Experiment No. 5

Title: Shell scripting in Linux

Date: 20/08/2024 **Batch: B-2** **Roll No: 16010422234** **Name: Chandana Galgali**

Experiment No: 5

Aim: Introduction to Shell Scripting.

Resources needed: Any open source OS / Cocalc online editor

Theory:
Pre lab/Prior concepts:

Create a script

As discussed earlier, shell scripts are stored in plain text files, generally one command per line. You can use a text editor such as vi or emacs. I recommend using vim (Vi Improved) as it is equipped with features such as syntax highlighting and indenting. Most modern Linux (or *BSD) distribution comes with vim. You can start vi or vim from shell prompt by typing any one of the following command:

```
$ vi myscript.bash
$ vi myscript.sh
$ vim myscript.bash
```

Make sure you use .bash or .sh file extension for each script. This ensures easy identification of shell scripts.

Setup executable permission

Once a script is created, you need to set up executable permission on a script. Why to set up executable permission, might be the next question in your mind, right?
Simple,

- Without executable permission, running a script is almost impossible.
- Besides executable permission, script must have a read permission.

Syntax to setup executable permission:
chmod permission your-script-name

Examples:

```
$ chmod +x your-script-name
$ chmod 755 your-script-name
```

Run a script (execute a script)

Now your script is ready with proper executable permission on it. Next, test the script by running it.

Syntax:

bash your-script-name sh your-script-name
./your-script-name
Examples:

```
$ bash bar
$ sh bar
$ ./bar
```

In the last example, you are using . (dot) command which reads and executes commands from filename in the current shell. If filename does not contain a slash, file names in PATH are used to find the directory containing filename. For example:

./bar
bar script executed from current directory. The specialty of dot (.) command is you do not have to set up an executable permission on script.

Debug a script if required

While programming shell sometimes you need to find out errors (bugs) in shell script and correct all errors (remove errors i.e. debug script). For this purpose you can pass -v and -x option to the sh/bash command to debug the shell script. General syntax is as follows:

sh option { shell-script-name }
OR
bash option { shell-script-name }
Where,

- **-v**: print shell input lines as they are read
- **-x**: after expanding each simple-command, bash displays the expanded value of the PS4 system variable, followed by the command and its expanded arguments.

Debugging discussed later in depth. Now you are ready to write the first shell script that will print "Knowledge is Power" on screen. See the common vi command list , if you are new to vi.

```
$ vi first
#
# My first shell script
#
clear
echo "Knowledge is Power"
```

After saving the above script, you can run the script as follows:
$ ./first

This script will not run since we have not set execute permission for our script first; to do this type command
$ chmod 755 first
$ ./first

First screen will be clear, then Knowledge is Power is printed on screen.

Script Command(s)	Meaning
$ vi first	Start vi editor
# # My first shell script #	# followed by any text is considered as a comment. Comment gives more information about script, logical explanation about shell script. *Syntax:* # comment-text
clear	clear the screen
echo "Knowledge is Power"	To print message or value of variables on screen, we use echo command, general form of echo command is as follows *syntax:* echo "Message"

Tip: For shell script files try to give file extension such as .sh, which can be easily identified by you as shell script.

Variables in Shell

To process our data/information, data must be kept in computers RAM memory. RAM memory is divided into small locations, and each location has a unique number called memory location/address, which is used to hold our data. Programmer can give a unique name to this memory location/address called memory variable or variable (It's a named storage location that may take different values, but only one at a time).

In Linux (Shell), there are two types of variable:

1. System variables - Created and maintained by Linux itself. This type of variable is defined in CAPITAL LETTERS.
2. User defined variables (UDV) - Created and maintained by the user. This type of variable is defined in lower letters.

You can see system variables by giving command like **$ set**, some of the important System variables are:

System Variable	Meaning
BASH=/bin/bash	Our shell name
BASH_VERSION=1.14.7(1)	Our shell version name
COLUMNS=80	No. of columns for our screen
HOME=/home/vivek	Our home directory
LINES=25	No. of columns for our screen
LOGNAME=students	students Our logging name
OSTYPE=Linux	Our Os type
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h \W]\$	Our prompt settings
PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=vivek	User name who is currently login to this PC

NOTE: Some of the above settings can be different in your PC/Linux environment. You can print any of the above variables contains as follows:
$ echo $USERNAME
$ echo $HOME

How to define User defined variables (UDV)?

