UNIX & SHELL PROGRAMMING

Course Overview

- Introduction
- Unix file system & Commands
- VI Editor
- Simple Filters
- Process
- Redirection
- Some more useful commands
- SED & AWK
- Shell Programming
- Other Application Interface

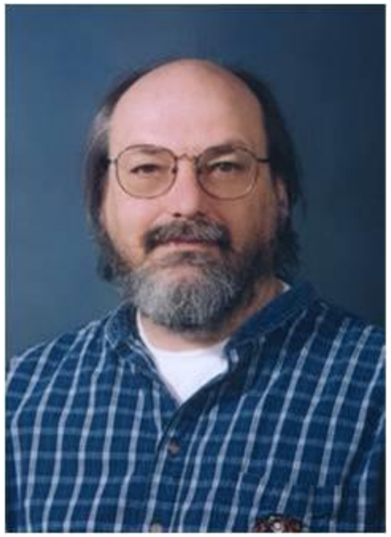
Introduction

What is an Operating System?

- ❖ A program that acts as an intermediary between a user of a computer and the computer hardware.
- ❖ Operating system goals:
  - ❖ Execute user programs and make solving user
  - ❖ problems easier.
  - ❖ Make the computer system convenient to use.
- ❖ Use the computer hardware in an efficient manner.

Creators of Unix

- ❖ Ken Thompson (left) and Dennis Ritchie (right)
- ❖ Turing award in 1983



Linux History

- Linux: free Unix-like kernel
  - can run almost all programs written for Unix
  - developed by Linus Torvalds, University of Helsinki, in 1992
  - small part of usable Linux system

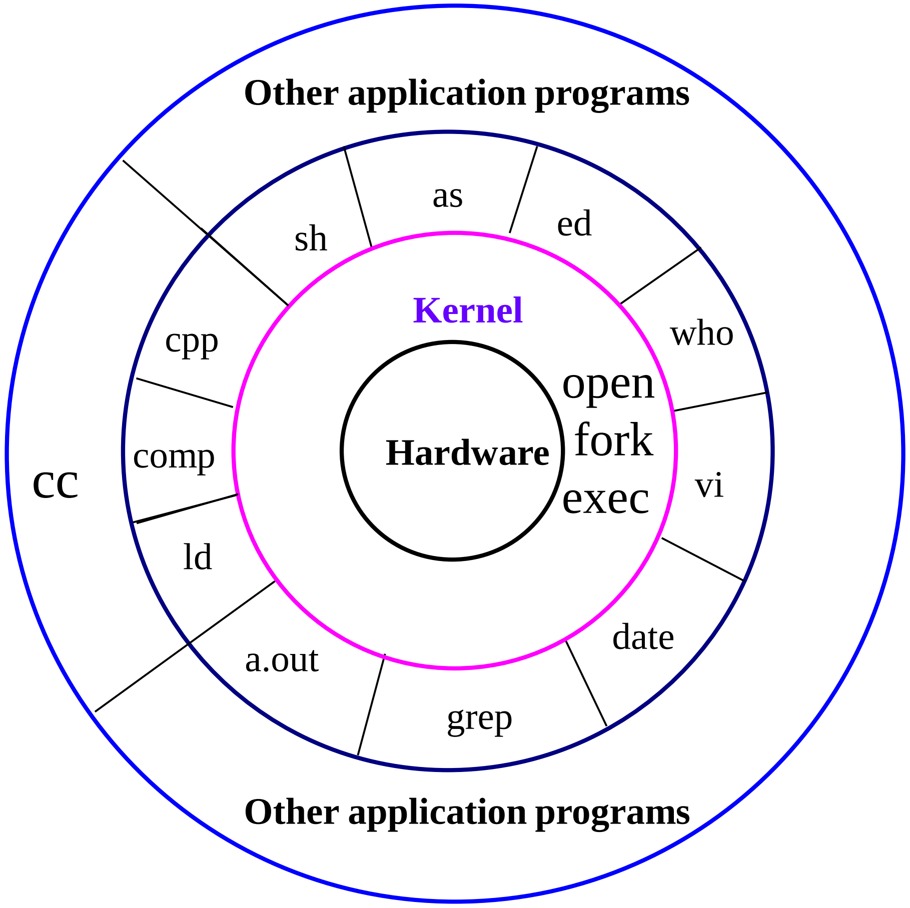
Linux System

- ❖ Linux kernel + many free tools
  - ❖ ready-to-install distributions (flavours)
  - ❖ RedHat, Fedora, Debian

- ❖ GNU project
  - ❖ free software, mass collaboration project
  - ❖ GNU (GNU's Not Unix), use Linux as kernel
  - ❖ GCC, G++, bash, GNU Emacs
  - ❖ Copyleft
  - ❖ Richard Stallman

UNIX Operating System

UNIX Architecture - Block diagram



The UNIX operating system

- Kernel
  - Core of the OS – a collection of routines/system calls written in C
  - Loaded into memory when the system is booted
  - Communicates with the hardware directly
- Shell
  - Interface between the user and the kernel
  - Acts as the command interpreter
  - Forms a simpler version of the command line and communicates with the kernel to see that command is executed

Logging in & Logging Out

- Logging in
  - To login in to the UNIX system, one should have an user account on the system.
  - The user account must be created by UNIX administrator also called as Super user or root
  - Logging in to the UNIX system requires two pieces of information
    - A username
    - A password
- When your account is created, a password is assigned. To change your password, type the command “passwd”
- Logging out
  - Once you are done using the system, you can logout by typing “exit” at the shell prompt.
  - Some shells will recognize other commands to log you out, like "logout" or even "bye".

Logging in to a remote system

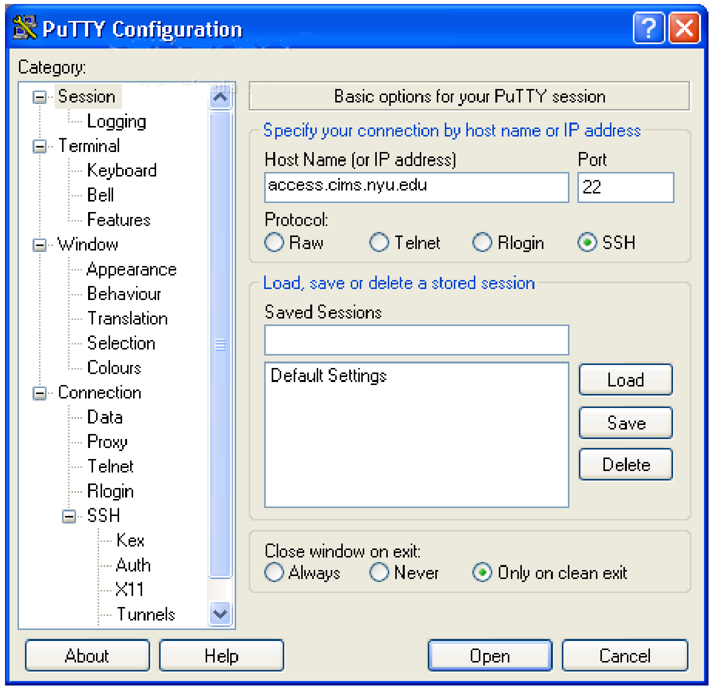
- If you have several UNIX systems at your site connected together by a network and If you have accounts on these remote systems you can login to them from the system you are currently using.
- Use rlogin or telnet to log in to remote system
- Logging-in using rlogin
  - To login to your account on a remote system use the command:

    rlogin remote_system_name
  - To login to a remote system using a different username, use the command:

    rlogin remote_system_name -l username

Remote Login

- Use Secure Shell (SSH)
- Windows
  - e.g. PuTTY



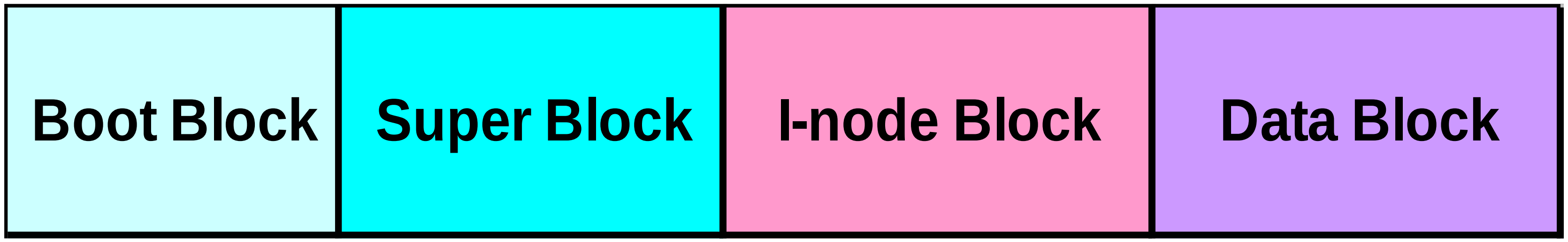
Host Name (or IP address): access.cims.nyu.edu
Port: 22
Protocol: Raw Telnet Rlogin [✓ SSH]
Close window on exit: Always Never [✓ Only on clean exit]

- UNIX-like OS
  - ssh name@access.cims.nyu.edu

Unix File System & Commands

UNIX File System

- A file system is a logical method for organizing and storing large amounts of information in a way which makes it easy to manage.
- There can be more than one file systems on any UNIX operating systems
- There can be more than one type of file systems
- Each disk file system on the UNIX operating system will have the following structure



