

Using UNIX

*Academic Computing Services
B66 Davis Hall
(817) 272-2208*

Table of Contents

ACS DOCUMENTATION CONVENTIONS	1
INTRODUCTION.....	2
LOGGING IN	2
AFTER LOGGING IN.....	3
CHANGING YOUR PASSWORD	4
LOGGING OUT.....	4
UNIX SYSTEM BASICS.....	5
SHELLS	5
SHELL COMMAND SYNTAX.....	5
GETTING HELP	6
VIEWING FILES	7
The cat Command	7
The more Command.....	8
The less Command.....	9
The head Command	10
The tail Command.....	10
INPUT/OUTPUT.....	10
PIPES.....	12
MANAGING FILES	13
MANAGING YOUR FILES WITH UNIX	13
The ls Command	13
The rm Command	13
The cp Command	14
The mv Command.....	14
The wc Command	15
The cmp Command	15
The diff Command	16
The grep Command.....	17
The sort Command.....	17
PRINTING FILES	18
MANAGING DIRECTORIES.....	19
UNDERSTANDING DIRECTORIES.....	19
USING WILDCARDS AND SYMBOLIC SUBSTITUTION	20
BASIC DIRECTORY COMMANDS	20
The pwd Command.....	20
The cd Command	21
The mkdir Command	22
The cp Command	22
The rmdir Command.....	23
PROTECTING FILES	24
The chmod Command.....	24
Checking Existing Permissions.....	25
USING THE VI EDITOR	26
GETTING STARTED.....	26
OPERATING MODES	27
<i>Text Input Mode</i>	27

<i>Command Mode</i>	27
COMMANDS	27
<i>Aborting Commands</i>	27
<i>Saving Files</i>	27
<i>Cursor Movement Commands</i>	28
<i>Delete Commands</i>	29
<i>Insert Commands</i>	29
<i>Search Commands</i>	30
<i>Change Commands</i>	30
<i>Pasting Commands</i>	31
<i>Marking Text</i>	31
Moving Marked Text	31
<i>Yank Command</i>	31
Copying Yanked Text	32
<i>Reading Other Files Into Current File</i>	32
<i>Repeating Commands</i>	32
<i>Joining Lines</i>	32
MULTITASKING	33
UNDERSTANDING FOREGROUND AND BACKGROUND PROCESSING	33
RUNNING A BACKGROUND JOB	33
The nice Command	33
The jobs Command	34
The fg Command	34
The stop Command	35
The kill Command	35
MISCELLANEOUS COMMANDS	37
The history Command	37
The alias Command	39
The cal Command	40
The date Command	40
The who Command	41
The quota Command	41
The limit Command	41
SHELL AND ENVIRONMENT VARIABLES	42
UNDERSTANDING SHELL AND ENVIRONMENT VARIABLES	42
The set Command	43
The echo Command	44
The printenv Command	45
The setenv Command	45
THE .LOGIN AND .CSHRC FILES	46
USING X WINDOWS SYSTEMS	47
USING X WINDOWS	47
<i>Using X Windows With a Workstation</i>	47
<i>Starting X Windows on an NCD X Windows System</i>	48
<i>Accessing UNIX Documentation Using X Windows</i>	48
THE BASICS OF DOING STATISTICS ON OMEGA FROM THE VIEWPOINT OF A VM/CMS USER . 50	
SAS AND SPSS	50
<i>Editors on OMEGA</i>	50
<i>Using SAS on OMEGA</i>	50
<i>Using SPSS on OMEGA</i>	51
<i>Using the IMSL Libraries on OMEGA</i>	51

RUNNING PROGRAMS ON UNIX	52
COMPILERS ON THE UNIX SYSTEMS	52
SAMPLE PROGRAMS	53
Example 1 n Sample C Program	53
Example 2 n Sample C Program	54
Example 3 n Sample C++ Program	55
Example 4 n Sample FORTRAN Program	56
Example 5 n Sample Pascal Program	57
<i>USING THE UTA UNIX SYSTEM.....</i>	58
<i>UNIX COMMAND CHART</i>	58

ACS Documentation Conventions

Throughout ACS documentation, commands you need to enter will appear in bold type:

This is a command that needs to be entered.

Items that you need to substitute information for will appear in bold-italic type:

This is something that you need to substitute information for.

Response prompts that request confirmation or information from you will appear in parentheses:

"This is a request for confirmation or information from you."

Titles of buttons, dialog boxes, and windows will appear in italics:

This is a button, dialog box, or window title.

Below is a list of standard key abbreviations used in ACS published documents.

<code><return></code> or <code><enter></code>	Return or Enter key
<code><backspace></code>	Backspace key
<code><tab></code>	Tab key
<code><esc></code>	Escape (Esc) key
<code><alt></code>	Alt key
<code><ctrl></code>	Control (Ctrl) key
<code></code>	Delete key
<code>f1, f2, ... f12</code>	Function keys
<code><pg-up></code>	Page-Up key
<code><pg-dn></code>	Page-Down key
<code><ctrl-i></code>	Press and hold the Ctrl key, while pressing the key i indicated.
<code><alt-i></code>	Press and hold the Alt key, while pressing the key i indicated.
<code><shift-i></code>	Press and hold the Shift key, while pressing the key i indicated.

Introduction

The UNIX operating system was developed by Ken Thompson in 1969 at AT&T Research Division at Bell Laboratories. In 1974, several universities acquired the source code of the UNIX operating system. The UNIX operating system includes many of the features found in the Berkeley Software Distribution (BSD) V4.3.

This document presents some basic UNIX commands and concepts. If you need more information about a particular topic, please use the **man** command or the **dxbook** (or **answerbook**) command (for information on the **man** command, see **Getting Help**; for information on the **dxbook** (or **answerbook**) command, see **Accessing UNIX Documentation Using X Windows**).

In all the examples given in this document, items in **bold** represent what you type at the keyboard. Keys that you press are represented in angle brackets. For example, the Enter key is represented by the symbol **<RETURN>**. If two keys need to be pressed at the same time, they will both be included inside the angle brackets. For example, **<CTRL-L>** means to press and hold the control key while pressing the **L** key. The standard input is the terminal and standard output is the screen. This document is not intended to be a complete guide to UNIX. Many of the commands discussed in this document have several options not included in the discussion. For a complete description of UNIX commands and options, please use the **man** command or the **dxbook** (or **answerbook**) command (for information on the **man** command, see **Getting Help**; for information on the **dxbook** (or **answerbook**) command, see **Accessing UNIX Documentation Using X Windows**).Using a UNIX System

Logging In

To use a UNIX system, you must have a valid User ID and password on that system. You can log in to a UNIX system from facilities on campus or from home.

When you log in, you'll need to type your User ID and password. When typing your User ID or password, be sure to type uppercase letters and lowercase letters as such. Case is significant. When typing your password, notice that the letters don't appear on your screen as you type them. This is for your account's security.

If you make a mistake while typing your User ID or password, you must use the <delete> key to correct it. The <backspace> key will not work.

After Logging In

After entering your password, the system checks both your User ID and your password. If either your User ID or password is invalid or incorrect, then you'll see the following error message ending with another login: prompt:

```
login: jsmith <RETURN>
Password: xxxxxxxx <RETURN>
Login incorrect
login:
```

After logging in to a UNIX system, you'll see three things: the date and time of your last login, any sign-on messages from the system's administration (optional), and finally the system prompt.

In the following example, the date and time of the last login (no sign-on message appears) and the system prompt are displayed:

```
login: jsmith <RETURN>
Password: xxxxxxxx <RETURN>
Last login: Fri Mar 03 10:07:03 from 129.107.1.100
%
```

When you log in to GAMMA or OMEGA, you'll see a prompt that includes that system's name. In this document, however, we'll use the generic C shell prompt, %. When you see your system's prompt, you are ready to enter UNIX commands.

The UNIX operating system is case sensitive. You must enter UNIX commands in lowercase letters just as you must enter your User ID in lowercase letters.

Changing Your Password

Your password on a UNIX system must follow these rules:

Your password must be at least 6 characters long and contain at least one numeric or special character.

To change your password on a UNIX system administered by Academic Computing Services, type **passwd** at the command prompt and enter your old password as instructed. After typing your new password, you must correctly retype it for verification. In the following example, the password is changed correctly:

```
% passwd <RETURN>
Changing password for jsmith
Old password: xxxxxxxx <RETURN>
New password: yyyyyyyy <RETURN>
Retype new password: yyyyyyyy <RETURN>
%
```

Passwords are not displayed on the screen.

In the following example, the password does not change because the new password was not retyped correctly:

```
% passwd <RETURN>
Changing passwd for jsmith
Old password: xxxxxxxx <RETURN>
New password: yyyyyyyy <RETURN>
Retype new password: zzzzzzzz <RETURN>
Mismatch - password unchanged.
%
```

Logging Out

To end a session on a UNIX system, type **logout**, **exit**, or press <ctrl-d> at the command prompt.

```
% logout <RETURN>
or
% exit <RETURN>
or
% <ctrl-d>
```

UNIX System Basics

Shells

After logging in to a UNIX system, you'll see the command prompt. At this point, you can enter shell commands. A shell acts as a translator between you and the UNIX operating system. The shell relays your commands to the operating system and returns any responses to you. There are two well-known shells. The first is called the Bourne shell and is named after its developer. The second shell is called the C shell and was developed at the University of California at Berkeley. All of the commands discussed in this guide are C shell commands (the majority of C shell commands are also the same as Bourne shell commands).

Shell Command Syntax

The basic syntax of a shell command is:

command [**options**] *objects*

where...

- command:** Is the name of the shell command.
- options:** Modify the way a command works. The options are usually single characters and are preceded by a dash (-).
- objects:** Are the objects that the command is applied against. Objects are *italicized* in this guide when showing the format of a command.

In the following example, the command **cat** is used with option **n** on the object **file1**:

```
% cat -n file1 <RETURN>
```

You can type several commands on the same line; just separate the commands with a semicolon (;). You can also type a command on more than one line; just place a backslash (\) at the end of the current line and continue with the rest of the command on the next line.

Getting Help

Two commands to get help on a UNIX system are **help** and **man**. A third command to get help while using X Windows is **dxbook**.

Typing **help** at the command prompt displays a screen of information about several commands. As indicated, use the **man** command for further information.

Typing **man** followed by a command displays the information pages for that command. The **man** command displays only one page at a time and at the bottom of the screen, the UNIX system displays the message A--More--@ if the information is continued on subsequent screens.

- * Press <spacebar> to scroll through the information one screen at a time.
- * Press <RETURN> to scroll through the information one line at a time.
- * Type **q** (for quit) to exit the information pages.

In the following example, the **man** command is used with the command **cat** as its object:

```
% man cat <RETURN>
```

```
cat(1)
```

```
NAME
```

```
cat - concatenate and print data
```

```
SYNOPSIS
```

```
cat [-b] [-e] [-n] [-s] [-t] [-u] [-v] file ...
```

```
DESCRIPTION
```

```
The cat command reads each file in sequence and displays it on the standard output.  
Therefore, to display the file on the standard output you type:
```

```
cat file
```

```
To concatenate two files and place the result on the third you type:
```

```
cat file1 file2 > file3
```

```
.  
.  
.q
```

```
%
```

Viewing Files

There are several UNIX commands for viewing files without starting an editing session. Some of these commands are listed below. To get a complete description of these commands and the different options used with these commands, use the **man** command or the **dxbook** command.

The **cat** Command

The **cat** command, followed by the name of a file, displays the contents of that file. If you specify more than one file name, then the **cat** command displays the contents of each file in sequence.

The **cat** command displays the entire file without stopping. In the following example, the **cat** command displays the contents of **file.test**:

```
% cat file.test <RETURN>
This is just a test file
This file contains only two lines.
%
```

The more Command

The **more** command also displays the contents of a file. The difference between the **more** command and the **cat** command is that the **more** command pauses between screens, indicates the percentage of the file which has been displayed, and waits for your response before continuing. While you are using **more**, you can use the following commands:

- * Press <RETURN> to scroll forward one line.
- * Press <spacebar> to scroll forward one screen.
- * Type **q** to quit and return to the system prompt.

In the following example, the **more** command displays the contents of **file1**. The <spacebar> and **q** commands are also used:

```
% more file1 <RETURN>
This is line number 1.
This is line number 2.
This is line number 3.
.
.
.
This is line number 23.
--More--(31%)<spacebar>
This is line number 24.
This is line number 25.
This is line number 26.
.
.
.
This is line number 46.
--More--(81%) q
%
```

The less Command

The **less** command also displays the contents of a file on the screen. While you are using **less**, you can use the following commands:

- * Press <RETURN> to scroll forward one line.
- * Type **b** to scroll backward one screen.
- * Type **f**, or press <spacebar>, to scroll forward one screen.
- * Type **q** to quit displaying the file and return to the system prompt.

In the following example, the **less** command displays the contents of **file1**. The commands **b**, **f**, and **q** are also used:

```
% less file1 <RETURN>
This is line number 1.
This is line number 2.
This is line number 3.
.
.
.
This is line number 23. f

This is line number 24.
This is line number 25.
This is line number 26.
.
.
.
This is line number 46. b

This is line number 1.
This is line number 2.
This is line number 3.
.
.
.
This is line number 23. q
%
```

The head Command

The **head** command displays the first 10 lines (by default) of a file. You can specify how many lines the **head** command shows by following the **head** command with a hyphen and the number of lines you want displayed. In the following example, the **head** command displays the first 6 lines in **file1**:

```
% head -6 file1 <RETURN>
This is line number 1.
This is line number 2.
This is line number 3.
This is line number 4.
This is line number 5.
This is line number 6.
%
```

The tail Command

The **tail** command displays the last 10 lines (by default) of a file. You can specify how many lines the **tail** command shows by following the **tail** command with a hyphen and the number of lines you want displayed. In the following example, the **tail** command displays the last 3 lines of **file1**:

```
% tail -3 file1 <RETURN>
This is line number 98.
This is line number 99.
This is line number 100.
%
```

Input/Output

Most of the commands on a UNIX system require input and produce output. For example, the **cat** command uses any file name following it as input and displays the contents of the file on the screen as output.

On a UNIX system, you can redirect both the source of the input and the destination of the output. This concept is called input/output redirection. Some of the redirection operators and their definitions are:

Operator	Meaning
>	Redirects standard output.
>>	Redirects and appends standard output.
>&	Redirects standard output and standard error.
>>&	Redirects and appends standard output and standard error.
<	Redirects standard input.
<<xxx	Reads input up to a line identical with xxx .
	Redirects standard output to another command (see Pipes).

In the following example, the output of the **cat** command is redirected to **file3**:

```
% cat file1 <RETURN>
This is a line in file 1.
% cat file2 <RETURN>
This is a line in file 2.
% cat file1 file2 > file3 <RETURN>
% cat file3 <RETURN>
This is a line in file 1.
This is a line in file 2.
%
```

In the following example, the output of the **cat** command is appended to **file3**:

```
% cat file3 <RETURN>
This is line 1 in file 3.
This is line 2 in file 3.
% cat file1 <RETURN>
This is line 1 in file 1.
% cat file1 >> file3 <RETURN>
% cat file3 <RETURN>
This is line 1 in file 3.
This is line 2 in file 3.
This is line 1 in file 1.
%
```

In the following example, the input is read from the terminal until the pattern **EOF** is encountered. All information received is stored in the file **file1**.

```
% cat > file1 << EOF <RETURN>
? This is the first line in file1. <RETURN>
? EOF <RETURN>
% cat file1 <RETURN>
This is the first line in file1.
%
```

When you want to redirect the input of a program, use the standard input redirection operator (<). The input of the program is then read from a file rather than from the keyboard.

Pipes

A pipe makes the output of one command become the input of another command without creating an intermediate file. A vertical bar (|) separates the commands forming a pipeline.

In the following example, the output of the **cat** command becomes the input to the **sort** command with the help of a pipe:

```
% cat test <RETURN>
wc
more
head
cat
% cat test | sort <RETURN>
cat
head
more
wc
%
```

Managing Files

Managing Your Files With UNIX

There are many UNIX commands to help you manage your files. Several of these commands are discussed in this chapter. To get a complete description of each command and its options, please use the **man** command or the **dxbook** command (for more information on the **man** command, see **Getting Help**; for information on the **dxbook** command, see **Accessing UNIX Documentation Using X Windows**).