To define UDV use following syntax Syntax:
variable name=value
'value' is assigned to a given 'variable name' and Value must be on the right side = sign. Example:
$ no=10# this is ok
$ 10=no# Error, NOT Ok, Value must be on the right side of = sign. To define variable called 'vech' having value Bus
$ vech=Bus
To define variable called n having value 10
$ n=10

Rules for Naming variable name (Both UDV and System Variable)

1. Variable names must begin with an Alphanumeric character or underscore character (_), followed by one or more Alphanumeric characters. For e.g. Valid shell variable are as follows

   HOME SYSTEM_VERSION
   vech no

(1) Don't put spaces on either side of the equal sign when assigning value to variable. For e.g. In following variable declaration there will be no error
```
$ no=10
```
But there will be problem for any of the following variable declaration:
```
$ no =10
$ no= 10
$ no = 10
```

(2) Variables are case-sensitive, just like filename in Linux. For e.g.
```
$ no=10
$ No=11
$ NO=20
$ nO=2
```
Above all are different variable name, so to print value 20 we have to use $ echo $NO and not any of the following
```
$ echo $no # will print 10 but not 20
$ echo $No # will print 11 but not 20
$ echo $nO # will print 2 but not 20
```

(3) You can define NULL variable as follows (NULL variable is variable which has no value at the time of definition) For e.g.
```
$ vech=
$ vech=""
```
Try to print it's value by issuing following command
```
$ echo $vech
```
Nothing will be shown because variable has no value i.e. NULL variable.

(4) Do not use ?,* etc, to name your variable names.

How to print or access the value of UDV (User defined variables)?

To print or access UDV use following syntax Syntax:
$variablename
Define variable vech and n as follows:
```
$ vech=Bus
$ n=10
```
To print contains of variable 'vech' type
```
$ echo $vech
```
It will print 'Bus',To print contains of variable 'n' type command as follows
```
$ echo $n
```

Caution: Do not try $ echo vech, as it will print vech instead of its value 'Bus' and $ echo n, as it will print n instead its value '10', You must use $ followed by variable name.

echo Command

Use echo command to display text or value of variable. echo [options] [string, variables...] Displays text or variable value on screen.

Options
-n Do not output the trailing new line.
-e Enable interpretation of the following backslash escaped characters in the strings:
\a alert (bell)
\b backspace
\c suppress trailing new line
\n new line
\r carriage return
\t horizontal tab
\\ backslash

For e.g. **$ echo -e "An apple a day keeps away \a\t\tdoctor\n**

Up to now, our scripts have not been interactive. That is, they did not require any input from the user. In this lesson, we will see how your scripts can ask questions, and get and use responses.

read
To get input from the keyboard, you use the read command. The read command takes input from the keyboard and assigns it to a variable.
Here is an example:

```
#!/bin/bash
echo -n "Enter some text > " read text
echo "You entered: $text"
```

As you can see, we displayed a prompt on line 3. Note that "-n" given to the echo command causes it to keep the cursor on the same line; i.e., it does not output a carriage return at the end of the prompt.
Next, we invoke the read command with "text" as its argument. What this does is wait for the user to type something followed by a carriage return (the Enter key) and then assign whatever was typed to the variable text.

Here is the script in action:

```
[me@linuxbox me]$ read_demo.bash Enter some text > this is some text You entered: this is some text
```

If you don't give the read command the name of a variable to assign its input, it will use the environment variable REPLY.
The read command also takes some command line options. The two most interesting ones are -t and -s. The -t option followed by a number of seconds provides an automatic

Time out for the read command. This means that the read command will give up after the specified number of seconds if no response has been received from the user. This option could be used in the case of a script that must continue (perhaps resorting to a default response) even if the user does not answer the prompts.

The -s option causes the user's typing not to be displayed. This is useful when you are asking the user to type in a password or other security related information.

Arithmetic

Since we are working on a computer, it is natural to expect that it can perform some simple arithmetic. The shell provides features for integer arithmetic.

What's an integer? That means whole numbers like 1, 2, 458, -2859. It does not mean fractional numbers like 0.5, .333, or 3.1415. If you must deal with fractional numbers, there is a separate program called bc which provides an arbitrary precision calculator language.

Variations

Arithmetic expansion with backticks (often used in conjunction with expr)

```
z=`expr $z + 3`          # The 'expr' command performs the
expansion.
```

```
z=$(($z+3))

z=$((z+3))                                    # Also correct.

                                              # Within double
parentheses,

                                              #+ parameter
dereferencing

                                              #+ is optional.



# $((EXPRESSION)) is arithmetic expansion. # Not to be confused
with

                                              #+ command
substitution.
```

Arithmetic expansion with double parentheses,and using let .The use of backticks (backquotes) in arithmetic expansion has been superseded by double parentheses -- ((...)) and $((...)) -- and also by the very convenient let construction.

```
# You may also use operations within double parentheses without
assignment.

  n=0
  echo "n = $n"                                       # n = 0

  (( n += 1 ))                                        # Increment.
# (( $n += 1 )) is incorrect!
  echo "n = $n"                                       # n = 1


let z=z+3
let "z += 3" # Quotes permit the use of spaces in variable
assignment.
              # The 'let' operator actually performs arithmetic
evaluation,
              #+ rather than expansion.
```

