EXPERIMENT No.1

TITLE:
To study and execute the commands and shell scripts.

OBJECTIVES: To study and execute the commands and shell scripts**.**

1: Date Command:

This command is used to display the current data and time.

Syntax:

$date

Various Date Command Formats

You can use formatting option to display date command in various formats using the following syntax:

$ date +%<format-option>

The following table displays various date command formatting options.

Format options	**Purpose of Option**	**Output**
date +%a	Displays Weekday name in short (like Mon, Tue, Wed)	Thu
date +%A	Displays Weekday name in full short (like Monday, Tuesday)	Thursday
date +%b	Displays Month name in short (like Jan, Feb, Mar)	Feb
date +%B	Displays Month name in full short (like January, February)	February
date +%d	Displays Day of month (e.g., 01)	07
date +%D	Displays Current Date; shown in MM/DD/YY	02/07/13
date +%F	Displays Date; shown in YYYY-MM-DD	2013-02-07
date +%j	Displays day of year (001..366)	038
date +%m	Displays month (01..12)	02
date +%M	Displays minute (00..59)	44
date +%S	Displays second (00..60)	17
date +%N	Displays nanoseconds (000000000..999999999)	573587606
date +%T	Displays time; shown as HH:MM:SS Note: Hours in 24 Format	23:44:17
date +%u	Displays day of week (1..7); 1 is Monday	4
date +%U	Displays week number of year, with Sunday as first day of week (00..53)	05
date +%Y	Displays full year i.e. YYYY	2013

date +%Z	alphabetic time zone abbreviation (e.g., EDT)	IS

Display Last Modification Time using -r option

In this example, the current time is 20:25:48

```
$ date
Sun May 20 20:25:48 PDT 2013
```

The timestamp of datefile is changed using touch command. This was done few seconds after the above date command's output.

```
$ touch datefile
```

The current time after the above touch command is 20:26:12

```
$ date
Sun May 20 20:26:12 PDT 2013
```

Finally, use the date command -r option to display the last modified timestamp of a file as shown below. In this example, it displays last modified time of datefile as 20:25:57. It is somewhere between 20:25:48 and 20:26:12 (which is when we execute the above touch command to modify the timestamp).

```
$ date -r datefile
Sun May 20 20:25:57 PDT 2013
```

2. Calendar Command :
This command is used to display the calendar of the year or the particular month of calendar year.
Syntax :
a.$cal <year>
b.$cal <month> <year>
Here the first syntax gives the entire calendar for given year & the second Syntax gives the calendar of reserved month of that year.

3. Echo Command :
This command is used to print the arguments on the screen.
Syntax: $echo <text>
Multi line echo command:
To have the output in the same line , the following commands can be used.
Syntax: $echo <text\>text
To have the output in different line, the following command can be used.
Syntax: $echo "text
>line2
>line3"

4.'who' Command :
It is used to display who are the users connected to our computer currently.
Syntax : $who – option‟s
Options : -
H–Display the output with headers.
b–Display the last booting date or time or when the system was lastely rebooted.

6.'who am i' Command :
Display the details of the current working directory.
Syntax : $who am i

7.'tty' Command :
It will display the terminal name.
Syntax : $tty

8.'Binary' Calculator Command :
It will change the „$‟ mode and in the new mode, arithematic operations such as +,-,*,/,%,n,sqrt(),length(),=, etc can be performed . This command is used to go to the binary calculus mode.
Syntax :
$bc operations
^d
$
1 base –inputbase
0 base – outputbase are used for base conversions.
Base :
Decimal = 1 Binary = 2 Octal = 8 Hexa = 16

9.'CLEAR' Command :
It is used to clear the screen.
Syntax : $clear

10.'MAN' Command :
It help us to know about the particular command and its options & working. It is like „help‟ command in windows .
Syntax : $man <command name>

11.MANIPULATION Command :
It is used to manipulate the screen.
Syntax : $tput <argument>

Arguments :
1.Clear – to clear the screen.
2.Longname – Display the complete name of the terminal.
3.SMSO – background become white and foreground become black color.
4.rmso – background become black and foreground becomes white color.

12.LIST Command :
It is used to list all the contents in the current working directory.
Syntax : $ ls – options <arguments>
If the command does not contain any argument means it is working in the Current directory.
Options :
a– used to list all the files including the hidden files.
c– list all the files columnwise.
d- list all the directories.
m- list the files separated by commas.
p- list files include „/" to all the directories.
r- list the files in reverse alphabetical order.
f- list the files based on the list modification date.
x-list in column wise sorted order.

DIRECTORY RELATED COMMANDS :

1.Present Working Directory Command :
To print the complete path of the current working directory.
Syntax : $pwd

2.MKDIR Command :
To create or make a new directory in a current directory .
Syntax : $mkdir <directory name>

3.CD Command :
To change or move the directory to the mentioned directory .
Syntax : $cd <directory name.

4.RMDIR Command :
To remove a directory in the current directory & not the current directory itself.
Syntax : $rmdir <directory name>

FILE RELATED COMMANDS :

1.CREATE A FILE :
To create a new file in the current directory we use CAT command.
Syntax :
$cat > <filename.

2.DISPLAY A FILE :
To display the content of file mentioned we use CAT command without „>" operator.
Syntax :
$cat <filename.
Options –s = to neglect the warning /error message.

3.COPYING CONTENTS :
To copy the content of one file with another. If file doesnot exist, a new file is created and if the file exists with some data then it is overwritten.
Syntax :
$ cat <filename source> >> <destination filename>
$ cat <source filename> >> <destination filename> it is avoid overwriting.
Options : -
-n content of file with numbers included with blank lines.
Syntax :
$cat –n <filename>

4.SORTING A FILE :
To sort the contents in alphabetical order in reverse order.
Syntax :
$sort <filename >
Option : $ sort –r <filename>

5.COPYING CONTENTS FROM ONE FILE TO ANOTHER :
To copy the contents from source to destination file . so that both contents are same.
Syntax :
$cp <source filename> <destination filename>
$cp <source filename path > <destination filename path>

6.MOVE Command :
To completely move the contents from source file to destination file and to remove the source file.
Syntax :
$ mv <source filename> <destination filename>

7.REMOVE Command :
To permanently remove the file we use this command .
Syntax :
$rm <filename>
8.WORD Command :
To list the content count of no of lines , words, characters .
Syntax :
$wc<filename>
Options :
-c – to display no of characters.
-l – to display only the lines.
-w – to display the no of words.

9.LINE PRINTER :
To print the line through the printer, we use lp command.
Syntax :
$lp <filename>

10.More Command :

If you are used to working with Linux, you will find a lot of file text in Linux world. Configuration files and log files are usually kept in text format. But those file are usually have long content. You can not view them all in one page. So we need to to pagination to those file. And to do that, we can use **more** command.

What is more command

More command is a command for displaying a long text file per page at a time. More command is a built-in command in Linux.

How to use more

To use more command, we just need to type :

$ more file_name

11. FILTERS AND PIPES

HEAD : It is used to display the top ten lines of file.
Syntax: $head<filename>

TAIL : This command is used to display the last ten lines of file.
Syntax: $tail<filename>

PAGE : This command shows the page by page a screenfull of information is displayed after which the page command displays a prompt and passes for the user to strike the enter key to continue scrolling.
Syntax: $ls –a\p

MORE : It also displays the file page by page .To continue scrolling with more command , press the space bar key.
Syntax: $more<filename>

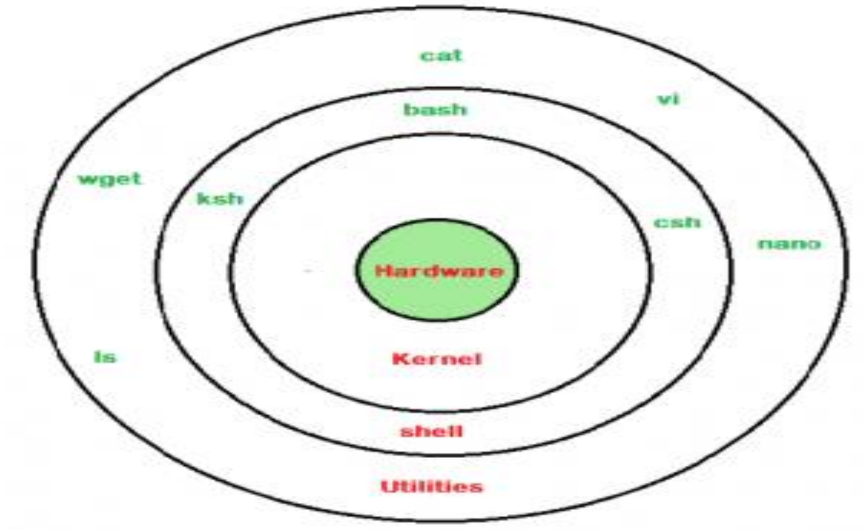
GREP : This command is used to search and print the specified patterns from the file. Sy**ntax:** $grep [option] pattern <filename>

SORT : This command is used to sort the datas in some order.