The ls Command

The **ls** (list) command displays the files in a specified directory. The **ls** command with no object displays the files in the current working directory. The format of the **ls** command is:

ls [**options**] *directory*

In the following example, the **ls** command displays the contents of user jsmith's home directory and the **/home/dept** directory (for information on directories, see **Managing Directories**):

```
% ls <RETURN>
file1 file2 file3 letters
% ls /home/dept <RETURN>
jsmith bcarson
%
```

The rm Command

The **rm** (remove) command is used to delete a file. The format for the **rm** command is:

rm [**options**] *file*

Use the rm command with caution. Once a file is deleted, you cannot recover it.

In the following example, the **rm** command is used to delete **file1**:

```
% cat file1 <RETURN>
This is the first line in file1.
% rm file1 <RETURN>
% cat file1 <RETURN>
file1: No such file or directory
%
```

Add the option **-i** (for interactive) to the **rm** command when you want to be asked to confirm deleting any files.

The cp Command

The **cp** (copy) command copies files. The format of the **cp** command is:

```
cp [options] source.file destination.file
```

Use the cp command with caution. If the destination file already exists, then the copy command replaces that file with the new file.

In the following example, the **cp** command copies **file1** to **file2**:

```
% cat file1 <RETURN>
abc def ghi
% cp file1 file2 <RETURN>
% cat file2 <RETURN>
abc def ghi
%
```

Add the option **-i** (for interactive) to the **cp** command when you want to be asked to confirm replacing any file that already exists.

The mv Command

The **mv** (move) command renames files. The format of the **mv** command is:

```
mv [options] old.filename new.filename
```

Use the mv command with caution. If the destination file already exists, then the move command replaces that file with the new file.

In the following example, the **mv** command renames **file1** as **file2**:

```
% cat file1 <RETURN>
abc 123 45678
% mv file1 file2 <RETURN>
% cat file2 <RETURN>
abc 123 45678
% cat file1 <RETURN>
file1: No such file or directory
%
```

Add the option **-i** (for interactive) to the **mv** command when you want to be asked to confirm replacing any file that already exists.

The **wc** Command

The **wc** (word count) command displays the number of lines, words, and characters in a file. In the following example, the **wc** command is used on **file1**:

```
% cat file1 <RETURN>
This is line number 1.
% wc file1 <RETURN>
 1  5 23 file1
%
```

Thus **file1** contains 1 line, 5 words, and 23 characters.

The **cmp** Command

The **cmp** (compare) command reports the differences between two files. The **cmp** command reports the first difference found between the two files. In the following example, the **cmp** command compares **letter1** and **letter2**:

```
% cat letter1 <RETURN>
abc
cde
fgh
ijk

% cat letter2 <RETURN>
abc
fgh
ijj
% cmp letter1 letter2 <RETURN>
letter1 letter2 differ: char 5, line 2
%
```

The diff Command

The **diff** (differences) command reports all the differences found between two files. Moreover, the **diff** command also displays the commands necessary to make the two files identical to each other using the ex editor (the ex editor is a standard UNIX editor). The format of the **diff** command is:

```
diff [options] file1 file2
```

The option **-e** used with the **diff** command produces a script of **a**, **c**, and **d** commands for the ex editor.

In the following example, the **diff** command compares **letter1** and **letter2**; the **-e** option is also used:

```
% cat letter1 <RETURN>
abc
cde
fgh
ijk
% cat letter2 <RETURN>
abc
fgh
ijj
% diff letter1 letter2 <RETURN>
2d1
< cde
4c3
< ijk
---
> ijj
% diff -e letter1 letter2 <RETURN>
4c
ijj
.
2d
%
```

The grep Command

The **grep** (get regular expression) command searches for a particular string of text in a file. The format of the **grep** command is:

```
grep string file
```

If **grep** finds the string, then the string is displayed on the screen and you'll return to the command prompt. If **grep** does not find the string, you'll simply return to the command prompt. In the following example, the **grep** command searches for the strings **efg** and **abc def** in the file **letter1**:

```
% cat letter1 <RETURN>
abc
cde
efg
ijk
% grep efg letter1 <RETURN>
efg
% grep "abc def" letter1 <RETURN>
%
```

The sort Command

The **sort** command sorts lines of the specified file and displays the sorted output on the screen. The default sort key is the entire line. The **sort** command does not alter the specified input file.

In the following example, the **sort** command sorts the contents of **file1**:

```
% cat file1 <RETURN>
hello
help
elephant
% sort file1 <RETURN>
elephant
hello
help
%
```

Printing Files

The **lpr** command sends a file to a default printer.

```
% lpr file1 <RETURN>  
%
```

To determine your default print queue, view the file **/etc/printcap** and search for the queue name with the entry **lp**. To print a file at another ACS computing facility, or to another print queue at the default site use the **-P** option followed by the name of the print queue with the **lpr** command. Refer to the ACS document *Using ACS Printers* for a complete list of print queues available.

Managing Directories

Understanding Directories

On a UNIX system, a directory contains a list of files and subdirectories. After logging in, you are placed in a directory called your home directory. Your home directory is the first level directory where you can save your files.

At the very top of the directory structure is the root directory, denoted by a single slash (/). Subdirectories are denoted by a slash followed by the subdirectory's name. Figure 1 represents a sample directory structure. In Figure 1, the home directory is a subdirectory of the root directory, departmental directories are subdirectories of the home directory, and individual user's directories are subdirectories of their departmental directory.

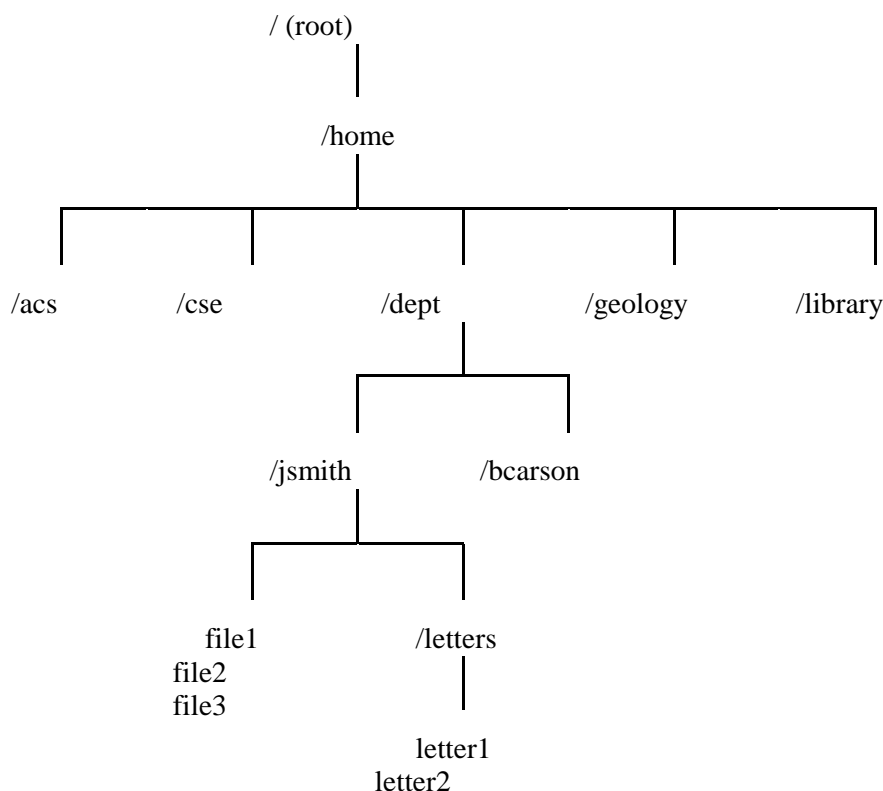


Figure 1 n A Sample Directory Tree Structure

According to Figure 1, three files, **file1**, **file2**, and **file3**, as well as the subdirectory **/letters** are in user **jsmith**'s home directory. (User **jsmith**'s home directory is **/home/dept/jsmith** which is a subdirectory under the **dept** directory which is a subdirectory under the **home** directory).

Two hidden files exist in each subdirectory, one designated by a single period (.) and the other designated by two periods (..). The single period file (.) is a pointer to the current working directory, whereas the double period file (..) is a pointer to its parent directory. A tilde (~) always represents your home directory.

Using Wildcards and Symbolic Substitution

You can use special characters, known as wildcards, to help you find filenames that match a certain pattern. The two most common wildcards are the question mark (?) and the asterisk (*). The rules for using ? and * are:

- ? Matches any one character.
- * Matches any group of zero or more characters.

You can also use metasequences to find file names. A metasequence is simply a list of characters, any one of which can be matched. To use a metasequence, simply place the characters to be matched between the brackets [].