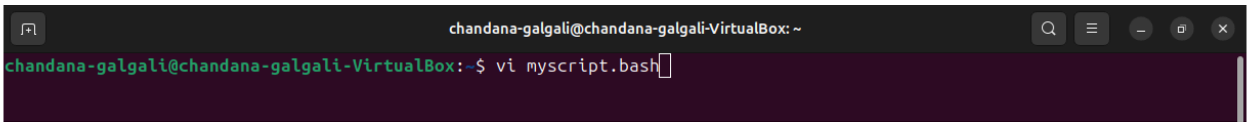
```
#!/bin/bash

# Demonstrating some of the uses of 'expr'
# ========================================

echo

# Arithmetic Operators
#

echo "Arithmetic Operators"
echo
a=`expr 5 + 3`
echo "5 + 3 = $a"

a=`expr $a + 1`
echo
echo "a + 1 = $a"
echo "(incrementing a variable)"

a=`expr 5 % 3`
```

```
# modulo
echo
echo "5 mod 3 = $a"

echo
echo
```

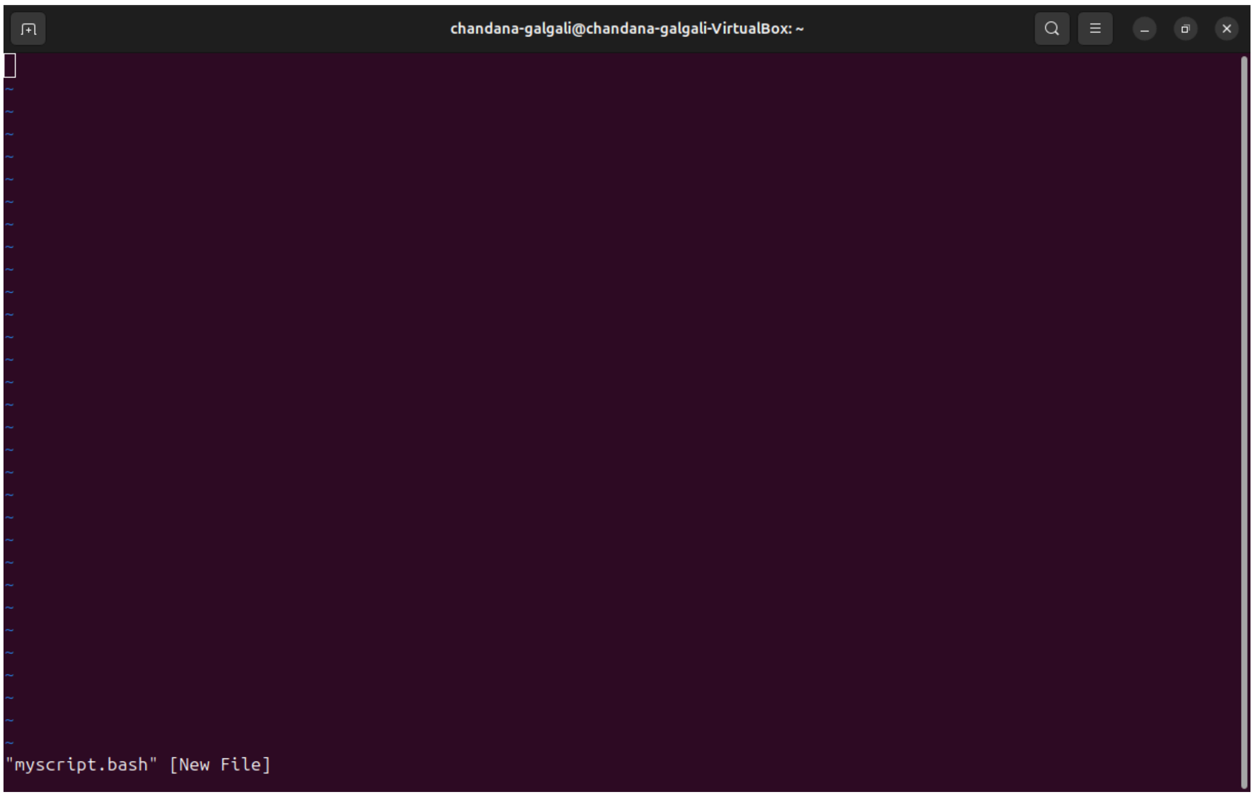
Activities:

1. How to Define variable x with value 10 and print it on screen.
2. How to Define variable xn with value Rani and print it on screen
3. Write a shell script to display your name(stored in UDV), name of the shell you are using, type of the operating system you are using, today's date and time, current month in the calendar.
4. Write a shell script to store three student's data (name, rollno and totalmarks) in separate variables and print this data each on a new line with proper spacing in between fields(use echo with - e option and escape characters \n,\t etc).
5. Write a shell program that can perform a variety of common (like +,-,*, /,% etc) arithmetic operations.
6. Write a shell program that formats an arbitrary number of seconds into hours and minutes.

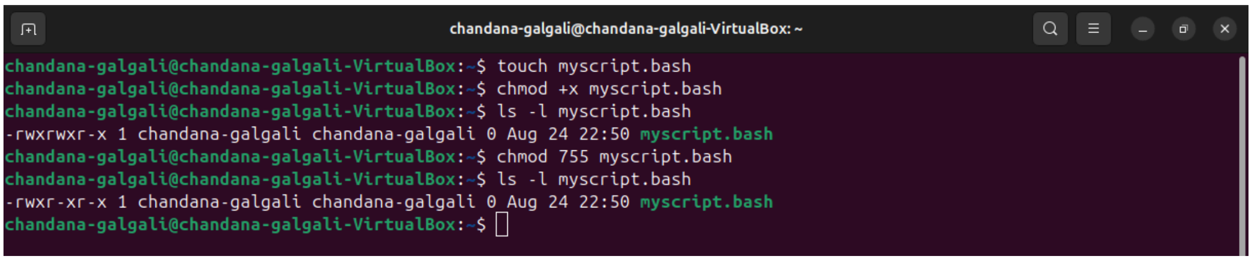
Results:



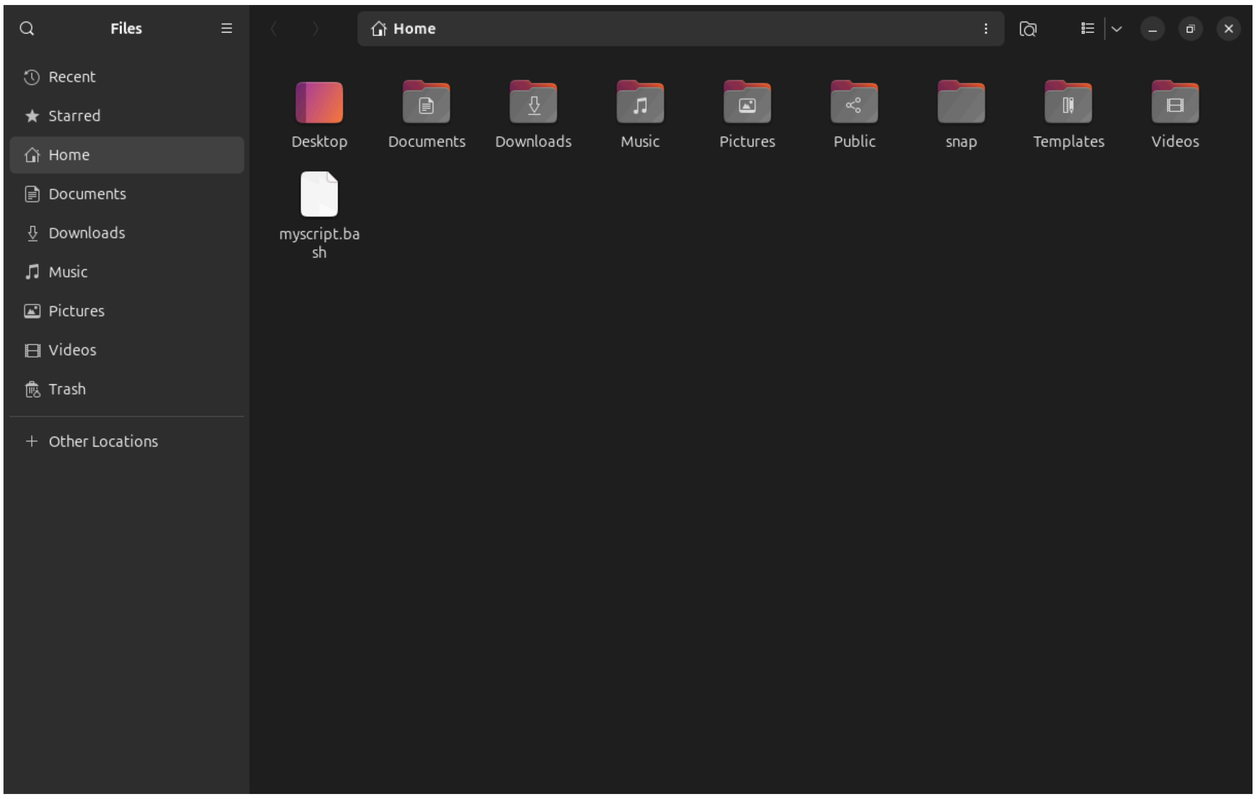
```
chandana-galgali@chandana-galgali-VirtualBox:~$ vi myscript.bash
```