Syntax: $sort<filename>
PIPE : It is a mechanism by which the output of one command can be channeled into the input of another command.
Syntax: $who | wc-l
TR : The tr filter is used to translate one set of characters from the standard inputs to another.
Syntax: $tr “[a-z]” “[A-Z]”

VI editor
The *VI editor* is a fast and powerful text editor of Unix system.
SAVING AND QUITING FROM vi :-
1.<ESC> w Command :
To save the given text present in the file.
Syntax : <ESC> : w
2.<ESC> q! Command :
To quit the given text without saving.
Syntax : <ESC> :q!
3.<ESC> wq Command :
This command quits the vi editor after saving the text in the mentioned file.
Syntax : <ESC> :wq
4.<ESC> x Command :
This command is same as „wq‟ command it saves and quit.
Syntax : <ESC> :x
5.<ESC> q Command :
This command would quit the window but it would ask for again to save the file.
Syntax : <ESC> : q

What is Shell? A sell is special user program which provide an interface to user to use operating system services. Shell accept human readable commands from user and convert them into something which kernel can understand. It is a command language interpreter that execute commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or start the terminal.



Linux shell
Shell is broadly classified into two categories –

- Command Line Shell-user using a command line interface. A special program called **Terminal** in linux/macOS or **Command Prompt** in Windows OS is provided to type in the human readable commands.
- Graphical shell-Graphical shells provide means for manipulating programs based on graphical user interface (GUI), by allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows.

Shell Scripting

Usually shells are interactive that mean, they accept command as input from users and execute them. However some time we want to execute a bunch of commands routinely, so we have type in all commands each time in terminal.
As shell can also take commands as input from file we can write these commands in a file and can execute them in shell to avoid this repetitive work. These files are called **Shell Scripts** or **Shell Programs**. Shell scripts are similar to the **batch file** in MS-DOS. Each shell script is saved with **.sh** file extension eg. **myscript.sh**

A shell script have syntax just like any other programming language. If you have any prior experience with any programming language like Python, C/C++ etc. it would be very easy to get started with it.
A shell script comprises following elements –

- Shell Keywords – if, else, break etc.
- Shell commands – cd, ls, echo, pwd, touch etc.
- Functions
- Control flow – if..then..else, case and shell loops etc.

Why do we need shell scripts?

There are many reasons to write shell scripts –

- To avoid repetitive work and automation
- System admins use shell scripting for routine backups
- System monitoring
- Adding new functionality to the shell etc.

Advantages of shell scripts

- The command and syntax are exactly the same as those directly entered in command line, so programmer do not need to switch to entirely different syntax
- Writing shell scripts are much quicker
- Quick start
- Interactive debugging etc.

Shell Programming

COMPARISON OF TWO STRINGS
Aim:
To write a shell program to compare the two strings.
Algorithm:
Step1: Enter into the vi editor and go to the insert mode for entering the code
Step2: Read the first string.
Step3: Read the second string
Step4: Compare the two strings using the if loop
Step5: If the condition satisfies then print that two strings are equal else print two strings are not equal.
Step6: Enter into the escape mode for the execution of the result and verify the output
Program:
Output:
Result:

MAXIMUM OF THREE NUMBERS
Aim:
To write a shell program to find greatest of three numbers.
Algorithm:
Step1: Declare the three variables.
Step2: Check if A is greater than B and C.
Step3: If so print A is greater.
Step4: Else check if B is greater than C.
Step5: If so print B is greater.
Step6: Else print C is greater.

C is greater

Program:

Output:

Result:

FIBONACCI SERIES

Aim:

To write a shell program to generate fibonacci series.

Algorithm :

Step 1 : Initialise a to 0 and b to 1.
Step 2 : Print the values of 'a' and 'b'.
Step 3 : Add the values of 'a' and 'b'. Store the added value in variable 'c'.
Step 4 : Print the value of 'c'.
Step 5 : Initialise 'a' to 'b' and 'b' to 'c'.
Step 6 : Repeat the steps 3,4,5 till the value of 'a' is less than 10.

Program:

Output:

Result:

ARITHMETIC OPERATIONS USING CASE

Aim:

To write a shell program to perform the arithmetic operations using case

Algorithm :

Step 1 : Read the input variables and assign the value
Step 2 : Print the various arithmetic operations which we are going to perform
Step 3 : Using the case operator assign the various functions for the arithmetic Operators.
Step 4: Check the values for all the corresponding operations.
Step 5: Print the result and stop the execution.

Program:

Output:

Result:

SIMPLE FUNCTION USING SHELL (sample Shell script)

<u>SOURCE CODE:</u>

```
program()
{
echo " 1 is $1 "
echo " 2 is $2 "
a= " goodbye "
}
a=hello
```

```
b=world
program $a $b
echo “ a is $a ”
echo “ b is $b ”
```

OUTPUT:

```
[examuser35@localhost Jebastin]$ sh function.sh
1 is hello
2 is world
a is goodbye
b is world
```

Write the function using shell script to accept and display employee payroll details.

Algorithm:

Aim: Write the function for employee payroll.

Step1: Declare the function get () to accept the employee information like employee id, name, deduction in salary, basic salary and allowance.

Step2: Calculate and display the employee net and gross salary

Program:

Output:

Result:

PROGRAM: Implement the program using shell script.

OUTPUT: