CHAPTER 1: INTRODUCTION TO UNIX

Evolution of UNIX- UNIX System Structure- Features of UNIX- Operating System Services. Architecture of UNIX system, architecture of kernel, features of Kernel.

➢ Definition of operating system

Operating System is the system software that manages computer hardware, software resources, and provides common services for computer programs.

➢ Features of operating system

1) **Memory management**
   Memory management module performs allocation and de-allocation of memory space.
2) **Processor management**
   It helps OS to create and delete processes.
3) **Device management**
   It keeps tracks of all devices.
4) **File management**
   It manages all file related activities such as organization, storage, retrieval, naming, sharing and protection of files.
5) **Security**
   Security modules protects the data and information of a computer system against authorized access.
6) **Job accounting**
   Keep track of time and resource used by various jobs and users.
7) **Control over system performance**
   Recording delays between request for a service and response from the system.

➢ How operating system works?

Operating system is loaded into memory when a computer is booted and remains active as long as machine is up. After any program has completed execution, the operating system cleans up the memory and registers and makes them available for the next program

➢ Examples of operating system

Microsoft Windows, UNIX, Linux, MacOS etc

➢ Definition of UNIX

UNIX is a portable, multitasking, multiuser, time sharing operating system(OS).

- **Founder of UNIX**

  The UNIX was founded by Ken Thompson, Dennis Ritchie and Brain kerninghan at AT& T Bell Labs research in 1969.

- **Founder of LINUX**

  Linus Benedict Torvalds in 1991.

- **Difference between UNIX and LINUX (linux is a unix clone)**

UNIX	LINUX
It is a multi tasking, multiuser operating system	It is a free and open source software.
Developed by ken Thompson, Dennis Ritchie and Brain Kerninghan	Developed by Linus Benedict Torvalds.
Source code is not available to general public.	Source code is available to general public
Not portable	Portable
Solaris, HP UNIX are some versions.	Ubuntu, Fedora are some versions.

- **Need of UNIX**

  Network capability:

  With other OS, additional software must be purchased for networking but with UNIX, network capability is the part of the operating system.

- **History of UNIX**

1960	Bell labs involved in the project with MIT, General Electric and Bell Laboratories to develop a time sharing system called MULTICS(Multiplexed Operating and Computing System).
1969	Ken Thompson wrote the first version of the UNIX called UNICS(Uniplexed Information and Computing System)
1970	Finally UNICS became UNIX.

- **Differences between UNIX and DOS**

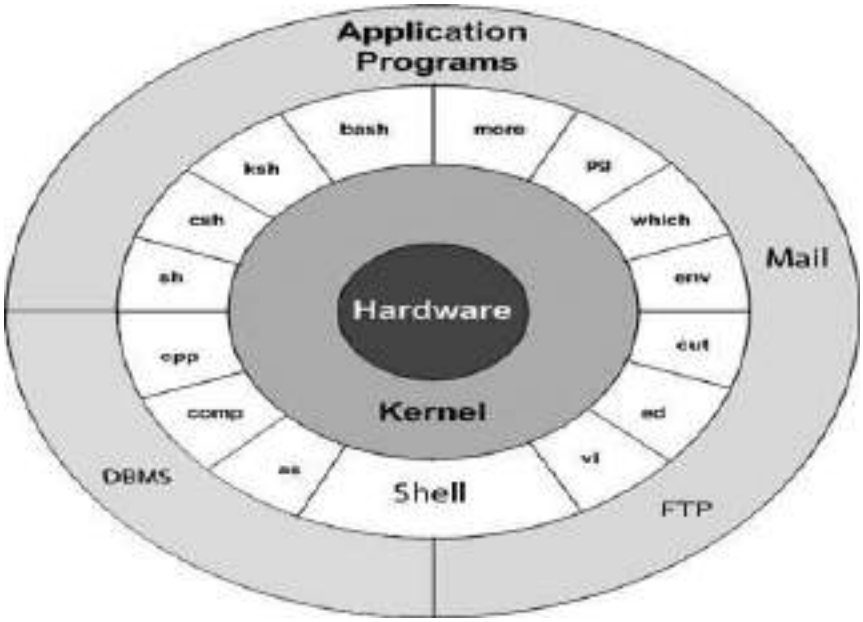
UNIX	DOS
UNIX can have GUI	DOS Can't have GUI
UNIX is more secure	DOS is not more secure compared to UNIX
UNIX is multitasking	DOS is singletasking
UNIX are multiuser	DOS is single user
UNIX is case sensitive	DOS is not case sensitive
UNIX is used in servers	DOS is used in embedded systems

- **Differences between UNIX and Windows**

UNIX	Windows
UNIX is an open source(free	Windows is not an open source(paid

download)	one)
UNIX has very high security system	Windows has low security system
UNIX is a command based operating system	Windows is not a command based operating system
The file system is arranged in hierarchical manner	The file system is arranged in parallel manner
UNIX is not a user friendly	Windows is a user friendly
Free office(such as excel, PowerPoint and others)	Pay for MS Office
Low Hardware cost	High Hardware cost
Customizable add features	Not customizable

➢ Architecture of UNIX System/ UNIX System Structure



The Architecture of the UNIX system is divided into 4 major components. They are:

1) The Kernel
2) The Shell
3) Files and Processes
4) System Calls

1. The Kernel

- The Kernel is the heart of the Operating System.
- It interface between Shell and Hardware.
- It performs Low Level Task.
- Eg: Device Management, Memory Management etc.

2. The Shell

- The Shell is a collection of UNIX Commands.
- The Shell acts as an interface between the user and the kernel.
- The Shell is a Command Line Interpreter(CLI)–Translates the commands provided by the user and converts it into a language that is understood by the Kernel.
- Only one Kernel running on the system, but several shells in action-one for each user who is logged in.
- Eg: C Shell, Bourne Shell, Korn Shell etc.

3. Files and Processes

- A ***File*** is an array of bytes and can contain anything.
- All the data in UNIX is organized into files.
- A ***Process*** is a program file under execution.
- Files and Processes belongs to a separate hierarchical structure.

4. System Calls

- UNIX is written in C.
- Thousands of commands in the system uses a handful of functions called ***System Calls*** to communicate with the kernel.

➢ Features of UNIX

1) Simple design, organization and functioning
The architecture of the UNIX is simple, organized and functional.

2) Portability
The code can be changed and compiled on a new machine.

3) UNIX Shell
The user interface UNIX Shell provides the services that the user wants.

4) Hierarchical file system
UNIX uses a hierarchical file structure to store information.

5) Multi user
UNIX allows more than one user to share the same computer system at the same time.

6) Multi-tasking
More than one program can be run at a time.

7) Security
UNIX provides high security level(System level security and File level security)

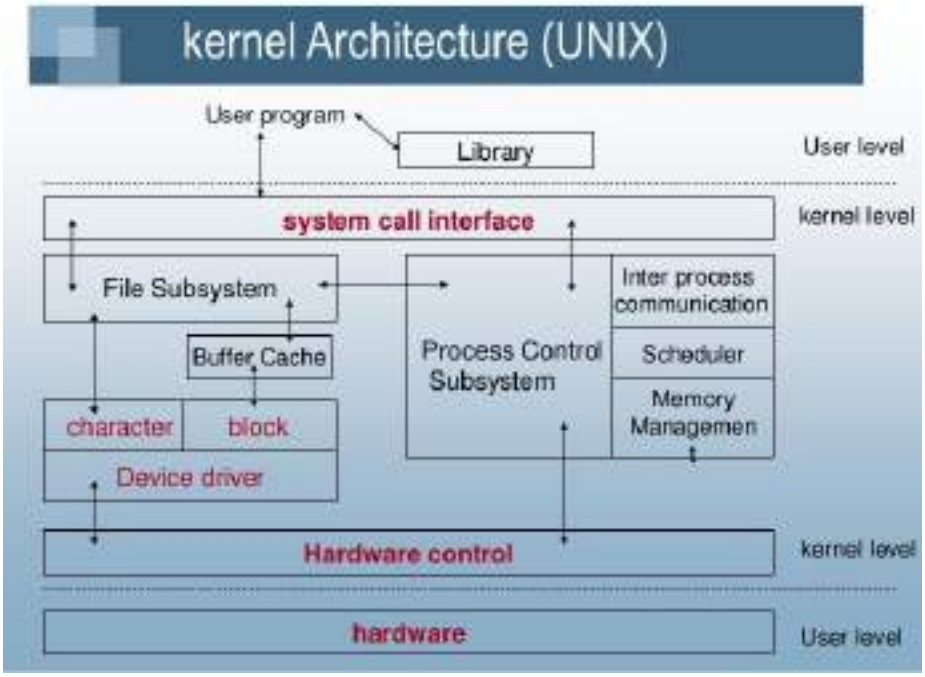
8) Pipes and Filters
In UNIX, we can create complex programs from simple programs.