The following table lists examples of wildcards and metasequences:

- *.* Filenames containing a period (.).
- *.com Filenames ending with .com.
- ?out Filenames with one character preceding the period and ending with .out.
- a[bc] Filenames **ab** and **ac**.
- pq[a-z] Filenames containing 3 characters in the range **pqa** through **pqz**.

Basic Directory Commands

All of the examples given in this section refer to the directory structure described in Figure 1. As shown in Figure 1, user jsmith's home directory is **/home/dept/jsmith**.

The pwd Command

The **pwd** (print working directory) command displays the working directory on the screen. In the following example, user jsmith's home directory is displayed:

```
% pwd <RETURN>
/home/dept/jsmith
%
```

The cd Command

The **cd** (change directory) command changes the current working directory. In the following example, user **jsmith** uses the **cd** command to change directories:

```
% pwd <RETURN>
/home/dept/jsmith
% cd letters <RETURN>
% pwd <RETURN>
/home/dept/jsmith/letters
%
```

In the following example, user **jsmith** returns to the directory above the current working directory:

```
% pwd <RETURN>
/home/dept/jsmith/letters
% cd .. <RETURN>
% pwd <RETURN>
/home/dept/jsmith
%
```

In the following example, user **jsmith** changes to the root directory:

```
% cd / <RETURN>
% pwd <RETURN>
/
%
```

You can change to your home directory from any other directory by using the **cd** command with no object or by using the **cd** command followed by a tilde (~). In the following example, user **jsmith** uses the **cd** command with no object:

```
% pwd <RETURN>
/
% cd <RETURN>
% pwd <RETURN>
/home/dept/jsmith
```

The mkdir Command

The **mkdir** (make directory) command creates a new directory. The format of the **mkdir** command is:

```
mkdir directory
```

In the following example, user jsmith uses the **mkdir** command to create a subdirectory, **newdir**:

```
% pwd <RETURN>
/home/dept/jsmith
% mkdir newdir <RETURN>
% ls <RETURN>
file1 file2 file3 letters newdir
%
```

The cp Command

The **cp** command copies files. In the following example, user jsmith uses the **cp** and **mv** commands to copy **file1** and move **file3** from the current working directory to the **newdir** directory:

```
% pwd <RETURN>
/home/dept/jsmith
% ls <RETURN>
file1 file2 file3 letters newdir
% cp file1 newdir <RETURN>
% mv file3 newdir <RETURN>
% ls <RETURN>
file1 file2 letters newdir
% ls newdir <RETURN>
file1 file3
%
```

The **rmdir** Command

The **rmdir** (remove directory) command is used to delete existing subdirectories. The **rmdir** command only removes subdirectories if they are empty. The format of the **rmdir** command is:

rmdir *directory*

In the following example, user jsmith uses the **rmdir** command to remove the subdirectory **newdir**:

```
% pwd <RETURN>
/home/dept/jsmith
% rmdir newdir <RETURN>
rmdir: newdir: Directory not empty
% cd newdir <RETURN>
% ls <RETURN>
file1 file3
% rm file1 <RETURN>
% rm file3 <RETURN>
% ls <RETURN>
% cd <RETURN>
% pwd <RETURN>
/home/dept/jsmith
% rmdir newdir <RETURN>
%
```

Protecting Files

On a UNIX system, the user's world is divided into three categories: User, Group, and Other.

The User (**u**) refers to the user who created and controls access to the file.

The Group (**g**) refers to the users who have the same group number as the owner (for example, all the students in the same class as the owner).

The Other (**o**) refers to the rest of the UNIX users.

You may grant users in any of these categories permission to access your files. There are three kinds of permission:

Read permission (r)	Permission to read a file.
Write permission (w)	Permission to modify or delete a file.
Execute permission (x)	Permission to run an executable file.

The **chmod** Command

The **chmod** (change mode) command allows you to change the protection assigned to a file or a directory. The **chmod** command has the following format:

chmod [mode] file

The mode field consists of three parts:

The category of users to be affected: user (**u**), group (**g**), or other (**o**).

The action to be performed: add (+); remove (-); or absolute (=), which replaces all existing permissions for that group.

The permission to be affected: read (**r**), write (**w**), or execute (**x**).

Checking Existing Permissions

You can check what permissions are currently in effect by using the **-l** (long) option with the **ls** command. In the following example, user **jsmith** uses the **ls -l** command to check existing permissions:

```
% pwd <RETURN>
/home/dept/jsmith
% ls -l <RETURN>
total 8
-rw-rw-rw-rw-    1  jsmith    1865   Nov 19  12:00  file1
-rw-rw-rw-rw-    1  jsmith     256   Nov 01   09:05  file2
-rw-rw-rw-rw-    1  jsmith     865   Dec 25   05:00  file3
drwxrwxrwx      2  jsmith     512   May 12  11:23  letters
%
```

The first character in the file description describes the type of file. A **Ad@** in the first column means that the file is a directory (the name of a subdirectory) and a **An@** in the first column means the file is a plain file containing ASCII or binary information.

The next nine columns list the file protection for user, group, and other, in that order.

The information after the list of file protection describes the number of links to the file, the owner of the file, the size of the file in bytes (a byte = 1 char), the last date and time the file was modified, and the name of the file.

A link is a logical connection between files and directories, or between directories. A file has at least one link to the directory file in which it resides. A directory file has at least two links, one to itself and one to its parent directory file.

In the following example, user `jsmith` uses the `chmod` command with the subdirectory `letters` and the file `file3` (note that the absolute action performed on `letters` replaces any existing permissions):

```
% pwd <RETURN>
/home/dept/jsmith
% ls -l <RETURN>
total 8
-rw-rw-rw-rw- 1 jsmith 1865 Nov 19 12:00 file1
-rw-rw-rw-rw- 1 jsmith 256 Nov 01 09:05 file2
-rw-rw-rw-rw- 1 jsmith 865 Dec 25 05:00 file3
drwxrwxrwx 2 jsmith 512 May 12 11:23 letters
% chmod o=w letters <RETURN>
% chmod g+r file3 <RETURN>
% ls -l <RETURN>
total 8
-rw-rw-rw-rw- 1 jsmith 1865 Nov 19 12:00 file1
-rw-rw-rw-rw- 1 jsmith 256 Nov 01 09:05 file2
-rw-rw-rw-rw- 1 jsmith 865 Dec 25 05:00 file3
drwxrwxrwn 2 jsmith 512 May 12 11:23 letters
%
```

Using the vi Editor

Getting Started

`vi` is a full-view line editor. You use `vi` to create and modify your programs and documents when you work on a UNIX-based system. There are several ways to start the `vi` editor, as demonstrated below.

<code>vi</code>	will invoke the editor, without opening a file
<code>vi filename</code>	will invoke the editor, and open the file or create the file if it does not exist
<code>vi /directory/filename</code>	will invoke the editor, and open/create the file

Operating Modes

Text Input Mode

vi has two text input modes, append and insert. Typing **a**, from command mode, will put you in append mode (text will be added behind the cursor). Typing **i**, from command mode, will put you in insert mode (text will be added before the cursor).

Command Mode

All vi commands must be entered from command mode. When the vi editor is started, you are automatically placed in command mode. To get to command mode from text input mode, press <ESC>. If you hear a beep when you press <ESC>, you are already in command mode. Note, vi commands are case sensitive - type uppercase and lowercase characters as such.

When you press <ESC> nothing appears on the screen.

Commands

Aborting Commands

To abort a command you can press <ESC>. If you abort a command, you will still be in command mode.

In addition to pressing <ESC>, you can type **u** (from command mode) to reverse the effect of the last command that changed the buffer. If you type **U** (from command mode), the effect of all the commands on the current line will be reversed, provided you have not moved from the line since the last change.

Also, you can correct typing errors from either of the text input modes by pressing the backspace key and typing over the incorrect text.

Saving Files

There are several ways to save files in vi, some of them are illustrated below.