UNIX File System structure

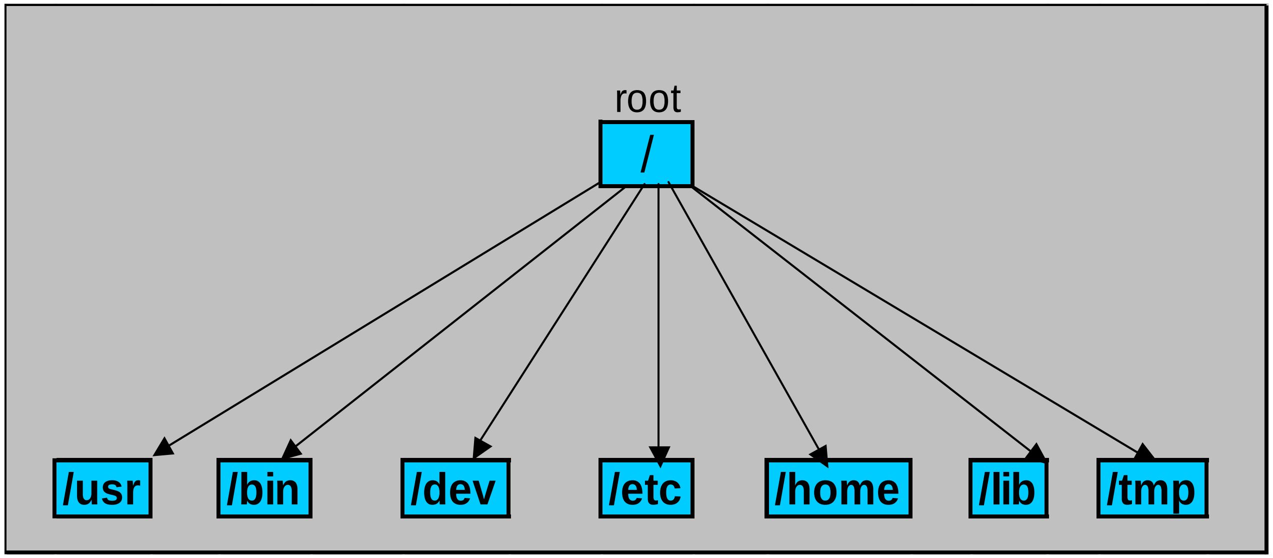
Boot Block	• Stores boot procedures
Super Block	• Contains file system size and status
	• Inode information (total number allocated, total free, free inode number)
	• One copy in memory, one copy on disk
Inode Table	• Type and permissions
	• Hard link count
	• UID of owner
	• GID of owner
	• size in bytes
	• date and time stamps (creation, last accessed, last modified)
	• Disk block addresses
Storage Blocks	• Data blocks
	• Regular files - file contents
	• Directory files - filenames and inode numbers

UNIX File System

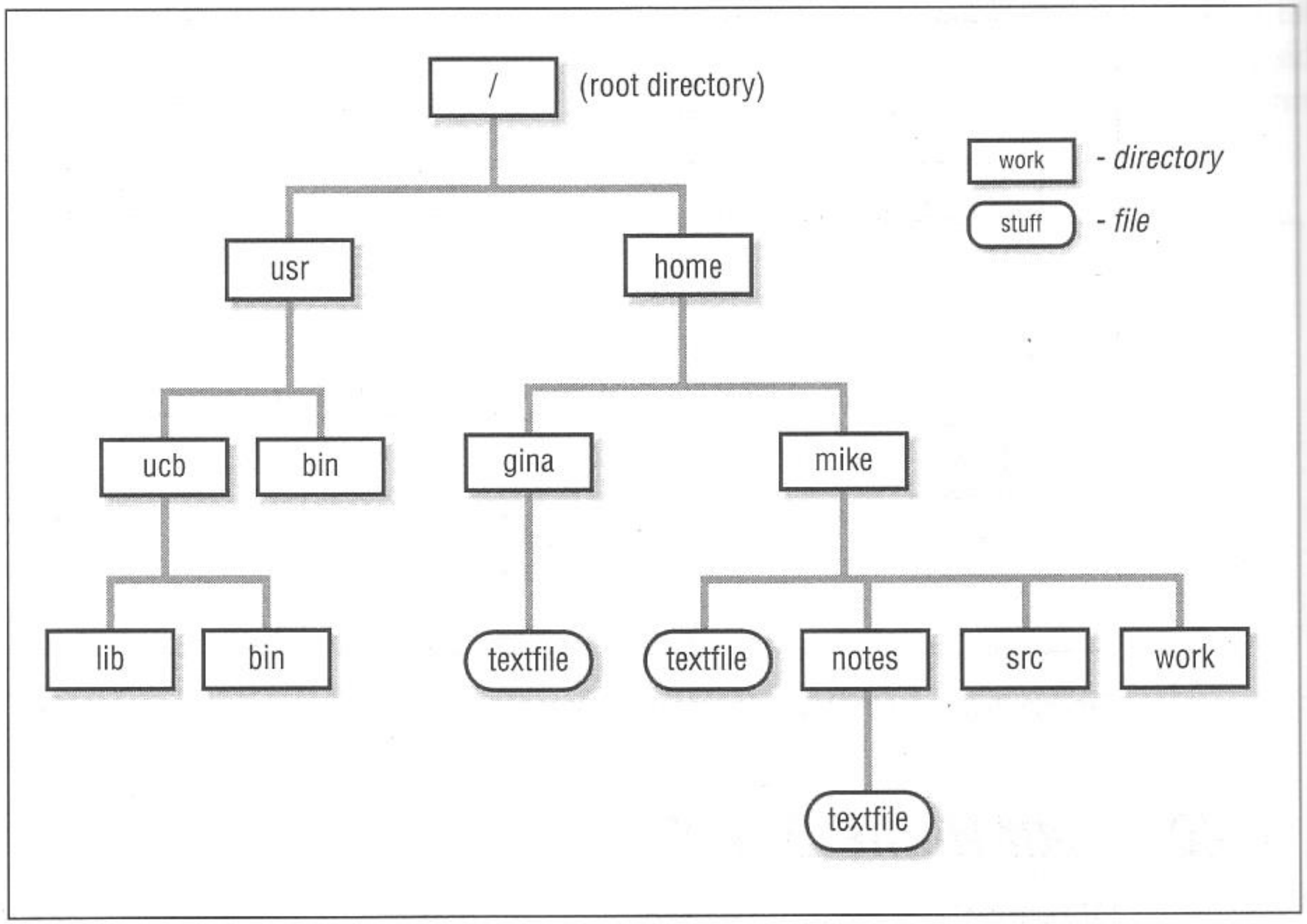
- Hierarchical file system
- UNIX file system comprises of a tree like structure consists of files and directories.
- It is logically organized into meaningful structure of directories, with each directory containing files and sub-directories

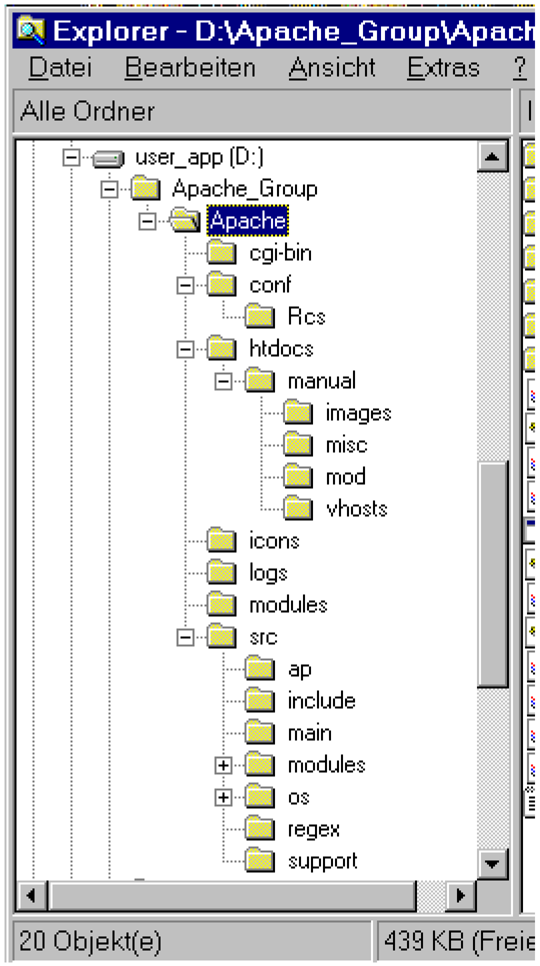
UNIX File System

- The standard system directories of the UNIX file system are shown below.
- Each directory below contains specific types of file.



The UNIX File Hierarchy





The UNIX File System - Files

- File is the smallest unit in which information is stored.
- The file name is a reference to the information stored on the file system.
- UNIX-legitimate filenames are any combination of these three classes of characters
  - Upper and lower case letters (A - Z and a - z)
  - Numbers (0 - 9)
  - Periods, underscores, hyphens (. _ -)

- UNIX file names are case sensitive
- Examples:
  - Good Filenames
    - “records.010802” A legitimate UNIX filename
    - “records.010802” Different from above because of the capital F

  - Bad Filenames
    - “records 01-08-02 “ Two files, one called "records" another called " 01-08-02 “ because of the line space between the words.
    - “records-01/08/02“ A mess because the Unix shell will attempt to interpret the slash characters as directory delimiter.

Path Names

- A pathname is a sequence of directories that lead to the file. A pathname can be either Absolute pathname or Relative pathname

- Absolute pathname
    - With reference to the root directory “/”
    - Each obsolete pathname is unique on an UNIX system

- Relative pathname
    - The name is relative to the current working directory
    - The path is valid only from that particular directory

UNIX - Commands

- UNIX command is an executable program
- The basic form of any Unix command is:

  command_name options argument(s)
- Most descriptions for commands such as those given in the On-line manual use a much more precise syntax. For example:

  cp [-iprR] filename ... directory
- This syntax has a few simple rules, which will help us understand how that command is to be used

man pages

- UNIX offers an online help facility in the man command.
- man displays the documentation of the specified command.

Example:

$ man wc displays help on *wc* command

- *man* uses a pager program, which displays this documentation one page at a time
- *man* is configured to be used with a specific pager.
- Two available pagers are : more, a Berkeley pager, as an alternative to the AT&T pg command less, the standard pager on Linux systems, also available on UNIX. It is modeled after vi editor and is more powerful than more.

