INTRODUCTION TO SHELL PROGRAMMING

A **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.

Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, 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 the **command prompt**, is issued by the shell. While the prompt is displayed, you can type a command.

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 the **date** command, which displays the current date and time −

```
$date
Thu Jun 25 08:30:19 MST 2009
```

You can customize your command prompt using the environment variable PS1 explained in the Environment tutorial.

Shell Types

In Unix, there are two major types of shells −

- **Bourne shell** − If you are using a Bourne-type shell, the **$** character is the default prompt.
- **C shell** − If you are using a C-type shell, the % character is the default prompt.

The Bourne Shell has the following subcategories −

- 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 the AT&T Bell Labs in New Jersey.

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

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 that can be used on different versions of Unix.

In this chapter, we are going to cover most of the Shell concepts that are based on the Borne Shell.

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 # 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

In Unix-like operating systems, the **chmod** command is used to change the access mode of a file. The name is an abbreviation of **change mode**.

Syntax :

chmod [reference][operator][mode] file...

The references are used to distinguish the users to whom the permissions apply i.e. they are list of letters that specifies whom to give permissions. The references are represented by one or more of the following letters:

Reference	Class	Description
u	owner	file's owner
g	group	users who are members ofthe file's group
o	others	users who are neither the file's owner nor members of the file's group
a	all	All three of the above, same as ugo

The operator is used to specify how the modes of a file should be adjusted. The following operators are accepted:

Operator	Description
+	Adds the specified modes to the specified classes
-	Removes the specified modes from the specified classes
=	The modes specified are to be made the exact modes for the specified classes

The modes indicate which permissions are to be granted or removed from the specified classes. There are three basic modes which correspond to the basic permissions:

r	Permission to read the file.
w	Permission to write (or delete) the file.
x	Permission to execute the file, or, in the case of a directory, search it.

Types of permissions which we will be changing using chmod command :
In linux terminal, to see all the permissions to different files, type ls -l command which lists the files in the working directory in long format

EX.NO:1 BASIC COMMANDS IN LINUX

AIM:

To Study the basic commands in Linux.

COMMANDS:

1. Task : To display system date and time.

   Syntax: Date

   Explanation: This command displays the current date and time on the screen.

2. Task: To display current month.

   Syntax: Date +%m

   Explanation: This command displays the current month on the screen.

3. Task : To display the name of the current month.

   Syntax: Date +%h

   Explanation: This command displays name of the current month on the screen.

4. Task: To display current system date.

   Syntax: Date +%d

   Explanation: This command displays the current system date on the screen.

5. Task: To display current year.

   Syntax: Date +%y

   Explanation: This command displays the current year on the screen.

6. Task: To display current system time.

   Syntax: Date +%H

   Explanation: This command displays the current system time on the screen.

7. Task: To display current system time in minutes.

   Syntax: Date +%m

   Explanation: This command displays the current system time in minutes on the screen.

8. Task: To display current system time in seconds.

   Syntax: Date +%s

   Explanation: This command displays the current system time in seconds on the screen.

9. Task: To display the calendar of current month.

   Syntax: cal

Explanation: This command displays the current month calender on the screen.

10. Task: To display user defined message.

    Syntax: echo ‘message’

    Explanation: This command displays the message.

SHELL PROGRAMMING

Ex.No.2a EVEN OR ODD

AIM:

To write a program to find whether a number is even or odd.

ALGORITHM:

STEP 1: Read the input number.

STEP 2: Perform modular division on input number by 2.

STEP 3: If remainder is 0 print the number is even.

STEP 4: Else print number is odd.

STEP 5: Stop the program.

PROGRAM

```
echo "enter the number"
read num
echo "enter the number"
read num
if [ `expr $num % 2` -eq 0 ]
then
echo "number is even"
else
echo "number is odd"
fi
```

OUTPUT:

enter the number: 5

the number is odd.

RESULT:

Thus the program has been executed successfully.

Ex.No.2b **BIGGEST OF TWO NUMBERS**

AIM :

To write a program to find biggest in two numbers.

ALGORITHM :

STEP 1: Read The Two Numbers.

STEP 2: If Value Of A Is Greater Than B Is Big.

STEP 3: Else Print B Is Big.

STEP 4: Stop The Program.

PROGRAM:

```
echo "enter the number"
read a b
if [ $a -gt $b ]
then
echo "A is big"
else
echo "B is big"
fi
```

OUTPUT:

Enter The Two Number:

23 67

B is Big.

RESULT:

Thus the program has been executed successfully.

Ex.No.2c **BIGGEST OF THREE NUMBERS**

AIM:

To Write a Program to Find Biggest In Three Numbers.

ALGORITHM:

STEP 1: Read The Three Numbers.

STEP 2: If A Is Greater Than B And A Is Greater Than C Then Print A Is Big.

STEP 3: Else If B is greater Than C Then C Is Big.

STEP 4: Else Print C Is Big.

STEP 5: Stop The Program.

PROGRAM:

```
echo "enter three numbers"
read  a b c
if [ $a -gt $b ] && [ $a -gt $c ]
then
echo "A is big"
else if [ $b -gt $c ]
then
echo "B is big"
else
echo "C is big"
fi
fi
```

OUTPUT:

ENTER THREE NUMBERS:

23 54 78

C IS BIG.

RESULT:

Thus the program has been executed successfully.

Ex.No.2d FACTORIAL OF NUMBER

AIM:

To find a factorial of a number using shell script.

ALGORITHM:

Step 1: read a number.

Step 2: Initialize fact as 1.

Step 3: Initialize I as 1.

Step 4: While I is lesser than or equal to no.

Step 5: Multiply the value of I and fact and assign to fact increment the value of I by 1.

Step 6: print the result.

Step 7: Stop the program.

PROGRAM:

```
echo "enter the number"
read n
fact=1
i=1
while [ $i -le $n ]
do
fact=`expr $i * $fact`
i=`expr $i + 1`
done
echo "the fcatorial number of $n  is $fact”
```

OUTPUT:

Enter the number :

4

The factorial of 4 is 24.

RESULT:

Thus the program has been executed successfully.

FIBONACCI SERIES

Ex.No.2e

AIM :

To write a program to display the Fibonacci series.

ALGORITHM:

Step 1: Initialize n1& n2 as 0 & 1.

Step 2: enter the limit for Fibonacci.

Step 3: initialize variable as 0

Step 4: Print the Fibonacci series n1 and n2.

Step 5: While the var number is lesser than lim-2

Step 6: Calculate n3=n1+n2.

Step 7: Set n1=n2 and n2=n3

Step 8: Increment var by 1 and print n2

Step 9: stop the program.

PROGRAM:

```
echo " ENTER THE LIMIT FOR FIBONNACI SERIES"
read lim
n1=0
n2=1
var=0
echo "FIBONACCI SERIES IS "
echo "$n1"
echo "$n2"
while [ $var -lt `expr $lim - 2` ]
do
n3=`expr $n1 + $n2 `
```

```
n1=`expr $n2 `
n2=`expr $n3 `
var=`expr $var + 1 `
echo "$n2"
done
```

OUTPUT :

enter the limit for Fibonacci:

5

The Fibonacci series is:

0

1

1

2

3

RESULT:

Thus the program has been executed successfully.

INTRODUCTION TO OPERATING SYSTEMS

An Operating System is a program that manages the Computer hardware. It controls and coordinates the use of the hardware among the various application programs for the various users.

A Process is a program in execution. As a process executes, it changes state

- New: The process is being created
- Running: Instructions are being executed
- Waiting: The process is waiting for some event to occur
- Ready: The process is waiting to be assigned to a process
- Terminated : The process has finished execution

Apart from the program code, it includes the current activity represented by

- Program Counter,
- Contents of Processor registers,
- Process Stack which contains temporary data like function parameters, return addresses and local variables
- Data section which contains global variables
- Heap for dynamic memory allocation

A Multi-programmed system can have many processes running simultaneously with the CPU multiplexed among them. By switching the CPU between the processes, the OS can make the computer more productive. There is Process Scheduler which selects the process among many processes that are ready, for program execution on the CPU. Switching the CPU to another process requires performing a state save of the current process and a state restore of new process, this is Context Switch.

Scheduling Algorithms

CPU Scheduler can select processes from ready queue based on various scheduling algorithms. Different scheduling algorithms have different properties, and the choice of a particular algorithm may favour one class of processes over another. The scheduling criteria include

- CPU utilization: