SHELL Programming (Part I)

LAB # 02

Spring 2023

CSE-204L Operating Systems Lab

Submitted by: **Ali Asghar**

Registration No.: **21PWCSE2059**

Class Section: **C**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Submitted to:

Engr. Madiha Sher

Date:

8th March 2023

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

OBJECTIVES:

In this lab we learnt about.

- Understanding what is a SHELL script
    - What is a SHELL script
    - Different kinds of SHELLs in UNIX
- Why and where it is used.
- First simple SHELL script
- SHELL variables
    - User defined variables
    - System variables
    - Read only variables and wiping out variables
    - Assigning values to variables
    - Reading input

INTRODUCTION

SHELL

A shell is a user interface used in computers that gives users access to the features of an operating system. By typing commands or running scripts and then receiving output, users can communicate with the computer system through the shell.

TYPES OF SHELL

Shells can be classified as either command-line shells or graphical user interface (GUI) shells. For entering commands and viewing output, command-line shells offer a text-based interface, whereas GUI shells offer a graphical interface with icons and menus. The most popular command-line shell for Unix-based systems is the Bash shell (short for Bourne-Again SHell), which is also accessible on Windows operating systems with the help of programmes like Cygwin or the Windows Subsystem for Linux (WSL).

SHELL SCRIPTS

A shell script or a shell program is a series of commands put in a file and executed by the Shell. We will use shell to create shell scripts.

WHY SHELL SCRIPTS?

Since the user cannot interact with the kernel directly, Shell programming skills are a must to be able to exploit the power of UNIX to the fullest extent. A shell script can be used for variety of tasks and some of them are listed below.

USES OF SHELL SCRIPTS

1. Customizing your work environment. For Example, Every time you login, if you want to see the current date, a welcome message, and the list of users who have logged in you can write a shell script for the same.
2. Automating your daily tasks. For example, to back up all the programs at the end of the day.
3. Automating repetitive tasks.
4. Executing important system procedures, like shutting down the system, formatting a disk, creating a file system etc.
5. Performing some operations on many files.

SHELL VARIABLES

The variables in the Shell are classified as:

User defined variables: defined by the user for his use (e.g. age=32).

Environmental variables: defined by shell for its own operations (PATH, HOME, TERM, LOGNAME, PS1, SHELL etc.).

Predefined variables: reserved variables used by the shell and UNIX commands for specifying the exit status of command, arguments to the shell scripts, the formal parameters etc.

EXAMPLE A:

```
name=Ali

echo $name

echo Hello $name ! , Welcome to $HOME
```

EXAMPLE 1:

```
# SS1

# Usage: SS1

ls

who

pwd
```

Save the file and type the file name in command line and the file is executed as follows.

```
console> SS1

SS1: Command not found.
```

The reason for this message is, the path of this command should be specified.

```
console> ./SS1

SS1: Permission Denied.
```

Now you do not have the permission to execute the file. That is the default permission given for any file is with out execute permission. How to know the default permission? Type the command umask. This command will give the masked permissions of a file while creation.

Change the permissions using chmod command and execute the file again.

To read standard input into a shell script use the read command. For example:

EXAMPLE 2:

```
# SS2

# Usage: SS2

# An interactive shell script

echo What is your name\?

read name

echo Hello $name. Assalam-o-Alaikum.
```

This prompts the user for input, assigns this to the variable name and then displays the value of this variable to standard output.

If there is more than one word in the input, each word can be assigned to a different variable. Any words left over are assigned to the last named variable. For example:

EXAMPLE 3:

```
# SS3
# Usage: SS3
echo "Please enter your surname\n"
echo "followed by your first name: \c"
read name1 name2
echo "Welcome to CSE Dept., UET, $name2 $name1"
```

EXAMPLE4:

```
# SS4
# Usage: SS4
# This script takes two file names and copies the first file into the second one
echo “Please Enter source file name: \c”
read source
echo “Enter the target file name :\c”
read target
cp $source $target
echo file $source  is copied into the $target
```

COMMAND SUBSTITUTION:

Format for command substitution is:

var=`command` (where ‘ ‘ is back quote)

EXAMPLE B:

```
echo ‘date’      # It will display  the output of date command
echo there are ‘who | wc –l’ users working on the system    # see output of this
```

ARITHMETIC IN SHELL SCRIPT:

Various forms for performing computations on shell variables using expr command are:

```
expr val_1  op  val_2  (Where op is operator)
expr $val_1  op  $val_2
val_3=’expr $val_1 op $val_2`
```

EXAMPLE C:

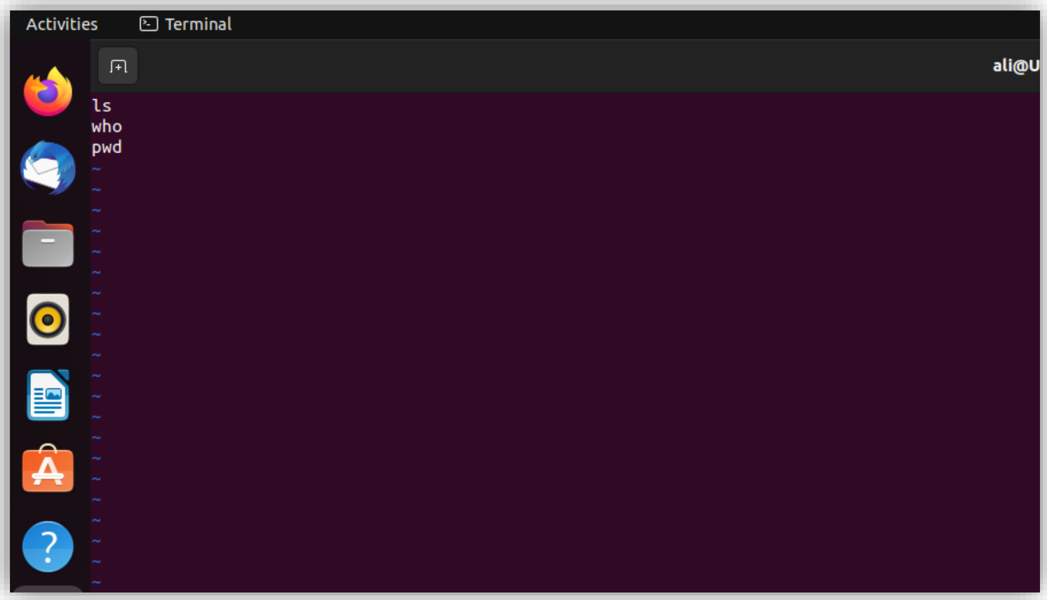
```
expr 5 + 7        # Gives 12
expr 6 – 3        # Gives 3
expr 3 \* 4      # Gives  12
expr 24 / 3      # Gives  8

sum=’expr 5 + 6’
echo $sum         # Gives 11
```

EXAMPLE 5:

```
# SS5
# Usage: SS5
a=12
b=90
echo sum is $a + $b          # Will display sum is 12 + 90
echo sum is `expr $a + $b`   # Gives sum is 102
```

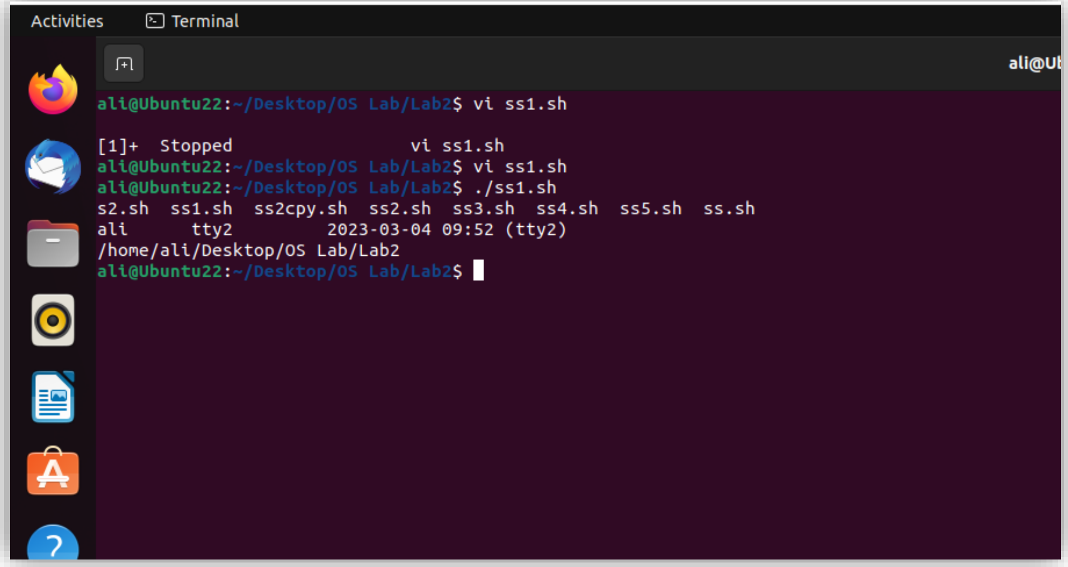
RESULTS AND EXPLANATION:

EXAMPLE 1:



```
ls
who
pwd
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Figure 1-1: Code Screenshot of Example 1



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi ss1.sh

[1]+  Stopped                 vi ss1.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi ss1.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./ss1.sh
s2.sh  ss1.sh  ss2cpy.sh  ss2.sh  ss3.sh  ss4.sh  ss5.sh  ss.sh
ali      tty2         2023-03-04 09:52 (tty2)
/home/ali/Desktop/OS Lab/Lab2
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ 
```

Figure 1-2: Output of Example 1

CODE EXPLANATION:

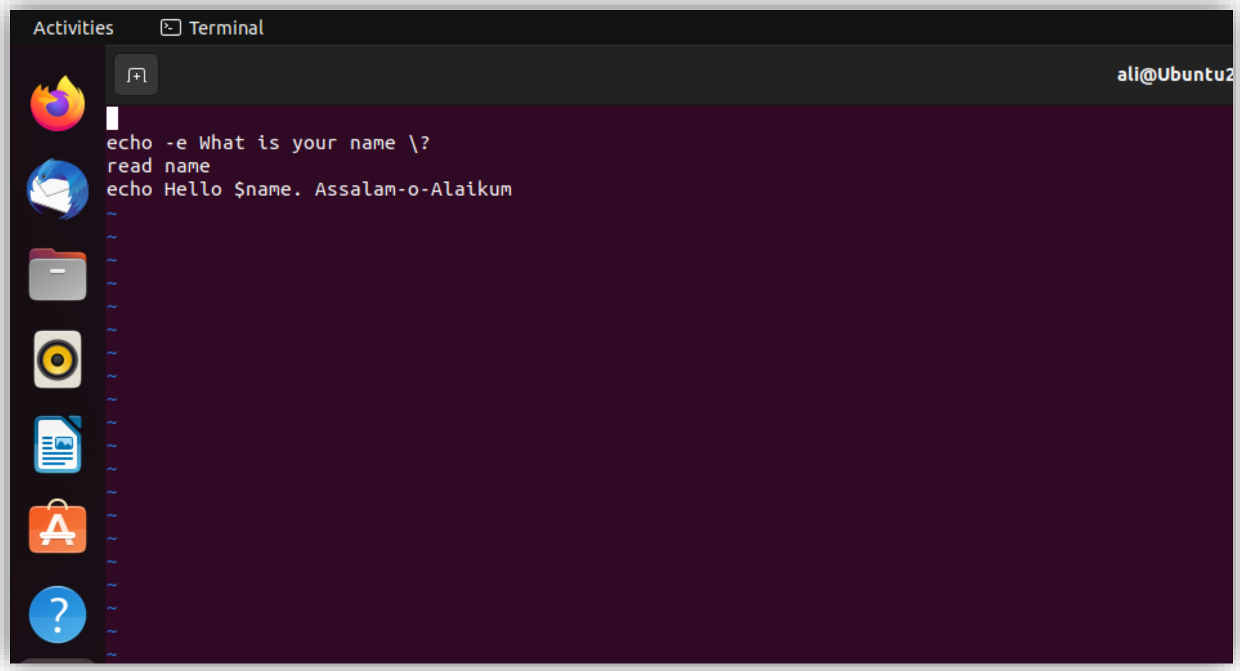
Figure 1-1 shows the code of Example 1. It shows a basic shell script which execute the following commands sequentially.

1. **ls** command which displays the contents of the current working directory.
2. **who** command displays the name of the currently logged in user**.**
3. **pwd** command displays the name of the current working directory.

OUTPUT EXPLANATION:

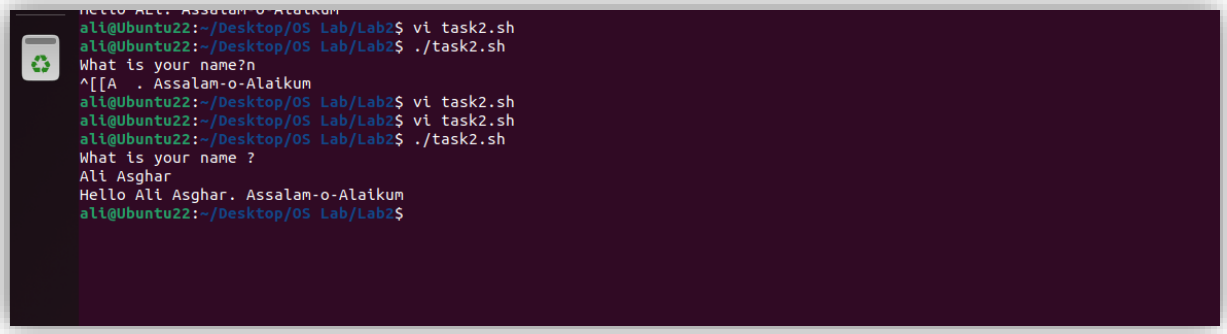
When this script is executed, it will run these commands sequentially and the output of each command will be displayed on terminal screen. Output is shown in Figure 1-2

EXAMPLE 2:



```
echo -e What is your name \?
read name
echo Hello $name. Assalam-o-Alaikum
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Figure 2-1: Code of Example 2



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi task2.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./task2.sh
What is your name?n
^[[A  . Assalam-o-Alaikum
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi task2.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi task2.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./task2.sh
What is your name ?
Ali Asghar
Hello Ali Asghar. Assalam-o-Alaikum
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$
```

Figure 2-2: Output of Example 2

CODE EXPLANATION:

Figure 2-1 shows the code of Example 2. It shows a basic shell script which prompt the user to enter his/her name and then display it on terminal screen.

In this code, First I displayed a message on terminal window using **echo command**. Then I read user input using the built-in **read command**. This command takes one or more than arguments from the user. In this case, I have passed a user-defined variable **name** to it. Finally, the output is displayed using the **echo command.**

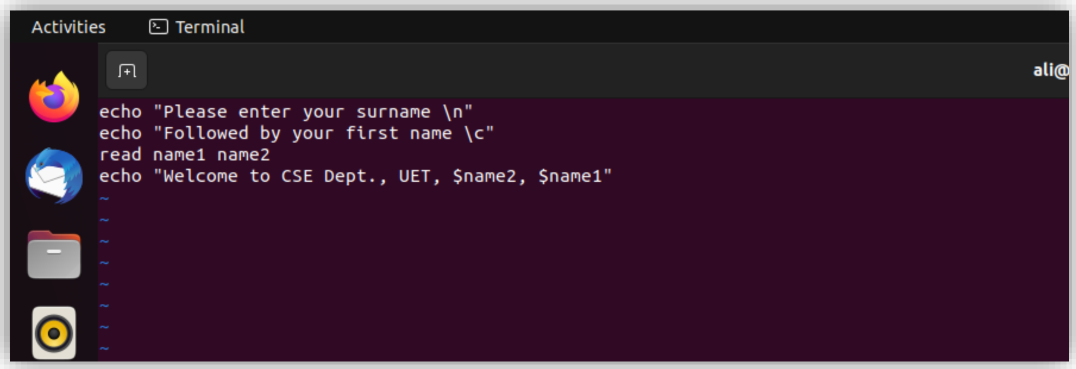
OUTPUT EXPLANATION:

When the shell script shown in Figure 2-1 is executed, it will first display the message “What is your name?” and then it will wait for the user input to enter a name. After the user inputs its name, it will display another message. This message will be like:

Hello + input name + Assalam-o-Alaikum

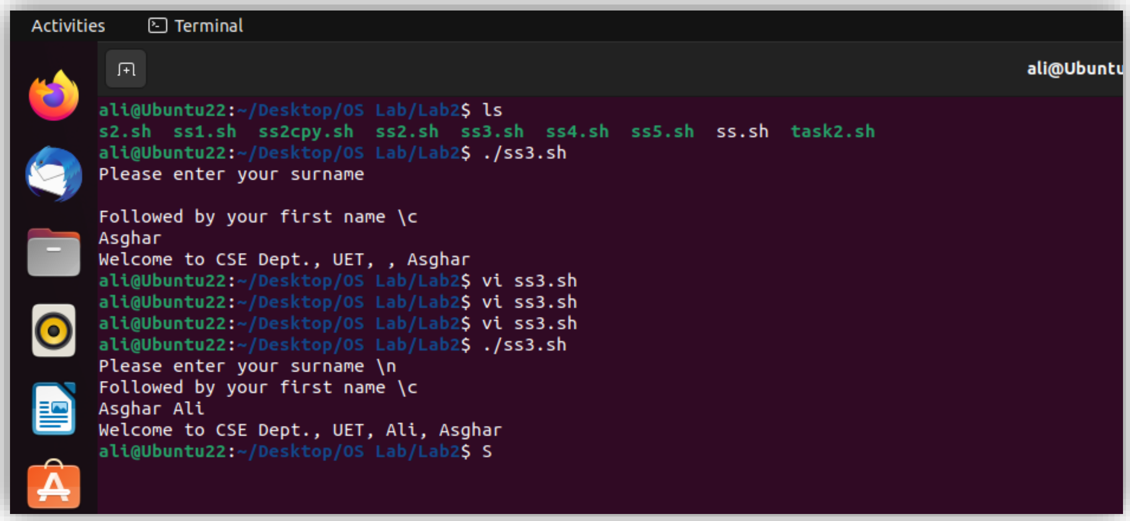
Output can be seen in Figure 2-2.

EXAMPLE 3:



```
echo "Please enter your surname \n"
echo "Followed by your first name \c"
read name1 name2
echo "Welcome to CSE Dept., UET, $name2, $name1"
~
~
~
~
~
~
~
~
```

Figure 3-1: Code of Example 3



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ls
s2.sh  ss1.sh  ss2cpy.sh  ss2.sh  ss3.sh  ss4.sh  ss5.sh  ss.sh  task2.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./ss3.sh
Please enter your surname

Followed by your first name \c
Asghar
Welcome to CSE Dept., UET, , Asghar
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi ss3.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi ss3.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi ss3.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./ss3.sh
Please enter your surname \n
Followed by your first name \c
Asghar Ali
Welcome to CSE Dept., UET, Ali, Asghar
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ S
```

Figure 3-2: Output of Example 3

CODE EXPLANATION:

Figure 3-1 shows the code of Example 3. This code is somehow similar to code shown in figure 2-1 but with a slight difference. It asks for two name inputs and then display them in a specific manner.

In this code, First I displayed a message on terminal window using **echo command**. Then I read user input using the built-in **read command**. This command takes one argument. In this case, I have passed two user-defined variable **name1** & **name2** to it. Finally, the output is displayed using the **echo command.**

This task is a little different than task 2 in terms of user input. This script (when executed) will take two inputs from the user i-e surname and first name. Then it will display them in a specific order such that **surname** is shown at first position and **first name** is shown at second position.

OUTPUT EXPLANATION:

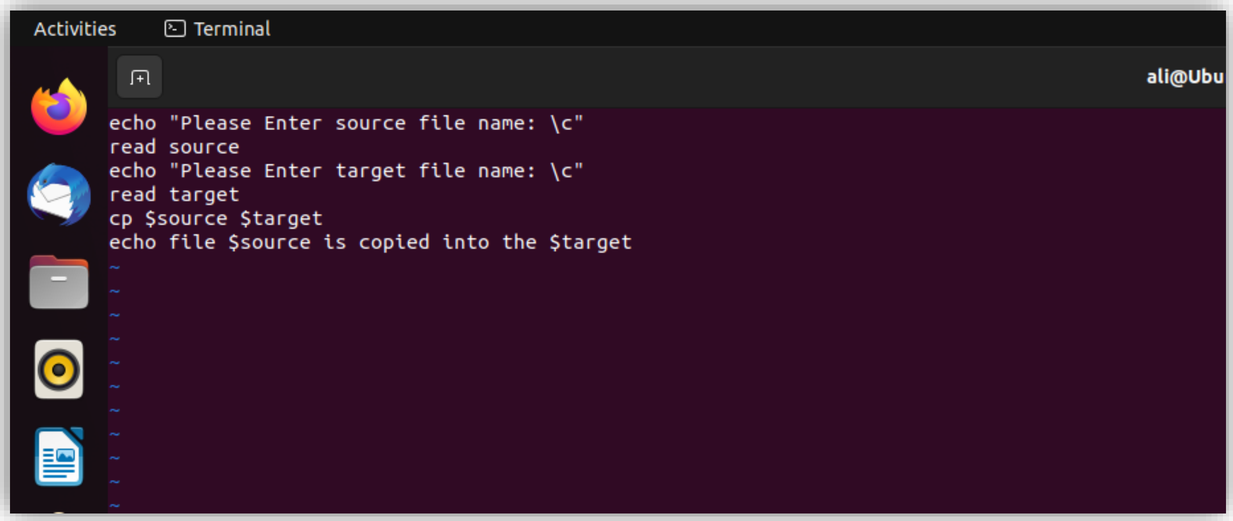
When the shell script shown in Figure 3-1 is executed, it will first display the message

Please enter your surname \n

Followed by your first name \c

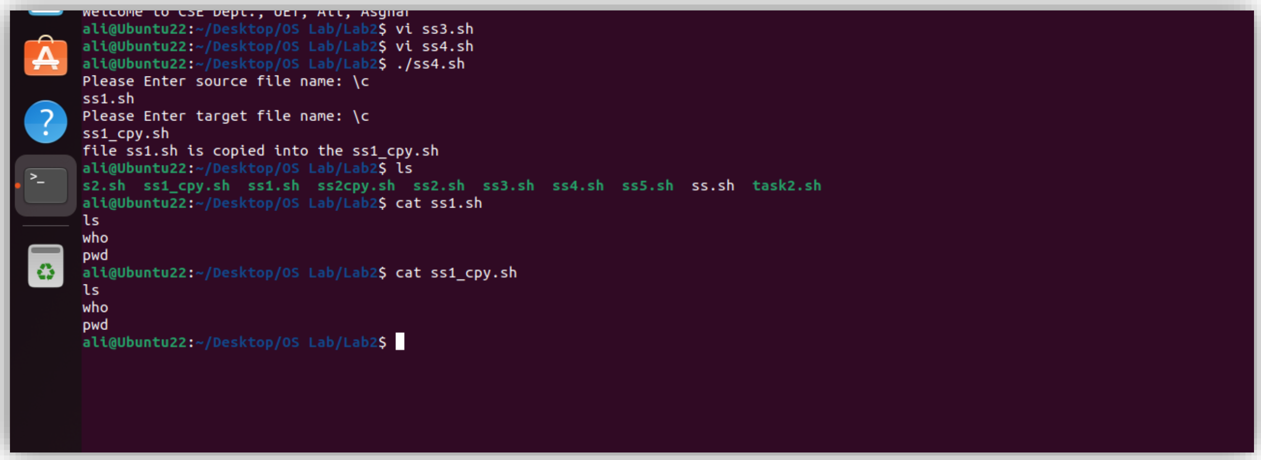
Then it will wait for user input and when user give two names in the input(i-e surname followed by first name). Finally, it will display the output which will contain the **surname** and **first name** given by user. Output can be seen in Figure 3-2.

EXAMPLE 4:



```
echo "Please Enter source file name: \c"
read source
echo "Please Enter target file name: \c"
read target
cp $source $target
echo file $source is copied into the $target
~
~
~
~
~
~
~
~
~
~
~
```

Figure 4-1: Code of Example 4



```
Welcome to CSE Dept., UET, Ali, Asghar
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi ss3.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi ss4.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./ss4.sh
Please Enter source file name: \c
ss1.sh
Please Enter target file name: \c
ss1_cpy.sh
file ss1.sh is copied into the ss1_cpy.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ls
s2.sh  ss1_cpy.sh  ss1.sh  ss2cpy.sh  ss2.sh  ss3.sh  ss4.sh  ss5.sh  ss.sh  task2.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ cat ss1.sh
ls
who
pwd
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ cat ss1_cpy.sh
ls
who
pwd
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ 
```

Figure 4-2: Output of Example 4

CODE EXPLANATION:

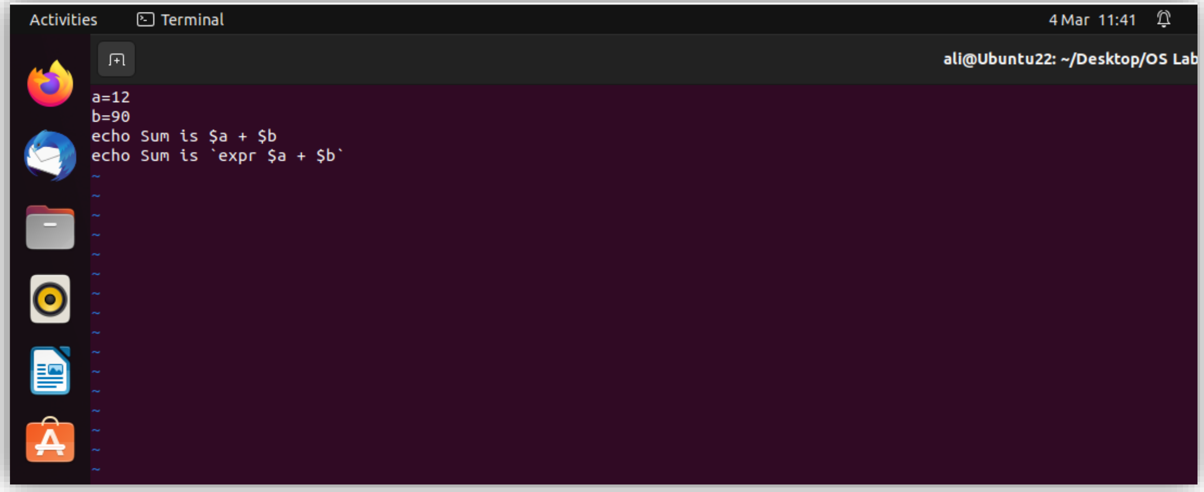
Figure 4-1 shows the code of Example 4. This code implements the copy paste functionality.

In this code, First I displayed a message on terminal window using **echo command**. Then I read user input using the built-in **read command** to get the exact location of file to be copied. Then I prompted the user to enter the destination location to paste the file in that location. Finally, the output is displayed using the **echo command.** After getting the address of these two locations, I used the built-in copy command **cp** and passed the two locations to perform copy paste operation. Finally, I prompt the message to display the success of this operation.

OUTPUT EXPLANATION:

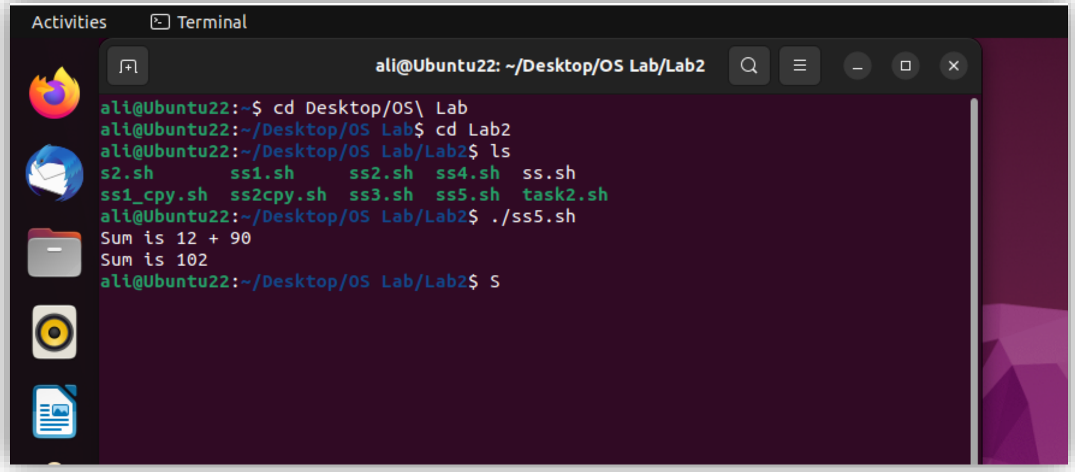
Output of code in Figure 4-1 can be seen in Figure 4-2. It performs the function as explained the above code explanation.

EXAMPLE 5:



```
a=12
b=90
echo Sum is $a + $b
echo Sum is `expr $a + $b`
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Figure 5-1: Code of Example 5



```
ali@Ubuntu22:~$ cd Desktop/OS\ Lab
ali@Ubuntu22:~/Desktop/OS Lab$ cd Lab2
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ls
s2.sh       ss1.sh     ss2.sh  ss4.sh  ss.sh
ss1_cpy.sh  ss2cpy.sh  ss3.sh  ss5.sh  task2.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./ss5.sh
Sum is 12 + 90
Sum is 102
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ S
```

Figure 5-2: Output of Example 5

CODE EXPLANATION:

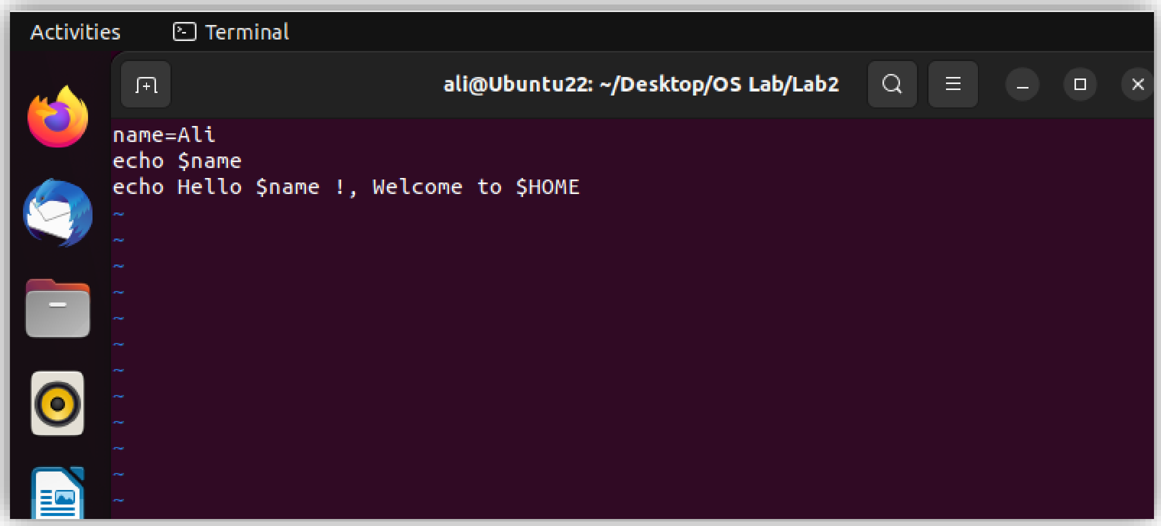
Figure 5-1 shows the code of Example 5. This code implements basic arithmetic operation(addition). It initialize two variables **a** and **b** and assign value of 12 and 90. Then I displayed the output using **echo command.** At first, I concatenated the values of **a** and **b** and after that I displayed the arithmetic sum using **`expr`** keyword.

OUTPUT EXPLANATION:

Output of code in Figure 4-1 can be seen in Figure 4-2. It performs the function as explained the above code explanation i-e arithmetic addition of **a** and **b.**

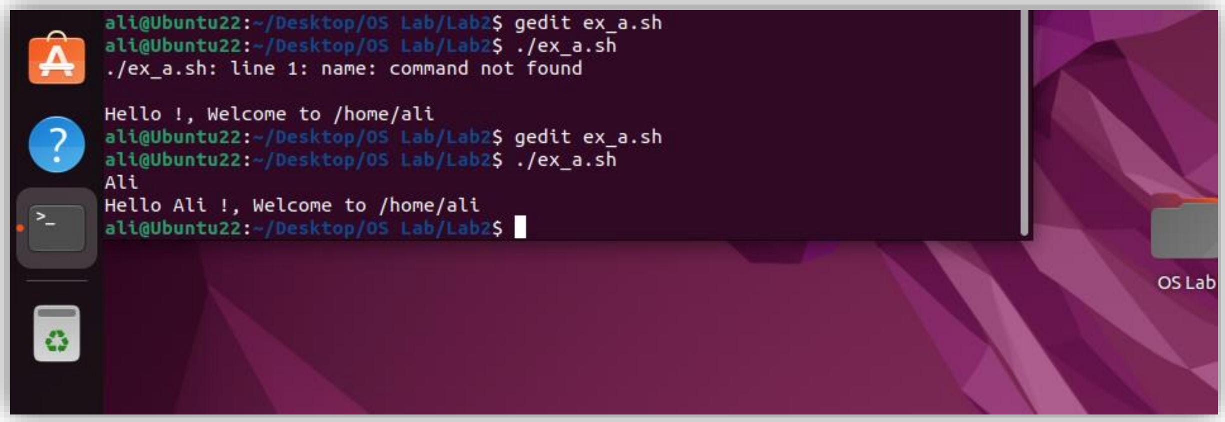
OTHER EXAMPLES:

EXAMPLE A:



```
name=Ali
echo $name
echo Hello $name !, Welcome to $HOME
~
~
~
~
~
~
~
~
~
~
~
~
```

Figure 6-1: Code of Example A



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ gedit ex_a.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./ex_a.sh
./ex_a.sh: line 1: name: command not found

Hello !, Welcome to /home/ali
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ gedit ex_a.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./ex_a.sh
Ali
Hello Ali !, Welcome to /home/ali
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$
```

Figure 6-2: Output of Example A

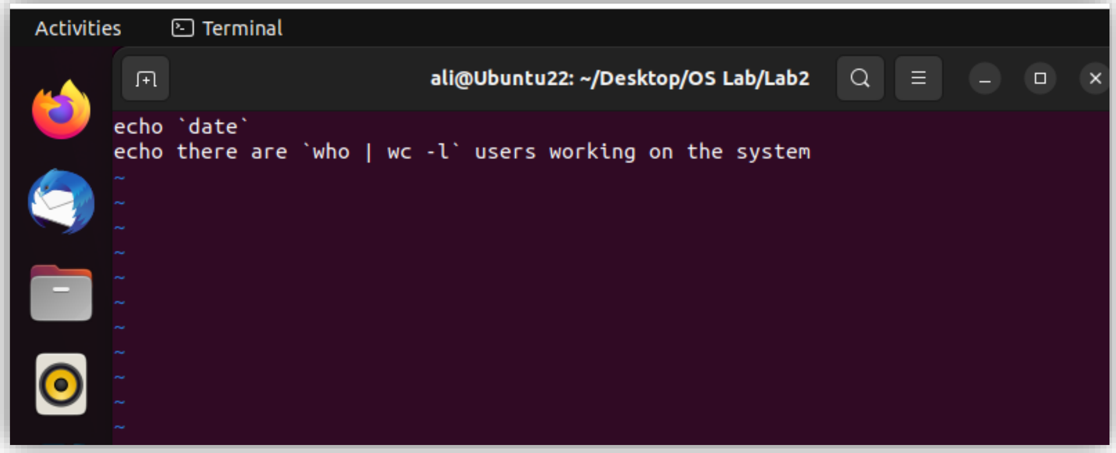
CODE EXPLANATION:

Figure 6-1 shows the code of Example A. This code print out the **user's name** and the **location** of their **home** directory. I set the variable "name" to "Ali" in the first line. The value of the "name" variable, which is "Ali," is printed out in the second line using the "echo" command. The path to the user's home directory, which is denoted by the **"$HOME"** environment variable, as well as the value of the "name" variable are printed out in the third line using the "echo" command once more.

OUTPUT EXPLANATION:

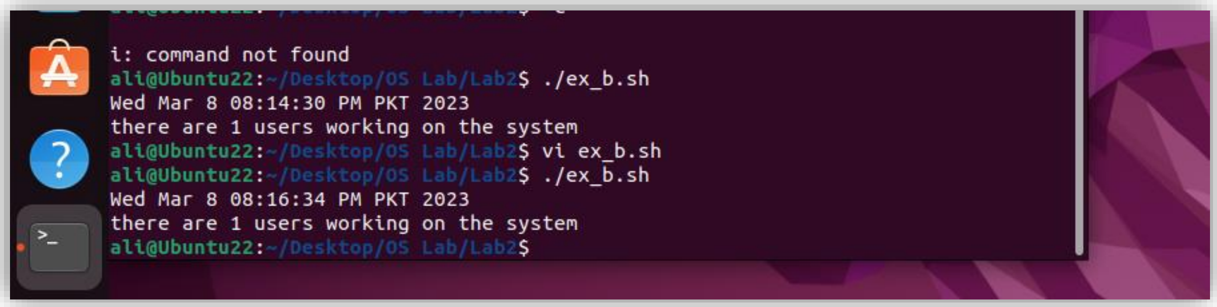
Output of code in Figure 6-1 can be seen in Figure 6-2.

EXAMPLE B:



```
echo `date`
echo there are `who | wc -l` users working on the system
~
~
~
~
~
~
~
~
~
~
~
```

Figure 7-1: Code of Example B



```
i: command not found
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./ex_b.sh
Wed Mar 8 08:14:30 PM PKT 2023
there are 1 users working on the system
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi ex_b.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./ex_b.sh
Wed Mar 8 08:16:34 PM PKT 2023
there are 1 users working on the system
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$
```

Figure 7-2: Output of Example B

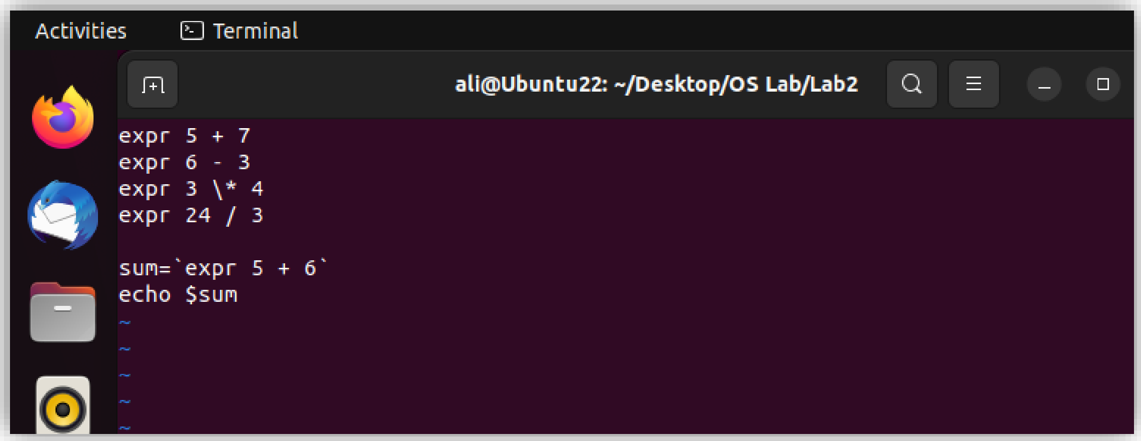
CODE EXPLANATION:

Figure 7-1 shows the code of Example B. This code displays some basic information about the current date and the number of **users logged** in to the **system**. In first line, I displayed **current date** and **time** using date command. In second line, I displayed the string "there are" followed by the output of the "**who | wc -l" command**, which **counts** the number of **users** currently logged in to the system.

OUTPUT EXPLANATION:

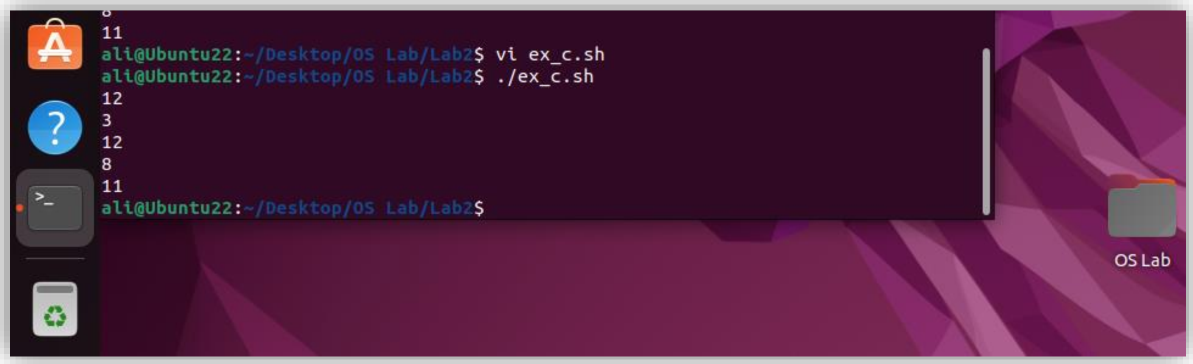
Output of code in Figure 7-1 can be seen in Figure 7-2.

EXAMPLE C



```
expr 5 + 7
expr 6 - 3
expr 3 \* 4
expr 24 / 3

sum=`expr 5 + 6`
echo $sum
~
~
~
~
~
```

Figure 8-1: Code of Example C



```
11
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ vi ex_c.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$ ./ex_c.sh
12
3
12
8
11
ali@Ubuntu22:~/Desktop/OS Lab/Lab2$
```

Figure 8-1: Output of Example C

CODE EXPLANATION:

Figure 8-1 shows the code of Example C. This is script uses the expr command to carry out simple arithmetic operations and saves the outcome of an expression in a variable.

The script's first four lines apply arithmetic operations on two numbers using the expr command, then output the outcome on the command line. Specifically:

expr (5 + 7) will add (5 + 7) and display (12) as the result.

expr 6 - 3 will display the result, which is 3, after subtracting 3 from 6.

expr 3 * 4 will add 3 and 4 together and show the result, which is 12.

expr 24 / 3 will give the outcome, which is 8 after dividing 24 by 3.

At last, we save some sum value in sum variable and then display it’s value using echo command.

OUTPUT EXPLANATION:

Output of code in Figure 8-1 can be seen in Figure 8-2.

CONCLUSION:

In this lab, I learned about **basic scripting** in **bash shell**. I learned how to **read** data from **user** using **read command** and how to **display** data using **echo command**. I also learned **arithmetic operations** in **shell scripting**. Finally, I learned how to make a shell script which can perform the **copy operation**.