Where are commands located & Connecting Commands

- ❖ Unix commands are executable binary files located in directories with the name bin (for binary). Many of these are located in the directory /usr/bin.
- ❖ When you enter the name of a command the shell checks to see if it is a built-in command.
- ❖ If it is not, it looks for the the binary file that the command name refers to in each of the directories that are defined in the PATH environment variable.
- ❖ which and whereis commands
- ❖ Unix allows you to link two or more commands together using a pipe.
- ❖ The pipe takes the standard output from one command and uses it as the standard input to another command.

  Example : **command1 | command2 | command3**
- ❖ The | (vertical bar) character is used to represent the pipeline connecting the commands.

Useful Commands

- cd – change directory (The tilda (~) represents your home directory)
- pwd – present working directory
- mkdir - make directories
- Syntax: mkdir [OPTION] DIRECTORY...

  Options: -m=mode, -p
- rmdir - make directories
- rm - remove files or directories

  Syntax: **rm** [OPTION]... FILE...

  Options:i,f,r,v

Useful Commands

- cp - copy files and directories

  Syntax:

  cp [OPTION]... SOURCE DEST

  cp [OPTION]... SOURCE... DIRECTORY

  Options:i,p,f,r,R,s,u

- mv - move (rename) files

  Syntax:

  mv [OPTION]... SOURCE DEST

  mv [OPTION]... SOURCE... DIRECTORY

  Options: b,i,f,u

Some useful UNIX Commands

ls - Lists directory contents

Syntax:

ls [OPTIONS]... [FILE]...

Options:

l – long listing

a – all files including hidden files

F – Flag file types

C – lists files in Columns

d – lists only directory information

i – Lists I-node for each file

R – lists files in all subdirectories

r – Lists files in reverse order

t – list with modification time stamp

1 – print 1 line of output per line

File Permissions

- The above two lines have 9 columns each, the table describes what they are

Type	Owner			Group			Everyone			#	Owner	Size	Date	Name
-,d,l	r	w	x	r	w	x	r	w	x					
-	r	w	x	r	-	x	r	-	-	1	cvwork	82349	Jan 20 2002	Readme.txt

- The first character indicates the type of the file as described below.
  - **-** : It is a regular file
  - **d**: It is a directory
  - **l** : It is a link to a file

File permissions

- The next 3 fields (9 characters) indicate the file access permissions for each kind of users to the file, and are represented by a set of 3 octal digits.
    - **rwx** - for owner (first digit)
    - **rwx** - for group (second digit)
    - **rwx** - for the other (third digit)

- Each character means the following
    - **r** - can read the file
    - **w** - can edit the file
    - **x** - execute the file
    - **-** - can not

- You can change/modify the permissions of the file using the chmod command, which will be discussed latter.

Chmod – Change Mode

- ❖ Chmod <options> <args>

Octal	Permissions	Significance
0	- - -	no permissions
1	- - x	execute only
2	- w -	write only
3	- w x	write and execute
4	r - -	read only
5	r - x	read and execute
6	r w -	read and write
r w x	read, write and execute	

- ❖ Using relative permission

  chmod a+rw xstart

- ❖ Using absolute permission

  chmod 666 xstart

  chmod 644 xstart

  chmod 761 xstart will assign all permissions to the owner, read and write permissions for the group and only execute permission to the others

Chown & chgrp

<u>chown</u>

- Changing ownership requires superuser permission, so use su command

```
ls -l note
-rwxr----x 1 kumar  metal  347  may  10  20:30  note
```

```
chown sharma note; ls -l note
-rwxr----x 1 sharma  metal  347  may  10  20:30  note
```

chgrp

- This command changes the file’s group owner
- No superuser permission is required

```
ls –l dept.lst
-rw-r--r--  1  kumar  metal  139  jun  8  16:43  dept.lst
chgrp dba dept.lst; ls –l dept.lst
-rw-r--r--  1  kumar  dba  139  jun  8  16:43  dept.lst
```

VI EDITOR

Vi - Special Keys

Special Keys		
Key	**Command Mode**	**Text Mode**
Arrow Keys	cursor movement	N/A
[CTRL]	used with other keys for extra commands	insert control characters in text
[ENTER] or **[RETURN]**	down 1 line	normal function
[ESC]	N/A	leave text mode
Space Bar	move right 1 character	normal function
[TAB]	N/A	normal function

Vi – Mode Changing Commands

Mode Change Commands	
a	append text after cursor
A	Apend text at the end of the line
i	insert text before cursor
I	Insert text at the beginning of the line
o	open new line after current line & add text
O	open new line before current line & add text
[ESC]	leave text mode

Vi – Saving and Exiting

Saving & Exiting Commands	
:w	write (save) the file
:w NAME	write the file and name it NAME
:w! NAME	rewrite the file named NAME
:wq	write the file and quit the editor
:q	quit the editor
:q!	quit the editor (abandoning any changes)

Vi Cursor motion commands

Cursor Motion Commands	
h	move back 1 character
l	move forward 1 character
j	move down 1 line
k	move up 1 line
b	back to beginning of word
e	forward to end of word
w	forward to beginning of next word
^	go to first displayable character of line
0	go to beginning of line
$	go to end of line
<CTRL>F	forward 1 screen
<CTRL>B	backward 1 screen
<CTRL>D	down (forward) 1/2 screen
<CTRL>U	up (backward) 1/2 screen

VI text delete commands

Delete Commands	
x	delete character
dw	delete rest of word
d0	delete from the beginig of the line
d$	delete rest of line
D	delete rest of line
dD	delete rest of file
dd	delete line

Vi – Yank and put commands

Yank Commands	
yw	yank rest of word
y$	yank rest of line
yy	yank entire line
Y	yank entire line

Put Commands	
p	put yanked/deleted text before cursor
P	put yanked/deleted text after cursor

Vi – Replace commands

Replace Commands	
r	replace character
rw	replace the entire word
R	replace the entire line
cw	change rest of word
c$	change rest of line
C	change rest of line
Ns	substitute text for N characters

Vi – Search commands

Search Commands	
/text	search forward for text
?text	search backward for text
n	search in same direction for next occurance of last-searched-for text
N	search in other direction for next occurance of last-searched-for text
Ns	substitute text for N characters

Vi – Undo command

- In the real vi, the undo command only undoes the last command, even if that is an undo command.
- Repeated undo commands simply toggle the effect of the last command before the series of undo commands.
- Some vi clones, such as vim, allow multiple undo's.

	Undo Command
u	undo last command (BE CAREFUL WITH THIS)

- R –redo
- S-substitution
- Vi Misc
- Repeat command (.)
- Join lines (J)
- :Set command
- Vi configuration file (.exrc)

SIMPLE FILTERS

head

- Displays the top of the file
- It displays the first 10 lines of the file, when used without an option
- head emp.lst
- -n to specify a line count
- head -n 3 emp.lst

tail

- Displays the end of the file
- It displays the last 10 lines of the file, when used without an option
- tail emp.lst
- -n to specify a line count
- tail -n 3 emp.lst
- Monitoring the file growth (-f)
- Extracting bytes rather than lines (-c)

cut

- It is used for slitting the file vertically
- head -n 5 emp.lst | tee shortlist

will select the first five lines of emp.lst and saves it to *shortlist*

- We can cut by using -c option with a list of column numbers, delimited by a comma (cutting columns)

cut -c 6-22,24-32 shortlist

cut -c -3,6-22,28-34,55- shortlist

cut

- ❖ Most files don’t contain fixed length lines, so we have to cut fields rather than columns (cutting fields)

-d for the field delimiter

-f for the field list

```
cut -d \ | -f 2,3 shortlist | tee cutlist1
```

will display the second and third columns of *shortlist* and saves the output in *cutlist1.* here | is escaped to prevent it as pipeline character

- ❖ To print the remaining fields, we have

```
cut –d \ | -f 1,4- shortlist > cutlist2
```

paste

- When we cut with cut, it can be pasted back with the paste command, *vertically*

```
paste cutlist1 cutlist2
```

- We can view two files side by side
- We can specify one or more delimiters with -d

```
paste -d “|” cutlist1 cutlist2
```

where each field will be separated by the delimiter |

Sort : Ordering a file

- ❖ Sorting is the ordering of data in ascending or descending sequence. The sort command orders a file and by default, the entire line is sorted

Example:

sort shortlist

-tchar	uses delimiter *char* to identify fields
-k n	sorts on nth field
-k m,n	starts sort on mth field and ends sort on nth field
-k m.n	starts sort on nth column of mth field
-u	removes repeated lines
-n	sorts numerically
-r	reverses sort order
-f	folds lowercase to equivalent uppercase
-m list	merges sorted files in list
-c	checks if file is sorted
-o flname	places output in file flname

Sort : Ordering a file Cont…

```
sort –t“|” –k 2 shortlist

sort –t”|” –r –k 2 shortlist                or

sort –t”|” –k 2r shortlist

sort –t”|” –k 3,3 –k 2,2 shortlist

sort –t”|” –k 5.7,5.8 shortlist

sort –n numfile
```

uniq command

Locate repeated and non-repeated lines

cat dept.lst

uniq dept.lst

sort : dept.lst | uniq - uniqlist

Selecting the nonrepeated lines (-u)

Selecting the duplicate lines (-d)

Counting frequency of occurrence (-c)

tr command

Manipulates the individual characters in a line. It translates characters using one or two compact expressions

tr options expn1 expn2 standard input

It takes input only from standard input, it doesn’t take a filename as argument

tr ‘|/’ ‘~-’ < emp.lst | head –n 3

grep

- ❖ It scans the file / input for a pattern and displays lines containing the pattern, the line numbers or filenames where the pattern occurs
- ❖ It's a command from a special family in UNIX for handling search requirements

grep *options pattern filename(s)*

grep options

-i	ignores case for matching
-v	doesn't display lines matching expression
-n	displays line numbers along with lines
-c	displays count of number of occurrences
-l	displays list of filenames only
-e exp	specifies expression with this option
-x	matches pattern with entire line
-f file	takes pattrens from file, one per line
-E	treats pattren as an extended RE
-F	matches multiple fixed strings

egrep

ch+	matches one or more occurrences of character ch
ch?	Matches zero or one occurrence of character ch
exp1|exp2	matches exp1 or exp2
(x1|x2)x3	matches x1x3 or x2x3

Process

What is a Process

- A running program in UNIX is called a *process*. Because UNIX is a multitasking system, many processes can run at the same time.
- A process is said to be born when the program starts execution. After the execution is complete, the process is said to die.
- Two important attributes of a process are,

 The process-id (PID): a unique identification number used to refer to the process.

