The Brief History of the Unix System

In the 1960s, a number of researchers at Bell Labs (the research arm of AT&T), participated in the development of an operating system called Multics (Multiplexed Information and Computing Service).

In 1969, Bell Labs withdrew from the Multics project. At about the same time (1969), one of these researchers, Ken Thompson, developed a simpler and smaller operating system for a PDP-7 minicomputer.

In searching for a name, Thompson compared his new system to Multics. The goal of Multics was to offer many features to multiple users at the same time.

Ken Thompson's system was smaller, less ambitious and, at the beginning, was used by one person at a time. Moreover, each part of the new system was designed to do only one thing and do it well. Ken Thompson decided to name his system Unics, which was soon changed to Unix.

Thompson's original UNIX system was written in assembly-language (PDP-7). In 1973 Dennis Ritchie rewrote the UNIX system in C.

Once the original assembly language programs were rewritten in C, it was possible to move the entire UNIX system from one environment to another with a minimum of difficulty.

During the early 70s, the UNIX system began to be used internally throughout the Bell system.

In 1975 Western Electric started licensing the UNIX system. The fee was nominal for academic institutions, encouraging many of them to use and further develop the UNIX system. However, there was little success in introducing it into mainstream businesses.

The main event during the 80s was the UNIX system's continuing internal struggle to define itself. For much of the decade there were two main factions: AT&T (System V) and Berkeley (BSD).

During the 1980s Unix remained a technological marvel, with impressive networking and communications' capabilities, but it never managed to crack the mainstream business market.

UNIX is a registered trademark of AT&T. For example, System V version 4 is referred as System V.4 (V.4 for Version/Release 4).

Unix is a more generic term to describe any operating system that meets certain specific standards.

BSD is one of the most important Unixes comes from the University of California at Berkeley. At first, Berkeley Unix was based on AT&T UNIX. The official name of Berkeley Unix is BSD, Berkeley Software Distribution.

Some of Unix systems are:

- AIX – IBM;
- HP-UX – Hewlett-Packard;
- Solaris – Oracle (former Sun Microsystems);
- NeXTSTEP – NeXT (bought by Apple in 1996 and became Mac OS X);
- BSD – Berkeley Kernel Organization;
- Linux – Linux Kernel Organization;
- GNU Hurd – GNU.

Free Unix System

Many noncommercial versions of Unix are available at little or no cost. The Unix systems are maintained by many people (mostly volunteers) around the world who cooperate and communicate over the Internet.

What Is an Operating System?

An Operating System (which is software) is a complex master control program whose principal function is to use the resources of a computer efficiently.

The operating system is always there, waiting to serve you and to manage the resources of your computer.

In Unix, the operating system can be divided into three layers: utilities, shell, and kernel.

Utilities: The utilities of an operating system are the standard commands and programs associated with the operating system.

Shell: A shell is a program that runs other programs.

Kernel: The kernel is a collection of software that provides the basic capabilities of the operating system.

"Unix" Is the Name of a Culture

Unix means much more than a family of operating systems.

In using Unix, we will learn to approach and solve problems by combining simple programs into elegant structures.

The Unix Connection

Host: The main computer that actually does most of the work.

Character Terminal:

A character terminal has nothing more than a screen and a keyboard, and can display only characters.

Graphics Terminal:

It can display everything that can be drawn on a screen using small dots: pictures, geometric shapes, and so on.

Most graphics terminals have a mouse and are designed to be used with a graphical user interface.

Console:

A display screen and a keyboard that are part of the host computer itself.

A Console is just another terminal.

What Happens When You Press a Key?

Each time you press a key, a signal is sent to the host. The host responds by sending its own signal back to your terminal telling it to display the appropriate character on the screen.

If the host computer is far away, you might not see the letters appear on the screen immediately after you press keys.

Network Connections

Network

A network refers to two or more computers connected together. People connect computers into networks in order to share resources.

Local Area Network (LAN)

When computers are connected directly by using some type of cable, we call the network LAN.

Wide Area Network (WAN)

Many LANs are connected to other networks, forming a bigger network that is called WAN.

Backbone

A high-speed link that ties together the smaller LANs into one large wide area network.

Gateways

Some computers, called gateways, will act as the links between the campus network and the outside world.

Internet

Around the world, the major wide area networks are connected to a system known as the Internet. Any computer on the Internet can connect to any other computer on the Internet.

Client-Server Relationship

Server

In network terminology, any program that offers a resource is called a server.

A program that provides access to files over the network is called a file server; A program that coordinates the printing of data using different printers is called a print server.

Sometimes the name server is used to refer to a real computer too (i.e., mail server, news server, …).

Client

A program that uses a resource is called a client.

Unix system programmers often talk about the connection between a client program and a server as the Client-Server Relationship.

VT-100 Terminal:

A very old terminal made by Digital Equipment Corporation.

Starting to Use Unix

System Manager/Administrator:

All Unix systems require administration and maintenance. The person who performs these duties is called the system manager or system administrator.

Userid: A name that identifies you to the system.

Password: A secret code that you must type in each time you use the system.

Account: Once you have permission to use a system, we say that you have a Unix account on that computer.

Logging In: Starting work with Unix

Logging Out: Stopping working with Unix

1. *logout* 2. *exit* 3. *login* 4. *Ctrl-D* (for some systems)

Getting Down to Work: The Shell Prompt

The program that reads and interprets your commands is called a "shell". When the shell is ready for you to type the next command, it will display a "prompt".

If you use the C-Shell, your prompt will be a `%`.

If you system manager has customized your environment, the prompt may be somewhat different, i.e., mars.acs.uwinnipeg.ca>.

With a Bourne shell, a `$` may be your prompt.

Upper- and Lowercase

Unix is case sensitive.

Some cases to use uppercase letters:

1. Passwords
2. Environment variables (TERM, HOME, …)
3. Writing programs
4. Electronic mail (E-mail) address

   Example: uwinnipeg.ca vs. Uwinnipeg.ca

Who Has Been Using Your Account: *last*

The command *last* can display login and logout information about users and terminals.

Example:

```
mars% last
sliao    pts/1        wnpgmb0426w-ds02 Sat Sep  2 15:40   still logged in
nischal  pts/1        acs-3d07b-f01.uw Fri Sep  1 12:23 - 12:24  (00:00)
acs2941  pts/1        acs-3d07b-f01.uw Fri Sep  1 12:22 - 12:23  (00:00)
. . .
aulakh-s pts/4        142.161.217.182  Sun Mar 19 03:28 - 04:03  (00:34)

wtmp begins Sun Mar 19 03:27:57 2023
mars%
```

passwd: You can change your password by using the command *passwd*.

Userid vs. User

A user is a person who utilizes a Unix system in some way. Unix itself only knows about userids. If someone logs in with your userid, Unix has no way of knowing whether or not it is really you.

Example:

```
mars% finger
Login     Name          Tty      Idle  Login Time   Office     Office Phone   Host
sliao     Simon Liao    pts/1          Sep  2 15:40
(wnpgmb0426w-ds02-202-50-88.dynamic.bellmts.net)
mars%
```

The Superuser Userid: root

Within Unix, all userids are considered equal, except the superuser root.

Using the Keyboard with Unix

TTYs: When the Unix was first developed, the programmers used Teletype ASR33 terminals. The terminals had letters, numbers, and a "Control" key. TTY (Teletype) quickly became a way to refer to any terminal.

tty is a command to display the name of your terminal.

```
mars% tty
/dev/pts/1
mars%
```

stty is a command to set up your terminal.

Another convention derived from Teletypes is how we use the word "print". Teletype printed output on paper. But now the same information would be displayed on a screen.

"Print" means "Display"

Examples: appreciate

pwd	Print Working Directory
lpr	Line Printer

How to Deal with Different Types of Terminals?

For older Unix systems, the descriptions of all different types of terminals into a single file, termcap database. The newer Unix systems should use the terminfo database and associated libraries.

How Does Unix Know What Terminal You Are Using?

There is a global variable named TERM whose value is the type of terminal you are using.

```
mars% echo $TERM
vt100
mars%
```

Understanding Your Keyboard

Unix must work with any terminals and there is no such thing as a standard keyboard. As a solution, Unix defines standard codes that are mapped into different keyboards.

Some codes:

erase:	Erase the last character that you typed.
werase:	Erase the last word you typed.
kill:	Erase the whole line.
intr:	Abort the program that is currently running (interrupt).
quit:	quit is designed for advanced programmers. When you stop a program with quit, it not only stops the program, but also makes a copy of the contents of memory at that instant.
stop:	Pause the screen display.
start:	Restart the screen display.
eof:	End of file

Checking the Special Keys for Your Terminals: *stty*

To check how your Unix system uses your particular terminal, you can use the *stty* (set terminal) command.

```
mars% stty
speed 38400 baud; line = 0;
kill = ^X;
-brkint -imaxbel
mars%
```

You can set some special keys in your .login file.

For example, add

```
stty kill ^X
```

to your .login file will change the kill key to ^X.

Summary of some Keyboard Codes:

Code	Key	Purpose
intr	^C	stop a program that is running
erase	<Backspace>,<Delete>	erase the last character typed
werase	^W	erase the last word typed
kill	^X, ^U	erase the entire line
quit	^\	stop a program, save **core** file
stop	^S	pause the screen display
start	^Q	restart the screen display
eof	^D	indicate there is no more data

Teletype (ASR33) Control Signals

<Ctrl-H>: Caused the print carriage to back up a single space before printing the next character.

<Ctrl-M>: Moved the print carriage to the beginning of the line.

<Ctrl-J>: Moved the paper up one line.

<Ctrl-M> <Ctrl-J>: Would position the carriage and the paper at the beginning of the next line.

<Ctrl-I>: Tab Setting

How Teletype Control Signals are Used by Unix

^H: When you press the <Backspace> key, Unix interprets the signal as being a ^H.

^I: When you press the <Tab> key, Unix interprets it as a ^I.

^M: Signal that you have reached the end of a line. (return)

^J: Mark the end of each line. (newline)

Unix treats the data typed at the keyboard the same as data read from a file.

When you display data, each *newline* (^J) is changed by Unix into a *return newline* (^M^J) combination.

Q. Can you press ^J instead of <Return> to enter a command at any time?

A. Yes!

Programs to Use Right Away

date The date command will display the current time and date.

Example:

```
mars% date
Tue Sep  5 15:29:51 CDT 2023
mars%
```

cal The cal command displays a calendar.

Examples:

```
mars% cal
   September 2023
Su Mo Tu We Th Fr Sa
                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

mars%
```

calendar

Unix does have a calendar command, which is different from cal.

