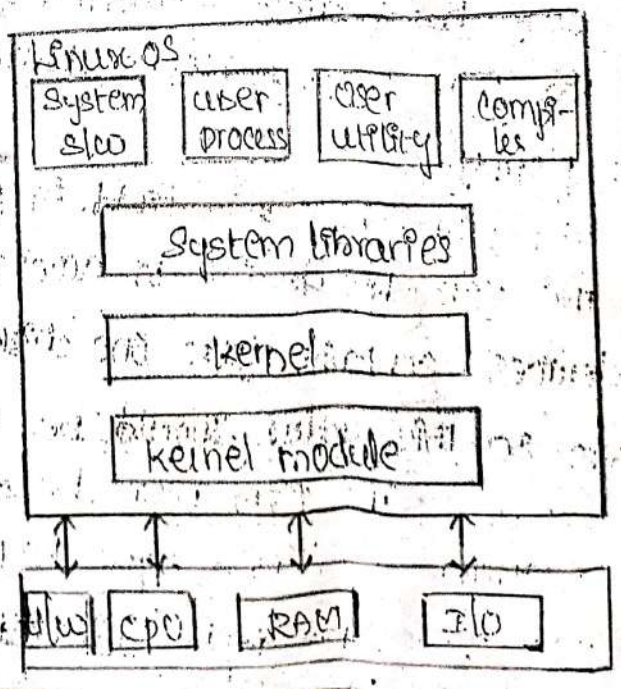
UNIT - V

LINUX

Introduction: LINUX was introduced by Linux torvalds in 1991 in Helsinki. Linux torvalds began a project that later became a Linux kernel. The source programs for hardware, he was using independent of an operating system because he wanted to use the functions of this new PC with 80386 processor. Development was done on minix using GNU 'C' compiler. It is still the main choice for compiling LINUX today. The code however can be built with other compilers, such as 'intel C' compiler.

Definition: LINUX is one of the popular version of UNIX operating system. It is open source as its source code is freely available. It is free to use. LINUX was designed considering UNIX compatability. Its functionality is quite similar to that of UNIX.

Components of LINUX System:



for all major activities of this operating system. It consists of various modules and interact directly with the underlying hardware kernel provides the required abstraction to hide low level hardware details to system or application program.

System Library:- system libraries are special functions or programs using which application programs of system utilities to access kernel features. This libraries implement most of the functionality with OS and do not require kernel modules and code access rights. System

System utility:- System utility programs are responsible to do specialized, individual level tasks.

Kernel mode (v/s) users mode:-

Kernel component code executes in a special privileged mode called kernel mode with full access to all resources of the computer. This code represents a single process executes in a single address space and do not require any switch hence it is very efficient and fast. kernel runs each process and provides system services to process and also provides protected access to hardware to process user program and other system programs. works in user mode, which as no access to the system hardware and kernel mode. User programs or utilities use system libraries to access kernel functions to get systems low level tasks.

Basic features of UNIX:-

There are some of the features of UNIX operating system is as follows.

(1) Portable: portability means slw can works on different types of hardware. Linux, kernel and application program supports installation on any kind of hlw's platform

(2) Open source:- Linux source code is freely available and it is community based development process project. Multiple teams works in collaboration to enhance the capability of Linux operating system and it is continuously evolving.

(3) Multi users:- Linux is a multiuser system that means multiple users can access system resources like memory or RAM or application process at same time.

(4) Multiprogramming:- Linux is a multiprogramming system that means multiple applications can run at the same time.

(5) Hierarical file system:- Linux provides a standard file structure in which system files or user files are arranged.

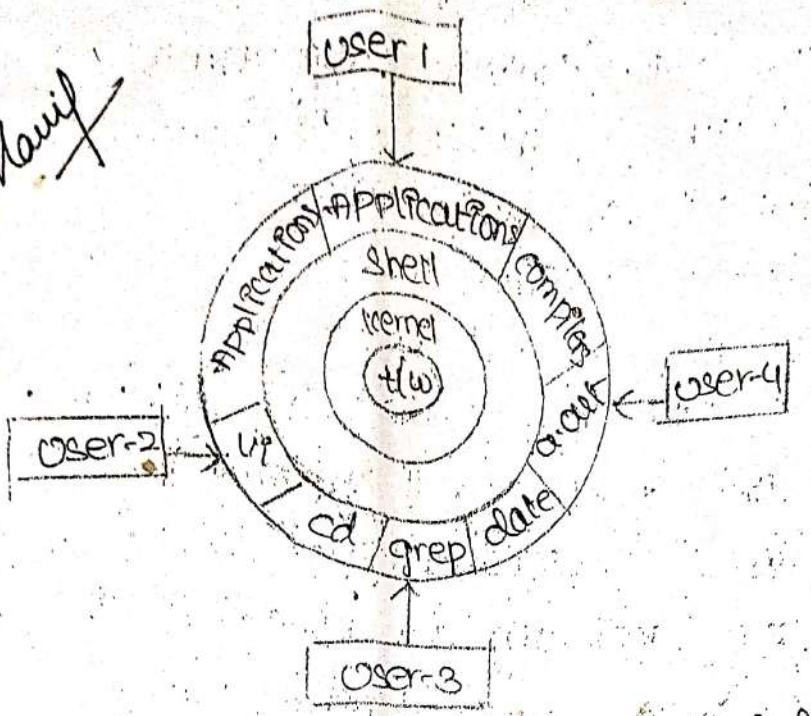
(6) Shell:- Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations call, application programs etc.

(7) Security:- Linux provides user security by using authentication features like password protection and control access to specific files or encryption of data.

* Linux Architecture:-

The following diagram shows the architecture of a Linux.

Manif



The architecture of a LINUX consists of following layers.

Hardware layer:- Hardware consists of all peripheral devices are CPU, hard disk etc.,

Kernel:- kernel is the heart of os. near it makes interaction with hardware, it can be represented as a program which controls all other programs on computer. It is responsible for creation and deletion of memory space which allows software to own all the component request for the required services. when the system is booted the kernel is read into memory. It resides in the memory as long as the system is running. The functions and services of kernel are as follows.

* File management & security
* I/O services
* process scheduling and management
* Memory management
* System accounting
* Error handling
* Date and Time services

kernel are of 2 types

Micro kernel: which contains basic functionalities

Monolithic kernel: which contains many drivers as a part of the kernel can be configured to accomodate like addition (or) deletion of device drivers

Shell: shell is a command interpreter, it acts as an interface b/w the user and the system. shell is a program that takes our command from the keyboard and gives them to os to perform a task. shell provides a way to separate users or tasks from each other while the kernel maintain overall control, shell provides a prompt ($ or %) symbol depends varies of shell being used. This is called as a command line interface (CLI).

Applications:- In addition to command and utilities a number of UNIX based application program that includes word processor, spread sheets, data manager are some of the examples of this application. For the kernel applications are just programs like commands and utilities. The applications usually specify their formats and maintain their own data bases on disk files. To access programs need to call the kernel services to access the device.

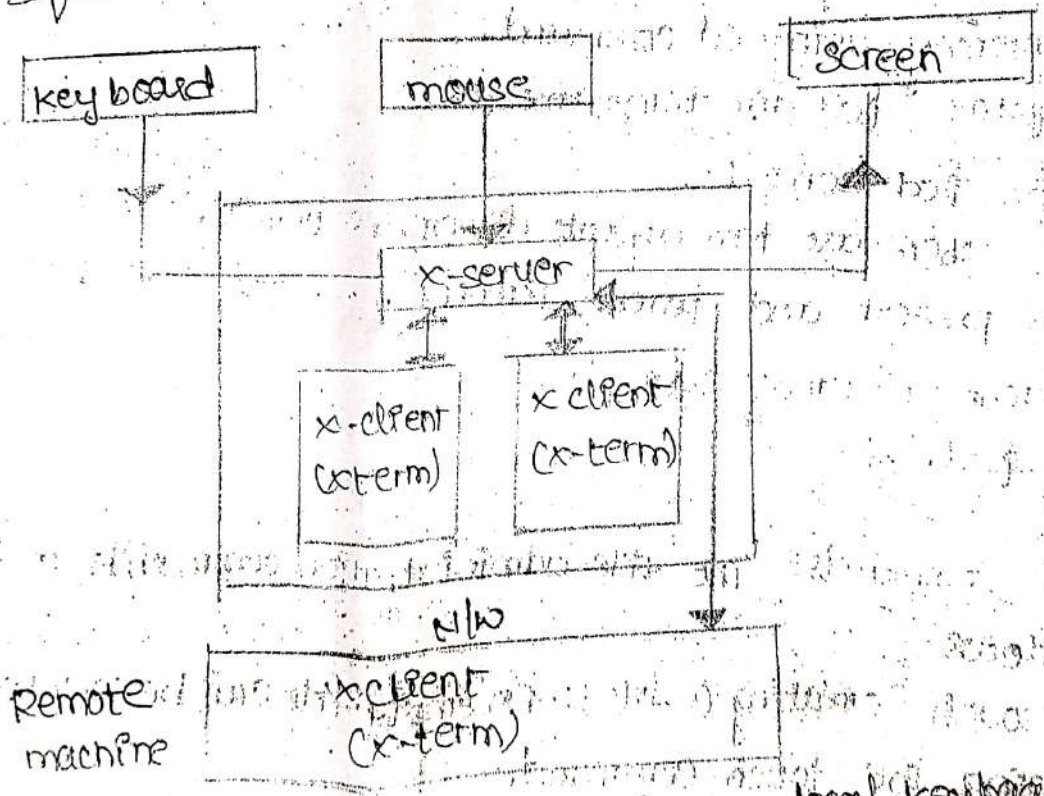
X-Windows:- The X-windows are refered as X or X-windows. It is an open cross platform client server system for managing a window graphical user interface in a distributed network. In general such systems are known as windowing systems. In X-window the client server relationship is reversed from the usual. Remote computer communications for client request to display management tasks.

The x-windoco system coas the olesult of research efforts on the early 1980's at Stanford University and MIT (Massachusetts Institute of technology), aided by IBM to develop a plat-form independent graphics protocal.

The x-windoco system is an open standard that is managed by x-ORG consoatium. Although microsoft has its ocon plat-form dependent windocoing system, cohich is an integral part of windoos | 95 | 98 | NT.

x-windoo system is a windocoing system for bit map displays common on Unix like computer operating system. x provides the basic framecoork for GUI environment such as dlacoing and moving coindoos on the display device and interacting coith mouse and keyboard.

The softcoare Architecture of x-coindoos is as follocos:



The x-server receives input from a local keyboard and mouse and displays to a screen. A coeb broocser and a terminal run on the user coork station and also run on the remote computer but it is controlled and monitored from the user machine.

* Linux Commands:-

** Directory Commands:-

1. Mkdir [making directory]:- A Directory is created using the Mkdir command.

Syntax: $ mkdir directoryname ↵

Eg:- $mkdir BCA2↵

2. Rmdir [Remove directory]: Rmdir removes a single or remains multiple directories at a time.

Syntax: $ rmdir directoryname ↵

Eg: $ rmdir BCA 2 ↵

3. Pwd [present working directory]: It displays the users present current directory.

Syntax: $ Pwd ↵

4. cd [change directory]: We can move in or move out of a directory using cd command.

Syntax: $cd directoryname ↵

Eg- $cd BCA2↵

There are two default directories i.e., (.), (..) which stands for present and parent directory, we can switch to a parent directory using '..'.

Eg: $cd.. ↵

** File Commands:- The file administration commands are as follows.

1. Touch [creating a file]: An empty file can be created by using the touch command.

Syntax $ Touch filename ↵

Eg: $ touch test1 test2 ↵

The above command creates two empty files.

2. cp [copying files]: cp command copies a file or group

Syntax: cp (options) -filename target filename ↵

options of cp command are:

1. -i: (interactive copying): coasts for confirmation, if necessary before an existing target file is overwritten.
2. -r (recursive copying):

Eg: $ cp -i test1 test2 ↵

$ cp -r test1 test2 ↵

3. rm (removing files): The 'rm' command is used to delete the files.

Syntax: $rm (options) -filename ↵

options of rm command are as follows:

1. -i (interactive copying): It coasts for confirmation before deleting a file

Eg: $ rm -i test1 ↵

2. -r (recursive copying): Deletes any existing file or subdirectory.

Eg: $ rm -r test2 ↵

4. mv (moving a file): mv command copies source file then rest of the original file.

Syntax: $mv (options) source-file object-file ↵

options of mv command is as follows:

1. -b [backup]: It creates a backup copy of the source file before moving

Eg: $ mv -b test3 test4 ↵

2. -i: coasts for confirmation, if necessary before an exists target file is over written

Eg: $ mv -i test3 test4 ↵

5. ls (existing files): This command provides the list of all files in the current directory

Syntax: ls(options) filename ↵

The options of ls command are:

-l: It displays the detailed list

Eg: ls -l test 5 ↵

-a: It is used to display the hidden files.

Eg: ls -a test 5 ↵

6. Ch mode: It is used to change the access permission to a file and can be executed by the owner of a file.

Syntax: $ch mod (options) filename ↵

mode: The mode has the following parameters that are.

* Group
* access
* accesstime

Group: Group accepts the following characters (owner) user(u), group(g), other (o)

Access: Access is guaranteed by (+) "+" and "-" is used for denied.

Access time: The access time is controlled by the following options read (r), write(w), execute(x).

Eg:- $chmod g+w ↵

$ chmod o-r ↵

7. chown: This command transfers the ownership of a file to any user.

Syntax: $chown (options) group filename ↵

Eg: $chown tr test1 ↵

8. ln: This ln command creates an internal link from the source file to the target file.

Syntax: $ln (options) source file target file ↵

Eg: The option of ln command is -s

-s: This option creates a symbolic link that points to the directory where the source file is located

Eg:- $ ch -S test1 test2

File Access commands:

1. cat: The cat command displays the contents of a file, prints the entire contents on the screen

Syntax: $ cat (option) file name ↵

eg: $ cat test2 ↵

2. grep: This grep command used to search a string in the specified file.

Syntax: $ grep (options) search string test1 ↵

eg: grep -i jeevitha test1 ↵

-i: ignore case Eg: $ grep -i raju test1 ↵

** Introduction to shell:

Shell is an interface between the user and the kernel

Shell starts and ends as a session for the user and provides users with a "work space".

UNIX supports four major shell implementations.

* Bourne shell
* C shell
* korn shell
* Bash shell

* Bourne shell
* C Shell
* ko korn shell
* Bash shell

Major features of shell:

1. Interactive processing: The users communicates with the system in the form of an interactive dialouge that the shell coordinates using the shell commands.

2. Background processing: A process can execute in the foreground or background. Usually one that requires interactive processing is ran in the foreground, and other long, time consuming non interactive tasks are peformed as background jobs. Sorting a large file could thus be performed as a background job.

3. Input/output redirection: Normally the input and output standard files usually the keyboard as a standard input and terminals as standard output. But shell provides a powerful redirection features that enables the user to take and send input/output to other files.

4. Pipes: These are class files in UNIX that provides the facility of passing the output of one command as input to the other command for further processing in a serial manner & with out the need for intermediate file pipes when connected together to allow complex functions to be performed easily.

5. Meta characters/ wild card characters: select * from stud;
Users can specify generalised patterns rather than fixed strings to match file names with a certain criteria. It provides the facility to select group of files for processing.

6. Shell Scripts: A commonly used sequences of shell commands to be grouped along with logical AND repetitive statement constructs to perform multiple task with a single execution of the shell scripts.

7. Shell variable: The user can control the behavior of the shell as well as other programs and utilizes by storing data in variables.

8. programming language constructs: The shell includes the feature that allows it to be used as a programming languages.

* Meta characters: UNIX utilizes file name as arguments the shell uses a set of symbols called meta characters to provide short hand notation for a group of file names shell also possesses the capability to expand this notation the meta characters are divided into following types.

* wild card characters
* Command line characters
* shell variables
* command characters

* wild card characters: This provides a short hand notation for a group of files.

characters	Meaning
?	It matches any one character
*	It matches multiple number of characters
[list of characters]	It matches any one character from the list of characters; square brackets signify a single character to match
[! list of characters]	It matches any one character excluding the list of characters

Eg: $ ls -lab? → This will map all the file names that starts with 'ab'

$ ls -l* → This will match all the file names in the current directory and complete long listing of the files will be the output.

$ cp chap[0123] progs → This will copy the files starting with 'chap' and followed by anyone of 0,1,2,3 into progs directory.

$ cat emp[!0-9] → This concatenates all the files beginning with the string emp and followed by non-numeric character

** Command line character:

1. Wild separates: spaces, tabs and new lines are termed as internal field separater that are defined by the shell variables. The shell interprets a command to be separated either starts from a new line or separated by semicolon.

Eg:- $ ls -i; cat file↵

ls -l runs first and then cat file1 and is executed
The semicolon seposates them as two commands.

2. Background process: Any command typed on the command lines when suffixed by '&' symbol is run as a background process or PID (process Identification) process.

Eg: $ cat file1 | printer &

This will displays the contents of file1 through the printer as a background job.

$ sort file2 &

This will sort file2 as a background process, each background processhas a unique PID number that can be used to sort background process or check its progress status.