 The parent PID (PPID): the number of the process (PID) that started this process.

- A process can be suspended, moved to the background or even be killed

ps: Process Status

- The process ID is simply a number that uniquely identifies each
- running process.
- To see what process IDs are associated with your process, use
- the ps command.

```
$ ps
PID   TTY        TIME   CMD
17717 pts/10   00:00:00 bash
27501 pts/10   00:00:01 find
27502 pts/10   00:00:00 ps
```

Here the login shell isn’t doing much work, its CPU usage is negligible.

ps options

POSIX Option	BSD Option	Description
-f	**F**	**Full Listing showing PPID of each process**
-e or –A	**aux**	**All processes including user and system processes**
-u *usr*	**U *usr***	**Processes of *usr* only**
-a		**Processes of all users excluding system processes**
-l	**l**	**Long listing showing memory-related information**
-t *term*	**t *term***	**Processes running on terminal *term* (say /dev/console)**

ps -f

$ ps -f

```
UID   PID        PPID      C   STIME     TTY      TIME CMD
srm   17717  1 0   15:57:07  pts/10   0:00 bash
srm   27501 3211         0   15:16:16  pts/10   0:01 find
srm   27502 2187         0   16:35:41  pts/10   0:00 ps –f
```

- UID – Login name of the user
- PID – Process ID
- PPID – Parent process ID
- C – An index of recent processor utilization, used by kernel for scheduling
- STIME – Starting time of the process in hours, minutes and seconds
- TTY – Terminal ID number
- TIME – Cumulative CPU time consumed by the process
- CMD – The name of the command being executed

Killing processes with Signals

- To use kill, use either of these forms:

  kill PID(s) OR kill –s NUMBER PID(s)

- To kill a process whose PID is 123 use,

```
$ kill 123
```

- To kill several processes whose PIDs are 123, 342, and 73 use,

```
$ kill 123 342 73
```

- Issuing the kill command sends a signal to a process. The default signal is SIGTERM signal (15). UNIX programs can send or receive more than 20 signals, each of which is represented by a number. (Use kill –l to list all signal names and numbers)
- If the process ignores the signal SIGTERM, you can kill it with SIGKILL signal (9) as,

  $ kill -9 123 OR $ kill –s KILL 123

Job Control

A job is a group of processes. A job is created by running a pipeline of two or more commands. You can use job control facilities to,

- ❖ Relegate a job to the background (bg)
- ❖ Bring it back to the foreground (fg)
- ❖ List the active jobs (jobs)
- ❖ Suspend a foreground job (*[Ctrl-z]*)
- ❖ Kill a job (kill)

Some more UNIX Utilities (Backup)

- The date utility
- The bc utility
- The expr utility
- The cal/calendar utilities
- The ping utility
- The uname utility
- The finger utility
- The clear utility

Process Commands

❖ Ps	Process State
❖ Kill	Kill Process
❖ Wait	Parent Wait for children
❖ Sleep	Sleep
❖ top	All the user status

Cron

- *cron* looks for instructions to be performed in a control file in
- /var/spool/cron/crontabs
- After executing them, it goes back to sleep, only to wake up the next minute.
- To a create a crontab file,
  - First use a editor to create a crontab file say cron.txt
  - Next use crontab command to place the file in the directory containing crontab files. crontab will create a file with filename same as user name and places it in /var/spool/cron/crontabs directory.
- Alternately you can use crontab with –e option.
- You can see the contents of your crontab file with crontab –l and remove them with crontab –r.

A typical entry in crontab file

minute hour day-of-month month-of-year day-of-week command

Time-Field Options for the crontab Command

Field	Range	
minute	00 through 59	Number of minutes after the hour
hour	00 through 23 (midnight is 00)	
day-of-month	01 through 31	
month-of-year	01 through 12	
day-of-week	01 through 07 (Monday is 01, Sunday is 07)	

The first five fields are time option fields. You must specify all five of these fields. Use an asterisk (*) in a field if you want to ignore that field.

A typical entry in crontab file

- ❖ 00-10 17 * 3.6.9.12 5 find / -newer .last_time –print >backuplist

The find command will be executed very minute in the first 10 minutes after 5 p.m. every Friday of the months March, June, September and December of every year.

To sort a file named /usr/wwr/sales/weekly and mail the output to a user named srm at 7:30 a.m. each Monday,

30 07 * * 01 sort /usr/wwr/sales/weekly |mail -s”Weekly Sales” srm

Customizing the Environment

The Shell

- ❖ UNIX shell is both an interpreter and a scripting language.
- ❖ An interactive shell runs a non-interactive shell when executing a shell script.
- ❖ Bourne Shell – Strong programming features, weak interpreter.
- ❖ C Shell – Improved interpretive features, wasn't suitable for programming.
- ❖ Korn Shell – Combines best of the two. Has features like aliases, command history. But lacks some features of C.
- ❖ Bash Shell – Superset that combined the features of Korn and C Shells. Conforms to POSIX shell specification.

Environmental Variables

- ❖ Shell variables are of two types: Local and environment.
- ❖ PATH, HOME, SHELL etc. are examples of environment variables.
- ❖ To include the directory /home/ahm/bin in the search use,

```
PATH=$PATH:/home/ahm/bin
```

The Initialization scripts

- The effect of assigning values to variables, defining aliases and using set options is applicable only for the login session; they revert to their default values when the user logs out.
- To make them permanent, use certain startup scripts.
- The startup scripts are executed when the user logs in.

  .profile (Bourne shell)

  .profile and .kshrc (Korn shell)

  .bash_profile (or .bash_login) and .bashrc (Bash)

  .login and .cshrc (C shell)
- You can also execute them by using a special command (called dot).

```
$ . .profile
```

The Shell's Wild-cards

Wild-card	Matches
*	Any number of characters including none
?	A single character
[ijk]	A single character – either i, j or k
[x-z]	A single character that is within ASCII range x and z
[!ijk]	A single character that is not an i, j or k (Not in C Shell)
[!x-z]	A single character that is not within the ASCII range x and z (Not in C shell)
{pat1,pat2…}	Pat1, pat2, etc. (Not in Bourne shell)

Example

Wild-card	Matches
*	Any number of characters including none
?	A single character
[ijk]	A single character – either i, j or k
[x-z]	A single character that is within ASCII range x and z
[!ijk]	A single character that is not an i, j or k (Not in C Shell)
[!x-z]	A single character that is not within the ASCII range x and z (Not in C shell)
{pat1,pat2…}	Pat1, pat2, etc. (Not in Bourne shell)

Escaping and Quoting

- Escaping: Providing a \ (backslash) before the wild-card to remove its special meaning.

- Quoting: Enclosing the wild-card, or even the entire pattern, within quotes. Anything within these quotes (barring a few exceptions) are left alone by the shell and not interpreted.

  echo ‘$HOME’ *displays $HOME*

Redirection

Redirection: The three standard files

- ❖ In the context of redirection, the terminal is a generic name that represents the screen or keyboard.
- ❖ We see the command output and error messages on the terminal and provide command input through the terminal (keyboard). The shell associates three files with the terminal. These files are actually streams of characters which many commands see as input and output.

There are three standard streams:

- ❖ Standard input – The file (stream) representing input, connected to keyboard.
- ❖ Standard output – The file (stream) representing output, connected to display.
- ❖ Standard error – The file (stream) representing error messages that emanate from the command or shell, connected to display.

Three Special Files

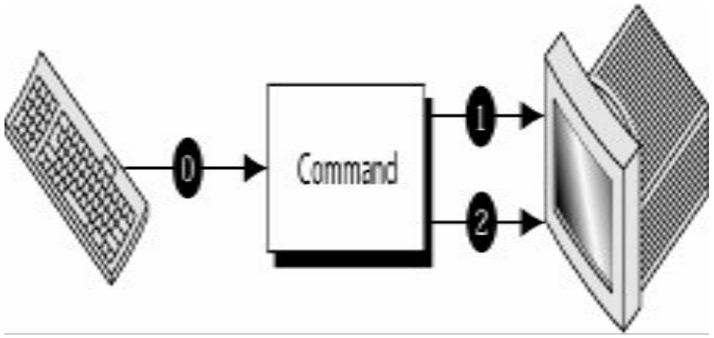
- Each of these three special files are represented by a number called, a file descriptor. A file is opened by using its pathname, but subsequent read and write operations identify the file by this file descriptor.

- The kernel maintains a table of the file descriptors. The first three slots are allocated to the three standard streams.