9) Utilities
UNIX has over 200 utility programs for various functions.

10) Machine Independence

The system hides the machine architecture from the user, making it easier to write applications that can run any computer system.

➢ Architecture of Kernel



- The ***system call and library interface*** represent the border between user programs and the kernel.
- The ***File subsystem*** manages files, allocating file space, administrating free space, controlling access to files and retrieving data for users.
- ***Kernel Interface to Hardware*** is responsible for handling interrupts and for communicating with the machine.
- ***Device drivers*** are the kernel modules that control the operation of peripheral devices.
- ***Block I/O devices*** are random access storage devices to reset the system.
- ***Scheduler*** module allocates CPU to processes**.**
- ***Memory management*** module controls allocation of memory.
- ***Inter-process communication (IPC)*** ranges from asynchronous signaling of events to synchronous transmission of messages between processes.

➢ Features of Kernel

1) **Concurrency**

   Many processes run concurrently to improve the performance of the system.

2) **Virtual Memory(VM)**

Memory management subsystem implements the virtual memory and users need not worry about the executable program size and RAM size.

3) **Paging**
   It is a technique to minimize the internal and external fragmentation in the physical memory.
4) **Virtual File System(VFS)**
   A VFS is a file system used to help the user to hide the different file systems complexities.
5) **Inter process communication**
   Itranges from asynchronous signaling of events to synchronous transmission of messages between processes.

- ## Operating System Services

1) **User interface**
   User interface refers to the software that allows a person to interact with the computer.
2) **Program execution**
   The system creates a special environment for that program.
3) **File system manipulation**
   File manipulation commands manipulates the file.
4) **Input/ Output operation**
   OS provides the capability to change where standard input comes from or where output goes using Input/Output concept.
5) **Communication**
   Communication commands makes proper communication between different computers, networks and remote users.
6) **Resource allocation**
   Resource allocation techniques allocates resources when a program need them.
7) **Error detection**
   Error detection techniques helps us to detect and correct errors.
8) **Accounting**
   It helps us to maintain account details of users.
9) **Security and protection**
   Only authorised users can access the files and directories.

*****

CHAPTER-2

UNIX FILE SYSTEM

Introduction

A file is one in which data can be stored.It is a sequence of bits,bytes or lines and stored on a storage device like a disk.Inunix everything is stored in terms of files.It could be a program,an executable code,a file,a set of instructions,a database,a directory or a subdirectory. A unix file is a tool or an application that defines the structure of the file and its format.

Unix Filenames

- Unix is case sensitive! So a file named my data. Txt is different from mydata.txt which is again different from mydata.txt.
- Unix filenames contain only letters,numbers and the_(underscore)&.(dot) characters.All other characters should be avoided. The /(slash) characters is especially important/since it is used to designate subdirectories.
- It is also possible to have additional dots in the filename.
- The part of the name that follows the dot is often used to designate the type of file.
    - ❖ Files that end in.TXT are text files.
    - ❖ Files thatend in.C are source code in the 'C' language.
    - ❖ Files that end in.HTML files for the web.
- But this is just a convention and not a rule enforced by the operating system.this is a good and sensible convention which you should follow.
- Most UNIX system allow a maximum of 14 characters as the length of a filename. However it depends on the UNIX variant used.

FILE SYSTEM

The unix file system is organized as a hierarchical tree structure.

The structure of a simple unix file system divided into 4 parts.

1. The Boot block
2. The Super block
3. The Inode block
4. The Data blocks

1) The Boot Block
    - The boot block is located at the beginning of the file system.
    - It can be accessed with minimal code incorporated in the computer's ROM bios.
    - It contains the initial bootstrap program used to load the os.

    It is usually a part of the disk label, aspecial set of blocks containing information on the disk layout.
2) The Super Block
    - It contains statistical information to keep track of the entire file system. Whenever disk manipulation is required,the super block is accessed.

    The Super Block contains the following information:
    1. Size of the file system: This is the storage size of the device.

2. List of storage blocks: The storage space is divided up into a series of standard size blocks.
3. Number of free blocks on the file system.
4. A list of free blocks with their location.
5. Index to next free block on the list.
6. A list of free inodes.

3) The Inode Block

- Information about each file in the file system is a special kernel structure called an inode.
- It contains a pointer to the disk blocks containing the data in the file,information such as type of file,permission bits,the owner and group,file size,file modification and so on.
- The name of each file is listed in the directory the file is associated with.A directory is special type of containing a list of filenames and associated inodes.
- When a user attempts to access a given file by name,the name is looked up in the directory. Where the corresponding inode is found.

An inode for a file contains the following information

- File owerid : It is the numeric id used in the password file to uniquely identify a user on the system.
- Group id (GID) :This identifies a group that can be granted special access by the ower.
- File type :It indicates whether inode represents a file,adirectory,aFIFO,character device.

File access permissions

User access: Access by the person who owns the file.
Group access: Access by the members of a specified group.
Other access: The rest of the world, who are not the owner.

Types of Access

Read access:To be able to read the data stored in file.
Write access : To be able to modify the data stored in file.
Execute access : To be able to request that the system attempt to execute the file.

4) Data Block

- It contain the actual data contained in the files.
- These blocks follows the inode table and occupy most of the storage device space.
- It allotted to one file cannot be allotted to another file, unless the two files are linked.

Advantages

- Data in small files can be accessed directly from the inode.
- Once read operation fectures the inode ,and another read fectures the first data block.
- Larger files can be accessed efficiently .
- Disk can be filled completely ,with little with wasted space.

Disadvantages

- Since inode information is kept separetly from data access of data often requires a long seek when files is initially accessed.
- Orginal file system uses only 512 bytes blocks, an insufficient transfer size.
- Free list quickly becomes scrambled increasing overhead of finding free blocks.

Types of Files

The unixos is built around the concept of the a file system which is which is used to store all of information. Including the operating sysem kernel itself.

There are 4 types of files:

1. Ordinary or Regular files
2. Directory files
3. Device files or special files
4. Hidden files

1. Regular or ordinary files: It can contain text, data,or program information, files cannot contain other files or directories. Instead use the underscore('_') symbol.it is ordinary file.

   An ordinary file can be of 2 types.

   i. Text file
   ii. Binary file

   I. Text File: A text file contains only printable characters.it contains line of character, each line terminated by a newline character .
   Ex:text files include c and Java program ,shell and perl scripts.

   II. Binary File:a binary file contains both printable and unprintable character(0 to 255 ASCII code).
   Ex:binary files include unix command ,object code of c programs ,pictures,sound,and video files.

2. Directory files : Directories are containers or folders that holdfiles,and other directories.Directories can point to other directories which are known as Sub directories.

   The directories generally contains files and other sub directories along with their link information . normally a directory contains 2 pieace of information.

   i. The file name
   ii. A unique identification number.

   When a user creates or remove a file ,the kernel updates the corresponding directory by adding or removing the inode number and filesname associated with the file.

3. Device files :To provide applications, with easy access to hardware devices.UNIX allows them to be used in much the same way as ordinary files. The video screen of your pc,RAM,disk,input ports and other such devices are usually accessed through device file.

   Two types of device files in unix

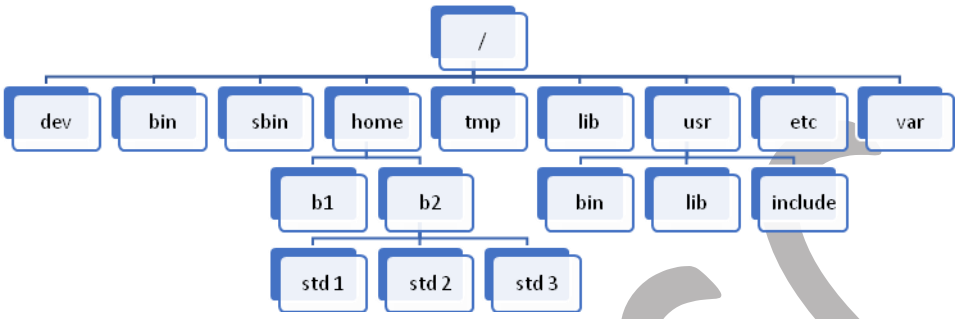
   - Block-oriented :Block-oriented devices which transfer data in blocks.
     EX: Hard disk
   - Character-oriented: Character oriented devices that transfer data on a byte -by-byte basis(eg.modems,printers and networks).

