

```
deroff -w Wake
```

then pipes that output to a sorting program that alphabetizes the words with

```
| sort
```

then pipes that output to a program that supresses duplicated words with

```
| uniq
```

then pipes *that* output to a program that counts the words with

```
| wc -c
```

and finally, saves the final result, the number of different words in file Wake, in a file called WakeWords with

```
> WakeWords
```

Whew!

A few more details that you should know:

- Your environment has been set up so that you won't accidentally clobber an existing file by inadvertently sending output to it. Rename or remove the file to free its name for reuse.

- Three additional options of redirecting output are:

  >! redirects output to a file even if the file already exists.

  >& makes the output file include error messages that would ordinarily print on the screen.

  >> appends or adds to an existing file without affecting its current contents.

Job Control

Recall that UNIX is a time-sharing system. Many people can use the system simultaneously because UNIX keeps each user's jobs separated. In fact, UNIX goes one step further. It lets each user run more than one job. This is termed **Job Control.**

To understand job control you have to understand the difference between **background** and **foreground** jobs.

- A foreground job is one that is running 'on the screen' so to speak. It may be an interactive job like mail or it may just do its thing and end, like ls. Most jobs work to the exclusion of all else until done.

- A background job runs independently of whatever else you are doing, without apparent input or output. You can continue to issue commands at the UNIX prompt and do other work while the background task is carried out "invisibly."

Tools of Job Control and Examples of Usage.

&	runs any new command in the background
ctrl-Z	temporarily suspends any job that is running in the foreground
fg	brings a job from the background into the foreground
bg	makes the job that just suspended start to run in the background
jobs	tells you what jobs are running in the background
kill	stops a job that is running in the background

Q. How do you use **&** to put a job in the background?

A. Suppose you want to mail a map of the Paris sewers to Cosette. Unfortunately , the system is heavily loaded. The job can be put in the background like this:

```
% mail cosette < sewer.map &
[1] 1686
%
```

UNIX tells you the **job number**, here [1], and its overall **process number**, here 1686. More on job and process numbers below.

Q. How do you use **ctrl-Z** to suspend a currently running job?

A. Say you want to temporarily suspend the mail job so that you can do something else, perhaps take a quick look at a file called *crust.locations*.

```
% mail cosette
Subject: Crusts of bread
I forgot to tell you -I know where you and your father can find some crusts
of bread. They are in
ctrl-Z
Stopped
%
```

IMPORTANT PIECE OF ADVICE: Before suspending an editing job with ctrl-Z, save the file you are working on. A common error made by beginners is to retype the editor command (e.g., "emacs"), forgetting that a copy of emacs is already running in the background. This could lead to the following situation:

```
% jobs
```

```
[1]   stopped        emacs boring
[2]   stopped        emacs boring
[3] - stopped        emacs boring
[4] + stopped        emacs boring
%
```

There are four different versions of the editing session for the file "boring"! Which one is the most recent? If you just keep typing "fg" to bring each of these back to the foreground, then save it and exit Emacs in each copy, you'll end up saving the *earliest* version of the file and losing all subsequent changes that you made. (This is because jobs are brought to the foreground in the reverse order in which they were created.)

Of course, the best thing to do is to avoid starting up a new copy of the editor, but it's sometimes easy to forget. If you always take the precaution of saving your file *before* you issue the CTRL-Z command, you can use the following recovery procedure to get rid of those extra jobs: bring each job into the foreground with fg and then quit the editor without saving the file. Ask a staff person for details if you don't understand this. It's an important point.

Q. How do you use **fg** to bring a job into the foreground?

A. The mail we suspended above can be brought back into the foreground so that you can finish the letter like this:

```
% more crust.locations
crust.locations: no such file or directory
% fg
Oh no Cosette. They're gone!
    --111a-bz
ctrl-D
%
```

Note: ctrl-D ends the letter as usual.

Q. How do you use **bg** to cause a suspended program to start in the background?

A. Suppose you started to compile a Pascal program and then got tired of waiting around for it. You can suspend the job and then put it in the background like this:

```
% pi homework.p &
ctrl -Z
Stopped
% bg
[2] pi homework.p &
```

```
%
```

Q. What do you use the **jobs** command for?

A. You can find out all your currently active job numbers with the jobs command:

```
% jobs
[1] - running          mail cosette < sewer.map &
[2] + running          pi homework.p
%
```

Jobs can be referred to by number as long as you precede the number with a percent sign. This is demonstrated below. In the examples, assume n to be the job's number.

```
% fg %n
```

This brings job n into the foreground.

```
% bg %n
```

This puts job n into the background.

Q. How do you **kill** a job?

A. Killing a job is equivalent to bringing it to the foreground and then entering a **ctrl-C**, the standard UNIX job-kill character. If a job won't die, kill it like this:

```
% kill -9 process number
```

But this is a special case you probably won't have to deal with.

One last point about job control: because it's such a convenient feature, people occasionally forget that jobs have been suspended when they try to log out. This error message appears:

```
% logout
There are stopped jobs.
%
```

If this happens, use the jobs command to find out what's going on and the fg, bg and kill commands to deal with the suspended jobs appropriately. You can log out while the job is running in the background, by the way. It will finish normally even though you aren't logged in.