

## **Experiment No. 5**

### **Aim:**

- a) Study of Shell, Types of Shell, Variables and Operators.
- b) Execute the following Scripts:
  - i) Write a shell script to perform arithmetic operations.
  - ii) Write a shell script to calculate simple interest.
  - iii) Write a shell script to determine largest among three integer numbers.
  - iv) Write a shell script to determine a given year is leap year or not.
  - v) Write a shell script to print multiplication table of given number using while statement.
  - vi) Write a shell script to compare two strings.

### **Objective:**

- i) To learn Unix general purpose commands and programming in Unix editor environment.
- ii) To understand file system management and user management commands in Unix.
- iii) To learn basic shell scripting.

### **Outcome:**

#### **LO :2,3,5**

#### **LO Statements:**

Identify the Unix general purpose commands.

Apply Unix commands for system administrative tasks such as file system management and user management.

Execute Unix commands for system administrative tasks such as process management and memory management.

### **Theory:**

Unix is a computer Operating System which is capable of handling activities from multiple users at the same time. The development of Unix started around 1969 at AT&T Bell Labs by Ken Thompson and Dennis Ritchie. Unix is security conscious, and can be used only by those persons who have an account. Telnet (Telephone Network) is a Terminal emulator program for TCP/IP networks that enables users to log on to remote servers. To logon, type telnet server\_ip address in run window. User has to authenticate himself by providing username and password. Once verified, a greeting and \$ prompt appears. The shell is now ready to receive commands from the user. Options suffixed with a hyphen (–) and arguments are separated by space.

**Study of Shells:**

A UNIX shell is a command-line interpreter that provides a user interface for the UNIX operating system. Users direct the operation of the computer by entering commands as text for a command line interpreter to execute or by creating text scripts of one or more such commands. 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. 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.

The commands can be combined using the pipeline (|) operator. For example, number of users logged in can be obtained as `who | wc -`

Example of the date command, which displays the current date and time –

\$date

Thu Mar 16 08:30:19 MST 2022

**Types of Shells:**

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.

**Bourne Shell:** The Bourne shell (sh) is a shell command-line interpreter for computer operating systems. The Bourne shell was the default shell for Version 7 Unix. Unix-like systems continue to have /bin/sh—which will be the Bourne shell, or a symbolic link or hard link to a compatible shell—even when other shells are used by most users. Developed by Stephen Bourne at Bell Labs, it was a replacement for the Thompson shell, whose executable file had the same name—sh. Although it is used as an interactive command interpreter, it was also intended as a scripting language and contains most of the features that are commonly considered to produce structured programs. Bourne Again Shell (Bash) is the free version of the Bourne shell distributed with Linux systems. The standard GNU shell, intuitive and flexible. Probably most advisable for beginning users while being at the same time a powerful tool for the advanced and professional user.

The Bourne Shell has the following subcategories –

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

**C Shell:** The syntax of this shell resembles that of the C programming language. The C shell (csh or the improved version, tcsh) is a Unix shell created by Bill. The C shell is a command processor which is typically run in a text window, allowing the user to type and execute commands. The C shell can also read commands from a file, called a script. Like all Unix shells, it supports filename wildcarding, piping, here documents, command substitution, variables and control structures for condition-testing and iteration. What differentiated the C shell from others, especially in the 1980s, were its interactive features and overall style. Its new features made it easier and faster to use. The overall style of the language looked more like C and was seen as more readable. tcsh is a superset of the common C shell, enhancing user-friendliness and speed. That is why some also call it the Turbo C shell.

The different C-type shells follow –

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

**Filtering Commands-** Filters are the central commands of the UNIX tool kit. It acts on data file where lines are records, fields delimited by a character not used by the data

Command	Function
head	used to display the first few records (10 records by default)
head stud	Displays first 10 records by default
head -5 stud head -1 stud   wc -c	Displays first 5 records length of first record
tail	used to display the last few records (10 records by default)
tail stud	Displays last 10 records by default
tail -5 stud   tee last5	Last 5 records listed & stored in file last5 using tee
cut	Used to extract specific fields. The d option specifies the delimiter and f for specifying the field list. The c option may be used if extraction is done character wise
sort stud	Sorted on 1st column by default
sort -t \  -k 3 stud	Sort as per 3rd column
tr	Translates characters. Can be used to change text case. It works with standard
uniq stud	Display unique entries in a sorted file

**Variable Types:** When a shell is running, three main types of variables are present –

- **Local Variables** – A local variable is a variable that is present within the current instance of the shell. It is not available to programs that are started by the shell. They are set at the command prompt.
- **Environment Variables** – An environment variable is available to any child process of the shell. Some programs need environment variables in order to function correctly. Usually,

a shell script defines only those environment variables that are needed by the programs that it runs.

- **Shell Variables** – A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly. Some of these variables are environment variables whereas others are local variables.

**Arithmetic Operators:** The following arithmetic operators are supported by Bourne Shell.

Assume variable **a** holds 10 and variable **b** holds 20 then –

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 hand 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)	Assigns 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.

**Relational Operators:** Bourne Shell supports the following relational operators that are specific to numeric values. These operators do not work for string values unless their value is numeric.

Assume variable **a** holds 10 and variable **b** holds 20 then –

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

Boolean Operators: The following Boolean operators are supported by the Bourne Shell.

Assume variable **a** holds 10 and variable **b** holds 20 then –

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

Assume variable **a** holds "abc" and variable **b** holds "efg" then –

Operator	Description	Example
=	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[ \$a = \$b ] is not true.
!=	Checks if the value of two operands are equal or not; if values are not equal then the 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 nonzero length, then it returns true.	[ -n \$a ] is not false.
str	Checks if <b>str</b> is not the empty string; if it is empty, then it returns false.	[ \$a ] is not false.

### Execute the following Scripts:

i) Write a shell script to perform arithmetic operations.

#### Program:

```
#!/bin/sh

echo -n "Enter the first number: "

read num1

echo -n "Enter the second number: "

read num2

echo "The addition of the $num1 and $num2 is `expr $num1 + $num2`"

echo "The subtraction of the $num1 and $num2 is `expr $num1 - $num2`"

echo "The multiplication of the $num1 and $num2 is `expr $num1 \* $num2`"

echo "The division of the $num1 and $num2 is `expr $num1 / $num2`"
```

**Output:**

```
soham32@soham32-VirtualBox:~$ gedit EXP%_1.sh
soham32@soham32-VirtualBox:~$ chmod +x EXP5_1.sh
soham32@soham32-VirtualBox:~$ ./EXP5_1.sh
Enter the first number: 9
Enter the second number: 2
The addition of the 9 and 2 is 11
The subtraction of the 9 and 2 is 7
The multiplication of the 9 and 2 is 18
The division of the 9 and 2 is 4
```

ii) Write a shell script to calculate simple interest.

**Program:**

```
#!/bin/sh

echo -n "Enter the principle value: "

read principle

echo -n "Enter the rate of interest: "

read rate

echo -n "Enter the time period: "

read time

SI=`expr $principle \* $rate \* $time / 100 `

echo "The Simple Interest is $SI"
```

**Output:**

```
soham32@soham32-VirtualBox:~$ gedit EXP5_2.sh
soham32@soham32-VirtualBox:~$ chmod +x EXP5_2.sh
soham32@soham32-VirtualBox:~$ ./EXP5_2.sh
Enter the principle value: 10000
Enter the rate of interest: 5
Enter the time period: 1
The Simple Interest is 500
```

iii) Write a shell script to determine largest among three integer numbers.

**Program:**

```
#!/bin/sh

echo -n "Enter the first number: "

read num1

echo -n "Enter the second number: "

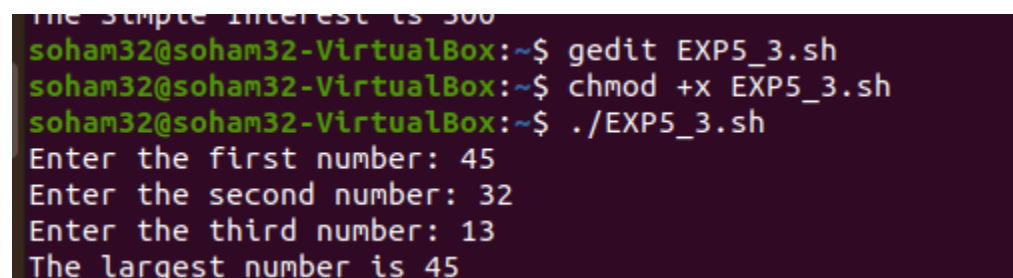
read num2

echo -n "Enter the third number: "

read num3

if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
    echo "The largest number is $num1"
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
then
    echo "The largest number is $num2"
else
    echo "The largest number is $num3"
fi
```

**Output:**



```
The Simple Interest is 300
soham32@soham32-VirtualBox:~$ gedit EXP5_3.sh
soham32@soham32-VirtualBox:~$ chmod +x EXP5_3.sh
soham32@soham32-VirtualBox:~$ ./EXP5_3.sh
Enter the first number: 45
Enter the second number: 32
Enter the third number: 13
The largest number is 45
```



iv) Write a shell script to determine a given year is leap year or not.

**Program:**

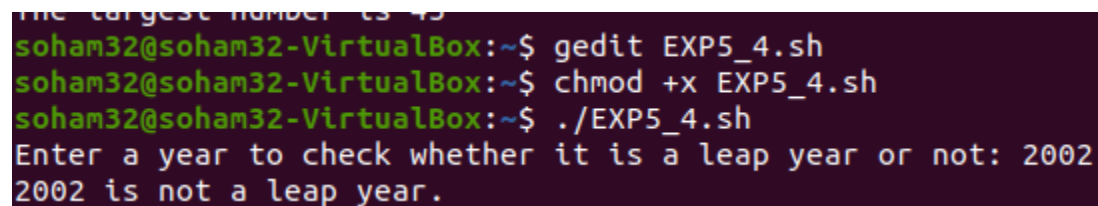
```
#!/bin/sh

echo -n "Enter a year to check whether it is a leap year or not: "

read year

if [ `expr $year % 4` -eq 0 ] && [ `expr $year % 100` -ne 0 ] || [ `expr $year % 400` -eq 0 ]
then
    echo "$year is a leap year."
else
    echo "$year is not a leap year."
fi
```

**Output:**



```
soham32@soham32-VirtualBox:~$ gedit EXP5_4.sh
soham32@soham32-VirtualBox:~$ chmod +x EXP5_4.sh
soham32@soham32-VirtualBox:~$ ./EXP5_4.sh
Enter a year to check whether it is a leap year or not: 2002
2002 is not a leap year.
```

v) Write a shell script to print multiplication table of given number using while statement.

**Program:**

```
#!/bin/sh

echo -n "Enter a number to get it's multiplication table: "

read num

i=1

echo "The Multiplication table for number $num is:"

while [ $i -le 12 ];do

    echo "$num x $i = `expr $num \* $i`"

    i=`expr $i + 1`
```

**Output:**

```
soham32@soham32-VirtualBox:~$ gedit EXP5_5.sh
soham32@soham32-VirtualBox:~$ chmod +x EXP5_5.sh
soham32@soham32-VirtualBox:~$ ./EXP5_5.sh
Enter a number to get it's multiplication table: 5
The Multiplication table for number 5 is:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
5 x 11 = 55
5 x 12 = 60
```

**Conclusion:** From this experiment we have learned how to do shell scripting and also how to run the programs.