Unit 1 - 3

DATE:
PAGE:

Getting Started with Unix, Unix shell (commands)

1) History of Unix :-

1965 : AT & T & MIT created multics multiple users.

1969 : Unix (single version of multics)
by : Ken Thomposon

1970 : Unix was created in assembly language version.

1973 : Unix version 3

1979 : Unix version 7

1981 : BSD (Besueley Software Distribution) developed at University of California.

1982 : System (3) iii was developed

1984 : System v (5) was developed

1987 : minix was developed

1991 : latter versions were developed, Redhat Ubuntu, Debain, fedora, Mapix etc are the different version of Linux.

2) Explain unix architecture.

DATE
PAGE

→ The unix operating system is a set of programs that act as a link between the computer and the user.
→ Unix was originally developed in 1969 by a group of AT&T employes Ken Thompson Dennis Ritchie at Bell Labs.
→ The computer program that allocates the systems resources and coordinates all the details of the computer internal is called the operating system or the kernel
→ Users communicate with the kernel through a program known as the shell.
→ The shell is a command line interpreter, it translates commands entered by the user and converts them into a language that is understood by the kernel.

Kernel :-

→ The kernel is the heart of the operating system.
→ It interacts with the hardware and most of the tasks like memory management tasks scheduling and file management.

— Shell :-

→ The shell is the utility that processes your requests.
→ When you type in a command at your terminal the shell interprets the command ~~at your~~ and calls the ~~files~~ program that you want.
→ C shell, Bourne shell and korn shell are the most famous shells which are availbe available with most of the unix.

DATE
PAGE

Commands and utilities :-

→ There are various commands and utilities which you can make use of in your day to day activities.

→ cp, mv, cut and grep etc are few example of commands and utilites.

→ There are over 250 standard commands others provided through 3rd party software

→ All the commands come along with various options.

Files and Directories :-

→ All the data of unix is organized into files

→ All files are then organized into directories

→ These directories are further organized into a tree-like structure called the filesystem

Q-2 what is directory files ?

→ The unix file system is a methodology for logically organing and storing large quantities of data such that the system is easy to manage

→ A file can be informally defined as a collection of data. A file is the smallest unit of storage in the unix file system

DATE:
PAGE:

Q-3 which Command is used to remove any file in Unix ?

→ The 'rm' Command is used to delete files.

Q-4 what is permission in Unix ?

→ Unix allows three forms of access to any file 1) read 2) write 3) execute for an ordinary file, if you have read (r) permission, you can use that file as input to any Command / Program

→ If you have write (w) Permission, you can make changes to the file.

Q-5 The 'who am I' Command used for what ?

→ when you have different Usernames on Several frequently used UNIX Computers you may in your whoami to determine your current Username and, hence which Computer you are using.

Q-6 Explain types of Shell in Unix ?

→ A Shell provides you with an interface to the Unix System.

→ It gathers input from you and executes programs based on their input

→ when a program finishes executing it displays that program's output.

→ There are different Shells available in Unix :

1) The Bourne shell :-

→ The Bourne shell, written by steve Bourne at AT&T Bell labs., is the original Unix shell.

→ It is the perpreferred shell for shell programing because of its compactness and speed.

→ Its Command full-path name is /bin/sh

User default prompt is $.

2) C shell :-

→ The C shell written by Bill Joy at the University of California at Berkeley.

→ Includes convenient programming features such as built-in arithmetic syntax.

→ Its Command full-Path name is /bin/csh

User default prompt in hostname %

3) Korn Shell :-

→ The korn shell written by David korn at AT&T Bell lads.

→ It supports everything in the Bourne shell. Have interactive features comparible to those in the C shell.

→ It is faster than the C shell.

→ It's Command full-path name is /bin/ksr. User default prompt is $

4) The Bourne-Again shell :-

→ The Bourne-Again shell (bash) is Compatible to the Bourne Shell.

→ it use features from the korn and C She
→ Have arrow keys that are are automatically
mapped for command recall and editing.
→ It's command full-path name is /bin/bash

Q-7 Explain head and tail commands.

→ head :-

→ It is used to display the line from the top of the file. If no argument is passed to this command, by default it displays first 10 lines of the files.

Syntax :-

$ head [-n] <file name>

(n is any positive no.)

Example :-

1) without option

$head abc.txt

1
2
3
4
5
6
7
8
9
10

2) with option

$head -2 abc.txt

1
2

tail Commands :-

→ It is used to display the bottom lines of a file. If no argument are specify it also display 10 lines from the bottom of the files.

Syntax :-

$ tail [-arg] <file name>

(arg is number of lines to be display)

Example :-

1) with argument
$tail -3 abc.txt
9
10
11

2) without argument
$tail abc.txt
2
3
4
5

Q-8 Explain Unix file System

→ The Unix file system contains several different types of files :-

ordinary files :-

→ Used to store your information, such as some text you have written.
→ Always located within/under a directory file.
→ Do not contain other files.

Directories :-

→ Branching points in the hierachical tree.
→ Used to organize groups of files.
→ May contain ordinary files, special files or other directories.

Special files :-

→ Used to represent a real physical device such as a printer, tape drive or terminal, used for input/output (I/O) operations.

DATE:
PAGE:

Q-2 Explain chmod, chown // chgrp commands

chmod :-

chmod changes the permissions of each given file according to mode. where mode describes the permissions to modify Mode can be specified with octal number or with letters.

⇒ Numeric mode :-

→ A numeric mode is from one to four octal digits (0-7), derived by adding up the bits with values 4, 2 and 1.

→ Permissions for the user who owns the file : read(4), write(2) and execute(1)

⇒ Examples :-

1) Read by owner only → $chmod 400 Sample.txt
2) Read by group only → $chmod 040 abc.txt
3) Read by anyone → $chmod 004 abc.txt
4) write by owner only → $chmod 200 abc.txt
5) write by group only → $chmod 020 abc.txt
6) write by anyone → $chmod 002 abc.txt
7) Execute by owner only → $chmod 100 abc.txt
8) Execute by group only → $chmod 010 abc.txt
9) Executed by anyone → $chmod 001 abc.txt
10) Allow read permission to owner and group and anyone → $chmod 444 abc.txt
11) allow everyone to read, write and execute file → $chmod 777 abc.txt

X Chown :-

To change owner, change the user and/or group ownership of each

given file to a new owner
→ Syntax :- chown [options] ... newowner file
Examples :-
1) change the owner of file
$ chown user sample.txt
2) change the group of file
$ chown :mygroup file.txt
3) change both the owner and group of file in single command
$ chown user:mygroup file.txt

X chgrp :-

To change group ownership, 'chgrp' command changes the group ownership of each given file to group which can be either a group name or a numeric group id)
→ Syntax :- chgrp [options] ... {group} file...
→ Example :-
→ To make oracleadmin the owner of the database directory
$ chgrp oracleadmin /user/database

Q-3) Explain : bc, kill, finger, mount and all commands
1) bc :-
bc is a unix command which stands for bench calculator. bc command is used to do some simple arithmetic

→ Syntax :- bc [options] [file..]
Example :- $x=20, $y=10
$ echo $x + $y | bc, 30

2) Kill :-
On Unix-like operating systems the kill command sends a signal to a process if you don't specify which signal to send, by default the TERM signal is sent, which terminates the process
Syntax :- kill Pid
Example :- $kill -3

3) finger :-
finger looks up and displays information about system users it is known as user information look up program.
Syntax :- finger [option] [user...]
Example :- finger → myuser

4) Mount :-
The mount command mounts a storage device or filesystem, making it accessible and attaching it to an existing directory structure.
Syntax :-
mount [option...] DEVICE-NAME DIRECTORY
Example :-
$mount -t type device dir.
options :-
use the -t option to specify the file system type

5) cal :-

To display a Calendar. A single parameter specifies the digit user (1-9999) to be displayed.

Two parameters denote the month (1-12) and year (0-1-9999) if arguments are not specified the current month is displayed A year start on 01 Jan

→ Syntax :- cal [-miy] [month] [year]

Options :-

-m :- displays monday as the first day of week

-j :- Displays julian dates days one based numbered from January -1)

-y :- Displays a calender for the current year

Unit - 4 Text Editing with vi and nano Editor. shell programming

DATE:
PAGE: 12

1) Explain positional parameters :-

→ A positional parameter is a variable within a shell program ; its value is set from one argument specified on the command line that invokes the program.

→ Positional Parameters are numbered and are referred to with a preceding "$" : $1, $2, $3 and so on.

Example :

sh Filename.sh First second third Fourth

$1 $2 $3 $4

Fifth Sixth seven eight

$5 $6 $7 $8

ninth

$9

→ Arguments are user supplied data that follow the file name on the command line and are input to the shell script.

→ positional parameters are predefined memory variables in the shell scripts

→ There are nine positional parameters $1 $2 .. $9 that are used to store the user arguments.

→ The file name is stores in $0.

→ when script is executed the shell puts the first positional parameter $1, second arg in second positional parameter $2 and so on.

Q) Explain decision statements in Unix

→ while writing a shell script, there may be a situation when you want to adopt one path out of the given two paths. So you need to make use of conditional statements that allow your program to make correct decisions and perform the right actions.

→ Unix shell supports conditional statements which are used to perform different actions based on different conditions.

→ Unix shell provides decisions making using

⇒ If - then - fi

```
if (condition)
then
execute command if condition is true
fi
```

⇒ If - else - fi

→ If gives condition is true, then command1 is executed else command2 is executed

Syntax :

```
if [condition]
then
  execute command1 if condition is true
else
  execute command2 if condition is false
fi
```

DATE:
PAGE:

if.. elif.. else.. fi

→ The if.. elif.. fi

The if.. elif.. fi statement is the one level advance form of control statement that allows shell to make correct decision out of several conditions.

Syntax :

```
if [condition 1]
then
    statement to be executed if con1 is true
elif [condition 2]
then
Statement to be executed if con 2 is true
else
statement to be executed if (exp is true
fi
```

→ The case.. esac

→ Unix shell support case.. esac stat which handles exactly this situation and it does so more efficiently than repeated if... elif statement.

→ There is only one form of case.. esac stat.

```
case "$variable" in
"$condition 1")
    Command...
    ;;
"$condition 2")
    Command...
    ;;
esac
```

3] System variable

→ These variables are used to configure the environment during the boot sequence or after user login. So they are also known as environment variables.

→ The shell has two prompts stored in PS1/PS2

* PS1 :- It is the primary prompt string like $ or #. The shell uses the primary prompt when the cmd is expected by the shell for changing the value of this prompt, we can use PS1 = "@".

As soon as it is set the new prompt is displayed. it can be restored back to $ prompt us ps1 - "$",

PS2 :- It is the secondary prompt; it is used when the command entered on the first time was not able to be completed The default Prompt is >

4] write a shell script to check the number is prime number or not.

```
#!/bin/bash
echo "Enter a number"
read n
i=2
if [$n -lt 2]
then
echo "$n is not a prime number";
exit
fi
```

```
while [$i -lt $n]
do
if ['expr $n % $i -eq 0]
then
 echo "$n is not a prime number"
 exit
fi
i=`expr $i+1`
done
echo "$n is a prime number"
```

5) write a shell script for the following pattern

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
#! /bin /bash

for r in {1...5}
do
for i in $(seq 1 $r)
 do
printf "$i"
done
printf "\n"
done
```

Q. Explain Array with Example

→ An Array is a structured arrangement of similar data elements within shell scripting an array is a variable that holds multiple values, whether they are of the same type or different types

How to Declare Array in shell scripting :

1) Indirect Declaration :

ARRAYNAME [INDEXNR] = value

2) Compound Assignment

ARRAYNAME = (value1 value2 value3)

Alternatively you can use index number to using values explicitly.

Example :

#!/bin/bash

Script to print the keys of the array

Declaring the ARRY

declare -a example_array=("welcome" "To" "Home")

Printing the keys

echo "${!example_array[@]}"

7) Example UDF with Example.

→ with the help of functions. overall functionality of a function can be divided into smaller or logical parts which can be called to perform their task

To create a function :
function functionName () {
Commands to be executed }

Variable Scope :

→ Global variables are defined as the variable which can be accessed anywhere within the script regardless of the scope By default all the variables are defined as global variables even if they are declared inside the function

* Return value

```
#! /bin/bash
# Setting up a return status for a function
print_it () {
        echo Hello $1
        return 5 }
print_it User
print_it reader
echo The previous function returned
    a value of $?
```

8) write a shell script to test for empty arguments. Use the -z operator to check if an arguments is empty

```
#!/bin/bash
Function Check_empty()
{
  if [-z "$1" ]; then
  echo "Error : Argument is empty"
    return 1
    fi
    Echo "Argument - $1" }
check_empty "apple"
check_empty ""
```

9) write a shell script to create a simple digital clock or terminal.

```
#!/bin/bash
  function digital_clock
  {
    clear
    while [1]
    do
      date '+%T'
      sleep 1
      clear
    done
}
digital_clock
```

DATE:
PAGE:

10] Explain looping structure in linux.

→ There is total 2 looping statements which can be used in bash programming.

* While statement :-

Here command is evaluated and based on the result loop will executed if command raise to false then loop will be terminated

Syntax :-

```
while command
do
Statement to be executed
done
```

* for Statement

→ The for loop operate on lists of items.

→ It repeats a set of commands for every item in a list.

→ Here var is the name of a variable and word1 to word n are sequences of characters separated by spaces.

Syntax :-

```
for var in word1 word2 ... wordn
do
Statement to be executed
done
```

Untill Loop :-

→ As an exit control loop. until loop is available

Syntax :

```
Until [condition]
do
    command 1. 2
done
```

The break Statement :

→ The break Statement is used to terminate the execution of the entire loop, after completing the execution of all the lines of code up to the break statement.

Syntax :

The following break Statement is used to come out of a loop.

```
break
```