The calendar program offers a reminder service based on a file named calendar in your home directory. The program calendar will check this file and display all the lines that have today's and tomorrow's date.

Example:

```
pearl% cat calendar
January 5       Day1
January 6       Day2
January 7       Day3
January 8       Day4
pearl% date
Thu Jan  5 17:13:26 CST 2019
pearl% calendar
Jan 05  Day1
Jan 06  Day2
pearl%
```

uptime

The uptime command will tell you that how long your particular computer has been up.

Example:

```
mars% uptime
 15:30:55 up 20 days,  5:49,  3 users,  load average: 0.00, 0.01, 0.05
mars%
```

In this case, **mars** has been up for 20 day, 5 hour and 49 minutes, and there are 3 users currently logged in. The last three numbers show the average number of jobs in the run queue over the last 1, 5 and 15 minutes, respectively.

hostname

The hostname command will display the name of the system you are using.

Example:

```
mars% hostname
mars-acs-uwinnipeg-ca
mars%
```

The Online Unix Manual

Unix comes with a large, built-in manual that is accessible at any time from your terminal.

The Online Manual is a collection of files, stored on disk, each of which contains the documentation about one Unix command or topic.

The Online Manual can be accessed at any time by using the *man* command.

Examples:

```
man cp
man man
man mv lpr ln
```

How Is the Online Manual Organized?

Section

1 Executable programs or shell commands
2 System calls (functions provided by the kernel)
3 Library calls (functions within program libraries)
4 Special files (usually found in /dev)
5 File formats and conventions (e.g. /etc/passwd)
6 Games
7 Miscellaneous (including macro packages and conventions)
8 System administration commands (usually only for root)
9 Kernel routines (nonstandard)

The most important section is Section 1. Section 2, 3, 4, and 5 may be important to programmers.

The following conventions apply to the SYNOPSIS section and can be used as a guide in other sections:

bold text	type exactly as shown
italic text	replace with appropriate argument
[-abc]	any or all arguments within [ ] are optional
-a | -b	options delimited by | cannot be used together
argument …	argument is repeatable
[expression] …	entire expression within [ ] is repeatable
…	

Examples:

`man kill`

will show the description of **`kill`** that resides in Section 1 of the manual;

`man -s 2 kill`

will show the description of **`kill`** that resides in Section 2 of the manual;

`man -s 7 man`

will show the description of **`man`** in Section 7;

`man umask`

will show the description of **`umask`** that resides in Section 1 of the manual:

```
mars% man kill

KILL(1)                 User Commends                        KILL(1)

NAME
       kill - terminate a process

SYNOPSIS
       kill [-s signal|-p] [-q sigval] [-a] [--] pid...
       kill -l [signal]
.
.
.
```

At the end of the kill man pages are the following a few lines, which tell us there are other pages related to this one:

```
SEE ALSO
       bash(1), tcsh(1), kill(2), sigvec(2), signal(7)

AUTHOR
       Taken from BSD 4.4. The ability to translate process names to
process ids was added by Salvatore Valente …

AVAILABILITY
       The kill command is part of the util-linux package and is available
from Linux Kernel Archive …
```

NAME: This is what the command is all about.

SYNOPSIS: Official explanation of how to enter the command.

DESCRIPTION: Could be divided into two separate sections: Description & Options.

FILES: This section shows the names of the files that are used by this command.

SEE ALSO: It shows you other places to look in the manual for more information.

A Quick Way to Find Out What a Command Does

Sometimes you don't want to view a full manual page and you are interested in just a quick description.

If you want to see the Name Section, which is a one line description, you can type man followed by –f.

Example:

```
mars% man -f date cal
date (1)              - print or set the system date and time
date (1p)             - write the date and time
cal  (1)              - display a calendar
cal  (1p)             - print a calendar
mars%
```

As a convenience, you can type the single word *whatis*, instead of *man –f*.

Example:

```
mars% whatis date cal
date (1)              - print or set the system date and time
date (1p)             - write the date and time
cal  (1)              - display a calendar
cal  (1p)             - print a calendar
mars%
```

What if you know what you want to do, but not sure which command(s) to use? You can use man –k (or apropos) followed by specified keywords.

Example:

```
mars% man -k what
.
.
.
w (1)                 - Show who is logged on and what they are doing.
whatis (1)            - display manual page descriptions
mars%
```

Command Syntax

The Unix Command Line

When you enter a command, the entire line that you type is called the command line. A command line can contain multiple commands separated by semicolons.

Example:

```
date; ls -l; lpq
```

Command Syntax

The formal description of how a command must be entered is called the command syntax.

Arguments: Options & Parameters

Options: Options come right after the command and consist of a – (minus sign) followed by a letter.

Parameters: Parameters come after the options.

Examples:

```
ls -a -l file1 file2
ls -al file1 file2
```

The Format of a Unix Command:

```
command-name options parameters
```

Whitespace

One or more consecutive spaces or tabs.

Two important expressions:

One or More: You must use at least one of something.

Zero or More: It is okay to leave it out.

Examples:

```
whatis man cp
man man
ls -l
ls -al file1 file2

ls
who
uptime
```

With **ls**, the default is the set of files in your working directory.

The Formal Description of a Command: Syntax

The syntax of a command is its "official" description.

The syntax that is used to describe Unix commands follows five simple rules:

1. Items in square brackets ([ ]) are optional.
2. Items not in square brackets are obligatory and ***must*** be entered as part of the command.
3. Anything in boldface ***must*** be typed ***exactly*** as written.
4. Anything in *italics* must be replaced by an appropriate value.
5. Any parameter that is followed by an ellipsis(…) may be repeated *any number* of times.

Example:

```
ls  [-aAbBcCdDfFgGhHiIklLmnNopqQrRsStTuUvwxXZ1] [file ...]
```

a) The command has 40 options. Since they are optional, they are enclosed in square brackets.

b) There is one parameter, ***file***, which is optional.

c) The name of the command and the options are printed in **boldface**. It means that they must be typed exactly as they appear.

d) The parameter is in *italics*. It means that you must replace it with an appropriate value.

e) The parameter is followed by "…". That means that you can use more than one parameter. Since the parameter is optional itself, the "Zero or More" will apply here.

Examples:

```
ls
ls -l
ls file1
ls file1 file2 file3
ls -f -l file4
ls -fl file4
```

The Shell

From the beginning, Unix was designed so that the shell is an actual program separated from the main part of the operating system.

What is a shell?

A shell is a command processor -- a program that reads and interprets the commands that you enter.

A shell is a programming language. You can write programs, called script, for a shell to interpret. These scripts can contain regular Unix commands, as well as special shell programming commands.

A shell is your main interface into Unix. Using the facilities that are built into your shell, you can create a highly customized environment for yourself.

There are several shells. Therefore, you can have a choice as to which interface you want to use.

The Bourne Shell Family

Bourne shell (*sh*)

The earliest Unix shell developed by Steven Bourne of AT&T Bell Labs. It is still in use.

Korn shell (*ksh*)

The Korn shell is an upwards compatible extension to Bourne shell. It was developed by David Korn, a Bell Labs scientist, in the mid-1980s.

Bash (*bash*)

Bash stands for “the Bourne Again SHell”. It is the product of the Free Software Foundation.

Bash extends the capabilities of the basic Bourne shell in a manner similar to the Korn shell.

Zsh (*zsh*)

Zsh was developed by Paul Falstad in 1990. Zsh offers all of the important features of the other Unix shells as well as new capabilities. It is popular among programmers and advanced Unix users.

The C-Shell Family

C-Shell (*csh*)

The C-Shell was designed by Bill Joy as the Berkeley Unix alternative to the Bourne shell.

The C-Shell offers many advantages over the Bourne shell. It is very popular among experienced Unix users, especially at universities and research organizations.

Tcsh (*tcsh*)

Tcsh is an enhanced C-Shell that offers advanced features.

What Shell Should You Use?

Worldwide, the most widely used shells are the Bourne shell and its replacement, the Korn shell. However, in the academic, research, and programming communities, the C-Shell is the most popular shell.

As a command processor, the C-Shell family provides a good all-around working environment.

The programming language used by the Bourne shell family is easier and more pleasant to use than that of the C-Shell.

The Relative Complexity of Different Shells

Name of Shell	Size of the Man page (bytes)	Relative Complexity
rc	37,885	1.00
Bourne	44,500	1.17
C-Shell	76,816	2.03
Bash	127,361	3.36
Zsh	133,565	3.53
Korn	141,391	3.73
Tcsh	199,834	5.27

Changing Your Shell Temporarily

When you log in, the shell Unix starts automatically is called your login shell.

Since a shell is a program itself, you can start a new shell any time by entering the name of that shell.

When you are finished with the new shell, you can stop it by entering either the *exit* command or press ^D, the eof key.

You must go back to the login shell before you can log out.

Using the C-Shell

Shell Variables

A shell variable is an item, known by a name, which represents a value of some type. The value of a shell variable can be changed.

There are two types of shell variables:

1. Variables that act as off/on switches.
2. Variables that store a particular value as a string of characters.

Shell Variables That Act as Switches: *set*, *unset*

To turn on switch variables, use the *set* command.

Syntax: set [*variable-name*]

Examples:

```
set ignoreeof
set filec
```

To turn off a switch, use unset command.

Examples:

```
unset ignoreeof
unset filec
```

To display all shell variables and their current settings, enter the command set without arguments:

```
set
```

If a variable is set, its name will appear in the list. If a variable is unset, its name will not appear.

Built-in Shell Variables: Switches

Variable Name	Purpose
echo	display each command before execution
filec	enable filename completion
ignoreeof	must log out with logout instead of eof key (^D)
nobeep	no beep if filename is ambiguous
noclobber	do not allow redirected output to replace a file
noglob	inhibit expansion of filenames
nonomatch	no error if filename expansion matches nothing
notify	notify about job completions at any time
verbose	display full command after history substitution

Shell Variables That Store Values: set

Some of the shell variables can be set by you to modify the shell's behavior. Other values are set by the shell to pass information to you.

To set a variable of this type, use the set command with the following syntax:

`set` [*variable-name = value*]

Example:

```
set history=50
```

Built-in Shell Variables That Store Values

Variable Name	Purpose
argv	list of arguments for current command
cdpath	directories to search to find a subdirectory
cwd	pathname of current working directory
fignore	suffixes to ignore during the file name completion
hardpath	no symbolic link pathnames in directory stack
histchars	the two characters used for history substitution
history	size of the history list
home	pathname of your home directory
mail	pathnames where shell should check for mail
path	list of directories to search for programs
prompt	string of characters used for command prompt
savehist	number of history lines to save upon logout
shell	pathname of the shell program
status	return status of the last command
term	type of terminal you are using
time	threshold value for reporting of command timing
user	name of the userid currently logged in

On occasion, you may want to give a variable a value that contains spaces or other special characters. In that case, you must put single quotes around the value.