(i) Pipes: A special operator pipe is used as a connector between the two commands such that output of first command is input to the command after pipe

Eg: $ who | wc

(ii) who: The who command provides the information currently logged in. The output of who lists the user name, terminal name, date and time of login.

The output of who is passed directly as input to the wc (word count). No intermediate files are created instead a pipe line is formed.

(iii) Conditional execution: If the execution of a command is dependent on the success or failure of a related command then '&&' and '||' conditional operators can be applied.

Eg: $ grep mngr file1 && echo right entry

If the word mngr is found in file1 then the message right entry is displayed otherwise not

$ grep mngr file1 || echo wrong entry

The word mngr is not found in file1 then the message command fails then the message wrong entry

has variables that can be assigned as a value. Bourn shell has three major types of shell variables.

* system/environment variables (predefined variables)
* user defined variables
* positional variables

1. System/environment variables:

UNIX system has a set of predefined variables that governs the general operation of the environment. These are defined in upper case just to distinguish them from the user defined variables. To see a listing of these variables just type.

eg: $set

This displays a list of system variables and the values they are initialized.

A short list of common system variables are as follows

* EXINIT
* HOME
* IFS
* LOGNAME
* MAIL
* PATH
* PS1
* PS2

EXINIT: It stores the multiple actions that represents in the "ex" & "vi" editor

HOME: This stores the path name of users home directory. On logging in, user is automatically placed in the home directory. To echo a variable.

eg: $ echo $HOME

/usr/James

IFS: This sets a list of characters that are used to separate words in a command line. These are invisible character like space, tabs, new line or any other character.

Default is white space.

LOGNAME: This lists the users login name.

MAIL: All mail addressed to the user is placed in a directory, the name of the mail and notifies the user with a message if mail arrives.

PATH: All alternative paths in a sequences that can be searched to find the executable for any command or file that are stored in it. A colon (:) is used to separate path names, no spaces can be included.

PS1: This stores the values of primary prompt like $ for a Bourne shell and % for C shell and so on.

PS2: This is the secondary prompt, a prompt like do $ that occurs if a new line is started without finishing a command. we can continue on a new line by using a (\).

Eg: $ echo This is an exam \
ple of ps2 secondary prompt
$ set.

2. User defined variables: A user can define has many variables are needed. These variables can be used without type declaration or initialization. Unique of UNIX treats all variables as string variables and automatically shell variables are assigned values using the "=" assignment operator.

Eg: $ #x=37 : assigns the string 37 (not number) to variable x.

A Null string can be explicitly assigned as

Eg:- $ #x= (or) $ #x= " "

A variable name can consist of letters, digits and underscore with first character always begins with letter. To assign a multiword string to a variable.

eg: $ msg = "you have mail"

$ echo $msg

o/p: you have mail

3. Optional variables:- Shell scripts also access arguments from the command line. This is implemented through positional variables which are $0, $1, $2, ... $9, $0 gives the name of the program $1 to $9 represents the command line arguments. Though there are only $1 to $9 variables with the use of shift command we can use more than 9 command line arguments.

Command character:

We can assign the output of a command to a variable and then echo it.

1. Global and local variables: A shell variable is a local variable, it is known only to the shell that is created. For example $ name = James # parent shell

$ echo $name

James

$ sh # new shell (child shell)

$ echo $name

response is blank line i.e. in this shell this variables are undefined.

$ name = smith # name is assigned a value in subshell

$ echo $name

smith

$ ^D # exit out of subshell

$ echo $name # parent shell

James.

Here name is a local variable first to the parent shell and then to the sub shell. sometimes we want the sub shell to know the parent shell variables for this we use the "export" command. variables can be export from parent to child. but that not vice versa.

Eg: $export name
$name=James
$echo $name
James
$sh
$echo $name # The value of parent shell is displayed
James
$name=Smith
$echo $name
Smith
$.AD # EXIT Subshell
$echo $name
James.

To find out the variable which are already exported, type export without arguments

$export

It will list all the exported variable names.

* Shell programming :-

Shell Scripts: All the shell statements and Unix commands can be executed at the command prompt itself. however, when a group of commands have to be executed frequently then it is better to store them in file and execute them by running the file. All such files are called as shell scripts (or) shell programs.

Eg: To creative shell program that echo's time and date

The shell script can

echo "the date and time is" $date

:x

Here the data we create a file called example.sh by using vi editor. A shell script can have any name. the extension is not mandatory.

If the execution permission is set on example.sh. we can execute it by simply writing the file name at the command prompt as

$ example.sh

o/p: The date and time is

03 23 2018 10:30:15

If the execution permission is not set then either the user can set it to execute using the command (ch mode) or execute example.sh by using the following command.

$ sh example.sh

Shell Commands: Making script interactive – read.

The read command is used to make a shell script interactive, it can be followed by a single or list of variables. The input given through the standard input is used to read into the specified variables.

Eg: $ vi test.sh

```
echo 'enter your name'
read name
echo "given name is" $name
:x
```

To execute a program:

$ test.sh (or) $ sh test.sh

o/p: Enter your name

Reddys

Executing commands (dot (.)):

dot (.): The dot command runs commands in a file without creating a new shell. It does not allow the use of command

line arguments. The dot command does not create a new sub shell, a script executed to a dot command can change the values of a shell variable in the current shell.

export command: The export command can be used in two ways.

* It can be used to export the value of a variable to a sub shell.
* The point out of a variable already exported can be obtained by giving export command without arguments

set command: The set command is multi purpose command.

* It is used to produce a list of currently defined system shell variables.
* Eg: $1a, $2, $3 .... $9

$ set file1, file2 ....

Here, we can set $1 = file1 and $2 = file2 and so on.

* set is used to set options for the shell.

Eg: $ set -v , $ set -v

which causes the shell to echo each command before it is executed.

Eg: $ set - x

This option acts as a debugging tool for shell scripts. when the shell script behaves differently these options allows the user to check what is actually going in the script. These options turned off by using the set and the option letter with a '+' sign. disable

Eg: $ set +v

$ set +x

Exit Status of a command: Every command returns a value after execution. These values are called the exit status or return values of the command that is set to be true if it executes successfully and a command is set said to be

the exit status
a program when exit statement is encountered in a script. The execution is terminated and control returns to the calling program.

User can specify any argument value with the exit statement. This makes the script terminate with a return value which can be used to take a decision. The value of exit is assigned as $?

*Control statements / structures:-

Program is the set of statements which are normally executed in sequential order in which they appear. In some cases we may have to change execution order of statements based on certain conditions. Transfer of control from one statement to another statement within the program is called "Branching". If this Branching depends on any condition is called Conditional Branching. Shell supports the following types of decision making statements:

* If statement
* If then else statement
* Nested if statement
* case statement

If statement & If then else statement:

If statement is the simplest conditional statement. general form of if statement is as follows.

Syntax: if expression
execute commands

Syntax for else statement:

if expression
then
execute commands
else

execute commands
fi

If the expression is true then the commands after 'then' are executed. Otherwise the commands listed after 'else' are executed.

If a construct is ended with 'fi' statement then the execution continues as normal after the termination.

eg: $ vi
echo 'enter your name'
read name
echo 'given name is $ name
ctrl +d (execution)

Nested if statement: An if statement having another if statement is known as Nested if statement. The general form of Nested if statement is as follows.

Syntax:

if expression
then
if expression

execute commands.
else
execute commands
else
execute commands

Case statement: When there are a number of alternatives there the user can use case statement.

The syntax of case differs from 'c' language

The general form of the case is as follows.

Syntax: case expression in

Here program/label cases must identify various choices of actions. Generally case statement is used for setup of menus.

Eg:- echo "enter any number from 1 to 3"
read num
case $num in
1) echo "number is one";;
2) echo "number is two";;
3) echo "number is three";;
*) echo "number is not in Range";;
esac.

In the above program we can have a choice label * that matches any values as last choice to act has a default value for case statement.

Loops (or) Iteratives:-

For loop: The use of for loop is different in shell script when compared to for loop in 'c' language. The variable automatically takes value from the list one by one. ... will the list is exhausted that is for loop values are in list.

Syntax:
for variable in list of values
do
execute commands
done

Eg: for i in 1,2,3,4,5
do
echo $?
done

O/p: 1 2 3 4 5

The value of the list can be provided by several methods which are listed below.

Eg: for i in apple orange Banana
do
echo $i
done

O/P: apple orange Banana

The file name from the directory can be supplied as a list using shell metacharacters has shown below.

Eg: for file in *
do
cat $file
echo $file
done

2. While loop: The while loop performs a set of instruction till the control command returns a true exit status. The general form of this loop is as follows:

Syntax:
while condition is true
do
execute commands
done

Until loop: The until command is executed repeatedly as long as the condition remains false and terminates when condition succeeds. The general syntax of until loop is as follows.

Syntax:
until condition is false
do
execute commands
done