:wq or ZZ or :x	writes buffer to file, and exits vi
:w	writes buffer to file, and stays in vi
:q	exits vi without saving changes
:w filename	writes buffer to <i>filename</i> (will not write to an
:w! filename	writes buffer to <i>filename</i> (overwrites existing file)
:q!	exits vi, and aborts all changes

Cursor Movement Commands

vi has many ways of moving the cursor around in a document. Normally, the arrow keys on your keyboard will work, as long as you are in command mode. However, if you find that the arrow keys are not functioning properly you can choose any of the following commands.

h	moves one character left, within current line
l	moves one character right, within current line
j	moves down one line
k	moves up one line
b	moves left to beginning of next word, or punctuation mark
B	moves left to beginning of next word, ignoring punctuation
w	moves right to beginning of next word, or punctuation mark
W	moves right to beginning of next word, ignoring punctuation
<ctrl-f>	scrolls forward one full screen
<ctrl-b>	scrolls backward one full screen
<ctrl-d>	scrolls forward one-half page
<ctrl-u>	scrolls backward one-half page
H	moves cursor to top left of screen
M	moves cursor to middle left of screen
L	moves cursor to bottom left of screen
G	moves cursor to the beginning of the last line of file/document

In addition to the commands listed above, you can add numbers to the front of the command to make it perform its function multiple times. For example, typing **6b** will move the cursor back six words.

Delete Commands

Below are just a few of the commands you can use to delete characters, words, or lines.

x	deletes character at the cursor position
dd	deletes current line
dw	deletes one word right

You can prefix the deletion commands with numbers to process multiple functions simultaneously. For example, typing **2dd** will delete the next two lines.

Insert Commands

With vi you can insert text and/or blank lines. To enter text, issue one of the following commands. Remember to press <ESC>, to return to command mode.

i	puts you in insert mode, insertion occurs before the current cursor position
I	moves cursor to beginning of current line, and enters insert mode
o	creates a blank line following the current line, moves the cursor to the beginning of the new line, and enters insert mode
O	creates a blank line before the current line, moves the cursor to the beginning of the new line, and enters insert mode
a	puts you in append mode, text will be added after current cursor position
A	moves cursor to end of current line, and enters append mode

Search Commands

You can use the search command to help you locate a text string. If you want a match for just the characters entered as the string, be sure to include a space preceding and following the string.

<i>/string</i>	searches forward from cursor position for <i>string</i>
<i>?string</i>	searches backward from the cursor position for <i>string</i>
n	repeats search in same direction
N	repeats search in opposite direction

If you want to find all occurrences of the word ear, enter the command A/ ear @. If you type A/ear@, you will find all occurrences of the word ear, and also fear, earth, bear, etc.

Change Commands

vi has several commands that allow you to change characters, words, and lines. Some of the commands and descriptions are below.

r	allows you to replace one character at current cursor position
cw	allows you to replace one word at current cursor position
ctr	allows you to replace text from current cursor position to a specified character <i>x</i>
xp	transposes the letter at the cursor position with the following letter
ddp	swaps position of the current line and the following line
R	allows you to replace all characters beginning at the cursor position; at end of line editor enters insert mode

You can prefix **r** and **cw** with numbers to make multiple changes.

Pasting Commands

When a string is deleted it is stored in a temporary buffer. Deleted text stays in this buffer until you modify the buffer by deleting another string or exit the editor. The text strings stored in this buffer can be removed and placed back in the text. This can be done by using the commands below:

p	places last deleted text following the cursor
P	places last deleted text before the cursor

Marking Text

To make a bookmark in a document place the cursor at the location where you want the bookmark; then type **mx** (Where *x* is any letter you choose to symbolize that particular bookmark). To return to the marked position type **`x** (grave accent {backward single quote} followed by the character *x* you specified). Typing **'x** (apostrophe {forward single quote} followed by the character *x* you specified), will return you to the beginning of the line that contains the mark. You can have more than one bookmark in a file. Marks are forgotten when you exit vi or close that document.

Moving Marked Text

You can delete the text between your mark and the current cursor position by typing **d`x** (backward single quote). Then, you can move the cursor to where you want to add the text, and type **p** or **P**.

Yank Command

The yank command copies text into a temporary buffer. It does not change the original text when it makes the copy. Text remains in the buffer until you enter another command that places text into the buffer, or exit the editor.

Your original text is not deleted, just copied.

yy or Y	copies the line at the cursor into the buffer
nyy or nY	copies <i>n</i> lines from the cursor position into the buffer
yw	copies the word at the cursor into the buffer
nyw	copies <i>n</i> words following the cursor position into the buffer

Copying Yanked Text

You can use the yank command with the paste command to copy text to another location in the current document.

Reading Other Files Into Current File

You can insert another file into the current file by typing `:r filename`. The inserted file will be added starting on the line following the cursor position.

Repeating Commands

If you type a period (`.`), the last command that modified the buffer will be repeated. You can use this instead of having to retype commands.

Joining Lines

Typing `J` allows you to join the current line and the following line. Typing `nJ` will join *n* lines to the current line.

Multitasking

Understanding Foreground and Background Processing

UNIX systems are multitasking systems. This means that you can run more than one process at the same time. A process can be anything from listing the contents of a directory to running a program.

A foreground process is any task that you run on your terminal n you'll return to the command prompt only when you complete your entire task. You may have only one foreground process running at any time. You can suspend a foreground process by pressing <ctrl-z>. After suspending a foreground process, you return immediately to the command prompt.

To run multiple processes on a UNIX system, you must run them in the background. A process running in the background is called a background job. Background jobs should not require your intervention. You can run several background jobs simultaneously n you are notified when a background job is completed.

Running a Background Job

To run a process as a background job, follow the command for that process with an ampersand (&). The job is then assigned a job number and a unique process identification number (PID). After starting a job in the background, you'll return to the system prompt where you can execute other commands or start other jobs.

A trade off exists between running several jobs in the background simultaneously and the system response. Running several jobs in the background can significantly degrade the interactive response you receive on a UNIX system.

The nice Command

When running a process as a background job, you should precede the command for that process with the **nice** command. The format of the **nice** command used to run a process as a background job is:

```
nice [-n] command [options] &
```

where **n** is an optional number between 1 and 20, ***command*** is the command for that process, and ***options*** are any optional arguments for that command. The optional number, included after the **nice** command, determines the priority level of the background job n the lower the number, the greater the priority. If you do not include a number with the **nice** command, 10 is used as the default.

If you want to be able to logout of your account and have a job continue to run in background, precede the previous command by the word **nohup** . The previous command would become:

```
nohup nice [-n] command [options] &
```

In the following example, the **nice** command is used with the FORTRAN compiler command, **f77**, at a nice level of **20**; the object of the compiler command is the FORTRAN program called **random.f** (for information on compilers, please see **Running Programs on UNIX**; for information on the **jobs** command, see **The jobs Command**, discussed next):

```
% nice -20 f77 random.f & <RETURN>
[1] 20394
% jobs <RETURN>
[1] + Running          f77 random.f
%
```

The jobs Command

The **jobs** command displays the status of any background jobs. The format of the **jobs** command is simply:

jobs

In the following example, the **jobs** command displays the single background job running:

```
% jobs <RETURN>
[1] + Running          f77 random.f
%
```

If you have used the **nohup** command to run a job in background and then logged out of your account, issuing the **jobs** command after you log back in will not show the status of that job. To see its status, enter the following command:

ps auxww | grep userid

If the job is still running, its status will be shown. If the job has completed, it will not appear.

The fg Command

The **fg** (foreground) command returns a background job to the foreground. The format of the **fg** command is:

fg %n

where **n** is the job number. If the **fg** command is used without any objects, the most current job in the job list is brought into the foreground.

The stop Command

The **stop** command stops a job currently running in the background. The format of the **stop** command is:

```
stop %n
```

where *n* is the job number.

The kill Command

The **kill** command cancels a job running in the background. The format of the **kill** command is:

```
kill %n    or    kill -9 %n
```

where *n* is the job number. Specifying the **-9** option with the **kill** command kills the process under all circumstances.