Examples:

```
mars% set prompt='My Own Prompt>'
My Own Prompt>echo $prompt
My Own Prompt>
My Own Prompt>set prompt='mars% '
mars% echo $prompt
mars%
mars%
```

Displaying the Value of a Variable: *echo*

The command *echo* simply displays the value of anything you give it.

Example:

```
mars% echo Have a nice day!
Have a nice day!
mars%
```

To display the value of a single variable, use the *echo* command with the following syntax:

```
echo $variable-name
```

Examples:

```
mars% echo $history
100
mars% set history=200
mars% echo $history
200
mars%
```

```
mars% echo My working directory is $cwd and my home directory is $home.
My working directory is /home/sliao and my home directory is /home/sliao.
mars%
```

Environment Variables

The shell variables are used only within the shell to control preferences and settings.

There is a whole other set of variables that the shell maintains for passing values between programs. These are called Environment Variables or Global Variables.

All environment variables have uppercase names.

Common Environment Variables

Variable Name	Purpose
EDITOR	pathname of your text editor
HOME	pathname of your home directory
LOGNAME	name of the userid currently logged in
MAIL	pathname of your mail program
MANPATH	list of directories to search for manual pages
PAGER	name of the paging program you prefer
PATH	list of directories to search for programs
SHELL	pathname of the shell program
TERM	type of terminal you are using
USER	name of the userid currently logged in

The value of an environment variable is available to any program or shell.

Example:

Many programs look at the TERM variable to see what type of terminal you are using.

Setting the Value of an Environment Variable

setenv

The syntax is setenv [*variable-name value*]

Example: setenv TERM vt100

Displaying the Value of Environment Variable(s)

printenv

The syntax is

printenv [*variable-name*]

Examples:

printenv will display all the environment variables

printenv TERM

echo

Example: *echo $TERM*

How Environment and Shell Variables Are Connected

There are six common shell variables that have the same names as environment variables (except that environment variables have uppercase names).

1) home, shell vs. HOME, SHELL

home contains the pathname of your home directory and shell contains the pathname of the shell you use. Whenever you log in, Unix automatically sets the values of home and shell (and HOME and SHELL). Your programs will examine these variables from time to time, but you will probably never need to change them yourself.

2) term, path, user vs. TERM, PATH, USER

The local variables term, path, and user are tied to the corresponding global variables.

3) mail vs. MAIL They are not connected.

Commands That Are Built Into the Shell

When you enter a command, the shell will break the command line into parts. The first part of each command is the name, the other parts are options and parameters.

Some commands are internal to the shell. The shell can interpret these commands directly.

The Number of Internal Commands in Each Shell

Name of Shell	Internal Commands
Bourne	32
Korn	43
Bash	50
C-Shell	52
Tcsh	56
Zsh	73

The Search Path

If a command is not built into the shell, the shell must find the appropriate program to execute.

The path variable tells the shell where to look for programs. The value of path is a list of directory names called the search path.

When the shell is looking for a program to execute, it checks each directory in the search path in the order they are specified.

Example:

```
set path = ( . /usr/local/bin /usr/bin ~/bin )
```

bin is often used to indicate a directory that holds programs.

Tilde (~) stands for the name of your home directory.

Dot (.) is the current working directory.

The shell automatically copies the value of path to PATH.

Setting Up History Substitution: history

History Substitution

A feature that lets you change and re-enter a previous command without having to retype it.

At all times, the shell saves your commands in a list called the History List. Each command is given an identification number (starting at 1). Whenever you enter a command, the identification number increases by 1.

You can determine how long the history list should be by setting the shell variable history.

To display the history list, use the history command. The syntax is:

```
history [-r] [number]
```

Examples:

```
mars% history 6
    22  ls
    23  date
    24  w
    25  who
    26  finger sliao
    27  history 6
mars% history -r 3
    28  history -r 3
    27  history 6
    26  finger sliao
mars%
```

Event Number

In C-Shell, past commands are referred to as Events. The number that identifies each command is called an Event Number.

You can display the event number within a prompt by using an exclamation mark (!). Whenever the shell displays the prompt, it will replace the ! with the current event number.

Example:

```
mars% set prompt='mars [\!]% '
mars [121]% uptime
 20:18:26 up 64 days, 10:12,  2 users,  load average: 0.00, 0.01, 0.05
mars [122]%
```

Using History Substitution

The C-Shell supports a large variety of complex substitutions.

!! Re-use the previous command, exactly as you typed.

Example:

```
mars [124]% uptime
 20:19:37 up 64 days, 10:13,  2 users,  load average: 0.00, 0.01, 0.05
mars [125]% !!
uptime
 20:19:39 up 64 days, 10:13,  2 users,  load average: 0.00, 0.01, 0.05
mars [126]%
```

! event_number

To re-use an older command, use an ! followed by the event number for that command.

Example:

```
mars [127]% history 3
   125  20:19   uptime
   126  20:21   clear
   127  20:21   history 3
mars [128]% !125
uptime
 20:21:12 up 64 days, 10:15,  2 users,  load average: 0.02, 0.02, 0.05
mars [129]%
```

It is also possible to add characters to the end of a command.

Example:

```
mars [138]% history 3
   136  20:23   clear
   137  20:25   ls -l
   138  20:25   history 3
mars [139]% !137 calendar
ls -l calendar
-rw-------. 1 sliao sliao 100 Jan  2 20:36 calendar
mars [140]%
```

^^

To replace a string of characters in the previous command, type ^, followed by the characters you want to replace, followed by another ^, followed by the new characters.

Example:

```
mars [140]% historry 3
historry: Command not found.
mars [141]% ^rr^r
history 3
   139  20:25   ls -l calendar
   140  20:26   historry 3
   141  20:26   history 3
mars [142]%
```

!pattern

To re-use a command that begins with a particular pattern, enter ! followed by that pattern.

Example:

```
mars [144]% history 3
   142  20:27   clear
   143  20:27   uptime
   144  20:28   history 3
mars [145]% !u
uptime
 20:28:16 up 64 days, 10:22,  2 users,  load average: 0.00, 0.01, 0.05
mars [146]%
```

!?pattern?

You can also reference a command by specifying a pattern within question marks(?). The shell will execute the last command that contained this pattern.

Example:

```
mars [146]% history 4
   142  20:27   clear
   144  20:28   history 3
   145  20:28   uptime
   146  20:29   history 4
mars [147]% !?ti?
uptime
 20:30:08 up 64 days, 10:24,  2 users,  load average: 0.00, 0.01, 0.05
mars [148]%
```

!*

This combination stands for everything on the command line after the name of the command.

Example:

```
mars [156]% ls -l file1 calendar
-rw-------. 1 sliao sliao 100 Jan  2 20:36 calendar
-rw-------. 1 sliao sliao   8 Jan 10 20:32 file1
mars [157]% ls !*
ls -l file1 calendar
-rw-------. 1 sliao sliao 100 Jan  2 20:36 calendar
-rw-------. 1 sliao sliao   8 Jan 10 20:32 file1
mars [158]%
```

Example: Avoid Deleting the Wrong Files

```
mars% ls
extra1   extra11  extra2   extra22  temp1    temp2    temp5
mars% ls temp* extra?
extra1  extra2  temp1   temp2   temp5
mars% rm !*
rm temp* extra?
mars% ls
extra11  extra22
mars%
```

When you use **!*** in the above example to remove files, you are guaranteed to get what you want. If you retype the patterns, you might make a typing mistake.

Command Aliasing: alias, unalias

An alias is a name that you give to a command or list of command. You can then type the name of the alias instead of the commands.

To create an alias, use the alias command. The syntax is:

alias [*name* [*command*]]

Examples:

```
mars% ls -l File*
-rw-------. 1 sliao sliao 29 Jan 10 20:41 File1
-rw-------. 1 sliao sliao 21 Jan 10 20:41 File2
mars% lf
lf: Command not found.
mars% alias lf 'ls -l File*'
mars% lf
-rw-------. 1 sliao sliao 29 Jan 10 20:41 File1
-rw-------. 1 sliao sliao 21 Jan 10 20:41 File2
mars%

mars% mytime
mytime: Command not found.
mars% alias mytime 'date; uptime'
mars% mytime
Tue Jan 10 20:43:24 CST 2023
 20:43:24 up 64 days, 10:37,  2 users,  load average: 0.00, 0.01, 0.05
mars%
```

More examples:

alias — will display all the aliases

alias a alias

a h history

To remove an alias, use the unalias command. The syntax is

unalias alias-name

Using Arguments with an Alias

When you use an alias, you can add arguments (options and parameters) to the end of the command line.

Example:

```
mars% ll File* extra*
ll: Command not found.
mars% alias ll ls -l
mars% ll File* extra*
-rw-------. 1 sliao sliao 20 Jan 10 20:47 extra11
-rw-------. 1 sliao sliao 13 Jan 10 20:48 extra22
-rw-------. 1 sliao sliao 29 Jan 10 20:41 File1
-rw-------. 1 sliao sliao 21 Jan 10 20:41 File2
mars%
```

The shell will replace the ll alias and then tack the parameters onto the end of the command.

If you want to insert arguments into the middle of an alias, you can refer to them as !*.

Example:

```
mars% lld
lld: Command not found.
mars% alias lld 'ls -l \!*; date'
mars% lld File*
-rw-------. 1 sliao sliao 29 Jan 10 20:41 File1
-rw-------. 1 sliao sliao 21 Jan 10 20:41 File2
Tue Jan 10 20:51:02 CST 2023
mars%
```

The shell replaces `lld File*`

with

`ls -l File*; date`

Example: Keeping Track of Your Working Directory

cd	Change Directory
pwd	Print Working Directory
cwd	Shell variable, contains the name of your current working directory.

If you want to display the new working directory every time you change directories, you can make the following alias:

```
alias cd 'cd \!*; echo $cwd'
```

Then, whenever you change directories, you will know exactly where you are.

Example:

```
mars% alias cd 'cd \!*; echo $cwd'
mars% cd /usr/bin
/usr/bin
mars% cd
/home/sliao
mars%
```

Initialization and Termination Files

The shell provides a way to specify certain commands once and have them executed at the appropriate time.

The C-Shell recognizes three special files:

.cshrc — Every time a new shell is started, the commands (i.e., setting shell variables, defining aliases, …) in the .cshrc file are executed. It happens when you login, also happens whenever you run a shell script.

.login — The commands (i.e., setting up terminal, defining environment variables, setting user masks, …) in the .login file are executed when you log in (only once).

.logout — The commands in .logout file will be executed when you log out (only once).

Example:

Add `history | cut -c8- > ~/.history`

to your .logout file will carry the history list from last login.

Note: rc stands for "run commands", eg, .mailrc, .newsrc, .exrc

Communicating with Other People

One of the wonderful things about Unix is that every time you log on a system that is connected to the Internet, you become a member of a global electronic community.

Take a look around our local system:

users displays the name of each userid that is logged in

Example:

```
mars% users
acs2941 sliao sliao
mars%
```

who shows more information than users does

Example:

```
mars% who
mars% who
nischal  pts/0        2023-08-21 15:27 (:0)
chan-w51 pts/1        2023-09-14 11:57 (10.64.21.147)
rajput-s pts/2        2023-09-14 11:26 (10.141.255.89)
sliao    pts/4        2023-09-14 12:41 (wnpgmb0426w-ds02-202-50-
88.dynamic.bellmts.net)
mars%
```

Finding Out What Someone Is Doing: w

w [-hsu] [*userid*]

-h display the long report without the heading line

-s display a short report, which contains the information of User, tty, idle, and what

-u work as the uptime command

Example:

```
mars% w
 12:42:51 up 29 days,  3:01,  2 users,  load average: 0.10, 0.06, 0.05
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
rajput-s pts/2    10.141.255.89     11:26    1:16m  0.03s  0.03s -csh
sliao    pts/4    wnpgmb0426w-ds02 12:41    3.00s  0.08s  0.04s w
mars%
```

USER	userid
TTY	terminal name
LOGIN@	time of login
IDLE	time since the user last pressed a key (idle time)
JCPU	processor time used by all processes (jobs) since login
PCPU	processor time used by the current process
WHAT	the command (and its arguments) that is running

Public Information about a Userid: /etc/passwd

Unix maintains the information about userids, which is available to anyone within the same system. On many systems, this file is named /etc/passwd.

Example:

```
sliao:x:1000:1000:Simon Liao:/home/sliao:/bin/csh
```

The first field is the userid, i.e., sliao.

The second field was the encoded password.

The third and fourth fields contain the numeric value for the userid and the groupid.

The fifth part is used to hold personal data.

The sixth field shows the userid's home directory.

The last field contains the name of the login shell.

Displaying Public Information about a Userid: finger

finger [-lmp] [*username* ...]

The most common way to use the command finger is to specify the name of a particular userid.

Example:

```
mars% finger sliao
Login: sliao                            Name: Simon Liao
Directory: /home/sliao                  Shell: /bin/csh
On since Thu Sep 14 12:41 (CDT) on pts/4 from wnpgmb0426w-ds02-202-50-
88.dynamic.bellmts.net
   4 seconds idle
New mail received Thu Sep 14 11:36 2023 (CDT)
     Unread since Thu Sep 14 09:16 2023 (CDT)
Project:
.project line 1
.project line 2
.project line 3
.project line 4
Plan:
.plan line 1
.plan line 2
.plan line 3
.plan line 4
mars%
```

Checking to See if a Computer Is Alive: ping

If you want to check if a computer is actually connected to the Internet and is alive, you can use the ping command.

Networks and Addresses

Once you are a part of the Unix community, you can communicate with other people and transfer data all over the world. All you need is an electronic address and the knowledge of how to use the networking programs.

An Overview of the Unix Mail System

mail: electronic mail (not regular post office mail)

address: electronic mailing address

TCP/IP: the common name for a collection of more than 100 different protocols

protocol: a set of rules that allow different machines and programs to coordinate with one another

TCP: Transmission Control Protocol

When you send a message, it is TCP that breaks the data into packets, sequences the packets, adds some error control information and then sends the packets out to be delivered.

At the other end, TCP receives the packets, checks for errors, and combines all of the packets back into the original data. If there is an error somewhere, the recipient computer will ask the sender to re-send that packet.

IP: Internet Protocol

IP moves the data packets from one place to another. The computers that direct data from one network to another are called routes.

Tracing the Route from Your Computer to Another

traceroute *computer-name*

Examples:

```
mars% traceroute mars.uwinnipeg.ca
traceroute to mars.uwinnipeg.ca (142.132.32.4), 30 hops max, 60 byte packets
 1  mars-acs-uwinnipeg-ca (142.132.32.4)  0.059 ms  0.027 ms  0.029 ms
mars%


pearl% traceroute cs.ubc.ca
traceroute to cs.ubc.ca (142.103.6.5), 30 hops max, 60 byte packets
 1  142.132.145.9 (142.132.145.9)  0.464 ms  1.989 ms  2.235 ms
 2  142.132.140.97 (142.132.140.97)  0.265 ms  0.297 ms  0.365 ms
 3  142.132.192.1 (142.132.192.1)  1.346 ms  1.336 ms  1.325 ms
 4  mrsrouter-uofw.mrnet.mb.ca (192.139.69.105)  1.314 ms  1.303 ms  1.292 ms
 5  205.189.32.252 (205.189.32.252)  1.398 ms  1.387 ms  1.376 ms
 6  clgr2rtr1.canarie.ca (205.189.32.176)  15.545 ms  15.415 ms  15.378 ms
 7  vncv1rtr1.canarie.ca (205.189.32.174)  26.313 ms  25.871 ms  25.816 ms
 8  288-canarie-oran-cr1.vncv1.bc.net (205.189.32.173)  26.026 ms  26.015 ms  26.008 ms
 9  cr1-bb3900.vantx2.bc.net (206.12.0.34)  25.989 ms  25.984 ms  25.993 ms
10  343-oran-ubcab-cr1.vncv1.bc.net (134.87.2.233)  26.467 ms  26.456 ms  26.602 ms
11  a0-anguborder.net.ubc.ca (137.82.123.138)  26.500 ms  31.608 ms  31.600 ms
12  a1-a0.net.ubc.ca (142.103.78.249)  26.746 ms  26.737 ms  26.725 ms
13  137.82.73.13 (137.82.73.13)  26.883 ms  26.868 ms  26.861 ms
14  www.cs.ubc.ca (142.103.6.5)  27.025 ms  27.020 ms  27.006 ms
pearl%
```

```
pearl% traceroute cosc.canterbury.ac.nz
traceroute to cosc.canterbury.ac.nz (132.181.17.3), 30 hops max, 60 byte packets
 1  142.132.145.9 (142.132.145.9)  0.459 ms  1.124 ms  1.308 ms
 2  142.132.140.97 (142.132.140.97)  0.241 ms  0.310 ms  0.350 ms
 3  142.132.192.1 (142.132.192.1)  1.249 ms  1.240 ms  1.230 ms
 4  mrsrouter-uofw.mrnet.mb.ca (192.139.69.105)  1.229 ms  1.219 ms  1.208 ms
 5  205.189.32.252 (205.189.32.252)  1.246 ms  1.235 ms  1.225 ms
 6  clgr2rtr1.canarie.ca (205.189.32.176)  15.529 ms  15.424 ms  15.348 ms
 7  vncv1rtr1.canarie.ca (205.189.32.174)  26.293 ms  25.813 ms  25.829 ms
 8  aarnet-2-lo-jmb-706.sttlwa.pacificwave.net (207.231.240.4)  29.181 ms  29.232 ms  29.226 ms
 9  et-2-0-0.pe2.brwy.nsw.aarnet.net.au (113.197.15.98)  171.395 ms  171.388 ms  171.378 ms
10  et-2-3-0.pe1.sxt.alxd.nsw.aarnet.net.au (113.197.15.79)  172.095 ms  172.086 ms  172.076 ms
11  et-0-3-0.pe1.wnpa.akl.aarnet.net.au (113.197.15.77)  194.027 ms  193.661 ms  193.652 ms
12  et-1-0-0-202.and05-mdr.reannz.co.nz (182.255.119.205)  194.590 ms  194.616 ms  194.460 ms
13  210.7.37.209 (210.7.37.209)  207.873 ms  207.846 ms  207.800 ms
14  210.7.37.210 (210.7.37.210)  208.266 ms  208.224 ms  208.209 ms
15  202.36.179.100 (202.36.179.100)  209.008 ms  212.632 ms  212.627 ms
16  132.181.3.236 (132.181.3.236)  212.810 ms  213.375 ms  213.616 ms
17  * * 132.181.17.3 (132.181.17.3)  210.599 ms
pearl%
```

What Is the Internet

Technical Definition:

The Internet is a worldwide collection of networks that transmit data using the IP protocol.

Practical Definition:

The Internet is a worldwide network that offers the services of mail, file transfer, remote login, discussion groups, accessing information, and talking with other people.

Redirection and Pipes

The Unix toolbox makes Unix different from other operating systems.

The Unix Philosophy

1. Each program or command should be a tool that does only one thing and does it well.
2. When you need a new tool, it is better to combine existing tools than to write new ones.

Some people describe this philosophy as:

"Small is beautiful"

Standard Input and Standard Output

The concept of standard input and output is central for using Unix effectively.

The basic idea:

"Every program should be able to accept input from any source and write output to any target."

Advantages:

For users:

- You can define the input and output for a program as you see fit.
- Need to learn only one program for each task.

For programmers:

- Writing programs becomes a lot easier.
- Programmers don't have to worry about all the variations of input and output, and can concentrate on the details of their programs and depend on Unix to handle the standard resources for them.

Redirecting Standard Output

When you log in, the shell automatically sets standard input to be your keyboard and standard output to be your screen.

However, every time you enter a command, you can tell the shell to reset the standard input or output for the duration of that command.

If you want the output of a command to go to a file, type a **>** followed by the name of the file at the end of the command. For example,

```
sort > result
```

will send the output of sort to a file named result. If the file result does not exist, Unix will create it. If the file result already exist, its contents will be replaced.

You also can use **>>** to append data to the end of an existing file. For example,

```
sort >> result
```

will create a file named result if it does not exist. If the file result already exists, the new data will be appended to the end of the file result.

When we send the standard output to a file, we say that we redirect it. Both above commands redirect their output to the file result.

Protecting Files from Being Replaced by Redirection

In C-Shell, there is a variable noclobber. Once it is set, you can have built-in protection. Type:

```
set noclobber
```

Then, if the file result already exists and you enter

```
sort > result
```

You will see

```
result: File exists
```

If you really want to replace the file, type an **!** after the **>** character:

```
sort >! result
```

This will override the automatic check.