   2 Hidden files :The filename can begin with a dot character is called hidden file.

Types of users

1. The ower of file(user)
2. The users who belong to the same group as the file(group)
3. Everyone else(other)

UNIX Directory Structure



- Directories are organised into a hierarchical structure that fan out like an upside own tree.
- The top most directory is known as root and it is written as (/).
- The roots contains subdirectory and each of these can contains more subdirectory and so on.

Directory	Contents
/	The root directory
/bin	Essential low-level system utilities
/usr/bin	Higher-level system utilities and application programs
/sbin	Super user system utilities (for performing system administration tasks)
/lib	Progarm liberaries (collections of system calls that can be included in programs by a compiler)for low level system utilities.
/usr/lib	Program libraries for higher level user programs
/tmp	Temporary file storage space (can be used by any user)
/home	User home directories containing personal file space for each user.each directory is named after the login of the user.
/etc	Unix system configuration and information files.
/dev	Holds the files /device drivers necessary to operate peri-pherals such as terminals , keyboard,printer,hard disk etc
/var	Contains files that vary in size,mail directories,printer,spool file,logs etc..
/usr/include	Contains standard headed files used by c programs

File access methods

- Owner:

It is the owner of the file.the owner's permissions determine what actions the owner of the File can perform on the file.

- Group:

It gives the name of the group.the groups permissions determine what actions a user,who is a member ofthe group that a file belongs to , can perform on the file.

- Other:

the rest of the world who are not the owner . the permissions for others indicate what action

all other users can perform on the file.

File and Directory related commands

- **Ls command** : listing files

It is used to display all the files and sub directories in a current directory.

Synatx : ls options filenames

Ls options :

-a : list all files including hidden files . Hidden file names start with a dot character.

A single dot (.) refers to the current directory and a double dot (..) refer to the parent Dirctory .

-x : list the content in a row – wise format.

-r : list contents,sorted in reverse alphabetical order.

*t : list all files ending with letter t .

t* : list all filnames starting with letter t .

-u : list the contents based on the access time or using time .

- **Cat command** : it is used to

i. **Creating files :**

cat command can be used to create small files

syntax :

cat options inputfile

Example :

$cat > fruits

Apple

Banana

Orange

<ctrl d> or (ctrl + d)

ii. **Displaying contents of a file :**

cat display only the contents of the file

syntax :

$cat filename

Ex :

$cat > veg

Carrot

Beans

Tomato

< ctrl + d>

$cat fruits veg

Apple

Banana

orange

carrot

Beans

Tomato

iii. **Concatenation of files** :

Cat command can concatenate the contents of two or more files and start Store then in another file.

Syntax :

$cat test1 test2 > test3

Ex :

$cat fruits veg > food

$cat food

Apple

Banana

Orange

Carrot

Beans

Tomato

iv. **Append files :**

Cat command is to append or add data to the contents of the file

Syntax :
$cat >> filname1

Ex :
$cat fruit1
Grapes
(ctrl + d)

Options

-v : displays non-printable ASCII characters.
-n : numbers the lines in the file .
-s : silent (no error messages).

- **Cal command** :

Cal command is used to print the calender of a required month or year

Syntax :

cal { [month] year }

Ex :

$cal 2013 | more

- **Date and time command :**

it displays the current date with time .

syntax :
date options arguments

options :

d : day of the month.
y : last two digits of the year .
H,M& S : hour ,minutes,seconds.
D : date in mm/dd/yy format .
T :time in hh : mm : ss format .
h :month name.
a :day of week.

Ex :
$date
Sat jan 05 15:35:30 1st 2016

- **Who :**

Who command provides the login details of all current users in 3 columns format.

Syntax :
Who options ami

First column shows login names , second column shows the devices names of the terminals and third column shows the login date and time .

Options :

-H :displays headers for the columns .
U :displays more details including idle time ,PID and comments .

PID is the process – ID , which is a unique number identifying a process .
Whoami gives the self login details of a week .

- **Printf : print formatted output command .**

it is used to write formatted output .

syntax :
printf format arguments

- **TTY : terminal command**

It is used to known the name of the device file .

Syntax :
Tty

Ex :

$tty
/dev/pts /1

- **Sttycomman:**

Set teletype. It will change and print the terminal line settings.

Syntax:
Stty[-a/g] [-f device] [settings]

Options:
-a :print, all current setting in human readable form.
-g(save): print all current setting in a stty readable form.
-f:(file): open and use the specified device.

- **Uname : system name command**

It is used to know the name and certain features of the system

Syntax :
Uname options

Options

-v :prints the versions of os .
-a :options displays all deatails of the system .
-m :machine details .

- **Passwd :**

It is used to change password in passwd .

Syntax :
$passwd

- **echo :**

it is usually used in shell scripts to display messages on the terminal

syntax :
echo options arguments

Ex :

$echo I am studing in 2 Bca
→ I am studing in 2 Bca

$echo " I am studing in 2 Bca "

→ I am studing in 2 Bca

- **tput : reposting cursor command**

it is used to control the movement of the cursor on the screen .

syntax :
tput options row_num column _num

- **Bc : calculator command**

there are two types of calculator .

1. **Xcalc :**it is a graphical object which is used only on x- window system and is easy to use .
2. **bc :** it is a text – based command . it behaves both as base calculator and a small language .it can perform all arithmetic operations .

Syntax :
bc arguments

Ex :

```
 $bc       $bc          $bc
   15+5 sqrt(49)10/3
    20                                  7          3
    21                                  22         23
```

- **script command:**

it is a unix utility that records a terminal session. after executing yhe script command it starts recording everything printed on screen including I/p and O/P until exit.by default , all the terminal information is saved in the file typescript, if no argument is given.

Syntax:$script [options] filename

Ex:$script
script started, file is typescript

- **Spell command :**

Read one or more files and print a list of words that may be misspelled .
Simplest a sometimes most convenient of these spelling chekes the spell.
Spell compares words typed in the keyboard or the contents of a text file to those in a built in a directory files and there written all of the words not in the directory to a standard output (which displace screen by default)

Syntax :

$spell file1 file2 file3 > file 4

- **ispellcommand :**

If the directory contains any near missletters (that is words that differ by only a single character , a missing or extra letter , missing space or _

ispell is a spell checking prpgram available for many uniximplementation .

- **ispellfilename :**

r : replace the misspelled word completely .
a : accept the word for the rest of these ispell session .
I : accept the word capitalized as it is in the file and update the provide directory .
U :accept the word and add a lowercase session to the provide directory .

- **Pwd : print working directory command**

Pwd in a command that prints the absolute pathname of your current working directory

Syntax :
Pwd

Ex :
$pwd
/home/rama

- **The home directory :**

When you log into unix ,your current working directory is your user home directory. we can refer to home directory at any times as (tilde)" ~ " **. so ~**std2/play is another way for user std 1 tp specify an absolute path to the directory /home/std2/play ,user std2 may refer to the directory as ~ /play .

The home directory can be found using a shell variable called HOME as shown :
$ echo $HOME
/home/rama

- **Pathnames :**

It is the notation used to point to the particular file or directory

Pathnames is divided in to two
a) Absolute path
b) Relative path

a) Absolute path :

Absolute path all ways starts from root directory (/) .

Ex :
User /jen /person

b) Relative path :

It points to a file or directory relative to your current working directory .

Ex :
Person /art.html

- **Cd-changing the directory**

It is used to move from one directory to another.it uses a pathname as its arguments.

- .(single dot)means current directory.
- ..(double dot)means the parent directory.
- ~(tilde)means your home directory.
- A plain cd without pathname brings you back to your home directory.

Syntax
Cd pathname

Ex :$pwd
/home/rama
$mkdirpartha
$cd partha
$pwd
/home/rama/partha

- **mkdir:making directors**

mkdir command is used to creat new directories.

Syntax:
mkdir optionsdir-names

ex:$mkdirranjitha
$cd ranjitha
Ranjitha$ pwd
/home/ranjitha

- **rmdir:**

It removes one or more directories or subdirectories.