  0 – Standard Input

  1 – Standard Output

  2 – Standard Error

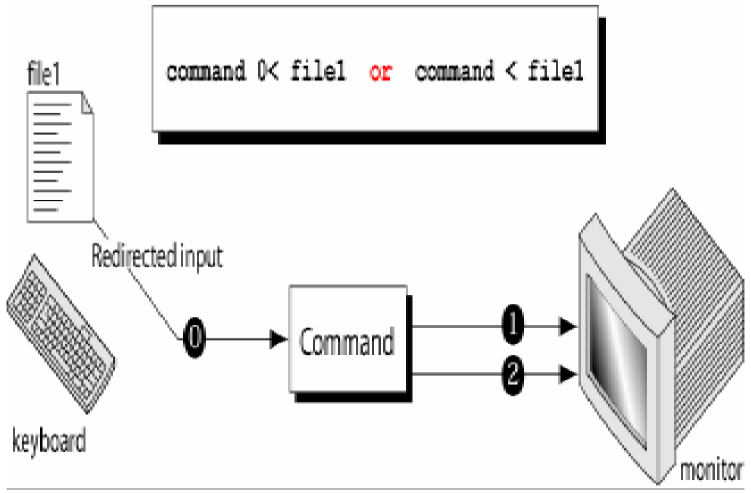


Standard Input

- When a command is used without arguments, it reads the file representing the standard input. It can represent three input sources viz.,
  - The keyboard, the default source
  - A file using redirection with the < symbol
  - Another program using a pipeline

Example:

- wc *(without any arguments)*
- wc < sample.txt
- ls | wc

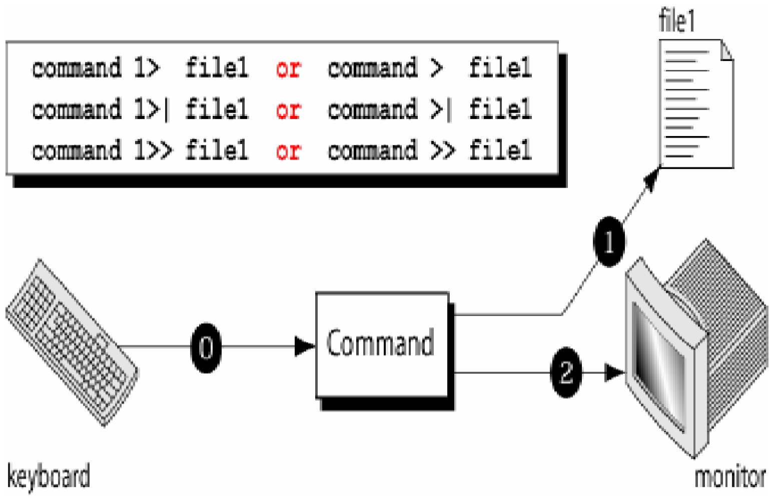


Standard Output

- All commands displaying output on the terminal actually write to the standard output file as a stream of characters. There are three possible destinations of this stream:
  - The terminal, the default destination
  - A file using the redirection symbols > and >>
  - As input to another program using a pipeline

Example:

- wc sample.txt
- wc sample.txt > outputFile
- who | wc –l
- cat *.c > all_C_progs.txt



```
command 1>  file1   or   command >  file1
command 1>| file1   or   command >| file1
command 1>> file1   or   command >> file1
```

Standard Error

- ❖ When you enter an incorrect command, or try to open nonexistent file, certain diagnostic messages show up on the screen. This is the standard error stream.
- ❖ For example, trying *cat* on a nonexistent file produces the error stream.
- ❖ We can redirect this stream to a file. But, standard error cannot be redirected in the same way standard output can be (with > or >>). To capture the standard error and redirect to a file we have to use 2> symbols.

Example:

```
cat file1 2> errorfile
cat file2 2>> errorfile
```

/dev/null & /dev/tty

/dev/null

- If you would like to execute a command but don’t like to see its contents on the screen, you may wish to redirect the output to a file called /dev/null.
- It is a special file that can accept any stream without growing in size. It’s size is always zero.

/dev/tty

- This file indicates one’s terminal.
- In a shell script, if you wish to redirect the output of some select statements explicitly to the terminal. In such cases you can redirect these explicitly to /dev/tty inside the script.

Pipes

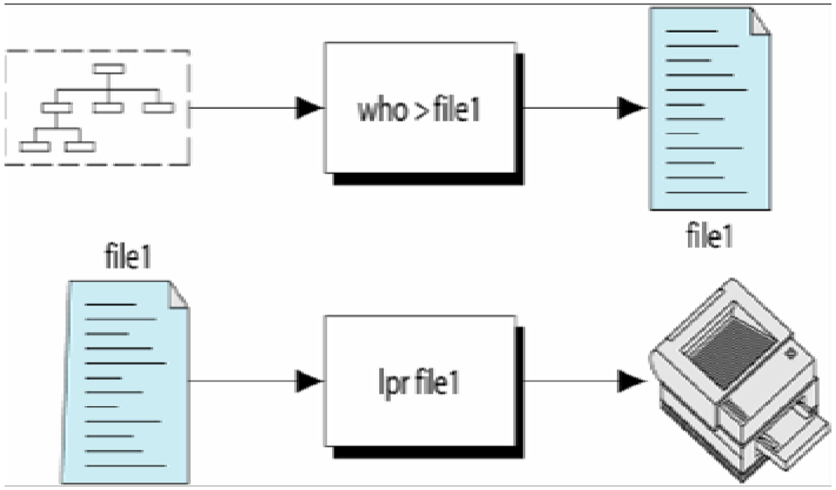
- ❖ UNIX without pipes is almost unthinkable

With piping, the output of a command can be used as input (piped) to a subsequent command.

Example: to print a list of users logged into the system.

- ❖ Without pipes,

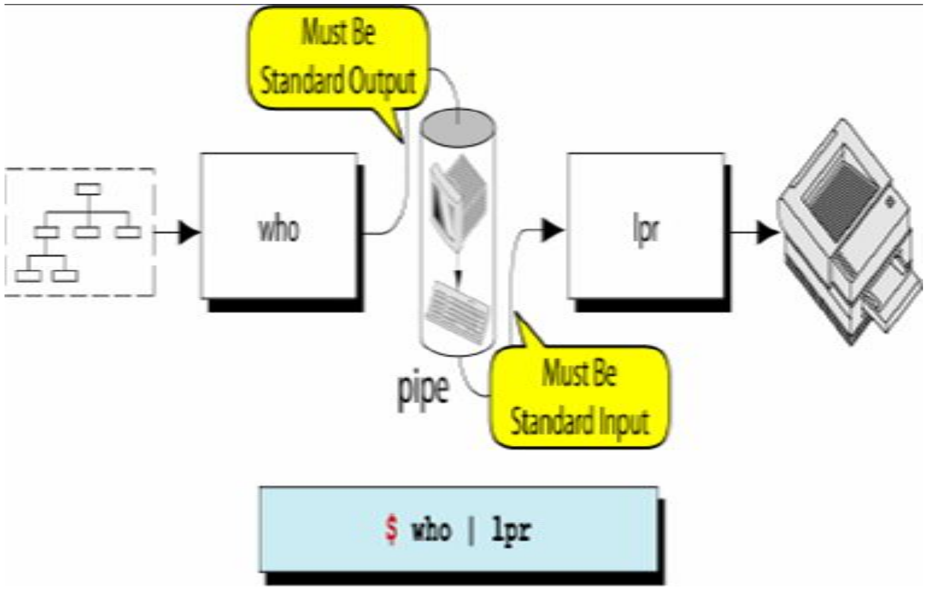
```
$ who > file1
$ lpr file1
```



Pipes

- Example: to print a list of users logged into the system.
- With pipes,

  $ who | lpr



Some More Useful Commands

Useful Commands

- ln - make links between files (hard and soft link)

  Syntax:

  ln [OPTION]... TARGET [LINK_NAME]

  ln [OPTION]... TARGET... DIRECTORY

  Options: b,f,i,s

- Two special director names

  . Current directory

  .. Parent directory

UNIX Utilities - find

- The "find" command is very powerful. It can search the entire filesystem for one or more files that you specify to look for.
- This is very helpful when a file has been "lost". You can also use the find command to locate files, and then perform some type of actions on that file.

Syntax: **find** [path...] [expression]

Options:

TESTS:

-name <name> : finds the file represented by name

-iname <name> : finds the file represented by name

UNIX Utilities - find

Options (cont):

-type <ftype> : find the file of type represented by ftype

ftype can be: d (directory) ,p (named pipes) ,f(regular files) ,l (smbolic links) ,s (sockets)

-perm <mode>: Finds the file with file permissions matches mode

-user <uname> : Finds the files owned by uname

ACTIONS:

-print : print the full file name on the standard output

-exec command

- ❖ Execute command. All following arguments to find are taken to be arguments to the command until an argument consisting of `;' is encountered.
- ❖ The string `{}' is replaced by the current file name being processed everywhere it occurs in the arguments to the command.
- ❖ Both of these constructions might need to be escaped (with a `\') or quoted to protect them from expansion by the shell.

UNIX Utilities - Find

Options (cont)

-ls :list current file in `ls -dils' format on standard output

-fls file : like –ls, but puts it in to a file

Examples:

- **find . print** : This command searches through the current directory for all the files and prints their location to screen.

- **find / -name File1** : This command searches through the root file system ("/") for the file named “File1". Prints the location to he screen if the file is found.

- **find /home/ -name File1 -type f -print** : This command searches through the "/home" directory for the file named "File1". Prints the location to he screen if the file is found.

UNIX Utilities - find

Examples (cont):

- **find /usr/opt /usr/local/ -name "*.csh" -type f -exec chmod 755 {} \;** : This command searches through the “/usr/opt" and “/usr/local" directories for files that end with the extension ".csh". When these files are found, their permission is changed to mode 755 (rwxr-xr-x). This example shows that the find command can easily search in more than one directories.
- **Find / -name core rm –rf {} \;** : This command deletes all the core files from the entire file system. Works only if you are root.

UNIX Utilities - df

- Reports file system disk space usage in terms of blocks
- Displays the disk space available on each file name argument. If no file name is specified displays this information for all the mounted file systems.

  Syntax: **df** [OPTION]... [FILE]...

  Options:

  a – all file systems

  h – Human readable form (Ex. 1.5K, 10.3M)

  k – In kilobyte blocks

  m – In megabyte blocks

  l – Limits to the local file system

UNIX Utilities - du

- Summarize disk usage of each file, recursively for all directories.

  Syntax: **du** [OPTION]... [FILE]...

  Options:

  a - writes count for all files, not just directories

  --block-size=<size in bytes> : use size byte blocks

  -b - prints size in bytes

  -c - prints the grand total

  -s - summarize, print only total of each directory

  -X FILE - Exclude files that match any pattern in FILE.