The following examples illustrate the concept of running jobs in the foreground and background:

```
% more file1 <RETURN>
This is line number 1
This is line number 2
.
.
.
--More--(57%)
<ctrl-z>
Suspended
% ls -l <RETURN>
total 8
nrwnnnnnnn    1  jsmith    1865  Nov 19  12:00  file1
nrwnrwnnnnn    1  jsmith     256  Nov 01  09:05  file2
nrwnrnnrwn     1  jsmith     865  Dec 25  05:00  file3
drwnrwnnnnn    2  jsmith     512  May 12  11:23  letters
<ctrl-z>
Suspended
% cat file1 file2 > file3 & <RETURN>
[3] 5421
% jobs <RETURN>
[1]  Suspended    more file1
[2]  - Suspended    ls -l
[3]  + Running     cat file1 file2 > file3
% diff file1 file2 > file.dif & <RETURN>
[4] 5427
%
```

The first number represents the job number and the second number is the unique process identification number (PID) assigned to the job.

```
% jobs <RETURN>
[1]  Suspended   more file1
[2]  Suspended   ls -l
[3]  - Running   cat file1 file2 > file3
[4]  + Running   diff file1 file2 > file.dif
%
[3]  Done        cat file1 file2 > file3
%
```

As shown in the above example, you are informed when any background job finishes.

```
% jobs <RETURN>
[1]  Suspended   more file1
[2]  - Suspended  ls -l
[4]  + Running   diff file1 file2 > file.dif
% fg %1 <RETURN>
more file1
This is line number 1
This is line number 2
.
.
.q
% kill %2 <RETURN>
[2]  Terminated ls -l
% kill %4 <RETURN>
[4]  Terminated diff file1 file2 > file.dif
% jobs <RETURN>
%
```

If you try logging out and you have any stopped jobs, you are notified by the system of those jobs. If you do not need the stopped jobs, you should kill them. In either case, you can use the **logout** command or **<ctrl-d>** a second time to log out.

Miscellaneous Commands

The **history** Command

The **history** command displays the last 20 commands used (by default), but this number can be increased or decreased (see the shell variable `history`).

You can reissue a previous command with one of the following commands:

- !!** Execute the last command entered.
- !*n*** Execute command number *n* on the history list.
- !*xxx*** Execute the most recent command on the history list starting with *xxx*.
- !*-n*** Execute the *n*th command ago.

The following example shows the **history** command being used:

```
% history <RETURN>
1 more file1
2 pwd
3 cd /
4 cd
5 history
% !2 <RETURN>
pwd
/home/dept/jsmith
% history <RETURN>
1 more file1
2 pwd
3 cd /
4 cd
5 history
6 pwd
7 history
% !-7 <RETURN>
more file1
This is line number 1
This is line number 2
.
.
.q
% history <RETURN>
1 more file1
2 pwd
3 cd /
4 cd
5 history
6 pwd
7 history
8 more file1
9 history
% !c <RETURN>
cd
% pwd <RETURN>
/home/dept/jsmith
%
```

The alias Command

The **alias** command is used for making a nickname or abbreviation for a UNIX command. You can consequently designate your own name for any UNIX command. The format of the **alias** command is:

```
alias nickname UNIX_command
```

To display a list of all current aliases, use the **alias** command with no object. In the following example, the **alias** command is used first to nickname the **more** command as **type** and then the **ls** command, including the **-l** (long) option, as **long**:

```
% alias type more <RETURN>
% alias long 'ls -l' <RETURN>
% type file1 <RETURN>
This is line number 1
This is line number 2
.
.q
% long <RETURN>
total 8
nrwnnnnnnn 1 jsmith 1865 Nov 19 12:00 file1
nrwnrwnnnn 1 jsmith 256 Nov 01 09:05 file2
nrwnrnrwn 1 jsmith 865 Dec 25 05:00 file3
drwnrwnnnn 2 jsmith 512 May 12 11:23 letters
% alias <RETURN>
type more
long ls -l
%
```

The cal Command

The **cal** (calendar) command displays the calendar for the specified month and year. The format of the **cal** command is:

cal month year

If you exclude the month number from the **cal** command, the calendar for the entire specified year is displayed.

```
% cal 4 1992 <RETURN>
April 1992
S   M   Tu  W   Th  F   S
      1   2   3   4
5   6   7   8   9  10  11
12  13  14  15  16  17  18
19  20  21  22  23  24  25
26  27  28  29  30

%
```

You must specify the entire year with the **cal** command. For example, **cal 90** displays the calendar for 90 AD, not 1990. To display the calendar for 1990, you must use **cal 1990**. The **cal** command with no object displays the calendar for the current month.

The date Command

The **date** command displays the current date and time on your screen:

```
% date <RETURN>
Fri Mar 3 10:07:03 CDT 2000
%
```

The who Command

The **who** command displays all users currently logged in to a UNIX system. The **who** command displays the user's login name, the device to which the user's terminal is connected, and the login time and date.

In the following example, the **who** command shows three users currently logged in:

```
% who <RETURN>
opr5    console  Mar 3   07:54
jsmith  tty0     Mar 3   09:38   (129.107.1.100)
bcarson tty3     Mar 3  10:43   (129.107.1.100)
%
```

The quota Command

The **quota** command displays the disk quota for your account. To check your disk quota, type the following:

```
% quota <RETURN>
Disk quotas for user sbh2284 (uid 4210):
Filesystem blocks quota limit grace files quota limit grace
/home 1084 4096 6144 84 0 0
```

The limit Command

The **limit** command shows the maximum computing resources that can be allocated to a process. To see this listing, type the following:

```
% limit <RETURN>
cputime    3:00:00
filesize  unlimited
datasize   81920 kbytes
stacksize  2048 kbytes
coredumpsize unlimited
memoryuse  1022184 kbytes
vmemoryuse 1048576 kbytes
descriptors 96
```

Shell and Environment Variables

Understanding Shell and Environment Variables

UNIX systems use two types of variables to define your working environment: shell variables and environment variables.

Shell variables are known only to the shell in which they were created. If you start a new shell (command interpreter), the shell variables created in the previous shell are not transferred to the current shell.

Environment variables are global variables and are independent of the shell. Environment variables stay in effect regardless of what shell is used.

On a UNIX system, some shell and environment variables are predefined by the system. Also, environment variables are usually represented by uppercase names whereas shell variables are represented by lowercase names. A shell variable with the same name as an environment variable still has the same function.

Some of the definable shell variables are:

Variable	Description
argv	Argument to the shell.
autologout	Number of minutes of idle time before a user is logged out.
cdpath	Alternative directory tree search.
history=<i>n</i>	Remember last <i>n</i> commands.
home	Home directory of user.
mail	Directory where shell checks for user's mail.
noclobber	Prevents overwriting an existing file during redirection.
user	Login User ID.
savehist=<i>n</i>	Remember last <i>n</i> commands at the beginning of next terminal session.
prompt=<i>string</i>	Change the default command prompt to <i>string</i> .
path	Current path specification - list of directories the system may search to resolve command requests.
shell	Shell used to process commands.
term	Terminal type.
status	Returned by last command: 0=no error, 1=error.

Some of the definable environment variables are:

Variable	Description
HOME	Home directory.
SHELL	User's login shell.
TERM	Terminal type.
USER	Login User ID.
PATH	Search path used to resolve command requests.
DISPLAY	Name of X Windows capable display device.

The set Command

The **set** command displays current shell variables and their values. You can also designate a new value for a variable with the **set** command. In the following example, current shell variables are shown using the **set** command:

```
% set <RETURN>
argv          ()
autologout    0
cdpath        /home/dept/jsmith
cwd           /home/dept/jsmith
history       20
home          /home/dept/jsmith
mail          /usr/spool/mail/jsmith
path          (. /home/dept/jsmith/bin /home/dept/bin )
prompt        %
shell         /bin/csh
status        0
term          vt52
user          jsmith
%
```

In the following example, the shell variables **term**, **prompt**, and **autologout** are assigned values using the **set** command:

```
% set term=vt100 <RETURN>
% set prompt="UNIX> " <RETURN>
UNIX> set autologout=20 <RETURN>
UNIX> set <RETURN>
argv          ()
autologout    20
cdpath        /home/dept/jsmith
cwd           /home/dept/jsmith
history       20
home          /home/dept/jsmith
mail          /usr/spool/mail/jsmith
path          (. /home/dept/jsmith/bin /home/dept/bin )
prompt        UNIX>
shell         /bin/csh
status        0
term          vt100
user          jsmith
UNIX>
```

The echo Command

To display the value of a shell variable, type **echo** followed by a dollar sign (\$) and the name of the variable. You must place a dollar sign before the variable name to display the value of the variable n otherwise the name of the variable is displayed. In the following example, the **echo** command is used to display both text and the current values for the **cwd** and **user** variables:

```
% echo "Hello there" <RETURN>
Hello there
% echo "cwd" <RETURN>
cwd
% echo "$cwd" <RETURN>
/home/dept/jsmith
% echo "$user" <RETURN>
jsmith
%
```

In the following example, the **echo** command displays the values of the environment variables **USER** and **TERM**:

```
% echo "USER" <RETURN>
USER
% echo "$USER" <RETURN>
jsmith
% echo "$TERM" <RETURN>
vt100
%
```

The **printenv** Command

The **printenv** command displays all environment variables and their values. In the following example, the **printenv** command displays the environment variables:

```
% printenv <RETURN>
TERM=vt100
HOME=/home/dept/jsmith
SHELL=/bin/csh
USER=jsmith
PATH=./home/dept/jsmith/bin:/home/dept/bin
%
```

The **setenv** Command

The **setenv** command sets the value of environment variables. In the following example, the environment variable **TERM** is set to **vt102** with the **setenv** command:

```
% setenv TERM vt102 <RETURN>
```

The .login and .cshrc Files

When you log in to a UNIX system using the C shell (the default shell), the system searches your home directory for two files, **.cshrc** and **.login**, and executes all of the shell commands in those files. The commands in the **.cshrc** file are executed first, then commands in the **.login** file are executed. The two files differ in that the commands in **.cshrc** are executed every time you start a new C shell, whereas the commands in **.login** are executed only when you log in. If you start a new C shell, the commands in **.login** are not re-executed. For users who invoke the Bourne shell when they login, the system searches for a **.profile** file and executes all the shell commands in that file.