Syntax
rmdirdir-names

ex: $cd\
$rmdirranjitha

SPECIAL TOOLS UTILITIES

Unix (shell) has some special features that are useful for interaction and programming. Before discussing further, we should know about *standard files.*

Standard files are the special files which are the streams of characters seen as input or output by the commands. There are three files representing three streams, each associated with a default device.

- Standard input(stdin)
- Standard output(stdout)
- Standard error(stderr)

Standard input: is a file or a stream representing input. Keyboard is the Standard input file .But we can use redirection with the **<** symbol or pipeline with the symbol **|** to specify other files as input files.

Standard output: is a file or a stream representing output. Terminal screen (monitor) is the Standard output file .But we can use redirection with the **>, >>** symbol s or pipeline with the symbol **|** to specify other files as output files.

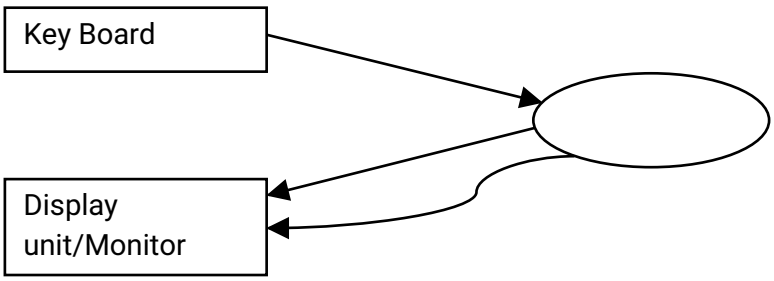
Standard error: it is a file or stream representing error messages that emerge from the command or the shell. This is also connected to display screen.

Each standard files are represented by a number called ***file descriptor***, they are

0→standard input

1→standard output

2→standard error



I/O Redirection: Redirection is a feature that reassigns the standard input and output files. For input redirection (input source) < operator is used, so that input is not from keyboard, but from another file. And for output redirection > or>> operators are used, which outputs to other files but not to the terminal.

Eg: $ banner < file1 [it takes input from file1 and displays it on monitor]

$ echo " good morning" > file2 [sends output to file2]

Note : 1. > operator is used for *overwriting*

2. >> operator is used for *appending*

Piping : it is a feature available in Unix where two or more commands can be combined together. Here output of first command is treated as input for second command. The symbol used for pipe is **| .** A sequence of commands using one or more pipes is called a pipeline. The advantage of pipe is that, a command line containing many commands can be executed as a background process.

For example: $ cat stdlist | sort

[here output of the command cat is directly send as input to sort command]

Filters: Filters are UNIX commands that take their inputs from the standard input file, process it and send it to the standard output file. A filter is actually a program that can read directly from files whose names are provided as arguments, process it or filer it and sends the result to the standard output, unless a redirection or a pipeline symbol is used. Some commonly used filter commands are :

- Θ cat
- Θ head
- Θ tail
- Θ cut
- Θ paste
- Θ uniq
- Θ tr
- Θ wc
- Θ sort
- Θ grep

NOTE: All filter commands does not change the contents of files but changes will appear only while displaying. The changes can be stored permanently by using redirection facility

cat: This command is a multipurpose filter command used for creating file, displaying contents of file, copy file, append to file and to concatenate two or more files.This command is already discussed in earlier chapter.

Options

-n : displays each line with its line number
-b : same as -n but it ignores blank lines
-u : the output is not blubbered
-s : silent about nonexistent files

head: This command is used to display the first few lines of a file. The user can specify the number of lines to be displayed by using the **n** argument. The default number of lines that are displayed is 10.

Syntax: head [-options] filename

Selected options:

-cn : print the first n characters of the file

-n : print the first n lines the default is 10

$ head dolls *[displays first 10 lines of dolls file]*

$ head -5 dolls [*displays first 5 lines from dolls file*]

$ head -c5 dolls [*displays first 5* characters (including white space) *from dolls file*]

tail: This command is used to display the last few lines from a file. The user can specify the number of lines to be displayed by using the **n** argument. The default number of lines that are displayed is 10.

Syntax: tail [-options] filename

Selected options:

-cn : print the last n characters of the file

-n : print the last lines the default is 10

+n : prints the lines from nth line to end of file

$ tail dolls *[displays last 10 lines of dolls file]*

$ tail -5 dolls [*displays last 5 lines from dolls file*]

$ tail –c5 dolls [displays last 5 characters (includes whitespace)]

usage

ls | tail -15 [displays last 15 lines of file listing]

head abc | **tail** -5 [prints lines 6 through 10 of file abc]

cut: This command is used to cut the file vertically i.e to copy (display) the specified columns to the standard output file. This command can be used to cut out parts of a file . it takes filenames as arguments. It can cut columns or fields in a file. ***However it does not delete the selected parts of the file***

Options

-b list : to specify the bite position

-c list : to extract the characters as specified position in the list . a dash (–) is used to indicate the range and comma(,) is used to separate items in the list.

-f list : to extract the fields as specified position in the list. a dash (–) is used to indicate the range and comma(,) is used to separate items in the list.

-dc : the character following the d is field delimiter and this option is used only with -f option

$ cut -c 10-15 stdlist

$ cut -d":" –f 1,3 stdlist

paste: This command is used to combine two or more files vertically. The lines of the input files must be in corresponding order before the paste utility can be used. If there is no corresponding field the command puts the delimiter and leaves it blank.

Options

-dc : to specify the delimiter character used in combined file

$ paste file1 file2

$ paste -d":" file1 file2

Note: paste expands table's (file's) width by increasing the number of columns. But cat expands a table's length by increasing the number of rows.

uniq: This command is used to remove the repeated lines from a file. The file must be in sorted order before using this command.

Options

-c: precedes each line in o/p with count of number of times the line occurred in i/p file

-d: selects only one copy of the duplicate entries

-u:selects lines which are not repeated

-i: ignore case

$ uniq names
$ uniq -c names
$ uniq -u names

tr: This command is used to translate characters. This command manipulates individual characters in a line. It takes input only from standard input. The syntax is:

tr [options] [expression 1] [expression 2] < input file

Options

-c: it complement the set of characters in the expression

-d: deletes specified range of characters

-s:sequeezes multiple occurrences of a character into a single character

$ tr "[a-z]" "[A-Z]" < fruits
$tr -d ':' <file1
$ tr -s ' ' < file1
$ tr -c 'a-zA-Z' '*' <file2 *[it matches all the characters that are not found in the first string and replaces the matched characters with character in the second string , here all spaces.dots and newline character are replaced with hyphens]*

wc: this command is used to count the number of lines ,words and characters in one or more files. It gives 4 column output, where the first column indicates number of lines in the file, second column indicates number of words, the third indicates number of characters and the last column indicates the file name.

Options

-c: it prints the counts of characters

-w: it prints the counts of words

-l : it prints the counts of lines

$ wc fruits
$ wc -c fruits
$wc -w fruits
$wc -l fruits

sort: this command is used to sort the contents of a file. Using this utility one or more files can be sorted in alphabetic or numeric order, and also it can reorder the file based on one or more of the fields.

Options

-d : sorts according to dictionary and ignores punctuation.
-f:ignores caps while sorting
-m : merges two or more given sorted files tnto single sorted file
-n : sorts according to numeric order
-r : sorts in reverse order
-u : removes duplicates and displays unique values
-o : specifies output file name
-t : specifies the delimiter used in the file
-k fieldno : sorts the data according the field number specified by the fieldno

$ sort file1
$ sort -o sfile file1
$ sort -r file1

grep: this command is used to searches its input for a pattern and displays lines containing the pattern (***G****lobally search a* **R**egular ***E****xpression and* ***P****rint it*) . The syntax is:

grep [options] [pattern] [file name].....

the pattern can be a string with single or multiple words or special characters. It is advised to enclose the pattern with single codes.

Options

-e : pattern : to specify multiple search pattern.
-i : prepares case while searching.
-n: used to display records (lines) along with line number
-v: prints those lines which do not match the pattern
-c: displays only the count of lines which contains pattern
-l: displays only the names of files containing the pattern

$ grep “bca” stdlist
$ grep -i “bca” stdlist
$ grep -n “BCA” stdlist

grep family contains three commands namely grep, egrep(extended grep) and fgrep(fixed grep)

egrep [Extended Global Regular Expressions Print] is a pattern searching command which belongs to the family of grep functions. It works the same way as ***grep -E*** does. It treats the pattern as an extended regular expression and prints out the lines that match the pattern. If there are several files with the matching pattern, it also displays the file names for each line.