UNIX Utilities – File Compression

- zip , unzip

  Compress or archive file and directories

  Syntax: zip [OPTIONS] file zipfle

  Examples

  - **zip -r foo foo** : This command will pack all he files and directories under foo in to foo.zip
  - **find . -name "*.[ch]" -print | zip source -**@ : this command will archive all the C source files in the current directory and its subdirectories.
  - **Unzip foo.zip** : will extract the package and recreate the entire structure

UNIX Utilities – File Compression

- ❖ gzip, gunzip, zcat
- ❖ GNU version of zip, unzip utilities, compress or expand files and directories, These are much efficient than zip and unzip.

    compress , uncompress

Similar to zip, unzip commands

- ❖ diff ,cmp,comm
- ❖ split - split a file into pieces
- ❖ split –n1000 file1 : splits the file file1 in to many files each with 1000 lines. The last file will have the remaining lines. You an use the cat command to get back he original file.

UNIX Utilities – tar

- tar – Tape archive utility
  - A archiving program designed to store and extract files from an archive file known as a tarfile.
  - A tarfile may be made on a tape drive, however, it is also common to write a tarfile to a normal file.
  - The first argument to tar must be one of the following options, followed by any optional argument

UNIX Utilities – tar

- Options

  a- append tar files to an archive

  c- create a new archive

  d- find differences between archive and file system

  r- append files to the end of an archive

  t- list the contents of an archive

  u- update only append files that are newer than copy in archive

UNIX Utilities – tar

- Examples:

  tar cvf dir1.tar dir1 : This command will archive the directory dir1 in to a tarfile dir1.tar

  tar xvf dir1.tar : This command will extract the contents of the tarfile fir1.tar to recreate the original directory tree

  tar tvf dir1.tar: This command will list the contents of the tarfile dir1.tar

  tar Avf dir1.tar anotherfile : This command will append the file anotherfile to the end of the tarfile dir1.tar

SED & AWK

sed

- sed stands for stream editor.
- sed is a non-interactive editor used to make global changes to entire files at once
- An interactive editor like vi would be too cumbersome to try to use to replace large amounts of information at once
- Syntax:
  - sed –e ‘command’ file(s)
  - sed –e ‘command’ –e ‘command’ … file(s)
  - sed –f scriptfile file(s)

sed

Whole line oriented functions

DELETE	d
APPEND	a
CHANGE	c
SUBSTITUTE	s
INSERT	I

Sed

- cat file

  I have three dogs and two cats

  sed -e 's/dog/cat/g' -e 's/cat/elephant/g' file

  I have three elephants and two elephants

- sed –e /^$/d foo

  deletes all blank lines

- sed -e 6d foo

  deletes line 6.

- sed 's/Tx/Texas/' foo

  replaces Tx with Texas in the file foo

Sed

- sed -e '1,10d' foo

  delete lines 1-10 from the file foo

  sed ‘1i\

  sed ‘a\

Awk

- awk is a pattern scanning and processing language.
- Named after its developers Aho, Weinberger, and Kernighan. (developed in 1977)]
- Search files to see if they contain lines that match specified patterns and then perform associated actions.

Running awk program:
There are several ways to run an awk program

awk '*program*' *input-file1 input-file2* ...
awk -f *program-file input-file1 input-file2* ...

- awk checks to see if the input records in the specified files satisfy the pattern
- If they do, awk executes the action associated with it.
- If no pattern is specified, the action affects every input record.
- A common use of *awk* is to process input files by formatting them, and then output the results in the chosen form.

Awk

- A sample data file named countries

```
Canada:3852:25:North America
USA:3615:237:North America
Brazil:3286:134:South America
England:94:56:Europe
France:211:55:Europe
Japan:144:120:Asia
Mexico:762:78:North America
China:3705:1032:Asia
India:1267:746:Asia
```

- country name, area (km^2), population density(10^6/km^2), continent

Awk

```
awk -F: '{ printf "%-10s \t%d \t%d \t%15s \n",$1,$2,$3,$4 }' countries
```

Outputs:

```
Canada      3852    25      North America
USA         3615    237     North America
Brazil      3286    134     South America
England     94      56      Europe
France      211     55      Europe
Japan       144     120     Asia
Mexico      762     78      North America
China       3705    1032    Asia
India       1267    746     Asia
```

Some built-in Variables

- NF - Number of fields in current record
- $NF - Last field of current record
- NR - Number of records processed so far
- FILENAME - name of current input file
- FS - Field separator, space or TAB by default
- $0   - Entire line
- $1, $2, …, $*n* - Field 1, 2, …, *n*

printf examples

printf syntax:

printf "control-string" arg1, arg2, ... , argn

- printf “I have %d %s\n”, how_many, animal_type
- printf “%10s %-4.2f %-6d\n”, name, interest_rate, account_number
- printf “\t%d\t%d\t%6.2f\t%s\n”, id_no, age, balance, name

Awk

- *awk* opens a file and reads it serially, one line at a time.
- By specifying a pattern, we can select only those lines that contain a certain string of characters.
- For example we could use a pattern to display all countries from our data file which are situated within Europe.

  ```
  awk '/Europe/' countries
  ```

Match operator

- A sample data file named countries

```
Canada:3852:25:North America
USA:3615:237:North America
Brazil:3286:134:South America
England:94:56:Europe
France:211:55:Europe
Japan:144:120:Asia
Mexico:762:78:North America
China:3705:1032:Asia
India:1267:746:Asia
```

- awk -F: '$3 == 55' countries
- Matching operators are :

==equal to;	**!= not equal to;**
> greater than;	**< less than;**
>= greater than or equal to;	**<= less than or equal to**

Logic Operations

Sample file named cars:

```
ford mondeo          1990    5800
ford fiesta          1991    4575
honda       accord          1991    6000
toyota      tercel          1992    6500
buick       centry          1990    5950
buick       centry          1991    6450
```

- $ awk '$3 >=1991 && $4 < 6250' cars
- $ awk '$1 == "ford" || $1 == "buick"' cars

BEGIN & END SECTIONS

- awk statements are applied to all lines selected by the address.
- If there are no addresses, then they are applied to every line of input.
- To print something before processing the first line, for example, a heading, then the BEGIN section can be used.
- Similarly, the end section useful in printing some totals after processing is over.
- The BEGIN and END sections are optional
- Syntax:

```
BEGIN     {action}
END       {action}
```

Note: Both of them use curly braces.

- These two sections, when present, are delimited by the body of the awk program

Awk

- Here BEGIN section prints a suitable heading offset by two tabs (\t \t).
- The END section prints the average pay (tot / count) for the selected lines.
- To execute this program, use the –f option:

```
$ awk –F “|” –f emp.awk emp.lst
```

- Built In Variables

The FS Variable: awk ues a contiguous string of spaces as the default field delimeter.

The OFS Variable: print statement used with comma-separated arguments, each argument was separated from the other by a space in the output.

Awk Variable declaration

- *Awk* variables take on numeric (floating point) or string values according to context. For example, in
- x = 1
- x = "3" + "4"
- assigns 7 to x.
- { s1 += $1; s2 += $2 }

Awk If Condition

<u>~ and !~ : The Regular Expression Operators:</u>

- if ($6 < 6000)
- da = 0.25*$6
- else
- da = 1000
- Example1: $2 ~ /[cC]ho[wu]dh?ury / || $2 ~ /sa[xk]s?ena /

  Matches second field
- Example2: $2 !~ /manager | chairman /

  Neither manager nor chairman

For & While

For

The for statement is also exactly that of C:

```
    for (i = 1; i <= NF; i++)
    print $i
for (i in array)
```

statement

While

```
i = 1
while (i <= NF) {
print $i
++i
}
```

Data processing

- ❖ Sample file named wages

```
Brooks    10       35
 Everest  8        40
 Hatcher  12       20
 Phillips 8        30
 Wilcox            12       40

 name,        $/hour,    hours/week
```

- ❖ Calculate $/week, tax/week, (25% on tax).

```
awk '{ print $1,$2,$3,$2*$3,$2*$3*0.25 }' wages
```

Shell Programming

Introduction to Shell

<u>Definition:</u>

Shell is an agency that sits between the user and the UNIX system.

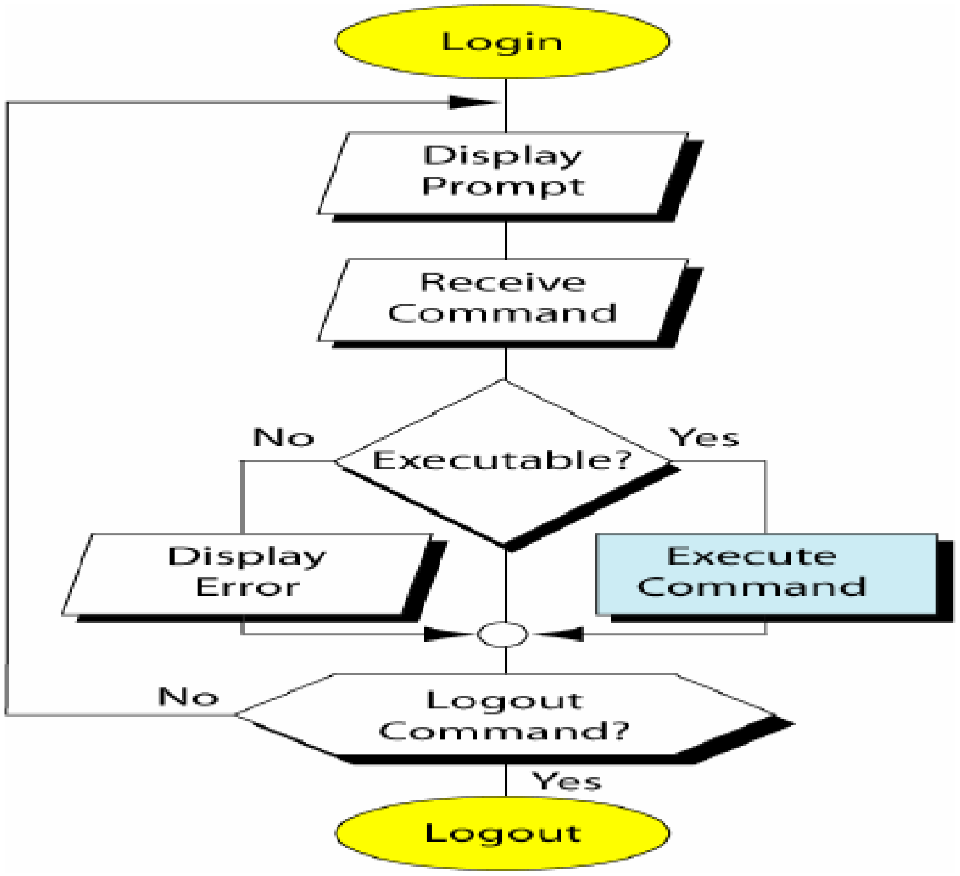
<u>Description:</u>

Understands all user directives and carries them out.
Processes the commands issued by the user.
Type of shell called Bourne shell.