When variable is set noclobber on and you try to append data to a file that does not exist, for example:

```
sort >> notafile
```

You will see a message like:

```
notafile: No such file or directory
```

If you really want to create a file, type

```
sort >>! notafile
```

Then you are going to override the automatic check.

Pipelines

Pipe: **|** (vertical bar)

If you want the output of a command to go to another program for further processing, use pipe. For example,

```
sort | lpr
```

will send the standard output of the sort program to the lpr program for printing. It reads:

"The sort program pipes its output to the lpr program."

Example:

```
cat file1 file2 | grep any_string | sort | lpr
```

will do the following tasks:

1. Combine two files, file1 and file2, with the cat program;
2. Send the output of the cat program to grep;

3. Program grep extracts all the lines of data that contain a specified string of characters;
4. Program sort sorts the output of program grep;
5. Program lpr prints the data (with a line printer).

When we combine commands in this manner, we call it a pipeline.

Redirecting Standard Input

By default, the standard input is set to your keyboard. For example, if you enter

```
sort
```

then the sort command is waiting for you to enter data. If you enter:

```
time
history
a
brief
of
^D
```

The sort program will sort all the data and write it to the standard output:

```
a
brief
history
of
time
```

If you want the shell to tell a program to read its data from a file, you can use **<**. For example,

```
sort < temp
```

will sort the data contained in a file named temp.

More Examples

Example 1:

```
Mail userid1 userid2 userid3 < notice
```

will send the file notice to three userids.

Example 2:

It is possible to redirect both the standard input and standard output at the same time:

```
sort < rawdata > result
```

reads data from a file named rawdata, sorts it, and writes the output to a file named result.

Splitting a Pipeline with Tees: tee

If you want to the output of a program to go to two or more places at the same time, you can use the tee command:

```
tee [-a] file...
```

Example:

```
cat file1 file2 | grep a_string | sort | tee save1 save2| lpr
```

will copy the output to two files, save1 and save2.

In the above example, if either save1 or save2 does not exist, tee will create it for you. If save1 or save2 already exists, tee will override it and the original contents will be lost.

If you want to have tee add data to the end of an existing file, use the **-a** option:

```
cat file1 file2 | grep a_string | sort | tee –a save| lpr
```

Filters

A filter is any program that reads from standard input and writes to standard output.

The Simplest Filter: cat

The simplest possible filter is cat. All cat does is to copy data from standard input to standard output. For example, after you enter

```
cat
```

the system will be waiting for you to input data. When you press <Return> at the end of each line, the line will be sent to cat, which will copy it to the standard output. The result is that each line you type will be displayed twice.

What's the point to use a filter that does not do anything?

Examples:

```
cat > a_file                (cat >> a_file)
```

will create a file named a_file.

```
cat < another_file
```

will display a file named another_file.

```
cat < file1 > copy_of_file1
```

will make a copy of file1 by redirecting both the standard input and output.

Look how much we can do with a filter that does nothing!

Increasing the Power of Filters

The definition of a filter requires it to read its data from the standard input.

What if we also have the option of reading from a file whose name is specified as a parameter. For example, instead of having to enter

```
cat < file
```

we could enter

```
cat file
```

The change might not be significant in this case, but it makes it possible to read from multiple files.

Here is an abbreviated version of the syntax for the cat command:

```
cat [file...]
```

where *file* is the name of a file which cat can read.

Examples:

```
cat file1 file2 file3
cat file1 file2 file3 | sort > file4
```

cat is more powerful now, but cat itself must be more complex.

Many filters are extended in this way.

A List of Useful Filters

Filters	Purpose
cat	combine files; copy standard input to standard output
colrm	remove specified columns from each line of data
cut	extract selected portions (columns) of each line
grep	extract lines that contain a specified pattern
head	display the first few lines of data
look	extract lines beginning with a specified pattern
more	display data, one screenful at a time
paste	combine columns of data
rev	reverse order of characters in each line pf data
sort	sort or merge data
spell	check data for spelling errors
tr	translate or delete selected characters
uniq	look for repeated lines
wc	count number of lines, words, or characters

Combining Files: cat

The cat program copies data, unchanged, to the standard output. The data can come from the standard input or from one or more files. The syntax is:

cat [-bn] [*file*…]

-n	place a line number in front of each line
-b	used with –n to tell cat not to number blank lines

Examples:

```
cat name address phone
cat name address phone > info
cat name address phone | sort
```

There is one thing that needs to be avoided:

```
cat name address phone > name
```

Unix sets up the output file *before* starting the cat program. Thus, the file name will be cleared out before cat reads and combines its input. By the time cat looks in name, it is already empty.

If you want to add address and phone to name, you can

cat address phone >> name

Extract Selected Columns of Each Line: cut

The cut command extracts columns of data. One of the syntaxes of the cut command is:

cut –c *list* [*file*…]

where *list* is a list of columns to be extracted.

Examples:

```
mars% cat students
012-34-5678  Ambercrombie, Al    01/01/72  555-1111
123-45-6789  Barton, Barbara     02/02/73  555-2222
234-56-7890  Canby, Charles      03/03/74  555-3333
345-67-8901  Danfield, Deann     04/04/75  555-4444
.    .    .    .    .    .    .    .    .    .
mars% cut -c 14-30 students
Ambercrombie, Al
Barton, Barbara
Canby, Charles
Danfield, Deann
  .    .    .
mars%
mars% cut -c 14-30,42-49 students
Ambercrombie, Al   555-11
Barton, Barbara    555-22
Canby, Charles     555-33
Danfield, Deann    555-44
  .    .    .        .
mars%
```

Another important syntax of the cut command is:

cut –f *list* [-d *delim*] [-s] [*file*…]

The *list* following the –f option is a list of fields assumed to be separated by a delimiter character.

Examples:

```
mars% cat students.backup
012-34-5678:Ambercrombie, Al:01/01/72:555-1111
123-45-6789:Barton, Barbara:02/02/73:555-2222
234-56-7890:Canby, Charles:03/03/74:555-3333
345-67-8901:Danfield, Deann:04/04/75:555-4444
mars% cut -f 1 –d ':' students.backup
012-34-5678
123-45-6789
234-56-7890
345-67-8901
mars% cut -f 2,4 –d ':' students.backup
Ambercrombie, Al:555-1111
Barton, Barbara:555-2222
Canby, Charles:555-3333
Danfield, Deann:555-4444
mars%
```

Combining Columns of Data: paste

The paste command combines columns of data and has a great deal of flexibility. The syntax of the paste command is:

paste [-d *char*] *file*… (d: delimiter)

where *char* is a character to be used as a separator.

Examples:

```
mars% cat id
012-34-5678
123-45-6789
234-56-7890
345-67-8901
mars% cat name
Ambercrombie, Al
Barton, Barbara
Canby, Charles
Danfield, Deann
mars% cat birthday
01/01/00
02/02/01
03/03/02
04/04/03
mars% cat phone
555-1111
555-2222
555-3333
555-4444
mars% paste -d'$' id name birthday phone > info
mars% cat info
012-34-5678$Ambercrombie, Al$01/01/00$555-1111
123-45-6789$Barton, Barbara$02/02/01$555-2222
234-56-7890$Canby, Charles$03/03/02$555-3333
345-67-8901$Danfield, Deann $04/04/03$555-4444
mars%
```

Check Data for Spelling Errors: spell

The spell command will read data and generate a list of all the words that look as if they are misspelled.

The spell command is actually one of a family of programs that uses a master file of stored words to provide a spell-checking service. This file is /usr/share/dict/words (could be a link).

The syntax of the spell command is:

spell *file*…

Examples:

```
mars% cat test_file
The background color of my cheque is light bluue.
mars% spell test_file
bluue
cheque
mars%
```

Extracting Lines Beginning with a Specified Pattern: look

The look command will search data that is in alphabetical order and will find all the lines that begin with a specified pattern. The syntax of the look command is:

look [-df] *string* [*file*…]

where *string* is the pattern to search for.

-d Tell look to consider only upper and lowercase letters, numbers, tabs, and spaces.

-f Tell look to treat uppercase the same as lowercase.

look cannot read from the standard input. Thus, it is not really a filter and cannot be used within a pipeline.

Example:

```
mars% look 3 info
345-67-8901$Danfield, Deann $04/04/03$555-4444
mars%
```

If you use look without specifying a source of input, look will examine the file /usr/share/dict/words, using both the –d and –f options.

Examples:

```
mars% look winni
Winni
Winnick
Winnie
Winnifred
winning
winningly
winningness
winnings
winninish
Winnipeg
winnipeg
Winnipegger
Winnipegosis
Winnipesaukee
Winnisquam
mars% look manito
manito
Manitoba
manitoba
Manitoban
manitos
Manitou
manitou
Manitoulin
manitous
Manitowoc
mars%
```

Counting Lines, Words, and Characters: wc

The wc (word count) command counts lines, words, and characters. (wc considers a "word" to be an unbroken sequence of characters.)

The syntax of the wc command is:

wc [-lwc] [*file*…]

-l counts lines

-w counts words

-c counts characters

Example:

```
mars% cat names
Barbara
Al
Al
Cathy
Barbara
mars% wc names
 5  5 28 names
mars%
```

In this case, the file names has 5 lines, 5 words, and 28 characters.

You can specify more than one file at a time. For example,

wc file1 file2 file3

might output

```
2       13       71 file1
3       17       85 file2
4       24       99 file3
9       54      255 total
```

Sorting Data: sort

The syntax for using sort to sort data is:

sort [-dfru] [-o *outfile*] [*infile*…]

-d considers only letters, numerals, and spaces.

-f treats uppercase letters as lowercase.

-r sorts the data in reverse order.

-u looks for identical lines and suppress all but one.

outfile is the name of a file to hold the output, and *infile* is the name of a file that contains input.

Examples:

```
mars% cat names
Barbara
Al
Al
Cathy
Barbara
mars% sort names
Al
Al
Barbara
Barbara
Cathy
mars% sort -u names
Al
Barbara
Cathy
mars% sort -ru names
Cathy
Barbara
Al
mars%
```

The `-o` option will allow you to save the output in the same file.

Examples:

```
sort -o names names
```

will sort the contents of names and save the sorted result in names.

```
sort -o names names oldnames extranames
```

will sort the data from names, oldnames, and extranames, and save the output in the file names.

The ASCII Code

By default, data is stored in ascending order according to a specification called the ASCII code. The ASCII code is a description of the entire set of 128 different characters.

The order of characters in the ASCII code is as follows:

- control characters (including the tab)
- the space character
- (symbols) `! " # $ % & ' ( ) * + , - . /`
- (the numerals) `0 1 2 3 4 5 6 7 8 9`
- (more symbols) `: ; < = > ? @`
- (uppercase letters) `A B C … Z`
- (more symbols) `[\ ] ^ _ ``
- (lowercase letters) `a b c … z`
- (more symbols) `{ | } ~`
- the del (null) character

Look for Repeated Lines: uniq

The uniq command will look for consecutive, duplicate lines. uniq can perform four different tasks: retain only duplicate lines, retain only unique lines, eliminate duplicate lines, and count how many times lines are duplicated.

The syntax if the uniq command is:

uniq [-cdu] [*infile* [*outfile*]]

- -c counts how many times each line is found.
- -d retains one copy of all lines that are duplicated.
- -u retains those lines that are not duplicated.

Examples:

```
mars% cat names
Barbara
Al
Al
Cathy
Barbara
mars% uniq -c names
      1 Barbara
      2 Al
      1 Cathy
      1 Barbara
mars% uniq -d names
Al
mars% uniq -u names
Barbara
Cathy
Barbara
mars% uniq names   (behaves as both -d and -u are on)
Barbara
Al
Cathy
Barbara
mars% sort names | uniq -u
Cathy
mars%
```

The real power of uniq is when you use it with sort in a pipeline. When data is sorted, it guarantees that all duplicate lines will be consecutive.

More examples:

I have two files that contain the names of students enrolled in two different courses: acs2941 and acs2947.

```
sort acs2941 acs2947 | uniq -d
```

will show the students who are taking both courses.

```
sort acs2941 acs2947 | uniq -u
```

will show the students who are taking one course only.

```
sort acs2941 acs2947 | uniq
(sort -u acs2941 acs2947)
```

will show all students' names without duplications.

```
sort acs2941 acs2947 | uniq -c
```

will show how many courses each student is taking.

Command Substitution

Command substitution allows you to use the output of one command as part of another command.

To use command substitution, you place part of a command within ` (backquote) characters. The shell will evaluate the part within backquotes as a command on its own. Then the shell will substitute the output of this command into the large command.

Examples:

```
mars% echo The time is date.
The time is date.
mars% echo The time is `date`.
The time is Tue Sep 19 15:15:44 CDT 2023.
mars% echo The current directory is pwd.
The current directory is pwd.
mars% echo The current directory is `pwd`.
The current directory is /home/sliao/2941.
mars%
```

Translate or Delete Selected Characters: tr

The tr command will read data and replace specified characters with other characters. The syntax of the tr command is:

tr [-csd] [set1 [set2]]

-d deletes all the characters that you specify (you only define one set of character in this case).

set1 and set2 are sets of characters.

tr reads data from the standard input and looks for any characters from set1. Whenever tr finds such a character, it replaces the character with the corresponding character from set2.

Examples:

```
mars% cat f1
aabbcc
mars% tr a A <f1 > f2
mars% cat f2
AAbbcc
mars%

mars% tr abc ABC < f1 > f2
mars% cat f2
AABBCC
mars%

mars% cat file1
abcdeabcde
mars% tr abcde Azzzz < file1 > file2
mars% cat file2
AzzzzAzzzz
mars%

mars% tr abcde Az < file1 > file2
mars% cat file2
AzzzzAzzzz
mars%

mars% cat file
a       b       c
mars% tr '\011' ' ' < file > newfile
mars% cat newfile
a b c
mars%

mars% cat file3
abcdeabcde
1234512345
mars% tr -d 12345 < file3 > file4
mars% cat file4
abcdeabcde

mars%
```

Extracting Lines that Contain a Specified Pattern: grep

The grep command will search for all lines in a collection of data that contain a specified pattern and write these lines to the standard output.

grep: Global Regular Expression Print

The grep command is actually part of a family, which provides a wide range of text searching capabilities. The other members are fgrep and egrep.

grep was designed to be the general purpose program. It can search for patterns that are exact characters, or for patterns that match a more general specification.

fgrep was designed to be a faster searching program. It can only search for exact characters, not for general specification. fgrep stands for "fixed character grep".

egrep was designed to be the most powerful program. It can search for more complex patterns than grep. The name egrep stands for "extended grep".

The syntax of the grep command is:

grep [-cilnvw] *pattern* [*file*…]

where *pattern* is the pattern to search for, and *file* is the name of input file.

-c displays the number of lines that have been extracted.

Example:

```
mars% w
 18:50:43 up 77 days,  8:45,  2 users,  load average: 0.02, 0.02, 0.05
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
sliao    pts/1    wnpgmb0412w-ds01 18:49    3.00s  0.05s  0.02s w
sliao    pts/5    wnpgmb0412w-ds01 18:50   40.00s  0.02s  0.02s -csh
mars% w | grep -c sliao
2
mars%
```

-i ignores the difference between upper- and lowercase letters when making a comparison.

Example:

```
mars% cat myfile
line 1
Line 2
line 3
Line 4
mars% grep line myfile
line 1
line 3
mars% grep -i line myfile
line 1
Line 2
line 3
Line 4
mars%
```

-l lists only the name of each file containing matched lines. Each file name is listed only once.

Examples:

```
mars% cat F1
line 1  F1
Line 2  F1
line 3  F1
Line 4  F1
mars% cat F2
line 1  F2
Line 2  F2
line 3  F2
Line 4  F2
mars% cat F3
line 1  F3
Line 2  F3
line 3  F3
Line 4  F3
mars% grep -l Line F1 F2 F3
F1
F2
F3
mars% grep Line F1 F2 F3
F1:Line 2  F1
F1:Line 4  F1
F2:Line 2  F2
F2:Line 4  F2
F3:Line 2  F3
F3:Line 4  F3
mars%
```

-v will select all the lines that do not contain the specified pattern.

Example:

```
mars% grep -v line F1
Line 2  F1
Line 4  F1
mars%
```

-n writes a relative line number in front of each line of output.

Example:

```
mars% grep -n Line F1 F2 F3
F1:2:Line 2  F1
F1:4:Line 4  F1
F2:2:Line 2  F2
F2:4:Line 4  F2
F3:2:Line 2  F3
F3:4:Line 4  F3
mars%
```

-w specifies that you want to search only for complete word.

Example:

```
mars% cat File
line 1
line_2
line 3
line_4
mars% grep line File
line 1
line_2
line 3
line_4
mars% grep -w line File
line 1
line 3
mars%
```

Regular Expressions

A Regular Expression is a compact way of specifying a general pattern of characters.

Regular expressions are an integral part of Unix and you can use regular expressions with many commands, including the vi editor.

Within a regular expression, certain symbols have special meanings.

Summary of Symbol Used in Regular Expressions

Symbol	Meaning
`.`	match any single character except **newline**
`*`	match zero or more of the preceding characters
`^`	match the beginning of a line
`$`	match the end of a line
`\<`	match the beginning of a word
`\>`	match the end of a word
`[ ]`	match one of the enclosed characters
`[^ ]`	match any character that is not enclosed
`\`	take the following symbol literally

`.` (period) will match any single character except newline.

Example:

```
mars% cat name
Ambercrombie, Al
Barton, Barbara
Canby, Charles
Danfield, Deann
mars% grep 'D....' name
Danfield, Deann
mars%
```

You need to place the pattern within single quotes whenever you use special characters. Otherwise, the shell may interpret some of them incorrectly.

Using the single quotes tells the shell to leave these characters alone and pass them on to the program.

***** will match multiple characters. It stands for zero or more occurrences of the preceding characters.

Examples:

```
grep ':.*:'
```

will search for all lines that contain a colon, followed by zero or more any other characters, followed by another colon.

```
mars% grep 'Winnip*' /usr/share/dict/words
Winni
Winnick
Winnie
Winnifred
Winnipeg
Winnipegger
Winnipegosis
Winnipesaukee
Winnisquam
mars% grep 'Winnipe*' /usr/share/dict/words
Winnipeg
Winnipegger
Winnipegosis
Winnipesaukee
mars% grep 'Winnipe' /usr/share/dict/words
Winnipeg
Winnipegger
Winnipegosis
Winnipesaukee
mars%
```

^ indicates that you want to match only patterns at the beginning of a line.

Examples:

```
mars% grep '^A' name
Ambercrombie, Al
mars% grep '^A*' name
Ambercrombie, Al
Barton, Barbara
Canby, Charles
Danfield, Deann
mars%
```

$ indicates that you want to match patterns at the end of a line.

Example:

```
mars% cat info
012-34-5678$Ambercrombie, Al$01/01/92$555-1111
123-45-6789$Barton, Barbara$02/02/93$555-2222
234-56-7890$Canby, Charles$03/03/94$555-3333
345-67-8901$Danfield, Deann $04/04/95$555-4444
mars% grep '4' info
012-34-5678$Ambercrombie, Al$01/01/92$555-1111
123-45-6789$Barton, Barbara$02/02/93$555-2222
234-56-7890$Canby, Charles$03/03/94$555-3333
345-67-8901$Danfield, Deann $04/04/95$555-4444
mars% grep '^4' info
mars% grep '4$' info
345-67-8901$Danfield, Deann $04/04/95$555-4444
mars%
```

\< indicates the beginning of the word.
\> indicates the end of the word.

Examples:

```
mars% grep '12' info
012-34-5678$Ambercrombie, Al$01/01/92$555-1111
123-45-6789$Barton, Barbara$02/02/93$555-2222
mars% grep '\<12' info
123-45-6789$Barton, Barbara$02/02/93$555-2222
mars%

mars% grep '01' info
012-34-5678$Ambercrombie, Al$01/01/92$555-1111
345-67-8901$Danfield, Deann $04/04/95$555-4444
mars% grep '\<01' info
012-34-5678$Ambercrombie, Al$01/01/92$555-1111
mars%

mars% grep 'n' info
123-45-6789$Barton, Barbara$02/02/93$555-2222
234-56-7890$Canby, Charles$03/03/94$555-3333
345-67-8901$Danfield, Deann $04/04/95$555-4444
mars% grep 'n\>' info
123-45-6789$Barton, Barbara$02/02/93$555-2222
345-67-8901$Danfield, Deann $04/04/95$555-4444
mars%
```

```
mars% grep 'ie' info
012-34-5678$Ambercrombie, Al$01/01/92$555-1111
345-67-8901$Danfield, Deann $04/04/95$555-4444
mars% grep 'ie\>' info
012-34-5678$Ambercrombie, Al$01/01/92$555-1111
mars%
```

[ ] matches *one* of the enclosed characters.

Examples:

```
mars% grep 'b[ae]r' info
012-34-5678$Ambercrombie, Al$01/01/92$555-1111
123-45-6789$Barton, Barbara$02/02/93$555-2222
mars%

mars% grep '[A-Z]a[a-z]' info
123-45-6789$Barton, Barbara$02/02/93$555-2222
234-56-7890$Canby, Charles$03/03/94$555-3333
345-67-8901$Danfield, Deann $04/04/95$555-4444
mars%
```

[^] matches any character that is *not* enclosed.

Examples:

```
mars% grep 'b[^a]r' info
012-34-5678$Ambercrombie, Al$01/01/92$555-1111
mars%

mars% grep '[^a-z]a[^A-Z]' info
123-45-6789$Barton, Barbara$02/02/93$555-2222
234-56-7890$Canby, Charles$03/03/94$555-3333
345-67-8901$Danfield, Deann $04/04/95$555-4444
mars%
```

**** takes the following symbol literally.

More examples:

```
mars% cat test
A
Al
Art
Ambercrombie
mars% grep 'A[a-z]*' test
A
Al
Art
Ambercrombie
mars% grep 'A[a-z][a-z]*' test
Al
Art
Ambercrombie
mars% grep 'A[a-z][a-z][a-z]*' test
Art
Ambercrombie
mars% grep 'A[a-z][a-z][a-z][a-z]*' test
Ambercrombie
mars%
```

```
grep '\\\*.*\$' file
```

will find all lines that contain the characters `\*`, followed by any characters, followed by `$`.

`\\` a single backslash
`\*` a single star
`.*` any number of other characters
`\$` a single dollar sign

What the following pipeline will do?

```
cat /usr/share/dict/words | grep '^[a-z][a-z]$'
```

The Unix File System

File System: To maintain all the data that is stored in the computer.

File: Any source from which data can be read, or any target to which data can be written.

The Three Types of Unix Files:

Ordinary Files:

Ordinary files contain data and are stored on disk or on tape. Sometimes, they are called regular files.

Text files are ordinary files that contain only ASCII characters, which can be generated by typing at a keyboard.

You can edit a text file with a text editor such as vi.

Binary files are ordinary files that contain non-textual data, which only makes sense when processed by a program.

Example:

C program	== >	Machine instructions
(source program)	(compiler)	(executable program)

Directories and Subdirectories

Unix uses directories to organize files into a hierarchical system.

The Tree-Structured File System

The outline of the Unix file system is like an upside-down tree. The name "root" was chosen to indicate the main trunk of the tree.

/ is used to indicate the root directory.

A Tour of the Root Directory

/bin contains basic programs that are part of Unix.

/dev contains the special files that represent the physical devices.

/etc contains the programs and files that are used for managing the system.

/lib contains libraries of programs used by programmers.

/tmp can be used for temporary storage.

A Tour of the /usr Directory

/usr holds a number of subdirectories of its own.

/usr/bin is used to hold executable programs.

/usr/include contains the files that used by (C) programmers.

/usr/lib contains libraries of programs and data used by programmers.

/usr/local is for local programs and documentation.

/usr/share/man contains the directories and files used by the online Unix manual.

/usr/share/dict contains files used by the Unix dictionary (words).

Home Directories

In a Unix system, each user (userid) is given a home directory. This is a directory associated with a particular userid. Within this home directory, the user can do whatever s/he wants.

~ can be used as an abbreviation for your home directory.

Examples:

```
mars% cd ~
mars% pwd
/home/sliao
mars% cd 2941
mars% pwd
/home/sliao/2941
mars% ls -ld
drwx--x--x. 7 sliao sliao 4096 Jan 18 12:36 .
mars% echo ~
/home/sliao
mars% echo $HOME
/home/sliao
mars%
```

Device File

The last type of file, a device file, is an internal representation of a physical device. For example, your keyboard, your display screen, a printer, the disk drive.

Working with Directories

Pathnames and Working Directory

A pathname is part of the name of a file. For example,

```
vi /home/sliao/2941/assignment1/ver1
```

When we write the name of a file in this manner, we call it a pathname. It shows the path through the directory tree from the root directory to the file.

Unix allows you to use one directory at a time as your working directory. When you want to use a file that is in your working directory, you don't need to specify the whole path.

Example:

```
mars% cd ~/2941/assignment1
mars% vi ver1
```

Absolute and Relative Pathnames

Absolute Pathname: A file name that begins with a **/**. It shows the full path to the file, starting from the root directory.

Examples:

/home/sliao/2941/test.txt
/home/sliao/2947/final.txt

Relative Pathname: When a file name that does not start with a **/**, we call it a relative pathname.

Pathname Abbreviations:

~	is home directory
..	refers to the parent directory.
.	refers to the working directory.

Moving Around the Directory Tree: cd, pwd

To change your working directory, use the cd (change directory) command. The syntax is:

cd [*directory*]

To display the name of your working directory, use the pwd (print working directory) command. The syntax is:

pwd

Examples:

```
mars% cd /
mars% pwd
/
mars% cd
mars% pwd
/home/sliao
mars% cd ~/2941
mars% pwd
/home/sliao/2941
mars% cd ../../..
mars% pwd
/
mars% cd home/sliao/2941
mars% pwd
/home/sliao/2941
mars%
```

Making a New Directory: mkdir

To make a directory, use the mkdir command. The syntax is:

mkdir *directory*...

where *directory* is the name of a directory, which can be either an absolute or relative pathname, you want to make.

Two Rules:

1. Within a single parent directory, you cannot make two directories with the name pathname.
2. You cannot make a subdirectory if its parent directory does not exist.

Example:

```
mkdir ~/2941 ~/2941/assignment1
```

You cannot

```
mkdir ~/2941/assignment1 ~/2941
```

"Tree": You cannot make a new branch that has nowhere to attach to the tree.

Removing a Directory: rmdir

You can use the command rmdir to remove a directory. The syntax is:

rmdir *directory*...

Two Rules:

1. You cannot remove a directory by using rmdir unless the directory is empty.
2. You cannot remove any directory that lies between your working directory and the root directory.

"Tree": You are sitting on a branch that is your working directory. You cannot cut off a branch that is holding up the one you are sitting on.

Example:

If your working directory is

```
/home/sliao/2941/assignment1
```

You cannot remove the `2941` directory.

Moving or Renaming a Directory: mv

We can use the mv command

(1) to rename a directory,
(2) to move a directory, and
(3) to move an entire sub-tree.

The syntax is:

mv *directory target*

Case 1: If there is a directory named assignment3 in my working directory. I want to change the name of this directory to Assignment3. Assuming that Assignment3 does not exist,

mv assignment3 Assignment3

will "rename" assignment3 to Assignment3.

Case 2: If the target directory (Assignment3) already exists,

mv assignment3 Assignment3

will "move" the directory assignment3 *into* Assignment3.

Example:

Assuming that we have two directories,

/home/sliao/2941

and

/home/sliao/assignment2

```
mv /home/sliao/assignment2 /home/sliao/2941
```

will move the assignment2 directory to lie within the 2941 directory. Afterward, the pathname of the assignment2 directory becomes

/home/sliao/2941/assignment2

When mv moves a directory, it also moves all the files and subdirectories that lie within that directory.

Listing the Contents of a Directory: ls

To display information about the contents of a directory, use the ls (list) command. The basic function of ls is to display an alphabetical list of names of files in a directory.

The syntax for the ls command is:

ls [*-options*] [*file...*]

where *file* is the name of a directory or an ordinary file.

By default, ls will display the names of all the files in your working directory.

-a (all) Lists all file names including the hidden files (dotfiles).

-A Lists all file names, except . (dot) and .. (dot-dot).

-t Sorts by time of last modification (latest first) instead of by name.

-d (directory) Displays information about the directory itself.

-l (long) Displays the mode, number of links, owner, group, size (in bytes), and time of last modification for each file, and pathname.

Example:

```
mars% pwd
/home/sliao/2941
mars% ls -ld
drwx--x--x. 4 sliao sliao 46 Dec 16 22:41 .
mars%
```

-r Displays the names in reverse order.

-R Will tell ls to list information about all the subdirectories and files that lie within the directory you name. It will display information about an entire sub-tree.

-F Will flag certain file names with an identification character.

- /: directory
- *: executable program

-1 Will force ls to write one line per file name.

-C Will force ls to write columns to a file or pipeline.

Examples:

```
mars% ls file*
file  file1  file2  file3  file4  file5  file6
mars% ls file* > filea
mars% ls -1 file* > fileb
mars% ls -C file* > filec
mars% cat filea
file
file1
file2
file3
file4
file5
file6
filea
mars% cat fileb
file
file1
file2
file3
file4
file5
file6
filea
fileb
mars% cat filec
file  file1  file2  file3  file4  file5  file6  filea  fileb  filec
mars%
```

Working with Files

How do you create a file?

1. Many programs will create a file on your behalf.

   Example:

   vi myfile

   will create a file named myfile if it does not exist.

2. Redirect output to a file.

   Example:

   ls ~ > myhome

   will create a file named myhome if it does not exist.

3. Make a copy of a file. Unix will create the new file automatically.

We have mkdir command to make a new directory. However, we do *not* have a similar command to make an ordinary file.

touch

The touch command can change the modification time of a file to the current time. It works by reading a character from the file and writing it back.

The syntax is

touch *filename*...

Side effect: If the file that you specify does not exist, touch will create it (an empty file).

Choosing a File Name

For Unix, there are only two basic rules for naming files:

1. File names can be up to 255 characters long.
2. A file name can contain any character except **/**, which has a special meaning within a pathname.

Bad File Names:

Any name that contains a character that has a special meaning (<, >, |, **!**, **;**, …).

Examples:

```
ls -l Assignment3;Newversion
ls -l Assignment3>Newversion
ls -info
```

Characters that Are Safe to Use in File Names

a,b,c...	(lowercase letters)
A,B,C...	(uppercase letters)
0,1,2...	(numbers)
.	(period)
=	(equals sign)
_	(underscore)

File names that begin with uppercase letters are reserved for files that are important in some special way. (Uppercase comes before lowercase in the ASCII code.)

Example: README

Some programs expect to use files whose names end in a period followed by one or more specific letters (extension).

Examples: a1q6.c and programs.Z

Copying a File: cp

To make a copy of a file, use the cp command:

cp [-ip] *file1 file2*

where *file1* is the name of an existing file.

If file2 does not exist, cp will create it.

If file2 does exist, cp will replace it.

-i Tells cp to ask your permission before replacing a file that already exists.

Example:

```
mars% ls -l
total 12
-rw-rw-r-- 1 sliao sliao 47 Jan 20 19:59 filea
-rw-rw-r-- 1 sliao sliao 53 Jan 20 19:59 fileb
-rw-rw-r-- 1 sliao sliao 67 Jan 20 19:59 filec
mars% cp -i filea fileb
cp: overwrite fileb? y
mars% ls -l
total 12
-rw-rw-r-- 1 sliao sliao 47 Jan 20 19:59 filea
-rw-rw-r-- 1 sliao sliao 47 Jan 20 20:00 fileb
-rw-rw-r-- 1 sliao sliao 67 Jan 20 19:59 filec
mars%
```

-p Makes the *file2* have the same modification time and permissions as the source file (*file1*).

Example:

```
mars% ls -l
total 8
-rw-rw-r-- 1 sliao sliao 47 Jan 20 20:00 fileb
-rw-rw-r-- 1 sliao sliao 67 Jan 20 19:59 filec
mars% cp -p fileb filec
mars% ls -l
total 8
-rw-rw-r-- 1 sliao sliao 47 Jan 20 20:00 fileb
-rw-rw-r-- 1 sliao sliao 47 Jan 20 20:00 filec
mars%
```

Copying Files to a Different Directory

The cp command can also copy one or more files to a different directory. The syntax is:

cp [-ip] *file*... *directory*

where *file* is the name of an existing file, and *directory* is the name of an existing directory.

Example:

```
mars% ls -l questions assignment1
-rw-rw-r-- 1 sliao sliao   47 Jan 20 20:31 questions

assignment1:
total 0
mars% cp questions assignment1
mars% ls -l questions assignment1
-rw-rw-r-- 1 sliao sliao   47 Jan 20 20:31 questions

assignment1:
total 4
-rw-rw-r-- 1 sliao sliao 47 Jan 20 20:32 questions
mars%
```

Copying a Directory to Another Directory

You can use cp to copy a directory and all of its files to another directory by using the –r option:

cp –r [-ip] *directory1*… *directory2*

where *directory1* is a source directory and *directory2* is the target directory.

If *directory2* does not exist, Unix will create one in your working directory and copy all files and subdirectories to it.

Examples:

```
mars% ls -l | grep assignment
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:32 assignment1
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:35 assignment2
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:36 assignment3
mars% ls -l assignment3
total 8
-rw-rw-r-- 1 sliao sliao 53 Jan 20 20:36 names
-rw-rw-r-- 1 sliao sliao 67 Jan 20 20:35 questions
mars% cp -r assignment3 assignment4
mars% ls -l | grep assignment
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:32 assignment1
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:35 assignment2
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:36 assignment3
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:36 assignment4
mars% ls -l assignment4
total 8
-rw-rw-r-- 1 sliao sliao 53 Jan 20 20:36 names
-rw-rw-r-- 1 sliao sliao 67 Jan 20 20:36 questions
mars%
```

If *directory2* does exist, Unix will create a subdirectory named *directory1* under *directory2* and copy all files and subdirectories to it.

Examples:

```
mars% ls -l | grep assignment
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:32 assignment1
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:35 assignment2
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:36 assignment3
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:43 assignment4
mars% ls -l assignment3
total 8
-rw-rw-r-- 1 sliao sliao 53 Jan 20 20:36 names
-rw-rw-r-- 1 sliao sliao 67 Jan 20 20:35 questions
mars% ls -l assignment4
total 0
mars% cp -r assignment3 assignment4
mars% ls -l assignment4
total 4
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:43 assignment3
mars% ls -l assignment4/assignment3
total 8
-rw-rw-r-- 1 sliao sliao 53 Jan 20 20:43 names
-rw-rw-r-- 1 sliao sliao 67 Jan 20 20:43 questions
mars%
```

Moving a File: mv

To move a file to a different directory, use the mv (move) command. The syntax is:

mv [-if] *file… directory*

where *file* is the name of an existing file, and *directory* is the name of target directory.

Examples:

```
mars% ls -l
total 16
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:46 backups
-rw-rw-r-- 1 sliao sliao   47 Jan 20 20:46 file1
-rw-rw-r-- 1 sliao sliao   53 Jan 20 20:46 file2
-rw-rw-r-- 1 sliao sliao   22 Jan 20 20:46 file3
mars% ls -l backups
total 4
-rw-rw-r-- 1 sliao sliao 22 Jan 20 20:46 file1
mars% mv file[123] backups
mars% ls -l
total 4
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:47 backups
mars% ls -l backups
total 12
-rw-rw-r-- 1 sliao sliao 47 Jan 20 20:46 file1
-rw-rw-r-- 1 sliao sliao 53 Jan 20 20:46 file2
-rw-rw-r-- 1 sliao sliao 22 Jan 20 20:46 file3
mars%
```

-i (interactive) option will tell mv to ask your permission before replacing a file.

-f (force) will force mv to replace a file. It will override the –i option as well as restrictions imposed by file permissions.

Use -f with care!

Renaming a File or Directory: mv

You can also use mv command to rename a file or directory. The syntax is:

mv [-if] *oldname newname*

where *oldname* is the name of an existing file or directory, and *newname* is the new name.

Examples:

```
mars% ls -l
total 4
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:47 backups
mars% mv backups backupfiles     <-- rename backups to backupfiles
mars% ls -l
total 4
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:47 backupfiles
mars%

mars% ls -l
total 8
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:53 backupfiles
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:53 backups
mars% ls -l backupfiles
total 8
-rw-rw-r-- 1 sliao sliao 47 Jan 20 20:46 file1
-rw-rw-r-- 1 sliao sliao 53 Jan 20 20:46 file2
mars% ls -l backups
total 0
mars% mv backupfiles backups<-- move backfiles under backups
mars% ls -l
total 4
drwxrwxr-x 3 sliao sliao 4096 Jan 20 20:54 backups
mars% ls -l backups
total 4
drwxrwxr-x 2 sliao sliao 4096 Jan 20 20:53 backupfiles
mars% ls -l backups/backupfiles
total 8
-rw-rw-r-- 1 sliao sliao 47 Jan 20 20:46 file1
-rw-rw-r-- 1 sliao sliao 53 Jan 20 20:46 file2
mars%
```

Removing a File: rm

To remove (delete) a file, use the rm command. The syntax is:

rm [-fir] *file*…

where *file* is the name of a file you want to remove.

Examples:

```
rm file1
rm ~/file2
rm /usr/tmp/file3
rm ~/tmp/file[123]
rm *
```

Once you remove a file, it is gone for good.

-i tells rm to ask your permission before removing each file.

Example:

```
alias erase 'rm -i'
```

-f tells rm to remove all the files you specify regardless of file permissions. –f will override –i.

-r will remove an entire sub-tree.

Example:

```
rm -r ~/2941
```

will remove the directory 2941 and all files and subdirectories under 2941.

File Permissions

Unix maintains a set of file permissions for each file. These permissions control who can access the file and in what way.

Three types of permissions:

- read permission
- write permission
- execute permission

Ordinary File

Read:	you can read from the file
Write:	you can write to the file
Execute:	you can execute the file

Directory

Read:	you can read the directory
Write:	you can create, move, copy, or remove entries
Execute:	you can search the directory

How Unix maintains file permissions?

You:	read, write, execute
Your group:	read, write, execute
Everybody:	read, write, execute

To display your userid and your groupid, use **`id`** command.

Example:

```
mars% id
uid=1000(sliao) gid=1000(sliao)
groups=1000(sliao)
mars%
```

Displaying File Permissions: `ls -l`

To display the file permissions for a file, use the `ls` command with the `-l` (long listing) option.

Example:

```
mars% ls -l
total 36
-rwxrwxrwx 1 sliao sliao  6 Jan 27 21:25 prog.everybody
-rwxrwx--- 1 sliao sliao 11 Jan 27 21:25 prog.group
-rwx------ 1 sliao sliao 11 Jan 27 21:25 prog.user
-rw-rw-rw- 1 sliao sliao 22 Jan 27 21:25 text.everybody
-rw-rw---- 1 sliao sliao 12 Jan 27 21:25 text.group
-rw------- 1 sliao sliao 10 Jan 27 21:25 text.user
-r--r--r-- 1 sliao sliao 14 Jan 27 21:25 tran.everybody
-r--r----- 1 sliao sliao 47 Jan 27 21:25 tran.group
-r-------- 1 sliao sliao 53 Jan 27 21:25 tran.user
mars%
```

r: read permission w: write permission

x: execute permission -: permission not granted

File Modes

read permission	=	4
write permission	=	2
execute permission	=	1
no permission	=	0

For each set of permissions, we add the appropriate numbers.

read	write	execute	VALUE	read		write		execute
-	-	-	0	0	+	0	+	0
-	-	yes	1	0	+	0	+	1
-	yes	-	2	0	+	2	+	0
-	yes	yes	3	0	+	2	+	1
yes	-	-	4	4	+	0	+	0
yes	-	yes	5	4	+	0	+	1
yes	yes	-	6	4	+	2	+	0
yes	yes	yes	7	4	+	2	+	1

Example:

```
owner    group    everybody                        mode

rwx 7    rwx 7      rwx 7   <-- prog.everbody      777
rwx 7    rwx 7      --- 0    <-- prog.group        770
rwx 7    --- 0      --- 0    <-- prog.user         700
rw- 6    rw- 6      rw- 6    <-- text.everybody    666
rw- 6    rw- 6      --- 0    <-- text.group        660
rw- 6    --- 0      --- 0    <-- text.user         600
r-- 4    r-- 4      r-- 4    <-- tran.everybody    444
r-- 4    r-- 4      --- 0    <-- tran.group        440
r-- 4    --- 0      --- 0    <-- tran.user         400
```

Changing File Permissions: chmod

To change the permissions for a file, use the chmod (change mode) command. The syntax is:

chmod *mode file…*

where *mode* is the new file mode, and *file* is the name of a file or directory.

Examples:

```
chmod 644 file1 file2
```

will change the mode for the specified files to give read and write permissions to the owner, and read permission to the group and everybody else.

```
chmod 755 program
```

will give the owner read, write, and execute permissions and read, execute permissions for the group and everybody else.

How Unix Assigns Permissions to a New File: umask

When Unix create a new file, it starts with a file mode of:

666: for non-executable ordinary files
777: for executable ordinary files
777: for directories

From this initial mode, Unix *subtracts* the value of the user mask. The user mask is a mode, set by you, showing which permissions you want to restrict.

To set the user mask, use the umask command. The syntax is:

```
umask [mode]
```

where *mode* specifies which permissions you want to restrict.

Examples:

```
umask 022 (umask 22)
umask 077 (umask 77)
```

It is a good idea to put a umask command in your initialization file so that your user mask will be set automatically each time you log in. For the C-Shell, this file is .cshrc.