The *egrep* command used mainly due to the fact that it is faster than the grep command. The egrep command treats the meta-characters as they are and do not require to be

escaped as is the case with grep. This allows reducing the overhead of replacing these characters while pattern matching making egrep faster than *grep* or *fgrep*.

Syntax:

egrep [options] '<*regular expression*>' <*filename*>

Some common options are:

-c for counting the number of successful matches and not printing the actual matches

-i to make the search case insensitive, -n to print the line number before each match printout

-v to take the complement of the regular expression (i.e. return the lines which *don't* match),

-l to print the filenames of files with lines which match the expression.

$ egrep 'bca|bcom' studfile
$ egrep –i 'bca|bcom' studfile
$ egrep –f pfile studfile where pfile contains pattern

fgrep: This command searches for fixed-character strings in a file or files. "Fixed-character" means the string is interpreted literally — metacharacters do not exist, and therefore regular expressions cannot be used.

fgrep is useful when you need to search for strings which contain lots of regular expression metacharacters, such as "$", "^", etc. By specifying that your search string contains fixed characters, you don't need to escape each of them with a backslash.

Running fgrep is the same as running grep with the -F option.

$ egrep 'bca
Bcom
bsc' studfile
$ egrep –i 'bca
bcom' studfile

Difference between egrep and fgrep

1. Both egrep and fgrep are derived from the base grep command. The "egrep" stands for "extended grep" while the fgrep stands for "fixed-string grep."

2.An egrep command is used to search for multiple patterns inside a file or other kind of data repository while frgrep is used to look for strings.

3.The term "egrep" is commonly expressed as "grep-E" while "fgrep" is encoded as "grep-F."

4.The egrep command allows the use of extended regular expressions while grep only searches for the matching word or term that the user specified in the command. The frep doesn't recognize or understand regular or extended regular expression.

5.Compared to the other search commands, the search process for fgrep is very fast since it is only concerned with the provided search word.

6.The egrep command usually uses operators in order to yield a more progressive or specific search research. A plus sign and the question mark deal with single, regular expressions or search terms. On the other hand, vertical bars and parentheses are used for multiple, regular expressions with opposing functions. The vertical bar separates the expressions while the parentheses operator groups them.

tar

The Linux 'tar' stands for tape archive, is used to create Archive and extract the Archive files. tar command in Linux is one of the important command which provides archiving functionality in Linux. We can use Linux tar command to create compressed or uncompressed Archive files and also maintain and modify them.

Syntax:
tar [ptions] filename

Options:

-c : Creates Archive
-x : Extract the archive
-f : creates archive with given filename
-t : displays or lists files in archived file
-u : archives and adds to an existing archive file
-v : Displays Verbose Information
-A : Concatenates the archive files
-z : zip, tells tar command that create tar file using gzip
-j : filter archive tar file using tbzip
-W : Verify a archive file
-r : update or add file or directory in already existed .tar file

An *Archive file* is a file that is composed of one or more files along with metadata. Archive files are used to collect multiple data files together into a single file for easier portability and storage, or simply to compress files to use less storage space.

Examples:

Creating an uncompressed tar Archive using option -cvf : This command creates a tar file called file.tar which is the Archive of all .c files in current directory.

```
$ tar cvf file.tar *.c
```

Extracting files from Archive using option -xvf : This command extracts files from Archives.

```
$ tar xvf file.tar
```

gzip compression on the tar Archive, using option -z : This command creates a tar file called file.tar.gz which is the Archive of .c files.

```
$ tar cvzf file.tar.gz *.c
```

Extracting a gzip tar Archive *.tar.gz using option -xvzf : This command extracts files from tar archived file.tar.gz files.

```
$ tar xvzf file.tar.gz
```

Creating compressed tar archive file in Linux using option -j : This command compresses and creates archive file less than the size of the gzip. Both compress and decompress takes more time then gzip.

```
$ tar cvfj file.tar.tbz example.cpp
```

untar single tar file or specified directory in Linux : This command will Untar a file in current directory or in a specified directory using -C option.

```
$ tar xvfj file.tar
```

Untar multiple .tar, .tar.gz, .tar.tbz file in Linux : This command will extract or untar multiple files from the tar, tar.gz and tar.bz2 archive file. For example the above command will extract "fileA" "fileB" from the archive files.

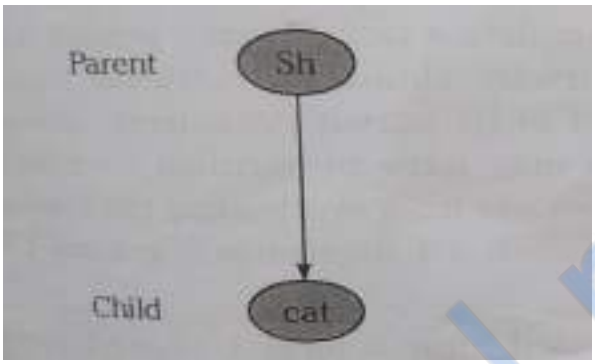
```
$ tar xvf file.tar "fileA" "fileB"
```

Chapter 4

Process management

Process :- Process can be defined as a program under Execution. Unix runs many programs at the same time by using Round-robin Scheduling algorithm.

Shell process (sh):-



shell creates a process for executing the catcommand.

The shell process(sh) is a parent process and the cat process is a child process. As long as process is running, it is alive. After completing the job, it becomes inactive and is said to be dead.

Parent and child process:-

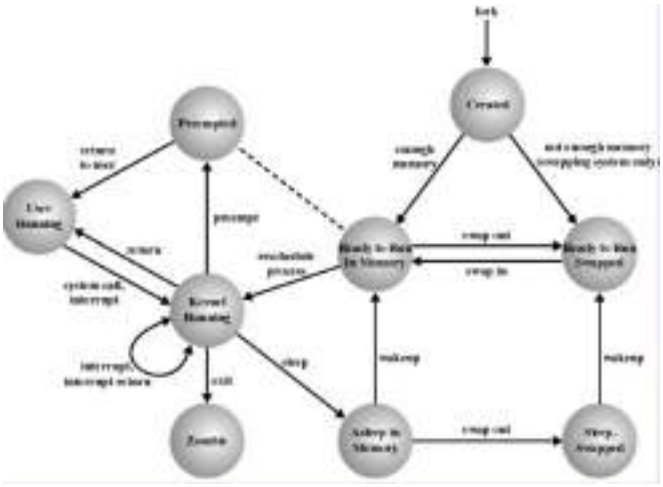
In Unix one process can generate another process. The process which generates another process is called Parent process. Newly generated process is called child process.

The parent can have one or more children

Eg:- $ cat fruits | grep orange fruits

The shell creates two child process cat and grep simultaneously.

Process state:-



Process state diagram

The main five-state model of any operating system are:-

1) New : The process is created.

2) Running: The process is being executed.

3) Waiting : The process is waiting for some required resources like input and output devices

4) Ready : The process is waiting to be allocated to a processor. The process comes to this state immediately after creation.

5) Terminated: A process terminates (exited) after finishing its execution.

Unix process state (system process) :-

In Unix there are 7 Process states same as five-state model

UNIX Process States are :-

1) Created:- Just created but not yet ready to run.

2)

i. Ready (in Memory):- Ready to run as soon as kernel schedules it.

ii. Ready (Swapped):- Ready to run, but needs to be swapped into memory.

3)

i. Asleep (in memory):- The process is blocked and waiting for an event in

memory.

ii. Asleep (swapped):-The process is swapped out and waiting for an event on the disk.

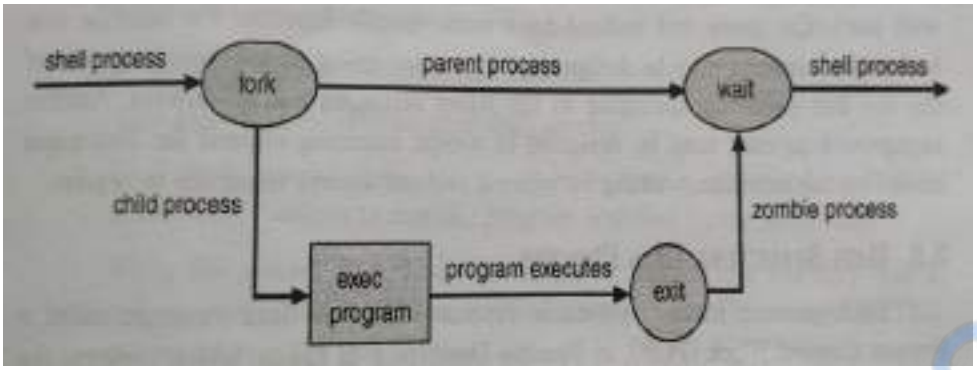
4) Running (kernel):- Executing in kernel mode.

5) Running (user):- Executing in user mode.

6) Zombie:- A zombie process in UNIX is a process that has terminated, but whose parent has not waited for it. May be a parent process exited without waiting for it to terminate

7) Pre-empted:- is a process returning from kernel mode to user model, since it in preempted by kernel, to schedule another process.

Unix process creation:-



★ Parent is the orginal process.

★ new process is called child.

★ child cantain same code, same data of its parent.

★ the parent can either wait for child to complete , or continue executing in parallel with the child.

★ child is created by system call **fork( ).**

★ fork( ) returns 0 (zero) in child process.

★ fork( ) returns **PID** of new child in parent process.

★ fork( ) system call is not successful, it returns **-1**.

★ Resource sharing: a process needs certain resource like CPU time, Memory, I/O devices etc.

★ **exec( )** system call is used after fork( ), to start another different program.

★ **ps** command is used display a listing of currently active processes in the system.

jobs in foreground and background :-

The unix supports both multi user as well as multi processsing.

There are 3 types of processses.

1.inter-active (foreground)process:-

All the processses creted by the user using the shell and atached to the terminals are called fore ground process.

-> when user pass a command to shell ,the shell passes ,rebuilds and sends it to the kernal for execution.

-> duiring the execution of one process the user should wait for the kernal until compilatiton of that process.

-> no further command can run during execution of older one.

2.non-interactive(background)process:-

The process which can without using the terminal are know as background process.

-> background process take input from a file ,process them without holding up the terminal and write output on other file.

-> these commands are ends with '&' sign .

i.e. $sort -o std.sort std.sort&

-> for ex:- sorting and searching of files/content from long files.

limitatitons:-

-> background process don't report their successful or unsuccessful executiton .

-> pid is used to find execution state .

-> too many background processes can effect system effeciency .

-> nohup (no hang up) which avoids deadlocking of ome background process during unusual logouts.

runaway process :- when background process (using & at the end) and logout without closing or killing the process .such a processs is called runaway process.

#chnanging process priority with nice command

Nice command :- It is used to change or set the priority of a process

syntax: $nice -value cat filename

★ The default priority of a process in **unix** is **20**

★ the value range from 0 to 39, in linux -9 to 20 .where 0 is high and 39 is lower value.

★ the default value of reduction is 10.

★ the priority of a process can be increased only by adminstrator using double minus(--).

eg:- $nice --15 cat last.txt

★ The priority of a process can be made lower using the **nice** command.

★ **for example:** if a process is already running and using a lot of cpu time; then it can be reniced.

I.e. $nice cat last. txt

$nice -10 cat last. txt

$nice --15 cat last. txt

Daemon Processes:-

Daemon processes are processes that are constantly running without using associated terminal or login shell, and keeps waiting for some instructions either from the system or user and immediately starts performing that task.

The characteristic features of daemon processes are as follows:

★ They start running as soon as the system is initialized.

★ The lifetime of the daemons is as long as the system is running.

★ The daemons cannot be killed prematurely.

★ init process is one of the first programs loaded, after bootstrapping.

★ The scheduler process is used to manage and schedule other processes.

★ Process vhand, which stands for virtual memory handler, is loaded into the system to swap the active processes between memory and disk, when they are waiting for CPU time.

★ Process bdflush is responsible for disk I/O.

Process Termination:-

There are situations when the user has to terminate a process prematurely. Several reasonS are possible for process termination such as:

★ The terminal hangs.

★ user logs off .

★ Program execution has gone into endless loop.

★ Error and fault conditions.

★ Time limit exceeded.

★ Memory unavailable

★ I/O failure.

★ Data misuse.

★ system performance slow due to too many background processes running.

★ Operating system intervention (for example to resolve, a deadlock).When a UNIX process is terminated normally, it

★ Close all files .

★ save usage status.

★ Makes init process the parent of live children.

★ Changes run state to zombie.

Communication commands:-

1) kill command:

Termination of a process forcibly is called killing.

★ Background process can be terminated by using kill command

★ A foreground process is terminated using **del key** or **break key .**

★ PID is used to select the process.

Syntax:- $ kill PID .

★ More than one process can be killed using a single kill command.

★A special variable **$!** (That holds PID of last background process) used to kill last background process.

★A special variable **$$** (That holds PID of current shell) used to kill current shell.

★ **$kill 0 :** to terminate all process of a user.

★ **$kill -9 0 :** to terminate all process of a user including the login shell.

2) mesg command:- is used to change the write permission of a user.

Syntax:- $ mesg y #grant the write permission.

$ mesg n #denise the write permission.

$ mesg #current write status.

★ If user doesn't want to be disturbed, he can deny the write permission.

★ But super can send message irrespective of permission.

3) write command:- allows two way communication between two users who are currently logged in and have given write permission.

Syntax:- $write username

★ The user **A** can send messages to user **B** who is logged in.

★ Then user **B** get message with a beep sound, then **B** replies.

★ But both the users must be logged in.

4) finger command:- is similar to who command, it shows current loggin details and shows asterisk symbol for those who have permission to accept messages.

Syntax:- $ finger

5) wall command:- wall stands for **write all**. Wall command is used only by the super user to send messages to all users on the system.

★ It is also known as broadcasting a message to all users , irrespective of there permissions.

Syntax:- $ wall

★★★★★

File access permission (FAP)

The files that users create will not be accessible to other users.

However UNIX allows to explicitly make their files available to others.

Every file of folder in UNIX hasacces permission.

There are three types of permission:

- Read access
- Write access
- Execute access

Read permission allows users to view the contents of a file and to copy it

writepermission allows users to modify the files

Execute permission allows users *to run the file, if it is executable.*

Types of users

- *The owner of the file(user)*
- *The users who belong to the same group as the file(group)*
- *Everyone else (other)*

File permissions notation

- *Textual representation like "-rwxr- -r- -"*
- *Numeric(octal) representation like "644"*

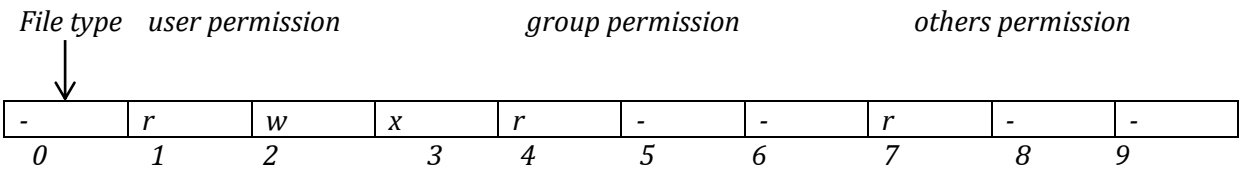
Textual representation

It is used in UNIX long directory listing.it is consists of 10 character. The first character shows the file type. Next 9 characters are permission, consisting of 3 group: Owner, group , others.

Special tools and utilites

Each group consists of threesymbols : r w x (in this order)

If some permission is denied, then a dash "-"is used instead.



Numeric representation:

If a numeric representation is used (like in chmod command) then it is in the octal format(base of 8)

Digits involved are 0 to 7. Every octal digit combines read, write and execute permission together,respective access rights for owner, group and other are the last three digits of the numeric file permissions representation.

Octal digits	*Text equivalent*	*Binary value*	*Meaning*
0	*---*	*000*	*All type of access are denied*
1	*--r*	*001*	*Execute access is allowed only*
2	*-w-*	*010*	*Write access is allowed only*
3	*-wx*	*011*	*Write and execute access are allowed*
4	*r--*	*100*	*Read access allowed only*
5	*r-w*	*101*	*Read and execute access are allowed*
6	*Rw-*	*110*	*Read and write access are allowed*
7	*rwx*	*111*	*Everything as allowed*

Head :-*displaying first few lines.*

The head command display the first few lines of one or more file .this command is used to verify the contents of a file. Without any option. By default it display the first 10 lines of the file

Syntax*: head [option] file name selected*

Options:

-cn: print the first n characters of the file.

-n: print the first n lines. the default is 10.

Special tools and utilites

Tail : displaying last few lines

The tail command displays the last few lines of a file .by default, it displays the last 10 lines of the file. It connot be used for multiple files

- Tail bcalist displays the last 10 lines of the file.
- Tail -5 bcalist displays last 5 lines of the bcalist.
- Tail +8 bcalist display all the lines starting from line number 8 upto end of file.

WC:

Wc command is a filter used to count the number of lines , words and character of one or more files

- It take one one or more filename as its arguments
- It gives the output in 4 columns:,
  1 is number of lines
  2 is number of words
  3 number of characters and the lost column indicates the filename

Wc options:

-c : prints the number of bytes.

-l : prints the line count.

-L : prints the length of the longest line.

-m: prints the number of character .

-w: counts words delimited by white space character or new line character.

TR: translate command

Tr commands is used to translate character .it is a filter that manipulates individual character in a line

Syntax :tr[options][expression1][expression]<input file

- Tr command take twoarguments .eacharguments may be a character or a string of the character.
- The behaviour of tr command can be explained with an

- example:

  $cat list1
  A friend in need is a friend indeed.
  $tr 'frie' FRI'<LIST1
  AFRIInd InnIId Is a FRIIndInddIId.
  $tr 'fri' 'FRIE' < list1
  A FRIend In need Is a FRIend Indeed.

  Cut command :

  Cut command splits files vertically .this command can be used to extract the required fields or columns from the file
  Example:

  $cut –c 1-3 stdlist
  001
  010
  007
  008
  003

  Cutoptions :

  - -b list : the list after –b specifies the byte positions
  - -c list : the list following -c specifies the character positions such as –c 1-7 which passes the first 7 characters of each line .
  - -f list : the list following -f is a list of fields , separated in the file by a delimiter character (-d)

Sort :

The sort command is used to sort the information in a text file in ascending or descending order .it is also used to merge sorted files it takes zero, one or more filename as arguments

Sort option :

Sort command has several options .some of the most command options are given below.

- **-d :** sorts according to dictionary and ignores the punctuation.
- **-f :** ignores caps while sorting .

Special tools and utilites

- **-n** : sorts according to numeric order.
- **-r** : sorts in reverse order.
- **-n** : removes duplicates and display unique values.
- **-0 outfile** : place the sorted output in the file out file.

Grep:- searching for a pattern

UNIX consists of a special family of commands for handling search requirement, known as the grep family

Globally searcha regular expression and print it

Syntax :$grep [options] [pattern][filename]….

Grep options :

- **-c :-** this is the count option.this option count the records or lines that contain the specified pattern in all the files given as arguments. It displays only the count.
- **-i :-** generally grep differentiates between uppercase and lowercase Alphabets .this option ignores case and searches for all patterns specified.
- **-l :-** when -l option is used,it display only the filenames containing the specific pattern . both options –I and –i can be used together as –il
- **-v :-** this option is known as inverse option .it prints or displays only those lines or records that do not match the specified pattern.
- **-n :-** print the matched line and its line number.

Egrep (extended grep) :-

Extendable global regular expressions Print is a pattern searching print is a pattern searching command which belong to the family grep function.

It works the same way as grep –e das it treats the pattern as an extched regular expressions and prints out the lines that match the pattern.

It there are several files with the matching pattern it also displaythe file name for each line.

The e-grep command used mainly due to the fact that it is faster than grepcommand.the grep command treats the meta character they are

and donot required to be escape as the case with grep .this allows reducing the overhead of replacing these characters while pattern

Matching making e-grep faster than grep or e-grep

Syntax:egrep[option]<regular expr><filename>

Egrepcommand options are:

-c :for counting the number of successful match and not printing the actual matches.

-I :to make the search case insensitive .

-n : to print the line number before each match print out.

-v : to take the compliment of the regular expressions (return the lines which does not match)

-l : to print the file names of files with lines which match the expression.

f-grep

the command when you need to search for string which contain lots of regular expressions meta character $ and ^etc. by specifying that your search and strings contains fixed character you donot need to escape each of then with a (back slash \)

Special tools and utilites

Special tools and utilites

Special tools and utilites

Special tools and utilites

Chapter-6

Shell Programming

The shell provides you with an interface to the UNIX system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

A shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of shells, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Shell Prompt:

The prompt, which is called command prompt, is issued by the shell. While the prompt is displayed, you can type a command.

The shell reads your input after you press Enter. It determines the command you want executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words.

Following is a simple example of **date** command which displays current date and time:

$date

Thu Jun 2508:30:19 MST 2009

Shell Types:

In UNIX there are two major types of shells:

1. The Bourne shell. If you are using a Bourne-type shell, the default prompt is the $ character.
2. The C shell. If you are using a C-type shell, the default prompt is the % character.

There are again various subcategories for Bourne Shell which are listed as follows:

- Bourne shell (sh)
- Korn shell (ksh)
- Bourne Again shell (bash)
- POSIX shell (sh)

The different C-type shells follow:

- C shell (csh)
- TENEX/TOPS C shell (tcsh)

The original UNIX shell was written in the mid-1970s by Stephen R. Bourne while he was at AT&T Bell Labs in New Jersey.

The Bourne shell was the first shell to appear on UNIX systems, thus it is referred to as "the shell".

The Bourne shell is usually installed as /bin/sh on most versions of UNIX. For this reason, it is the shell of choice for writing scripts to use on several different versions of UNIX.

Shell Scripts:

The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by a pound sign, #, describing the steps.

There are conditional tests, such as value A is greater than value B, loops allowing us to go through massive amounts of data, files to read and store data, and variables to read and store data, and the script may include functions.

Shell scripts and functions are both interpreted. This means they are not compiled.

It supports less features. It supports input and output redirection operators.

Example Script:

Assume we create a **test.sh** script. Note all the scripts would have **.sh** extension. Before you add anything else to your script, you need to alert the system that a shell script is being started. This is done using the shebang construct. For example:

```
#!/bin/sh
```

This tells the system that the commands that follow are to be executed by the Bourne shell. It's called a shebang because the # symbol is called a hash, and the !symbol is called a bang.

To create a script containing these commands, you put the shebang line first and then add the commands:

```
#!/bin/bash
pwd
ls
```

Shell Comments:

You can put your comments in your script as follows:

```
#!/bin/bash

# this is sscasc
# Copyright (c)
sscasc.com #
Script follows
here: pwd
ls
```

Now you save the above content and make this script executable as follows:

```
$chmod +x test.sh
```

Now you have your shell script ready to be executed as

follows: $./test.sh

Extended Shell Scripts:

The shell is, after all, a real programming language, complete with variables, control structures, and so forth. No matter how complicated a script gets, however, it is still just a list of commands executed sequentially.

Following script use the **read** command which takes the input from the keyboard and assigns it as the value of the variable PERSON and finally prints it on STDOUT.

```
echo "What is
your name?"
read PERSON
echo "Hello,
$PERSON"
```

Here is sample run of the script:

```
$./test.sh
What is your
name? vvfgc
Hello, vvfgc
```

Variables:

Variable is value that always changes during execution of a program. It is an integral part of shell programming. They provide the ability to store and manipulate information.

There are 2 types of variables. They are

- Environment variables
- User defined variables

Environment variables:

These variables are the part of the system and these are created and maintained by the syatem itself. These variables always in capital letters only.

Variable	meaning
PS1	this is first prompt setting in Unix ($)
PS2	this is second prompt setting in Unix (>)
PATH	whether we are used absolute or relative path.
HOME	it stores the current root directory.
LOGNAME	it stores the login name of the user.

User defined variable:

Variables are defined as follows::

variable_name = variable_value

For example:

NAME = "sscasc"

Above example defines the variable NAME and assigns it the value "sscasc". Variables of this type are called scalar variables. A scalar variable can hold only one value at a time.

The shell enables you to store any value you want in a variable. For example:
VAR1="ssczsc "
VAR2=100

Accessing Values to variables:

To access the value stored in a variable, prefix its name with the dollar sign ($):

For example, following script would access the value of defined variable NAME and would print it on STDOUT:

```
NAME="v
vfgc "
echo
$NAME
```

This would produce following value:

Output: vvfgc

Read-only Variables:

The shell provides a way to mark variables as read-only by using the **"read only"** command. After a variable is marked read-only, its value cannot be changed.

For example, following script would give error while trying to change the value of NAME:

```
NAME="vvfg
c "readonly
NAME
NAME="Qadiri"
```

This would produce following result:

/bin/sh: NAME: This variable is read only.

Unsetting Variables:

Unsetting or deleting a variable tells the shell to remove the variable from the list of variables that it tracks. Once you unset a variable, you would not be able to access stored value in the variable.