When you log in, you are in one of five shells.

- ❖ Bourne Shll – /bin/sh – Steve Bourne (On AT & T UNIX)
- ❖ Bourne Again SHell – /bin/bash – Improved Bourne shell (On Linux)
- ❖ C Shell – /bin/csh – Bill Joy (On BSD UNIX)
- ❖ TC Shell – /bin/tcsh (On Linux)
- ❖ Korn Shell – /bin/ksh – David Korn (On AT & T UNIX)
- ❖ The system administrator determines which shell you start in.

The Shell's Interpretive cycle



Shell programming

- A simple shell script

  The following script (simple.sh) prints Hello World! in the screen

  #!/bin/sh

  echo “Hello World!”

- In the above shell script, the first line #!/bin/sh is the shell executable (in this case it is bourn shell located in /bin/ directory). This line serves as a interpreter for this script.

- You can also have a shell script without this line. In that case either we have to explicitly specify the shell to execute this script or it will be interpreted by the current shell.

Shell Programming

- ❖ To execute the above shell script, the file should have the execute permission set.
- ❖ To change/modify the permissions, use chmod command. At the shell prompt type the following command to set execute permission for the file simple.sh.

  chmod +x simple.sh

  To run use

  - ❖ **./simple.sh**
  - ❖ **Sh simple.sh**
  - ❖ **For debugging use sh –x simple.sh**
  - ❖ **For debugging inside the script use SET –x ,SET +x and SET –v, SET +v**

  ⇒ **set -x** : Display commands and their arguments as they are executed.

  ⇒ **set -v** : Display shell input lines as they are read.

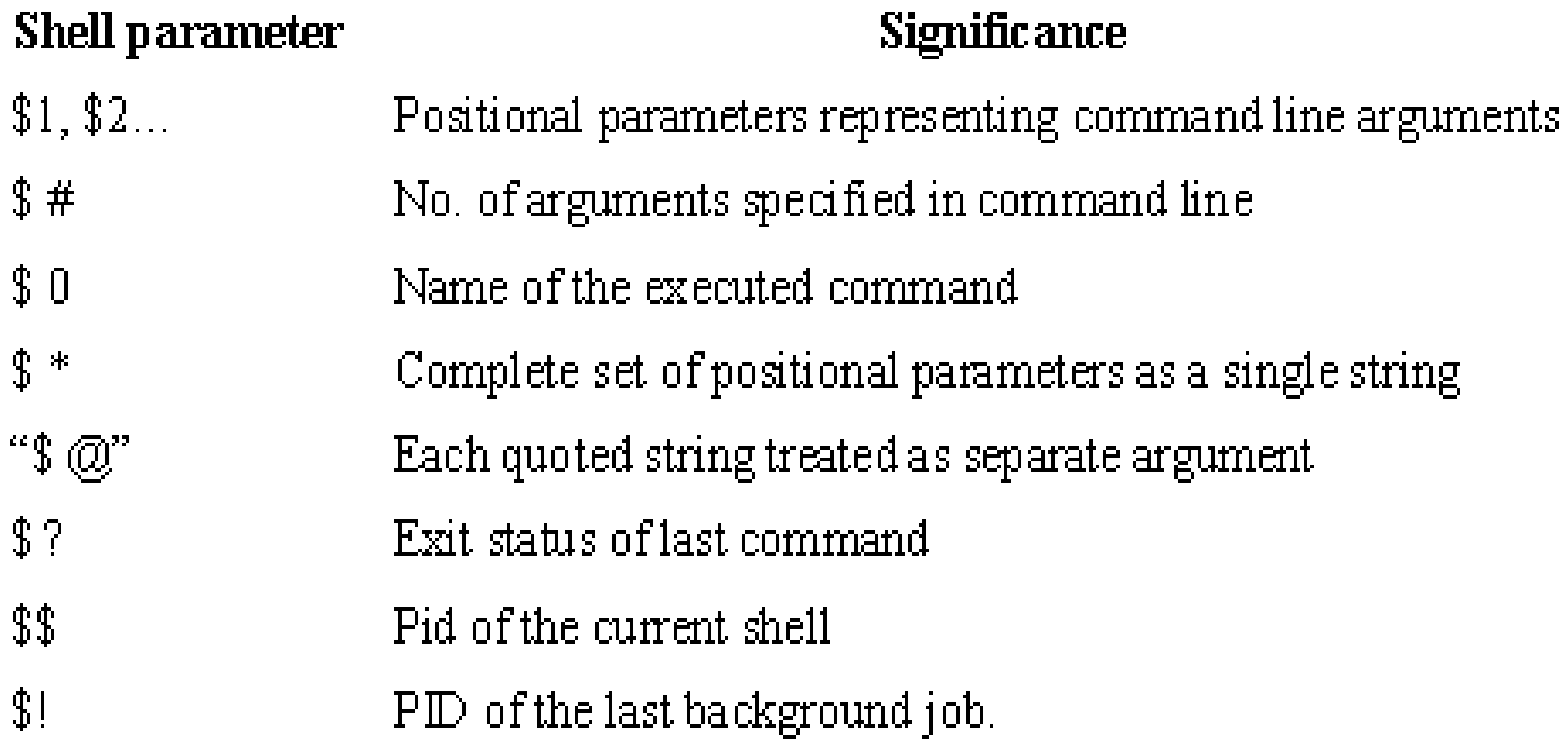
Read: Making Scripts Interactive

Example:

A shell script that uses read to take a search string and filename from the terminal.

```
#! /bin/sh
# emp1.sh: Interactive version, uses read to accept two
# inputs
    echo “Enter the pattern to be searched: \c”
# No newline
    read pname
```

Using Command Line Arguments



Shell parameter	Significance
$1, $2…	Positional parameters representing command line arguments
$ #	No. of arguments specified in command line
$ 0	Name of the executed command
$ *	Complete set of positional parameters as a single string
"$ @"	Each quoted string treated as separate argument
$?	Exit status of last command
$$	Pid of the current shell
$!	PID of the last background job.

Variables and arrays

- Variables
- a=2
- b=2.2
- c=‘A’
- d=“Hello”
- Arrays
- A[$i]=$value
- Display ${a[i]}

The if Conditional

Form 1

```
if command is successful
then
 execute commands
fi
```

Form 2

```
if command is successful
then
execute commands
else
 execute commands
fi
```

Form 3

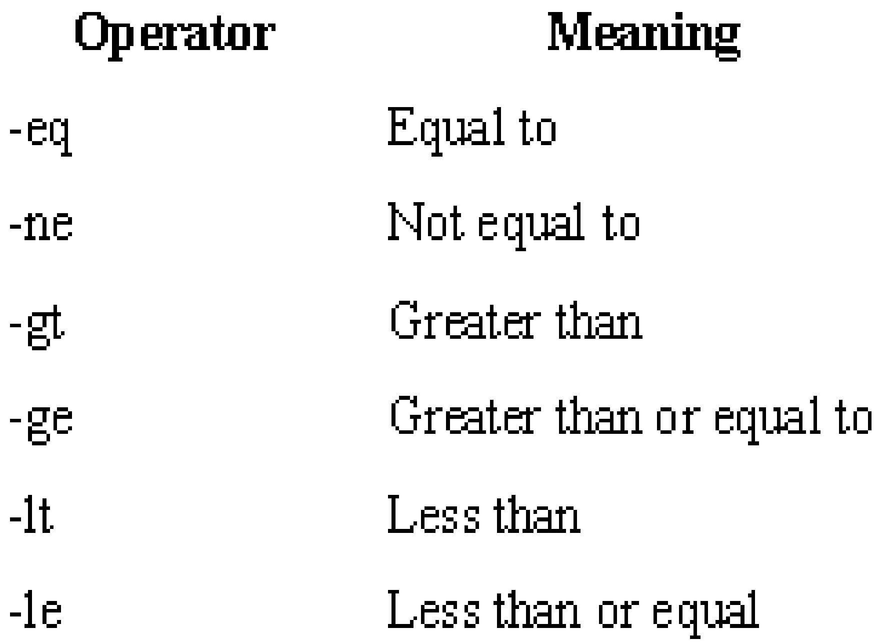
```
if command is successful
then
execute commands
elif command is successful
then...
else...
fi
```

If the command succeeds, the statements within if are executed or else statements in else block are executed (if else is present).