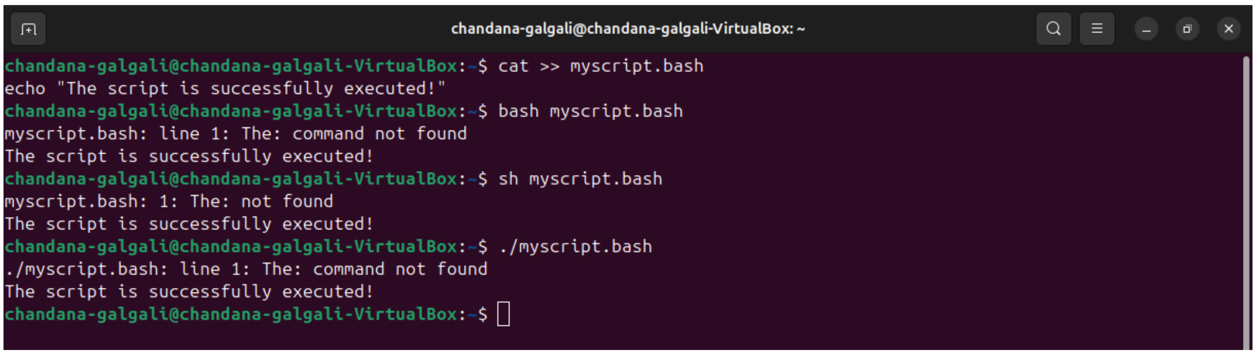


```
~
~
~
"myscript.bash" [New File]
```

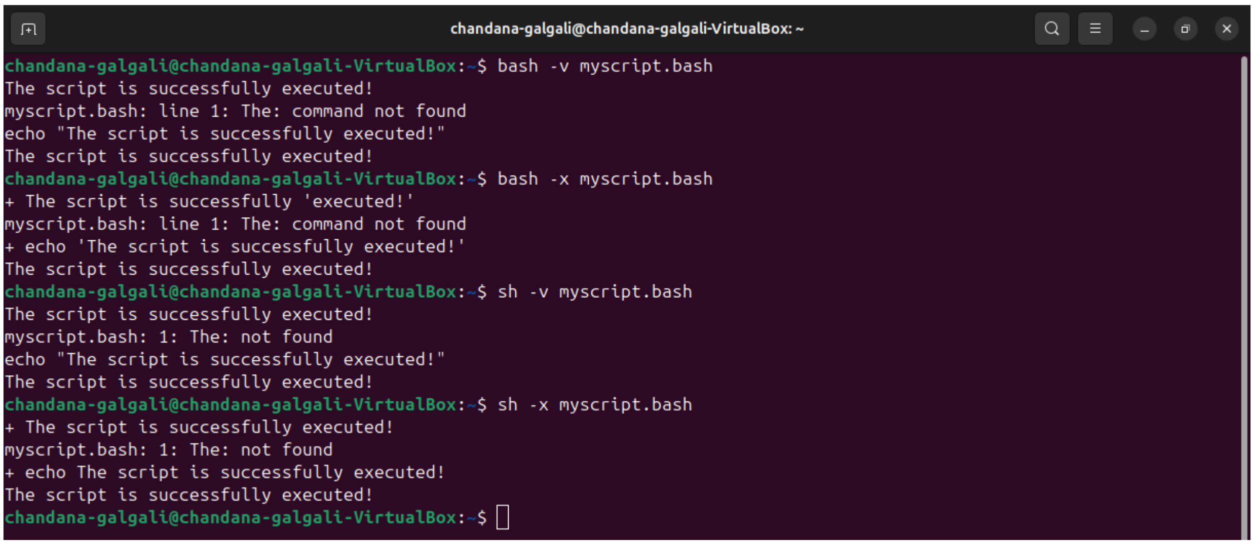


```
chandana-galgali@chandana-galgali-VirtualBox:~$ touch myscript.bash
chandana-galgali@chandana-galgali-VirtualBox:~$ chmod +x myscript.bash
chandana-galgali@chandana-galgali-VirtualBox:~$ ls -l myscript.bash
-rwxrwxr-x 1 chandana-galgali chandana-galgali 0 Aug 24 22:50 myscript.bash
chandana-galgali@chandana-galgali-VirtualBox:~$ chmod 755 myscript.bash
chandana-galgali@chandana-galgali-VirtualBox:~$ ls -l myscript.bash
-rwxr-xr-x 1 chandana-galgali chandana-galgali 0 Aug 24 22:50 myscript.bash
chandana-galgali@chandana-galgali-VirtualBox:~$ 
```