Following is the syntax to unset a defined variable using the **unset**

command: unset variable_name

Above command would unset the value of a defined variable. Here is a simple example

```
NAME="v
vfgc"
unset
NAME
```

echo
$NAME

Above example would not print anything. You cannot use the unset command to **unset** variables that are marked **readonly**.

Read command

This command is used to take the input from the user.

Syntax: $read var1 var2 var3 ….. var n

Syntax: $ read var1

The variable used along with the read command need not be preceded by the

$ symbol. Ex: clear echo “enter ur name” read name echo “hello

$name”

Output: enter urname :vvfgc

Hello vvfgc

Expr command

This command is used to perform mathematical caluculations.

Syntax: ` expr operand1 operator operand2 `

Where (`) this symbol is known as grep symbol. There is always must a space

between symbol and the expr command. Ex: clear echo “enter 2 numbers”

read a b

echo “ sum of 2 numbers is ` expr $a + $b ` “
echo “ sub of 2 numbers is ` expr $a - $b ` “

echo “ product of 2 numbers is ` expr $a * $b

` “ echo “ quotient of 2 numbers is ` expr $a /

$b ` “ Note:

In unix multiplication purpose we use the symbol of **"*"** because only * is wild card character.

Test operator [numerical test]

Arithmetic Operators:

There are following arithmetic operators supported by Bourne Shell.

Assume variable "a" holds 10 and variable "b" holds 20 then:

Show Examples

Operator	Description	Example
+	Addition - Adds values on either side of the operator	`expr $a + $b` will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
*	Multiplication - Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/	Division - Divides left hand operand by right operand	`expr $b / $a` will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
=	Assignment - Assign right operand in left operand	a=$b would assign value of b into a
= =	Equality - Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!=	Not Equality - Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

It is very important to note here that all the conditional expressions would be put inside square braces with one spaces around them, for example [$a == $b] is correct where as [$a==$b] is incorrect.

All the arithmetical calculations are done using long integers.

Relational Operators:

Bourne Shell supports following relational operators which are specific to numeric values. These operators would not work for string values unless their value is numeric.

For example, following operators would work to check a relation between 10 and 20 as well as in between "10" and "20" but not in between "ten" and "twenty".

Assume variable “a” holds 10 and variable “b” holds 20 then:

Show Examples

Operator	Description	Example
-eq	Checks if the value of two operands is equal or not, if yes then condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand, if yes then condition true. becomes true.	[$a -lt $b] is
-ge	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	[$a -le $b] is true.

It is very important to note here that all the conditional expressions would be put inside square braces with one spaces around them, for example [$a <= $b] is correct where as [$a <= $b] is incorrect.

Boolean Operators (or) logical operators:

There are following Boolean operators supported by Bourne Shell.

Assume variable “a” holds 10 and variable “b” holds 20 then:
Show Examples

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical OR. If one of the operands is true then condition would be true.	[$a -lt 20 -o $b -gt 100] is true.
	This is logical AND. If both the operands are	[$a -lt 20 -a $b -gt 100] is

-a	true then condition would be true otherwise it f alse. would be false.	

String test Operators

There are following string operators supported by Bourne Shell.

Assume variable a holds "abc" and variable b holds "efg" then:

Show Examples

Operator	Description	Example
=	Checks if the value of two operands is equal or not, if yes then condition becomes true.	[$a = $b] is not true.
!=	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero. If it is zero length then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero. If it is non-zero length then it returns true. Check if str is not the empty string. If it is str empty then it returns false.	[$a] is not false.

File Test Operators

There are following operators to test various properties associated with a Unix file.

Assume a variable **file** holds an existing file name "test" whose size is 100 bytes and has read, write and execute permission on:

Show Examples

Operator	Description	Example
-b file	Checks if file is a block special file if yes then condition becomes true.	[-b $file] is false.
-c file	Checks if file is a character special file if yes then condition becomes true.	[-b $file] is false.
-d file	Check if file is a directory if yes then condition	[-d $file] is not true.

	becomes true.	
-f file	Check if file is an ordinary file as opposed to a directory or special file if yes then condition becomes true.	[-f $file] is true.
-g file	Checks if file has its set group ID (SGID) bit set if yes then condition becomes true.	[-g $file] is false.
-k file	Checks if file has its sticky bit set if yes then condition becomes true.	[-k $file] is false.
-p file	Checks if file is a named pipe if yes then condition becomes true.	[-p $file] is false.
-t file	Checks if file descriptor is open and associated with a terminal if yes then condition becomes true.	[-t $file] is false.
-u file	Checks if file has its set user id (SUID) bit set if yes then condition becomes true.	[-u $file] is false.
-r file	Checks if file is readable if yes then condition becomes true.	[-r $file] is true.
-w file	Check if file is writable if yes then condition becomes true.	[-w $file] is true.
-x file	Check if file is execute if yes then condition becomes true.	[-x $file] is true.
-s file	Check if file has size greater than 0 if yes then condition becomes true.	[-s $file] is true.
-e file	Check if file exists. Is true even if file is a directory but exists.	[-e $file] is true.

Control statements

The ability to control the flow of execution of program is known as control statements.

The different types of control structures are

- Sequence control structure
- Selection control structure
- Looping control structure

Selection control structure

This type of instruction allows the shell script to be executed depending on the condition.

There are mainly 4 types of decision making instructions. They are

- if-then-fi statement
- if-then-else-fi statement
- if-then-elif-then-else-fi statement case-esac statement **if-then-fi statement:**

in this first check the condition. That condition is true then only the if block statements will be executed otherwise the cursor transfer outside the if condition.

Syntax:	**ex:**
if [condition]	if [$a –gt 18]
then	then
statements	echo “eligible for vote”
fi	fi

if-then-else-fi statement:

in this first check the condition. That condition is true then only the if block statements will be executed otherwise the else block statements will be executed.

Syntax:	**ex:**
if [condition]	if [$a –gt $b]
then	then
statements	echo “a is larger than b”
else	else
statements	echo “b is larger than a”
fi	fi

if-then-elif-then-else-fi statement:

in this first check the condition. That condition is true then only the if block statements will be executed otherwise the cursor checks the next condition then the second

condition will be true then inside that statements will be executed and so on. If any conditions were not true then the else block statements will be executed.

Syntax:

```
if [ condition 1 ]
]
then
statements
elif [ condition 2]
then
statements
else
statements
fi
```

ex:

```
if [ $a –gt $b –a $a –gt $c
then
echo ”a is larger”
elif [ $b –gt $c ]
then
echo “ b is larger”
else
echo “c is larger”
fi
```

Case-esac statement:

Syntax: case $variable in

```
  [match1] )statements ; ;
  [match2]   statements ; ;
  [match3]   statements ; ;
:                :
:                :
     *)          statements ; ;
esac
```

- here match1,match2 etc are the case labels.
- When a case statement is evaluated the value of variable is matched in any one of the choices.
- When a match is found then shell executes that corresponding match statements.
- The pair of semicolon at the end of every choices. It identifies break.
- *) indicates default class. Ex: clear echo “enter a character”

readch

```
case $ch in

[a-z]) echo “entered character is lowercase letters” ; ;

[A-Z] echo “entered character is uppercase letters” ; ;

[0-9] echo “entered character isdigit” ; ;

  *)  echo “invalid choice” ;;

esac
```

Looping Control statements

In this all statements are executed repeatedly again and again as long as condition is true. This is also known as repetition or iteration.

Shell allows various types of looping. They are

- While loop
- Until loop
- For loop

While loop:

This is the pretested loop or entry controlled loop. In this first check the condition, if that condition was true then control enter inside the loop otherwise control transferred outside the loop.

Syntax:

```
while [ condition ]

do

Statements
done
```

ex:

```
while [ i –le 10 ]

do

echo “$i”
        i =` expr $i + 1 `

done
```

until loop:

This is also pretested loop or entry controlled loop. In this first check the condition, if that condition was false then control enter inside the loop otherwise control transferred outside the loop.

Syntax:

```
until [ condition ]

do
```

ex:

```
until [ i -ge 10 ]

do
```

Statements echo “$i”

done i =`

expr $i + 1 `

done

for loop:

This is a fixed execution loop. This loop is allow to execute list of statements certain period of time.

Syntax:

for variable in value1 value2 value3

….. value n do

statements

done

ex:

for i

in 1 2

3 4 5

do

echo $i

i=` expr $i + 1 `

done