Using test and [ ] to Evaluate Expressions

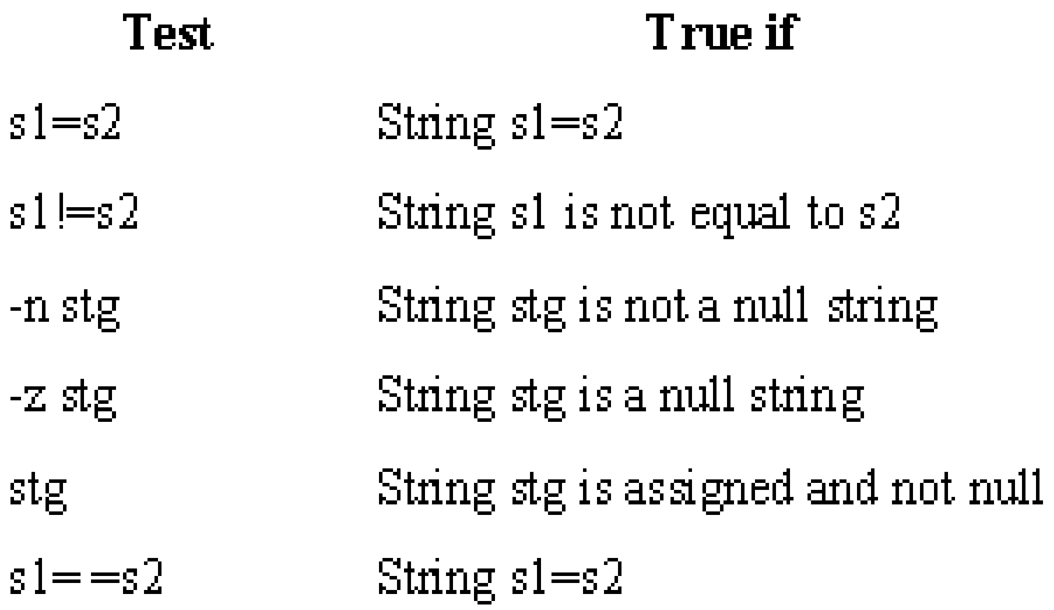
- Test statement is used to handle the true or false value returned by expressions.
- Test uses certain operators to evaluate the condition on its right
- Returns either a true or false exit status
- Is used by if for making decisions.
- Test works in three ways:
  - Compare two numbers
  - Compares two strings or a single one for a   null value
  - Checks files attributes
- Test doesn’t display any output but simply returns a value that sets the parameters $?

Using test and [ ] to Evaluate Expressions



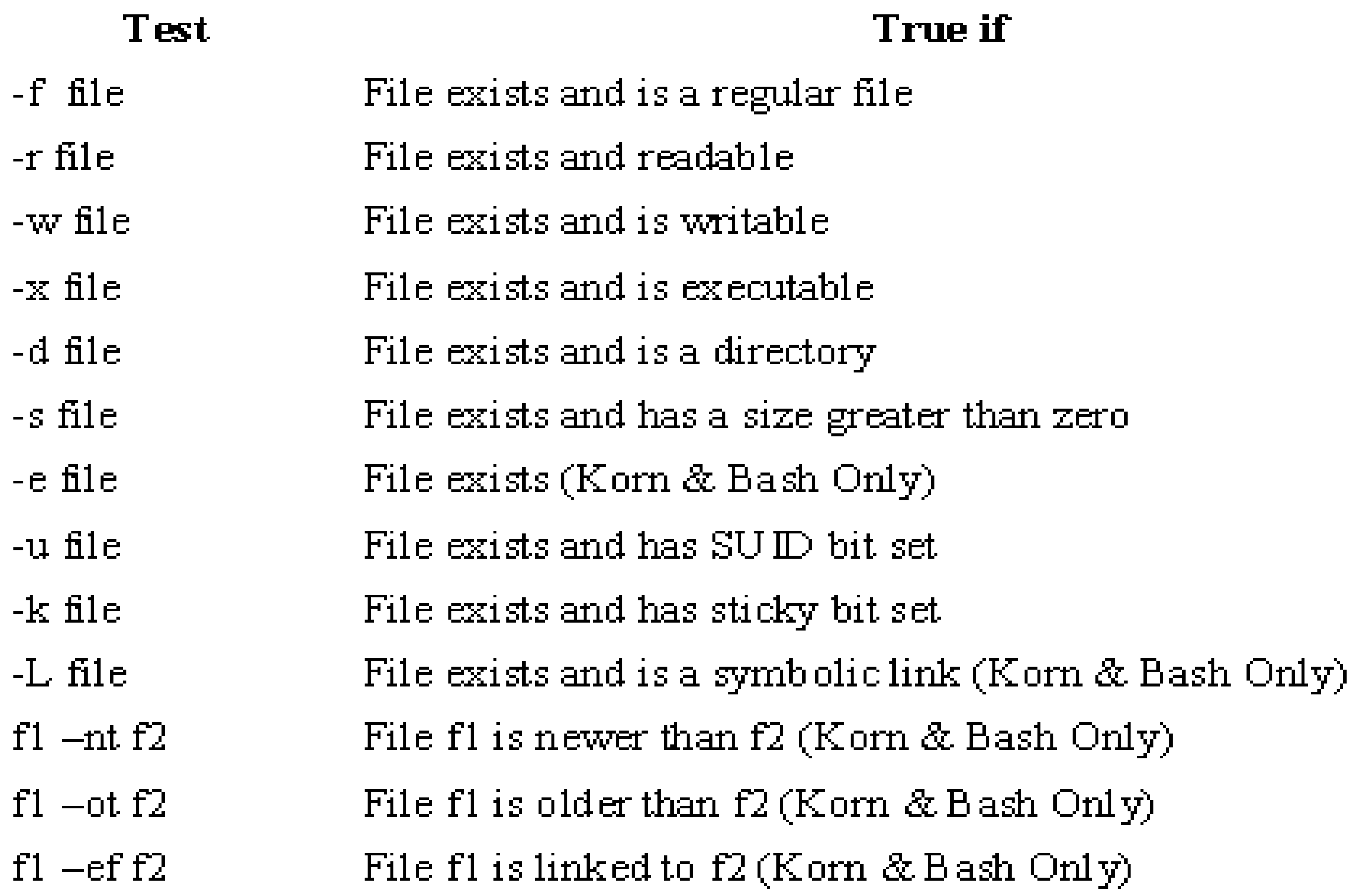
Operator	Meaning
-eq	Equal to
-ne	Not equal to
-gt	Greater than
-ge	Greater than or equal to
-lt	Less than
-le	Less than or equal

Using test and [ ] to Evaluate Expressions



Test	True if
s1=s2	String s1=s2
s1!=s2	String s1 is not equal to s2
-n stg	String stg is not a null string
-z stg	String stg is a null string
stg	String stg is assigned and not null
s1==s2	String s1=s2

File Tests



Test	True if
-f file	File exists and is a regular file
-r file	File exists and readable
-w file	File exists and is writable
-x file	File exists and is executable
-d file	File exists and is a directory
-s file	File exists and has a size greater than zero
-e file	File exists (Korn & Bash Only)
-u file	File exists and has SUID bit set
-k file	File exists and has sticky bit set
-L file	File exists and is a symbolic link (Korn & Bash Only)
f1 –nt f2	File f1 is newer than f2 (Korn & Bash Only)
f1 –ot f2	File f1 is older than f2 (Korn & Bash Only)
f1 –ef f2	File f1 is linked to f2 (Korn & Bash Only)

The Case Conditional

Syntax:

```
case expression in
Pattern1) commands1 ;;
Pattern2) commands2 ;;
Pattern3) commands3 ;;
…
esac
```

expr: Computation and String Handling

- ❖ The Bourne shell uses expr command to perform computations and string manipulation.
- ❖ expr can perform the four basic arithmetic operations (+, -, *, /), as well as modulus (%) functions.

<u>Computation:</u>

Example1 :$ x=3 y=5

```
$ expr  $x+$y
8                                   → Output
```

Example 2:$ expr 3 * 5

```
15                                  → Output
```

*Note: \ is used to prevent the shell from interpreting * as metacharacter*

expr: Computation and String Handling

Length of the string:

The regular expression .* is used to print the number of characters matching the pattern .

Example: $expr “abcdefg” : ‘.*’ (o/p → 7)

Extracting a substring:

expr can extract a string enclosed by the escape characters \ (and \).

Example:$ st=2007

$ expr “$st” :’..\(..\)’ (o/p → 07)

Locating position of a character:

expr can return the location of the first occurrence of a character inside a string.

Example: $ stg = abcdefgh ; expr “$stg” : ‘[^d]*d’

(o/p →4)

For : Looping

- More examples of for loop

```
for i in *
do
   cat $i
done | grep sh  > sh.out #note where loop output goes
```

- Try these examples of some for loop header lines

```
for i in $*
for i in `cat file.lst`
for i in 2 4 6
```

While : Looping

- While loop

```
while <command_list>
do
   <command_list>
Done
```

Example:

```
cnt=1
while test $cnt -ne 5
do
    echo $cnt
 cnt = `expr $cnt + 1`
 done
```

Functions

Function Call

```
Func a b
Stat=$?
```

Function definition

```
Func()
{
arg1=$1
Arg2=$2……
Commands;
Return value;
}
```

Other Application Interface

sqlplus

```
#!/usr/bin/ksh
Blah=$1
....
sqlplus system/manager@testdb1 << EndOfFile
select * from dual ;
{blah blah blah}
quit
EndOfFile
{more shell script}
```

sqlplus

- ❖ sqlplus -s scott/tiger <<EOF

  SET SERVEROUTPUT ON

  SET FEED OFF spool aaa.txt BEGIN DBMS_OUTPUT.PUT_LINE('xyz'); END; / spool off EOF

- ❖ sqlplus -s scott/tiger <<EOF

```
SET SERVEROUTPUT ON
SET FEED OFF spool aaa.txt
BEGIN
DBMS_OUTPUT.PUT_LINE('xyz');
END; / spool off EOF
```

set serveroutput <ON | OFF> - Display Output From DBMS_OUTPUT.PUT_LINE built-in package

SET COLSEP <column separator> - Column Separators - SET COLSEP ‘,’

SET HEAD <OFF | ON> - display headers

SET LINESIZE <integer> , SET PAGESIZE <integer>

exec demoexec – Run Stored Procedure

Run Script : @ <path_and_script_name> @c:\oracle\product\ora102\rdbms\admin\catplan.sql

Calling Other Programs

Java

- ❖ Define Your class path if nessasary , if not it will take the global classpath
- ❖ Java cp -$classpath –Dvar1 –Dvar2 com.db.dbiq.classname
- ❖ When you run with java check the JAVA_HOME and PATH C or C++

  gcc <main.c> -o <main.out>

  g++ main.cpp then type main.out