Usually, all of the environment variables are defined in **.login** and all of the shell variables are defined in **.cshrc**. In the following example, the **cat** command displays the contents of files **.login** and **.cshrc**:

```
% cat .login <RETURN>
setenv SHELL /bin/csh
set mail=/usr/spool/mail/$user
% cat .cshrc <RETURN>
set path=(. $HOME/bin /home/dept/bin /bin)
set history=20
alias dir 'ls -l'
alias rm 'rm -i'
alias cp 'cp -i'
alias mv 'mv -i'
set noclobber
set autologout=20
%
```

You can start a new C shell under the parent shell by issuing the **cs** command. All of the commands defined in the **.cshrc** file are automatically executed in this newly created C shell. Pressing **<ctrl-d>** kills the most recently created C shell.

Using X Windows Systems

X Windows was developed by MIT to provide an efficient network protocol for windowing and graphics applications that is independent of workstation hardware and operating systems. With X Windows you can monitor several applications executing on several different systems under a variety of operating systems. You can shift between these applications simply by using a mouse.

Using X Windows

To use X Windows you must have a terminal or workstation that supports X Windows protocols. If you are uncertain whether your terminal or workstation can support the X Windows protocols, please consult your documentation or call Academic Computing Services at 272-2208.

The Teleray VT100 compatible terminals in the computing facilities do not support X Windows protocols. The terminals in the computing facilities that support X Windows protocols are the NCD HMXpro terminals.

If your terminal or workstation *does* support X Windows protocols, then you can use X Windows with any host system that supports X Windows applications.

Using X Windows With a Workstation

To use X Windows with a workstation, follow these steps:

1. Log in to the local workstation.
2. Start X Windows on the local workstation.
3. From the local workstation, enter the command **xhost +node** (where *node* is the name of the remote node where you will be running X Windows applications).
4. Connect and log in to the remote node.
5. If the remote system is administered by Academic Computing Services, then enter the command **setx** on the remote system to set the DISPLAY environment variable.

or

If the remote system is not administered by Academic Computing Services, then enter **setenv DISPLAY <IPaddress>:0**, where *<IPaddress>* is the IP address of your local system (you must type **setenv** in lowercase letters and **DISPLAY** in uppercase letters).

You may then start any X Windows applications on the remote system and they will display on the local workstation.

Starting X Windows on an NCD X Windows System

To start an X Windows session from an NCD X Windows System, follow these steps:

1. Select *host* from the chooser window.
2. Connect and log in to the remote node.

or (for a Telnet session):

1. Press the *setup* key and select *terminals, new terminal*.
2. Connect and log in to the remote node.
3. If the system is not administered by Academic Computing Services at UTA, then enter the following:

```
% setenv DISPLAY <terminal>:0 <RETURN>
```

where **<terminal>** is the number of the terminal which can be found at the top of the monitor. To enter this amount, you must be in the *csh* or *tcsh* shell and **setenv** must be in lowercase letters and **DISPLAY** must be in upper case letters. For example:

```
% setenv DISPLAY xncd02:0 <RETURN>
```

You may then start any X Windows applications on the remote system and they will display on your X Terminal.

If you subsequently use Telnet to connect with another system not administered by Academic Computing Services at UTA, you must perform step 3 listed above once again.

Accessing UNIX Documentation Using X Windows

To access UNIX documentation while using X Windows, type **dxbook** on the Omega system or **answerbook** on the Gamma system. These applications display online documentation on your workstation screen. When you start the application, you will see two types of windows.

The selection window displays lists of available online documentation. When you do not have a book open, the selection window contains lists of related documentation grouped into bookshelves. When you do have a book open, the selection window contains the table of contents or index of that book.

The topic window lists the contents of a book after that book is opened.

To open a book, follow these steps:

1. Open that book's shelf by double-clicking on the shelf title n once opened, you'll see a list of books and bookshelves in the selection window.
2. Open a book by double-clicking on that book's title n once opened, you'll see the Table of Contents for that book in the selection window and that book's first page in the topic window.

For more information on using the **dxbook** or **answerbook** commands, including a complete list of options, use the **man** command.

The Basics of Doing Statistics on OMEGA from the Viewpoint of a VM/CMS User

SAS and SPSS

SAS and SPSS are statistical packages available on ACS=s OMEGA system. The file extension for SAS is **.sas**, and the file extension for SPSS is **.sps**.

Editors on OMEGA

OMEGA has an editor called Athe@ which is a clone of the XEDIT editor on VM/CMS. If you have an X-Windows terminal, which most of you do not, it works almost identically to XEDIT on VM/CMS. If you are connected to the campus network with a PC running Windows 95, you need to connect to OMEGA choosing a **vt220** terminal emulation from the telnet window. This will enable you to use function key **f7** to page-up, **f8** to page-down, **f5** will be the quit key (**f3** performs this function on VM), and the insert key will work correctly when using Athe@. (So far, the keys do not work correctly when using Athe@ on PCs running operating systems other than Windows 95.) Also, INPUT MODE does not seem to work with any machine setup we have tried. If you have a setup on which the function keys do not work, contact ACS at Ext. 2208 to obtain a copy of the escape sequences which can be used to perform the functions of these keys. Also, if you wish to use Athe@ you will need to acquire a file called **.there** for your account.

Using SAS on OMEGA

If you are logging in to OMEGA from a PC, SAS works much the same on OMEGA as it did on VM/CMS. You create your program with an editor. Save the program in a file with filetype (extension) **.sas**. Then, at the OMEGA system prompt, type:

```
% sas filename <RETURN>
```

Remember to type **sas** in lower case and your filename in the correct case. It should then run and create a file called **filename.log** which is your saslog and a file called **filename.lst** which is your listing. The sas commands you used in your programs on VM/CMS should basically be identical to those you will use on OMEGA with the exception that filenames used in FILE, INFILE, and LIBNAME statements will have to conform to UNIX conventions.

Using SPSS on OMEGA

Again, if you are using SPSS on OMEGA from a PC, you will create your SPSS program with an editor. Save it in a file with filetype (extension) **sps**. Then, at the OMEGA system prompt, issue the command:

```
% spss -m filename.sps > filename.lst <RETURN>
```

SPSS will then run and put your results in the file *filename.lst*. Again, the basic commands you used in your SPSS programs on VM/CMS should port over to OMEGA with the exception of the filenames which will have to be changed to conform to UNIX conventions.

Using the IMSL Libraries on OMEGA

If you are going to run a FORTRAN simulation using the IMSL Libraries, you will need to create your FORTRAN program as usual using the editor of your choice and save it in a file with the extension **.f**. Where you used to use FILEDEF commands to link your input and output files to your programs, you will now have to include the appropriate OPEN statements within your programs. Before compiling and linking your program, issue the command:

```
% source /usr/local/vni/00setup.imsl <RETURN>
```

Then, to compile and link, issue the command:

```
% $FC -o filename $FFLAGS filename.f $LINK_FNL <RETURN>
```

Then, to run your program, issue the command:

```
% filename > filename.lst <RETURN>
```

Your diagnostics will appear in *filename.lst*. You will be pleasantly surprised at how much faster your simulations run on OMEGA. With all of the above ways of running programs, you must wait until the program has finished running before you can continue with any other work.

Note: In all cases above, the word *filename* should be replaced by the actual name you have given the file.

Running Programs on UNIX

Compilers on the UNIX Systems

Currently there are two C compilers (DEC C and GNU C), a GNU C++ compiler, and a FORTRAN compiler available on both ACS supported UNIX systems. There is also a Pascal compiler available on OMEGA. The file extension expected by the C compiler is **.c**. The file extension expected by the C++ compiler is **.C**, **.cc**, or **.cxx**. The file extension expected by the FORTRAN compiler is **.f**. The file extension expected by the Pascal compiler is **.p**. Sample programs for each of these compilers are given in this section.

For more information about any of these compilers, please use the **man** command or the **dxbook** command (for more information on the **man** command, see **Getting Help**; for information on the **dxbook** command, see **Accessing UNIX Documentation Using X Windows**).

Sample Programs

Example 1 n Sample C Program

The **cc** command invokes the DEC C compiler and linkage editor. The **gcc** command invokes the GNU C compiler. The **cc** command takes a C source file as input and, if no syntax or linkage errors are found in the C source code, creates an executable image file called **a.out**. If a file named **a.out** already exists, it is deleted when the new image file is created.

In the following example, the C program **random.c** generates ten random numbers with the help of the C **rand** function.

```
% cat random.c <RETURN>
#include <stdio.h>
main ()
{
int i, k;

srand(1);
for (k=0; k !=10; k++)
{
i = rand();
printf ("Random number %2d = %d\n", k, i);
}
exit (1);
}
% cc random.c <RETURN>
% a.out <RETURN>
Random number 0 = 1103527590
Random number 1 = 377401575
Random number 2 = 662824084
Random number 3 = 1147902781
Random number 4 = 2035015474
Random number 5 = 368800899
Random number 6 = 1508029952
Random number 7 = 486256185
Random number 8 = 1062517886
Random number 9 = 267834847
%
```

Example 2 n Sample C Program

In the following example, the C program **printmain.c** prints a line to the screen and then calls an external subroutine **sec** which is defined in the file **printsub.c**. The subroutine prints a line to the screen and transfers control back to the calling program which in turn returns control back to the operating system.

```
% cat printmain.c <RETURN>
#include <stdio.h>

extern void sec();

main()
{
    printf ("This is a line in the main C program\n");
    sec ();
    exit (1);
}

% cat printsub.c <RETURN>
#include <stdio.h>

void sec ()
{
    printf("This is a line in the SEC subroutine\n");
    return;
}

% cc -c printmain.c <RETURN>
% cc -c printsub.c <RETURN>
% cc printmain.o printsub.o -o print <RETURN>
% print <RETURN>
This is a line in the main C program
This is a line in the SEC subroutine
%
```

In this example:

The **-c** option used in the first two **cc** commands created two object modules called **printmain.o** and **printsub.o**.

The **cc** command was then used to link the two object modules together.

The **-o** option created an executable image called **print** rather than **a.out**.

The following example demonstrates another method of creating the executable image of the files **printmain.c** and **printsub.c**:

```
% cc printmain.c printsub.c <RETURN>
printmain.c:
printsub.c:
% a.out <RETURN>
This is a line in the main C program
This is a line in the SEC subroutine
%
```

Example 3 n Sample C++ Program

In the following example, the C++ program **test.cc** displays two values contained within the program.

```
% cat test.cc <RETURN>
#include <iostream.h>
#include <String.h>
#include <stdlib.h>

class myclass {
public:
    int a;
    myclass (int i) {a = i;};
    void showit() { cout << "a = " << a << "\n"; };
};

int main() {
    myclass obj1(5), obj2(10);
    obj1.showit();
    obj2.showit();
    exit(0);
}
% g++ test.cc <RETURN>
% a.out <RETURN>
a = 5
a = 10
%
```

Example 4 n Sample FORTRAN Program

In the following example, the FORTRAN program **random.f** generates ten random numbers with the help of the FORTRAN rand function.

```
% cat random.f <RETURN>
  program randm
  integer i, j
  do 10 i=1,10
    j = irand()
    write (6,*) 'Random Number = ', j
10 continue
  stop
  end
```

```
% f77 random.f <RETURN>
% a.out <RETURN>
Random Number =    16838
Random Number =    5758
Random Number =   10113
Random Number =   17515
Random Number =   31051
Random Number =    5627
Random Number =   23010
Random Number =    7419
Random Number =   16212
Random Number =    4086
%
```

Example 5 n Sample Pascal Program

In the following example, the Pascal program **random.p** generates five numbers from a seed number that the user is prompted to supply.

```
% cat random.p <RETURN>
program random(input,output);
var RNumber,StartNumber,Counter:integer;
function Random(var Seed:integer):integer;
  const
    MODULUS=65536;
    MULTIPLIER=25173;
    INCREMENT=13489;
  begin
    Seed:=((MULTIPLIER*Seed)+INCREMENT)mod MODULUS;
    Random:=Seed;
  end;
begin
  Counter:=1;
  writeln('Enter a seed value. ');
  readln(StartNumber);
  writeln('Here are five random numbers. ');
  while Counter<=5 do
    begin
      Rnumber:=Random(StartNumber);
      write(Rnumber);
      Counter:=Counter+1;
    end;
  writeln;
end.
% pc random.p <RETURN>
% a.out <RETURN>
Enter a seed value.
8 <RETURN>
Here are five random numbers.
  18265 63294 2119  8772 40261
```

Using the UTA UNIX System

Viewing Files

<i>cat</i>	Displays file without stopping.
<i>more</i>	Displays file one screen at a time; use <ENTER> to continue or q (quit).
<i>less</i>	Displays file one screen at a time; use b (back), f (forward), and q (quit).
<i>head -n</i>	Displays first <i>n</i> number lines of file (default 10).
<i>wc</i>	Displays the number of lines, words, and characters in a file.
<i>cmp</i>	Compares then reports the differences in two files.
<i>diff</i>	Compares then reports the differences in two files and the <i>ex</i> editor commands that will make them identical.
<i>grep xx</i>	Searches for a regular expression <i>xx</i> in a file.
<i>sort</i>	Alphabetically sorts the lines in a file.
<i>lpr</i>	Prints a file.
<i>tail -n</i>	Displays last <i>n</i> number of lines of file (default 10).

Managing Files

<i>ls</i>	Displays contents of a directory.
<i>rm</i>	Removes (deletes) a file from a directory.
<i>cp</i>	Copies a file.
<i>mv</i>	Moves (renames) a file.
<i>wc</i>	Displays number of lines, words, & characters in file.
<i>chmod</i>	Changes the protection assigned to a file.
<i>ls -l</i>	Displays protection assigned to each file.

Input/Output Redirection

>	Redirect standard output.
>>	Redirect and append standard output.
>&	Redirect standard output and standard error.
>>&	Redirect and append standard output and standard error.
<	Redirect standard input.
<< <i>xxx</i>	Redirect standard input up to a line identical with <i>xxx</i> .
	Redirect standard output to another command.

To see all options and arguments available with each command, type **man** followed by that command's name.

UNIX Command Chart

Managing Directories

<i>pwd</i>	Prints (displays) the current working directory.
<i>cd</i>	Changes the current working directory.
<i>mkdir</i>	Makes a new directory.
<i>rmdir</i>	Removes a directory.

Multitasking

<i>nice x &</i>	Runs command <i>x</i> as a background job.
<i>jobs</i>	Displays the status of any background jobs.
<i>fg</i>	Returns a background job to the foreground.
<i>stop</i>	Stops a currently running background job.
<i>kill</i>	Cancel a currently running background job.

Miscellaneous

<i>history</i>	Displays the last 20 (by default) commands used.
<i>!!</i>	Executes the last command used.
<i>!<i>n</i></i>	Executes the command numbered <i>n</i> on the history list (displayed by <i>history</i>).
<i>!<i>xxx</i></i>	Executes the most recent command starting with <i>xxx</i> .
<i>!<i>n</i></i>	Executes command used <i>n</i> commands ago.
<i>alias</i>	Creates a nickname for a command.
<i>cal</i>	Displays a calendar for the current month. If specifying year, specify all four digits.
<i>date</i>	Displays the current date and time.
<i>who</i>	Displays all users who are logged into the system.

Setting Variables

<i>set</i>	Displays current shell variables and their values. Also sets new value for specified variable.
<i>echo \$x</i>	Displays current value for variable <i>x</i> .
<i>printenv</i>	Displays current environment variables and values.
<i>setenv</i>	Sets current value for environment variable.

For an on-line help screen, type *help*.