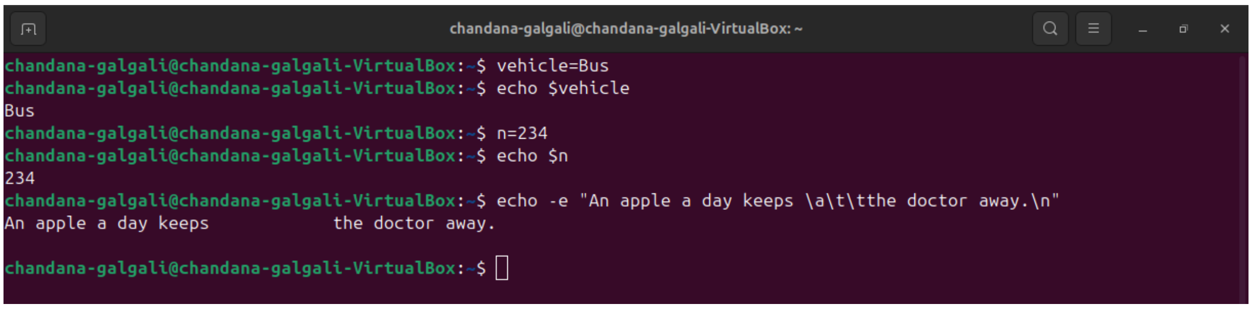




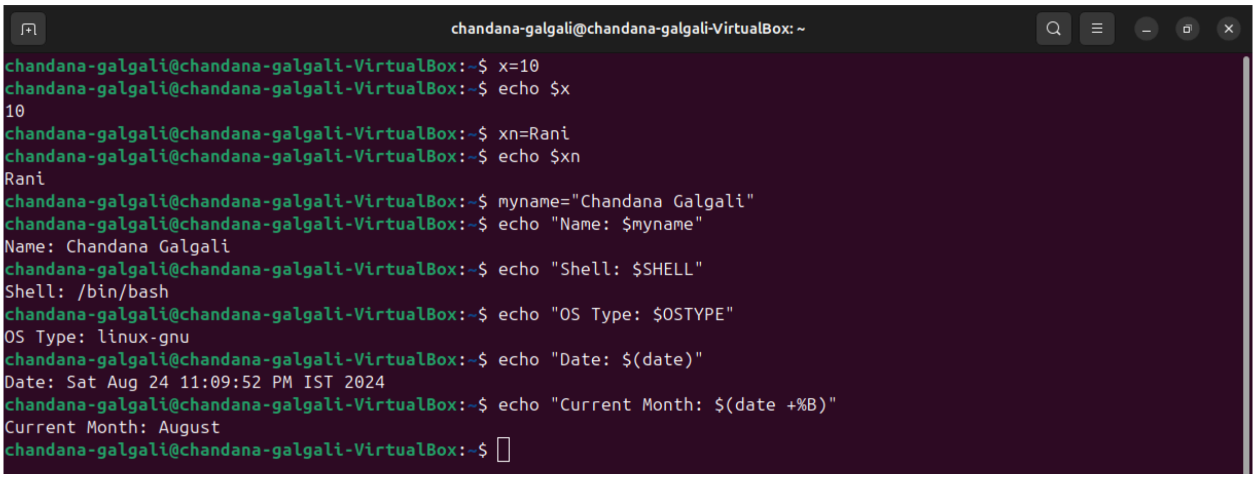
```
chandana-galgali@chandana-galgali-VirtualBox:~$ cat >> myscript.bash
echo "The script is successfully executed!"
chandana-galgali@chandana-galgali-VirtualBox:~$ bash myscript.bash
myscript.bash: line 1: The: command not found
The script is successfully executed!
chandana-galgali@chandana-galgali-VirtualBox:~$ sh myscript.bash
myscript.bash: 1: The: not found
The script is successfully executed!
chandana-galgali@chandana-galgali-VirtualBox:~$ ./myscript.bash
./myscript.bash: line 1: The: command not found
The script is successfully executed!
chandana-galgali@chandana-galgali-VirtualBox:~$ 
```


chandana-galgali@chandana-galgali-VirtualBox: ~

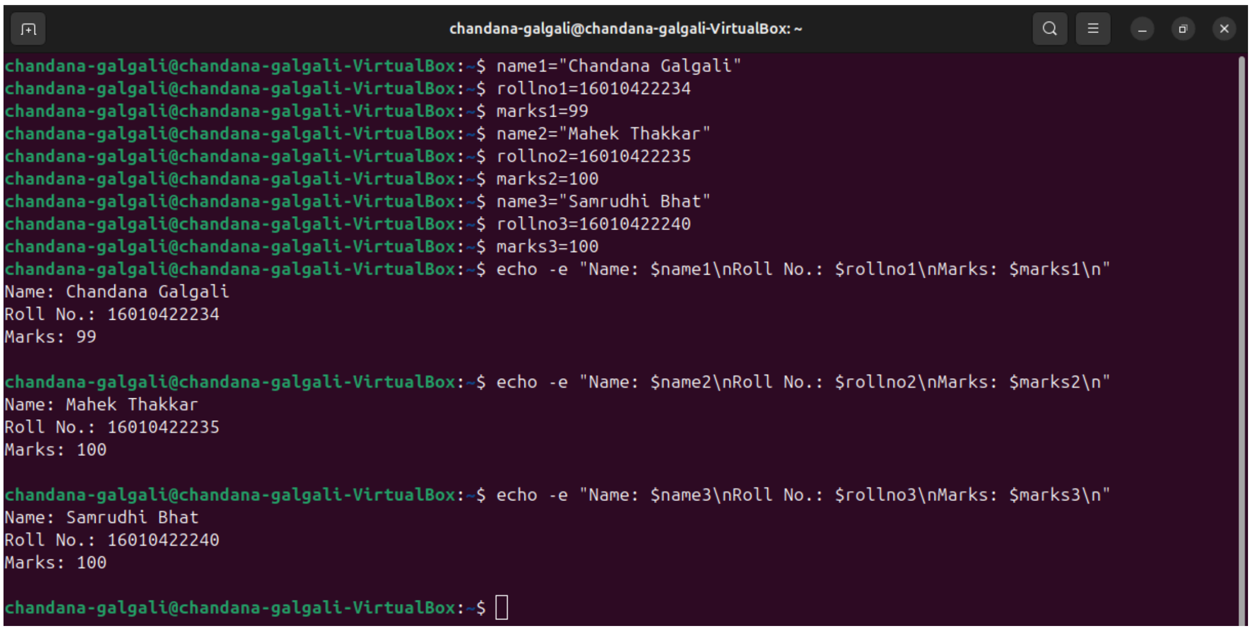
```
chandana-galgali@chandana-galgali-VirtualBox:~$ bash -v myscript.bash
The script is successfully executed!
myscript.bash: line 1: The: command not found
echo "The script is successfully executed!"
The script is successfully executed!
chandana-galgali@chandana-galgali-VirtualBox:~$ bash -x myscript.bash
+ The script is successfully 'executed!'
myscript.bash: line 1: The: command not found
+ echo 'The script is successfully executed!'
The script is successfully executed!
chandana-galgali@chandana-galgali-VirtualBox:~$ sh -v myscript.bash
The script is successfully executed!
myscript.bash: 1: The: not found
echo "The script is successfully executed!"
The script is successfully executed!
chandana-galgali@chandana-galgali-VirtualBox:~$ sh -x myscript.bash
+ The script is successfully executed!
myscript.bash: 1: The: not found
+ echo The script is successfully executed!
The script is successfully executed!
chandana-galgali@chandana-galgali-VirtualBox:~$ 
```


chandana-galgali@chandana-galgali-VirtualBox: ~

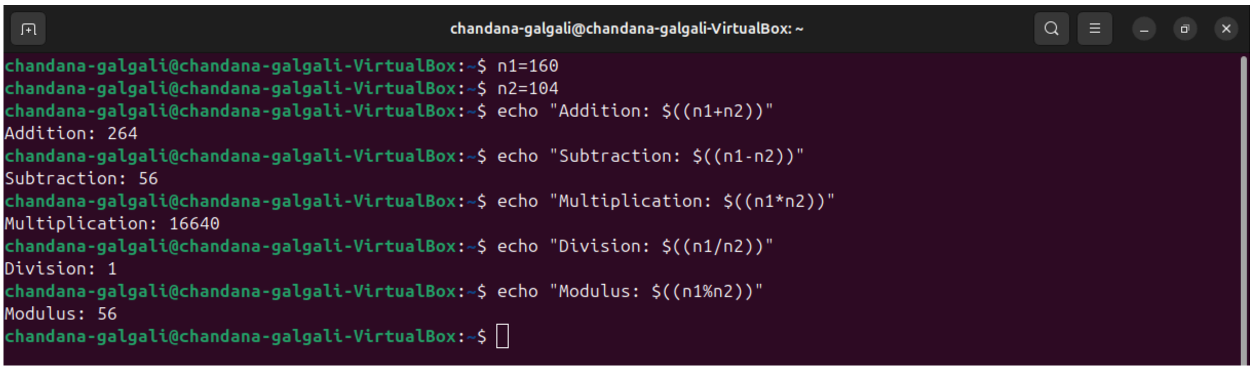
```
chandana-galgali@chandana-galgali-VirtualBox:~$ vehicle=Bus
chandana-galgali@chandana-galgali-VirtualBox:~$ echo $vehicle
Bus
chandana-galgali@chandana-galgali-VirtualBox:~$ n=234
chandana-galgali@chandana-galgali-VirtualBox:~$ echo $n
234
chandana-galgali@chandana-galgali-VirtualBox:~$ echo -e "An apple a day keeps \a\t\tthe doctor away.\n"
An apple a day keeps            the doctor away.

chandana-galgali@chandana-galgali-VirtualBox:~$ 
```


chandana-galgali@chandana-galgali-VirtualBox: ~

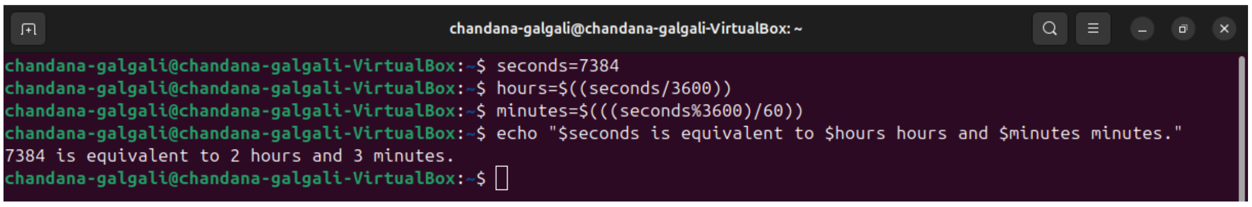
```
chandana-galgali@chandana-galgali-VirtualBox:~$ x=10
chandana-galgali@chandana-galgali-VirtualBox:~$ echo $x
10
chandana-galgali@chandana-galgali-VirtualBox:~$ xn=Rani
chandana-galgali@chandana-galgali-VirtualBox:~$ echo $xn
Rani
chandana-galgali@chandana-galgali-VirtualBox:~$ myname="Chandana Galgali"
chandana-galgali@chandana-galgali-VirtualBox:~$ echo "Name: $myname"
Name: Chandana Galgali
chandana-galgali@chandana-galgali-VirtualBox:~$ echo "Shell: $SHELL"
Shell: /bin/bash
chandana-galgali@chandana-galgali-VirtualBox:~$ echo "OS Type: $OSTYPE"
OS Type: linux-gnu
chandana-galgali@chandana-galgali-VirtualBox:~$ echo "Date: $(date)"
Date: Sat Aug 24 11:09:52 PM IST 2024
chandana-galgali@chandana-galgali-VirtualBox:~$ echo "Current Month: $(date +%B)"
Current Month: August
chandana-galgali@chandana-galgali-VirtualBox:~$ 
```



```
chandana-galgali@chandana-galgali-VirtualBox:~$ name1="Chandana Galgali"
chandana-galgali@chandana-galgali-VirtualBox:~$ rollno1=16010422234
chandana-galgali@chandana-galgali-VirtualBox:~$ marks1=99
chandana-galgali@chandana-galgali-VirtualBox:~$ name2="Mahek Thakkar"
chandana-galgali@chandana-galgali-VirtualBox:~$ rollno2=16010422235
chandana-galgali@chandana-galgali-VirtualBox:~$ marks2=100
chandana-galgali@chandana-galgali-VirtualBox:~$ name3="Samrudhi Bhat"
chandana-galgali@chandana-galgali-VirtualBox:~$ rollno3=16010422240
chandana-galgali@chandana-galgali-VirtualBox:~$ marks3=100
chandana-galgali@chandana-galgali-VirtualBox:~$ echo -e "Name: $name1\nRoll No.: $rollno1\nMarks: $marks1\n"
Name: Chandana Galgali
Roll No.: 16010422234
Marks: 99

chandana-galgali@chandana-galgali-VirtualBox:~$ echo -e "Name: $name2\nRoll No.: $rollno2\nMarks: $marks2\n"
Name: Mahek Thakkar
Roll No.: 16010422235
Marks: 100

chandana-galgali@chandana-galgali-VirtualBox:~$ echo -e "Name: $name3\nRoll No.: $rollno3\nMarks: $marks3\n"
Name: Samrudhi Bhat
Roll No.: 16010422240
Marks: 100

chandana-galgali@chandana-galgali-VirtualBox:~$ 
```



```
chandana-galgali@chandana-galgali-VirtualBox:~$ n1=160
chandana-galgali@chandana-galgali-VirtualBox:~$ n2=104
chandana-galgali@chandana-galgali-VirtualBox:~$ echo "Addition: $((n1+n2))"
Addition: 264
chandana-galgali@chandana-galgali-VirtualBox:~$ echo "Subtraction: $((n1-n2))"
Subtraction: 56
chandana-galgali@chandana-galgali-VirtualBox:~$ echo "Multiplication: $((n1*n2))"
Multiplication: 16640
chandana-galgali@chandana-galgali-VirtualBox:~$ echo "Division: $((n1/n2))"
Division: 1
chandana-galgali@chandana-galgali-VirtualBox:~$ echo "Modulus: $((n1%n2))"
Modulus: 56
chandana-galgali@chandana-galgali-VirtualBox:~$ 
```



```
chandana-galgali@chandana-galgali-VirtualBox:~$ seconds=7384
chandana-galgali@chandana-galgali-VirtualBox:~$ hours=$((seconds/3600))
chandana-galgali@chandana-galgali-VirtualBox:~$ minutes=$(((seconds%3600)/60))
chandana-galgali@chandana-galgali-VirtualBox:~$ echo "$seconds is equivalent to $hours hours and $minutes minutes."
7384 is equivalent to 2 hours and 3 minutes.
chandana-galgali@chandana-galgali-VirtualBox:~$ 
```

Outcomes: CO4 - Demonstrate open source standards usage.

Conclusion:
In this experiment, we explored the basics of shell scripting in a Linux environment. The experiment provided an introduction to creating and running shell scripts, setting up executable permissions, debugging scripts, and working with variables. We also learned how to make scripts interactive by taking user input and performing arithmetic operations. The practical exercises helped solidify our understanding of shell scripting fundamentals, including variable declaration, echo command usage, and arithmetic operations within the shell. This knowledge is crucial for automating tasks and managing systems in a Linux environment.

Grade: AA/AB/BB/BC/CC/CD/DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Richard Blum and Christine Bresnahan, “Linux Command Line & Shell Scripting”, II Edition, Wiley, 2012.
