SHELL Programming (Part II)

LAB # 03

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:

14th March 2023

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

OBJECTIVES:

The aim of this laboratory is to learn and practice SHELL scripts by writing small SHELL programs.
The following are the primary objectives of this lab session:

- SHELL keywords
- Arithmetic in SHELL script
- Control Structures
  - Decision control
  - Repetition control
- More UNIX commands
- Executing commands during login time

INTRODUCTION:

HANDLING SHELL VARIABLES

The shell has several variables which are automatically set whenever you login. The values of some of these variables are stored in names which collectively are called your user environment. Any name defined in the user environment, can be accessed from within a shell script. To include the value of a shell variable into the environment you must export it.

Special shell variables/Pre-defined shell variables (Parameters to shell scripts):

There are some variables which are set internally by the shell and which are available to the user:

Name	Description
$1 - $9	these variables are the positional parameters (Positional parameters).
$0	the name of the command currently being executed (The command name).
$#	number of positional arguments given to this invocation of the shell.
$$	the process number of this shell
$*	a string containing all the arguments to the shell, starting at $1 (All parameters).
$@@	same as above, except when quoted.

Passing arguments to the shell

Shell scripts can act like standard UNIX commands and take arguments from the command line. Arguments are passed from the command line into a shell program using the positional parameters $1 through to $9. Each parameter corresponds to the position of the argument on the command line. The positional parameter $0 refers to the command name or name of the executable file containing the shell script. Only nine command line arguments can be accessed, but you can access more than nine using the shift command.

All the positional parameters can be referred to using the special parameter $*. This is useful when passing filenames as arguments.

Examples of passing arguments to the shell

Write shell script which will accept 5 numbers as parameters and display their sum. Also display the contents of the different variables in the script.

Example1:

```
# Usage: SS1 param1 param2 param3 param 4 param5

# Script to accept 5 numbers and display their sum.

echo the parameters passed are : $1, $2, $3, $4, $5

echo the name of the script is : $0

echo the number of parameters passed are : $#

sum=`expr $1 + $2 + $3 + $4 + $5`

echo The sum is : $sum
```

Why need of shift command?

If more than 9 parameters are passed to a script, it is not possible to refer to the parameters beyond the 9th one. This is because shell accepts a single digit following the dollar sign as a positional parameter definition.

The shift command is used to shift the parameters one position to the left. On the execution of shift command the first parameter is overwritten by the second, the second by third and so on. This implies, that the contents of the first parameter are lost once the shift command is executed.

Example of shift:

Write a script which will accept different numbers and finds their sum. The number of parameters can vary.

```
sum=0
```

```
while [ $# -gt 0 ]
do
  sum='expr $sum + $1'
  shift
done
echo sum is $sum
```

Here, the parameter $1 is added to the variable sum always. After shift, the value of $1 will be lost and the value of $2 becomes the value of $1 and so on.

The above script can also be written without using the shift command as:

```
for i in $*
do
  sum='expr $sum + $i'
done
```

Usually only nine command line arguments can be accessed using positional parameters. The shift command gives access to command line arguments greater than nine by shifting each of the arguments.

The second argument ($2) becomes the first ($1), the third ($3) becomes the second ($2) and so on. This gives you access to the tenth command line argument by making it the ninth. The first argument is no longer available.

Successive shift commands make additional arguments available. Note that there is no "unshift" command to bring back arguments that are no longer available!

Another Example of using the shift Command

To successively shift the argument that is represented by each positional parameter:

Example 3:

```
  #Usage: SS3 param1 param2 param3 param4 param5 param6 param7 param8
```

```
          param9 param10 param11 param12
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
```

CONTROL STRUCTURES

Every UNIX command returns a value on exit which the shell can interrogate. This value is held in the read-only shell variable $?.

A value of 0 (zero) signifies success; anything other than 0 (zero) signifies failure.

The if statement

The if statement uses the exit status of the given command and conditionally executes the statements following. The general syntax is:

```
if test
then
  commands    (if condition is true)
else
  commands    (if condition is false)
fi
```

then, else and fi are shell reserved words and as such are only recognized after a new line or ; (semicolon). Make sure that you end each if construct with a fi statement.

Nested if statement :

```
if ...
then ...
else if ...
    ...
```

```
   fi
  fi
```

The elif statement can be used as shorthand for an else if statement. For example:

```
  if ...
  then ...
  elif ...
    ...
  fi
```

TEST COMMAND:

The UNIX system provides test command which investigates the exit status of the previous command and translate the result in the form of success or failure, i.e. either a 0 or 1.

The test command does not produce any output, but its exit status can be passed to the if statement to check whether the test failed or succeeded.

How to know exit status of any command?

All commands return the exit status to a pre-defined Shell Variable '?'. Which can be displayed using the echo command.

e.g.

```
        echo $?
```

If output of this is 0 (Zero) it means the previous command was successful and if output is 1 (One) it means previous command failed.

The test command has specific operators to operate on files, numeric values and strings which are explained below:

Operators on Numeric Variables used with test command:

-eq : equal to

-ne : not equals to

-gt : grater than

-lt : less than

-ge : greater than or equal to

-le : less than equal to

Examples:

```
 a=12; b=23
 test $a –eq $b
 echo $?     # Gives 1 (one) as output.(Indicates exit status  false)
```

Operators on String Variables used with test command:

= : equality of strings

!= : not equal

-z : zero length string (i.e. string containing zero character i.e. null string).

-n : String length is non zero.

Examples:

```
 name=”Ahmad”
 test –z $name           # will return the exit status 1 as the string name is not null.
 test –n $name           # will return 0 as the string is not null.
 test –z “$address”     # will return 0 as the variable has not been defined.
 test $name = “Ali”    # will return 1 as the value of name is not equal to “Ali”
```

Operators on files used with test command:

-f : the file exists.

-s : the file exists and the file size is non zero.

-d : directory exists.

-r : file exits and has read permission.

-w : file exists and has write permission.

-x : file exists and has execute permission.

Examples:

```
test –f “mydoc.doc”      # Will check for the file mydoc.doc, if exists, returns 0 else 1.
test –r “mydoc.doc”       # Will check for read permission for mydoc.doc
test –d “$HOME”         # Will check for the existence of the users home directory.
```

Logical Operators used with test command:

Combining more than one condition is done through the logical AND, OR and NOT operators.

-a : logical AND

-o : logical OR

! : logical NOT

Example:

```
 test –r “mydoc.doc”  –a   –w “mydoc.doc”   # Will check both the read and write permission for the file mydoc.doc and returns either 0 or 1 depending on result.
```

Example of using an if construct:

To carry out a conditional action:

Example 4:

```
a=10
b=20
if [ $a == $b ]
then
   echo "a is equal to b"
elif [ $a -gt $b ]
```

```
then
  echo "a is greater than b"
elif [ $a -lt $b ]
then
  echo "a is less than b"
else
  echo "None of the condition met"
fi
```

Example 5:

```
 if who | grep -s student > /dev/null
  then
    echo student is logged in
  else
    echo student is not available
 fi
```

This lists who is currently logged on to the system and pipes the output through grep to search for the username student.

The -s option causes grep to work silently and any error messages are directed to the file /dev/null instead of the standard output.

If the command is successful i.e. the username student is found in the list of users currently logged in then the message student is logged on is displayed, otherwise the second message is displayed

FLOW OF CONTROL STATEMENTS:

The Bourne shell provides several flow of control statements. Select an item for further information.

The case statement:

The case statement case is a flow control construct that provides for multi-way branching based on patterns.

Program flow is controlled on the basis of the wordgiven. This word is compared with each pattern in order until a match is found, at which point the associated command(s) are executed.

```
  case word in
  pattern1)        command(s) ;;
  pattern2)        command(s) ;;
  -----------------------
  -----------------------
  patternN)        command(s) ;;
  *) default       command   ;;
  esac
```

When all the commands are executed control is passed to the first statement after the esac. Each list of commands must end with a double semi-colon (;;).

A command can be associated with more than one pattern. Patterns can be separated from each other by a | symbol.

For example:

```
  case word in
  pattern1|pattern2) command
  ...     ;;
```

Patterns are checked for a match in the order in which they appear. A command is always carried out after the first instance of a pattern.

The * character can be used to specify a default pattern as the * character is the shell wildcard character.

Example 6:

```
# Display a menu of options and depending upon the user's choice,

#execute associated command

#Display the options to the users

clear

echo "1. Date and time"

echo

echo "2. Directory listing"

echo

echo "3. Users information”

echo

echo "4. Current Directory"

echo

echo "Enter choice (1,2,3 or 4 ) :\c"

read choice

case $choice in

1)     date;;

2)     ls -l;;

3)     who ;;

4)        pwd ;;

*)     echo wrong choice;;
```

The for statement:

The for loop notation has the general form:

```
for var in list-of-words
do
  commands
done
```

commands is a sequence of one or more commands separated by a newline or ; (semicolon).

The reserved words do and done must be preceded by a newline or ; (semicolon). Small loops can be written on a single line.

For example:

```
for var in list; do commands; done
```

Examples of using the for statement :

To take each argument in turn and see if that person is logged onto the system.

Example 7:

```
# see if a number of people are logged in
for i in $*
do
  if who | grep -s $i > /dev/null
  then
    echo $i is logged in
  else
    echo $i not available
  fi
done
```

For each username given as an argument an if statement is used to test if that person is logged on and an appropriate message is then displayed.

The while and until statements:

The while statement has the general form:

```
  while command-list1
  do
    command-list2
  done
```

The commands in command-list1 are executed; and if the exit status of the last command in that list is 0 (zero),

the commands in command-list2 are executed.

The sequence is repeated as long as the exit status of command-list1 is 0 (zero).

The until statement has the general form:

```
  until command-list1
  do
    command-list2
  done
```

This is identical in function to the while command except that the loop is executed as long as the exit status of

command-list1 is non-zero.

The exit status of a while/until command is the exit status of the last command executed in command-list2. If no such command list is executed, a while/until has an exit status of 0 (zero).

The break and continue statements:

It is often necessary to handle exception conditions within loops. The statements break and continue are used for this.

The break command terminates the execution of the innermost enclosing loop, causing execution to resume after the nearest done statement.

To exit from n levels, use the command:

```
break n
```

This will cause execution to resume after the done n levels up.

The continue command causes execution to resume at the while, until or for statement which begins the loop containing the continue command.

You can also specify an argument n|FR to continue which will cause execution to continue at the n|FRth enclosing loop up.

Example of using the break and continue statements

Example 8:

```
while echo "Please enter command"
read response
do
  case "$response" in
  'done') break          # no more commands
           ;;
  "")    continue        # null command
           ;;
```

```
   *)      eval $response  # do the command
                   ;;
   esac
  done
```

This prompts the user to enter a command. While they enter a command or null string the script continues to run. To stop the command the user enters done at the prompt.

Some more examples for writing shell scripts :

```
#SS9
# To show use of case statement .
echo What kind of tree bears acorns\ ?
read responce
case $responce in
[Oo][Aa][Kk]) echo $responce is correct ;;
*) echo Sorry, responce  is wrong
esac
```

```
#SS10
# To show use of while statement
clear
echo What is the Capital of Saudi Arabia \?
read answer
while test $answer != Riyadh
do
echo No, Wrong please try again.
```

```
read answer
done
echo This is correct.

#SS11
# Example to show use of until statement
# Accept the login name from the user
clear
echo "Please Enter the user login name: \c"
read login_name
until who | grep $login_name
do
   sleep 30
done
echo The user $login_name has logged in

#SS12
#To show use of  if  statement
# Read three numbers and display largest
clear
echo "Enter the first number :\c"
read num1

echo "Enter the second number :\c"
read num2

echo "Enter the third number :\c"
read num3
```

```
if test $num1 -gt $num2
then
     if test  $num1 -gt $num3
        then
             echo $num1 is the largest
    else
            echo $num3 is the largest
    fi
else
       if test  $num2 -gt $num3
          then
              echo $num2 is largest
       else
              echo $num3 is the largest
       fi
fi
```

Assignment Problems on UNIX SHELL programming

1. Run all the programs given in the Lab Notes, and observe the output for each program.

2. Write a shell script that takes a keyword as a command line argument and lists the filenames containing the keyword

3. Write a shell script that takes a command line argument and reports whether it is a directory, or a file or a link.

4. Write a script to find the number of sub directories in a given directory.

5. Write a menu driven program that has the following options.

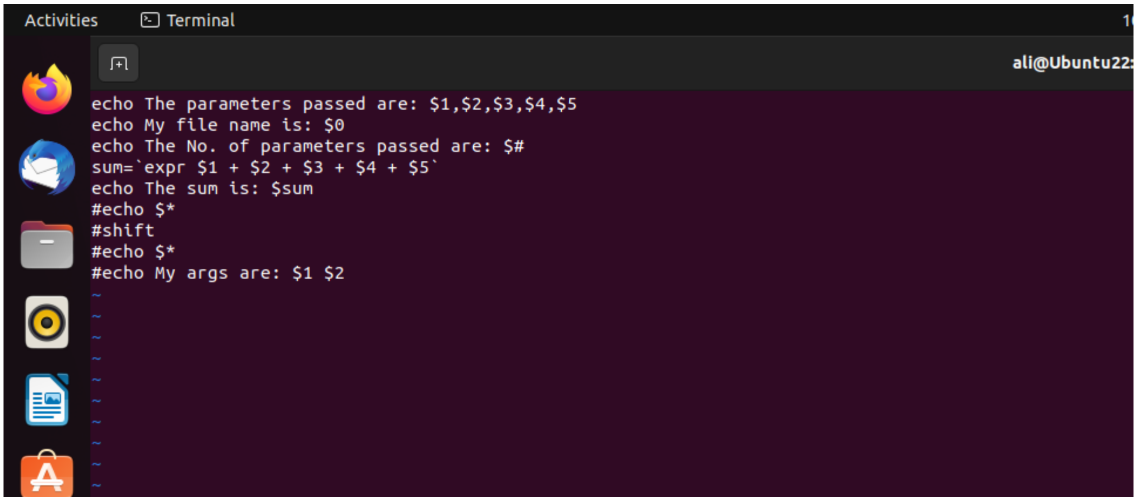
5.1. Search a given file is in the directory or not.

5.2. Display the names of the users logged in.

RESULTS AND EXPLANATION:

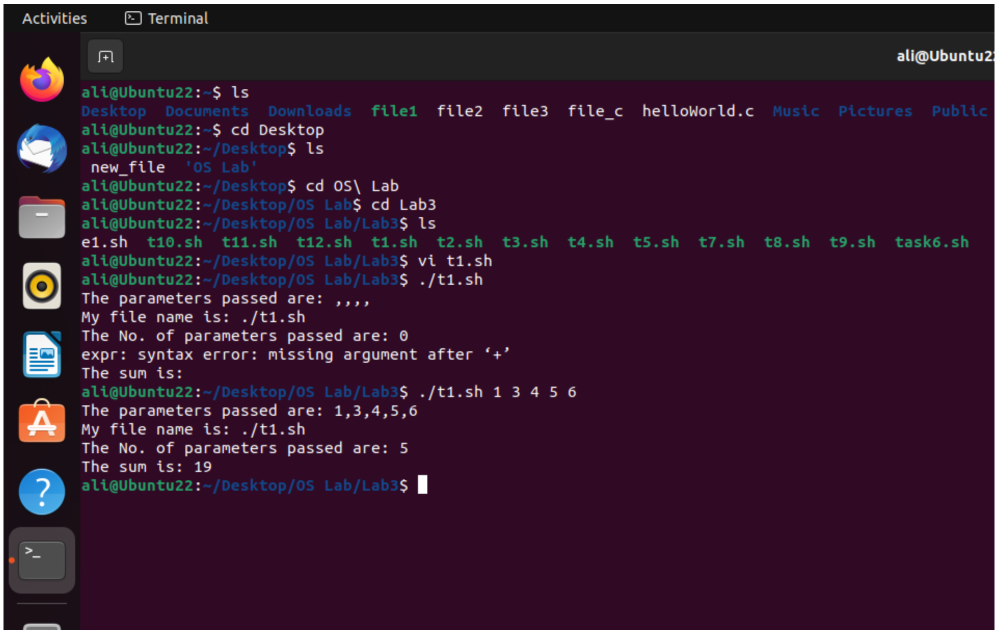
1. Run all the programs given in the Lab Notes, and observe the output for each program.

EXAMPLE 1:



```
echo The parameters passed are: $1,$2,$3,$4,$5
echo My file name is: $0
echo The No. of parameters passed are: $#
sum=`expr $1 + $2 + $3 + $4 + $5`
echo The sum is: $sum
#echo $*
#shift
#echo $*
#echo My args are: $1 $2
~
~
~
~
~
~
~
~
~
~
```

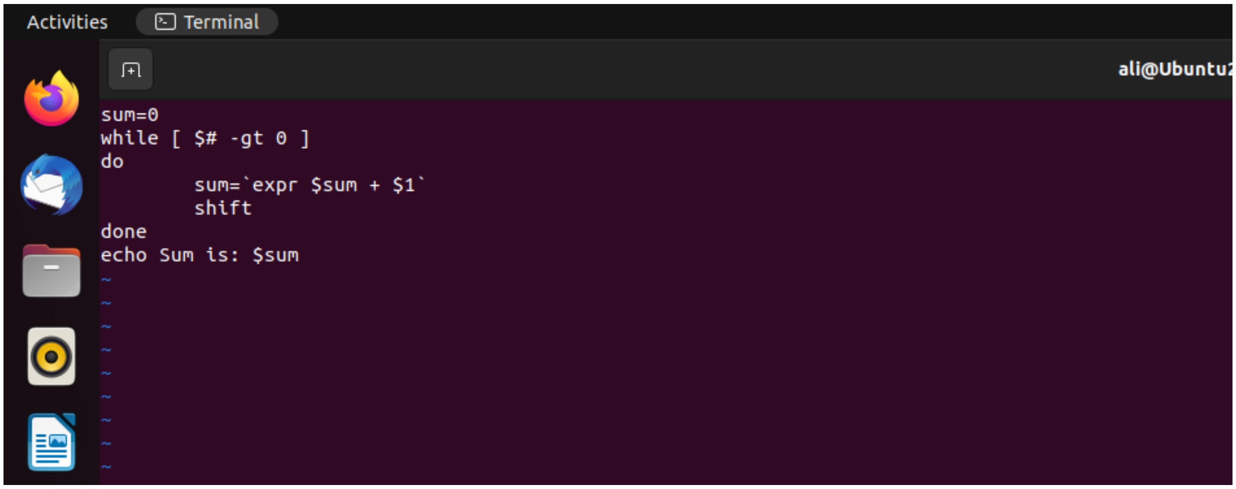
Figure 1-1: Code of Example 1



```
ali@Ubuntu22:~$ ls
Desktop  Documents  Downloads  file1  file2  file3  file_c  helloWorld.c  Music  Pictures  Public
ali@Ubuntu22:~$ cd Desktop
ali@Ubuntu22:~/Desktop$ ls
 new_file  'OS Lab'
ali@Ubuntu22:~/Desktop$ cd OS\ Lab
ali@Ubuntu22:~/Desktop/OS Lab$ cd Lab3
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ls
e1.sh  t10.sh  t11.sh  t12.sh  t1.sh  t2.sh  t3.sh  t4.sh  t5.sh  t7.sh  t8.sh  t9.sh  task6.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t1.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t1.sh
The parameters passed are: ,,,,
My file name is: ./t1.sh
The No. of parameters passed are: 0
expr: syntax error: missing argument after ‘+’
The sum is:
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t1.sh 1 3 4 5 6
The parameters passed are: 1,3,4,5,6
My file name is: ./t1.sh
The No. of parameters passed are: 5
The sum is: 19
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ █
```

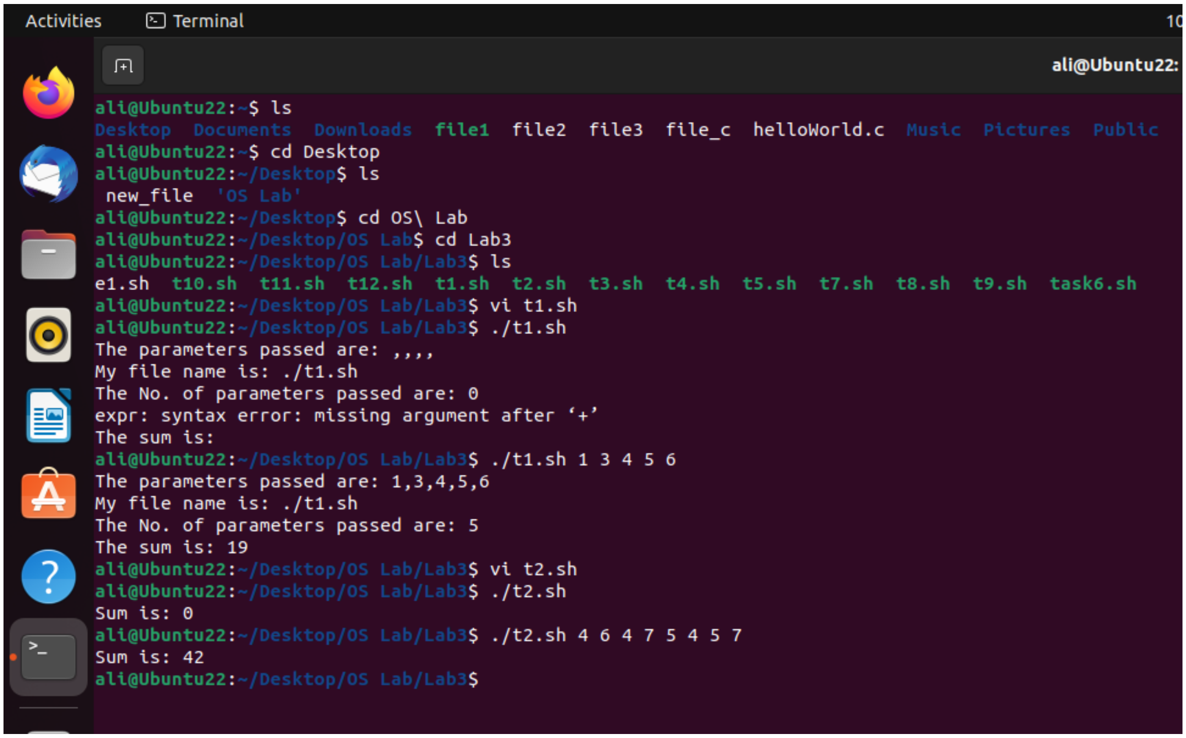
Figure 1-2: Output of Example 1

EXAMPLE 2:



```
sum=0
while [ $# -gt 0 ]
do
        sum=`expr $sum + $1`
        shift
done
echo Sum is: $sum
~
~
~
~
~
~
~
~
~
```

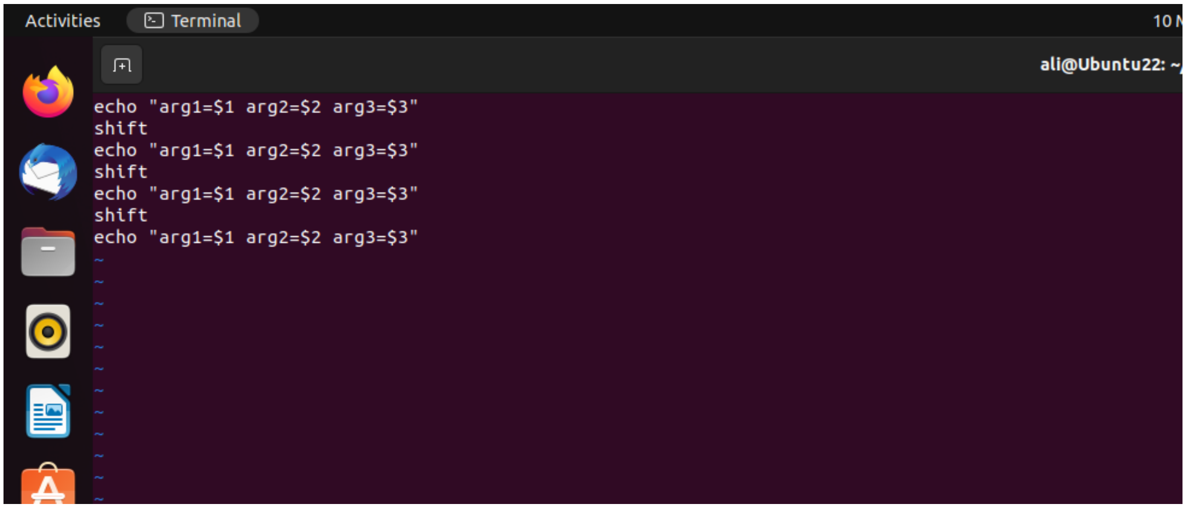
Figure 2-1: Code of Example 2



```
ali@Ubuntu22:~$ ls
Desktop  Documents  Downloads  file1  file2  file3  file_c  helloWorld.c  Music  Pictures  Public
ali@Ubuntu22:~$ cd Desktop
ali@Ubuntu22:~/Desktop$ ls
 new_file  'OS Lab'
ali@Ubuntu22:~/Desktop$ cd OS\ Lab
ali@Ubuntu22:~/Desktop/OS Lab$ cd Lab3
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ls
e1.sh  t10.sh  t11.sh  t12.sh  t1.sh  t2.sh  t3.sh  t4.sh  t5.sh  t7.sh  t8.sh  t9.sh  task6.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t1.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t1.sh
The parameters passed are: ,,,,
My file name is: ./t1.sh
The No. of parameters passed are: 0
expr: syntax error: missing argument after ‘+’
The sum is:
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t1.sh 1 3 4 5 6
The parameters passed are: 1,3,4,5,6
My file name is: ./t1.sh
The No. of parameters passed are: 5
The sum is: 19
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t2.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t2.sh
Sum is: 0
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t2.sh 4 6 4 7 5 4 5 7
Sum is: 42
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

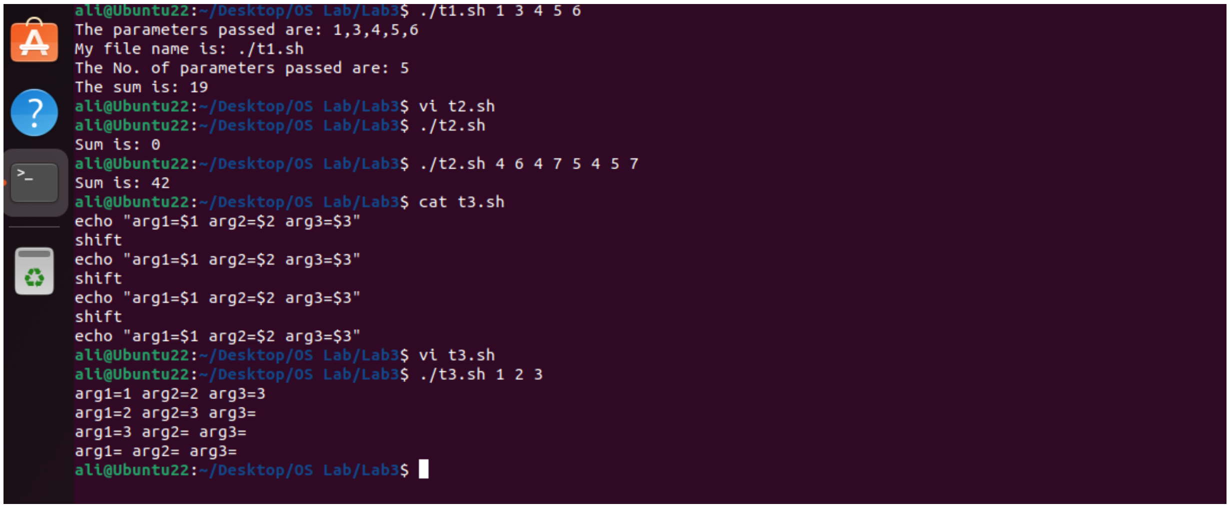
Figure 2-2: Output of Example 2

EXAMPLE 3:



```
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
```

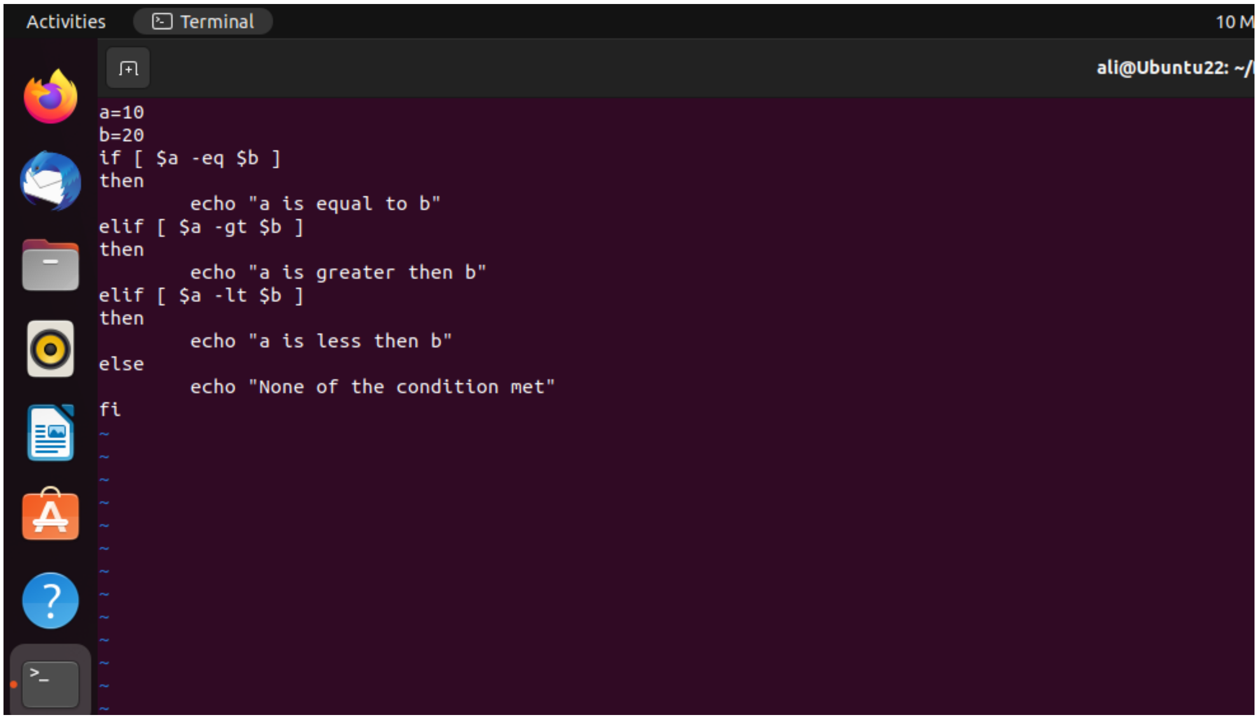
Figure 3-1: Code of Example 3



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t1.sh 1 3 4 5 6
The parameters passed are: 1,3,4,5,6
My file name is: ./t1.sh
The No. of parameters passed are: 5
The sum is: 19
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t2.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t2.sh
Sum is: 0
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t2.sh 4 6 4 7 5 4 5 7
Sum is: 42
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ cat t3.sh
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t3.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t3.sh 1 2 3
arg1=1 arg2=2 arg3=3
arg1=2 arg2=3 arg3=
arg1=3 arg2= arg3=
arg1= arg2= arg3=
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

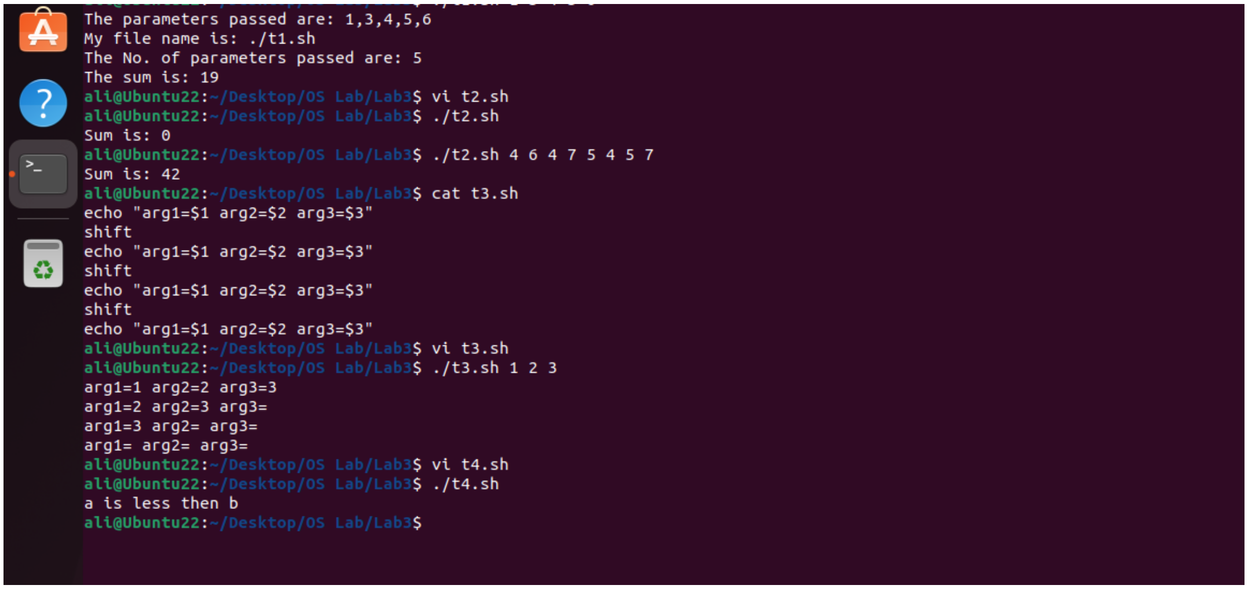
Figure 3-1: Output of Example 3

EXAMPLE 4:



```
a=10
b=20
if [ $a -eq $b ]
then
        echo "a is equal to b"
elif [ $a -gt $b ]
then
        echo "a is greater then b"
elif [ $a -lt $b ]
then
        echo "a is less then b"
else
        echo "None of the condition met"
fi
~
~
~
~
~
~
~
~
~
~
~
~
~
```

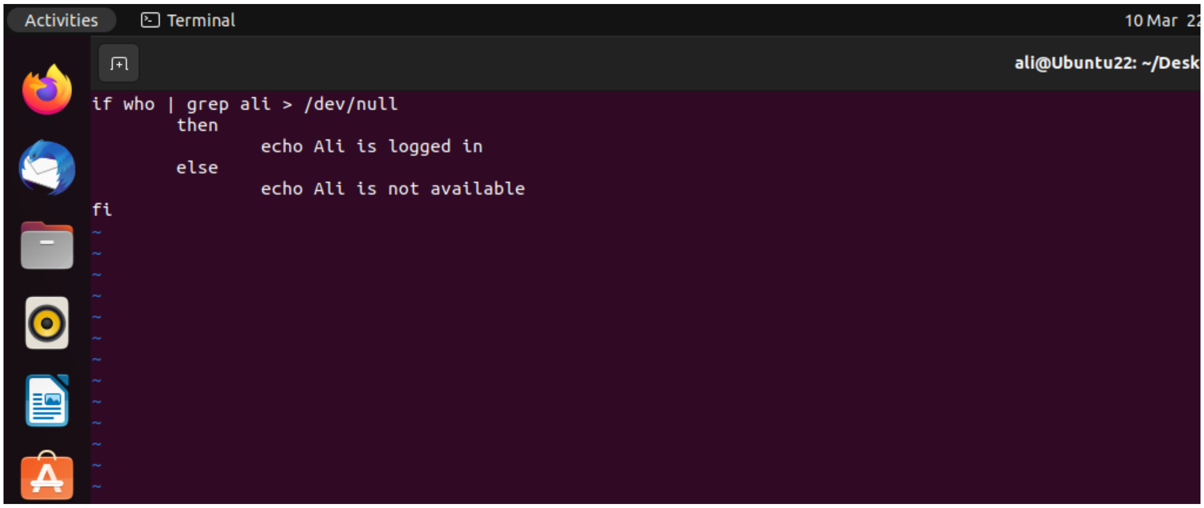
Figure 4-1: Code of Example 4



```
The parameters passed are: 1,3,4,5,6
My file name is: ./t1.sh
The No. of parameters passed are: 5
The sum is: 19
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t2.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t2.sh
Sum is: 0
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t2.sh 4 6 4 7 5 4 5 7
Sum is: 42
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ cat t3.sh
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t3.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t3.sh 1 2 3
arg1=1 arg2=2 arg3=3
arg1=2 arg2=3 arg3=
arg1=3 arg2= arg3=
arg1= arg2= arg3=
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t4.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t4.sh
a is less then b
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

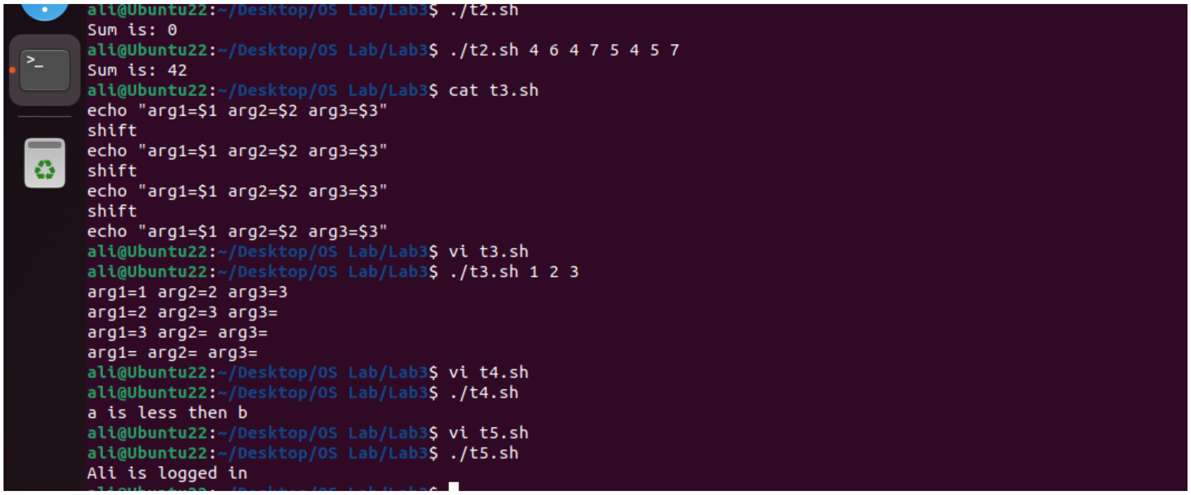
Figure 4-2: Output of Example 4

EXAMPLE 5:



```
if who | grep ali > /dev/null
        then
                echo Ali is logged in
        else
                echo Ali is not available
fi
~
~
~
~
~
~
~
~
~
~
~
~
~
```

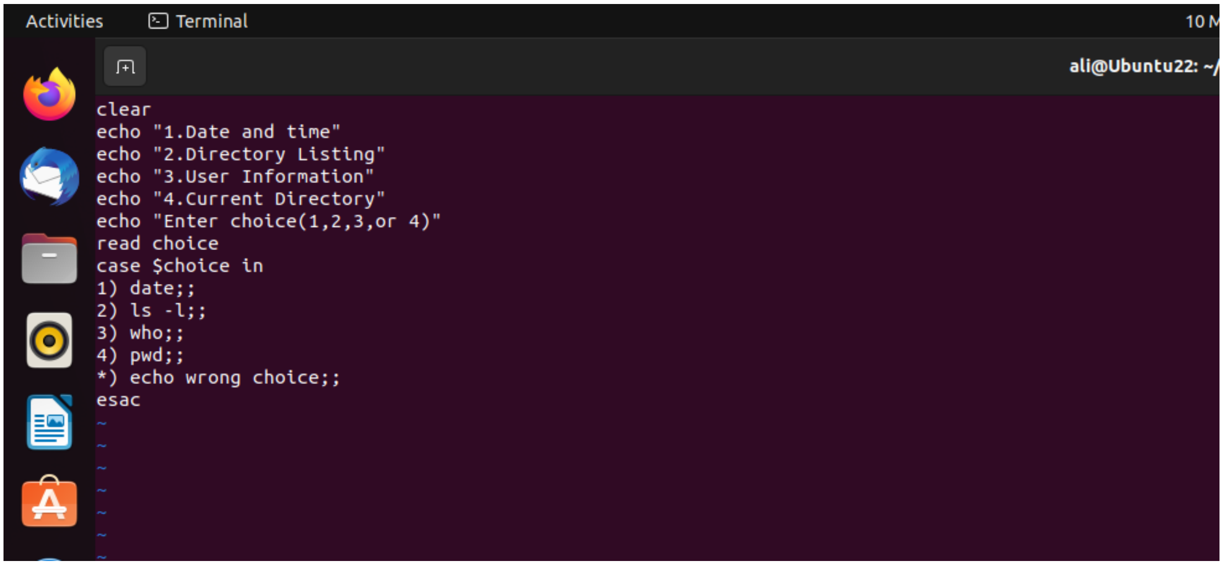
Figure 5-1: Code of Example 5



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t2.sh
Sum is: 0
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t2.sh 4 6 4 7 5 4 5 7
Sum is: 42
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ cat t3.sh
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
shift
echo "arg1=$1 arg2=$2 arg3=$3"
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t3.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t3.sh 1 2 3
arg1=1 arg2=2 arg3=3
arg1=2 arg2=3 arg3=
arg1=3 arg2= arg3=
arg1= arg2= arg3=
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t4.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t4.sh
a is less then b
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t5.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t5.sh
Ali is logged in
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

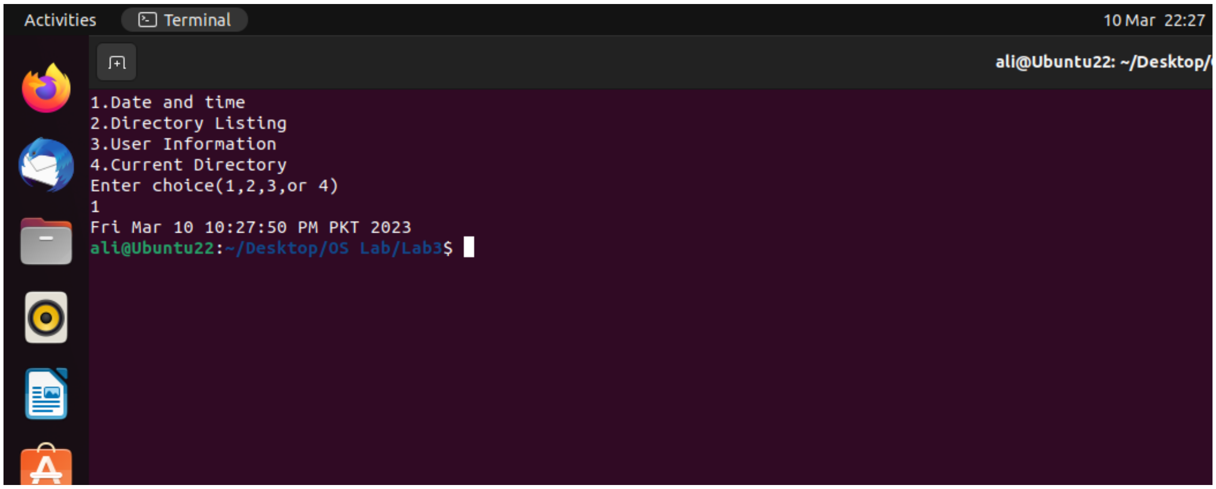
Figure 5-2: Output of Example 5

EXAMPLE 6:



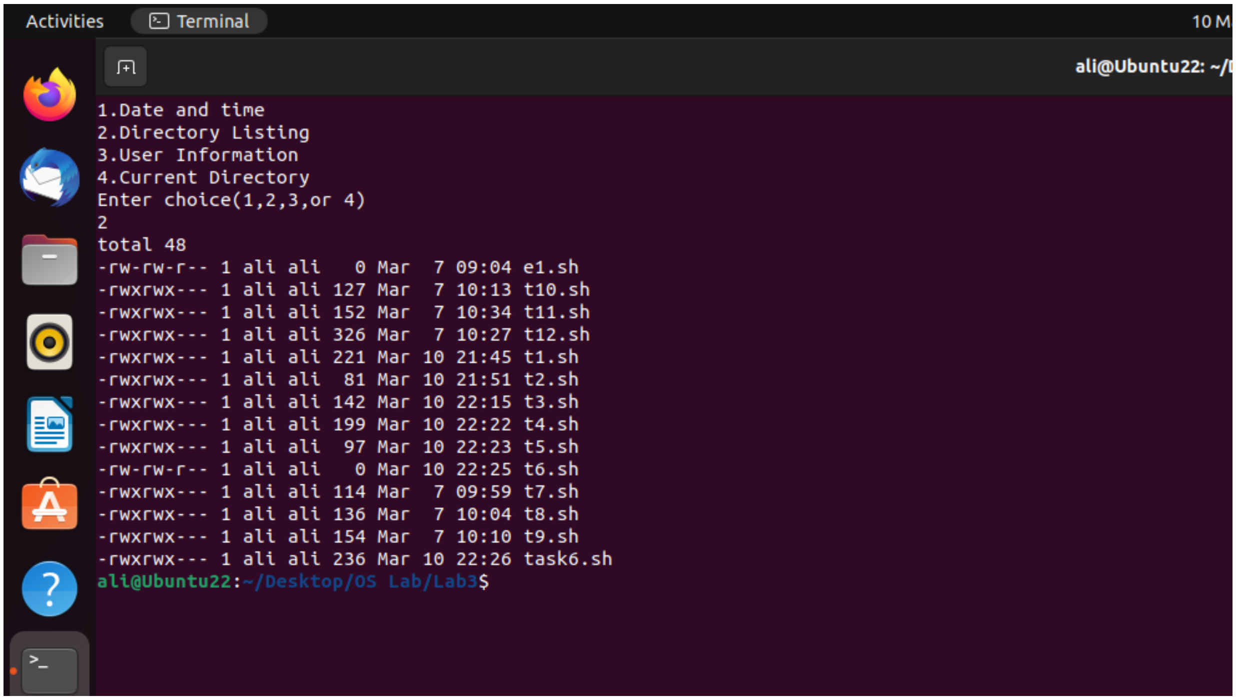
```
clear
echo "1.Date and time"
echo "2.Directory Listing"
echo "3.User Information"
echo "4.Current Directory"
echo "Enter choice(1,2,3,or 4)"
read choice
case $choice in
1) date;;
2) ls -l;;
3) who;;
4) pwd;;
*) echo wrong choice;;
esac
~
~
~
~
~
~
```

Figure 6-1: Code of Example 6



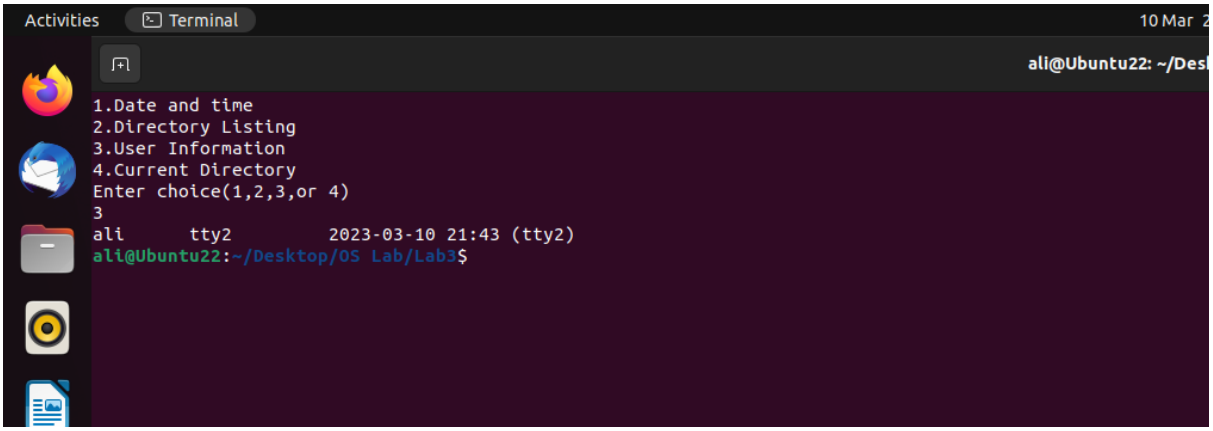
```
1.Date and time
2.Directory Listing
3.User Information
4.Current Directory
Enter choice(1,2,3,or 4)
1
Fri Mar 10 10:27:50 PM PKT 2023
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ 
```

Figure 6-2: Output of Example 6



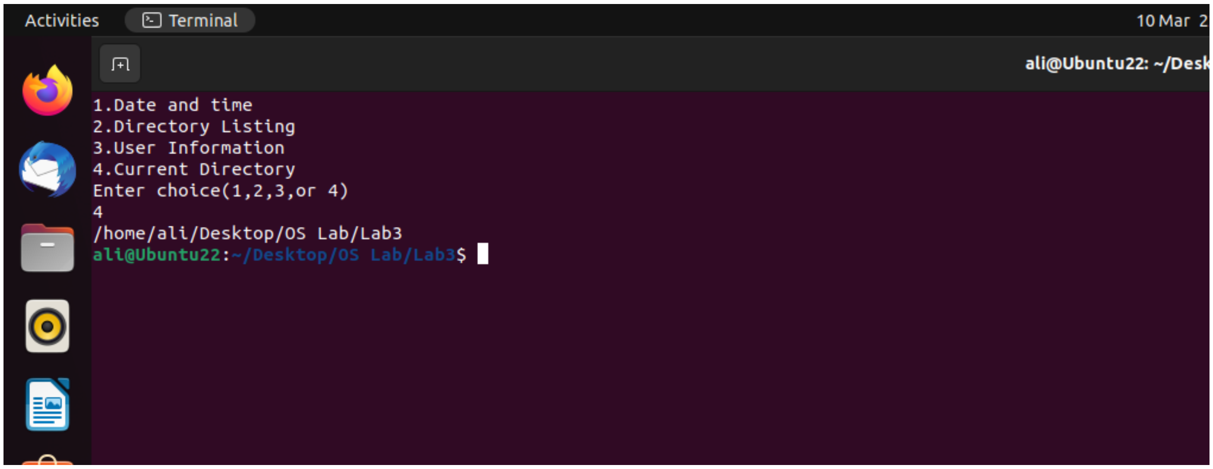
```
1.Date and time
2.Directory Listing
3.User Information
4.Current Directory
Enter choice(1,2,3,or 4)
2
total 48
-rw-rw-r-- 1 ali ali   0 Mar  7 09:04 e1.sh
-rwxrwx--- 1 ali ali 127 Mar  7 10:13 t10.sh
-rwxrwx--- 1 ali ali 152 Mar  7 10:34 t11.sh
-rwxrwx--- 1 ali ali 326 Mar  7 10:27 t12.sh
-rwxrwx--- 1 ali ali 221 Mar 10 21:45 t1.sh
-rwxrwx--- 1 ali ali  81 Mar 10 21:51 t2.sh
-rwxrwx--- 1 ali ali 142 Mar 10 22:15 t3.sh
-rwxrwx--- 1 ali ali 199 Mar 10 22:22 t4.sh
-rwxrwx--- 1 ali ali  97 Mar 10 22:23 t5.sh
-rw-rw-r-- 1 ali ali   0 Mar 10 22:25 t6.sh
-rwxrwx--- 1 ali ali 114 Mar  7 09:59 t7.sh
-rwxrwx--- 1 ali ali 136 Mar  7 10:04 t8.sh
-rwxrwx--- 1 ali ali 154 Mar  7 10:10 t9.sh
-rwxrwx--- 1 ali ali 236 Mar 10 22:26 task6.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

Figure 6-3: Output of Example 6



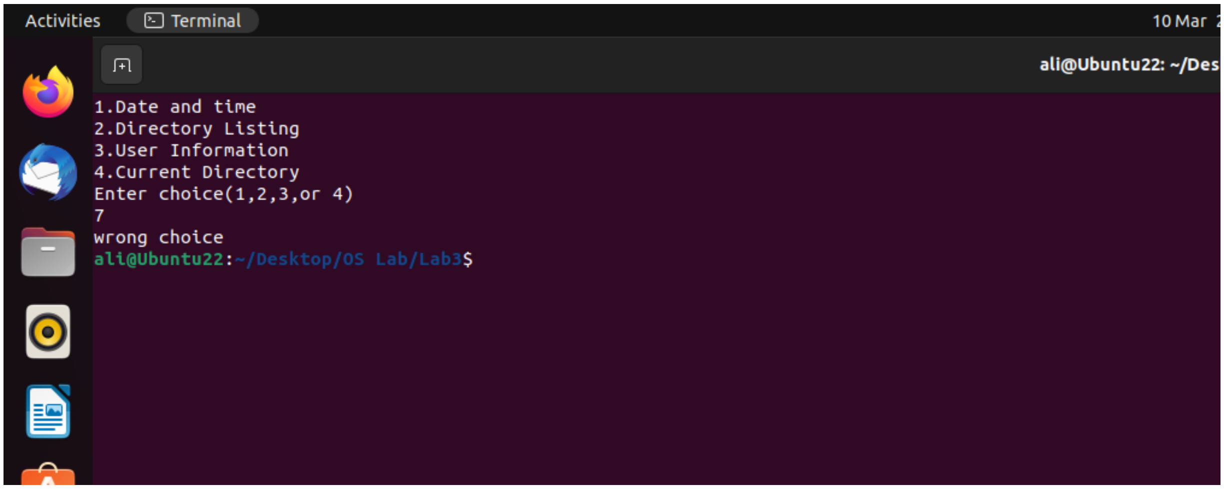
```
1.Date and time
2.Directory Listing
3.User Information
4.Current Directory
Enter choice(1,2,3,or 4)
3
ali       tty2         2023-03-10 21:43 (tty2)
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

Figure 6-4: Output of Example 6



```
1.Date and time
2.Directory Listing
3.User Information
4.Current Directory
Enter choice(1,2,3,or 4)
4
/home/ali/Desktop/OS Lab/Lab3
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

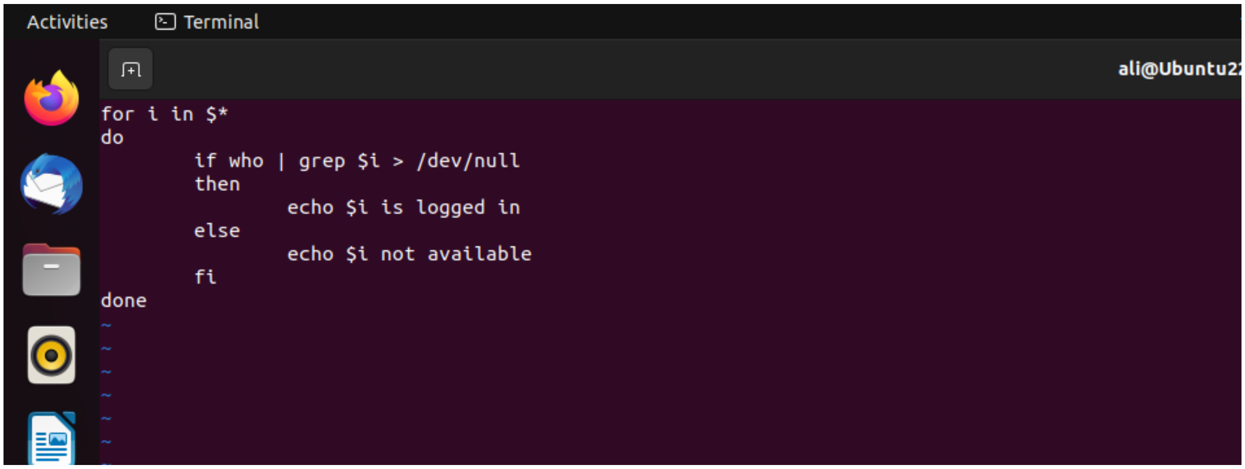
Figure 6-5: Output of Example 6



```
1.Date and time
2.Directory Listing
3.User Information
4.Current Directory
Enter choice(1,2,3,or 4)
7
wrong choice
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

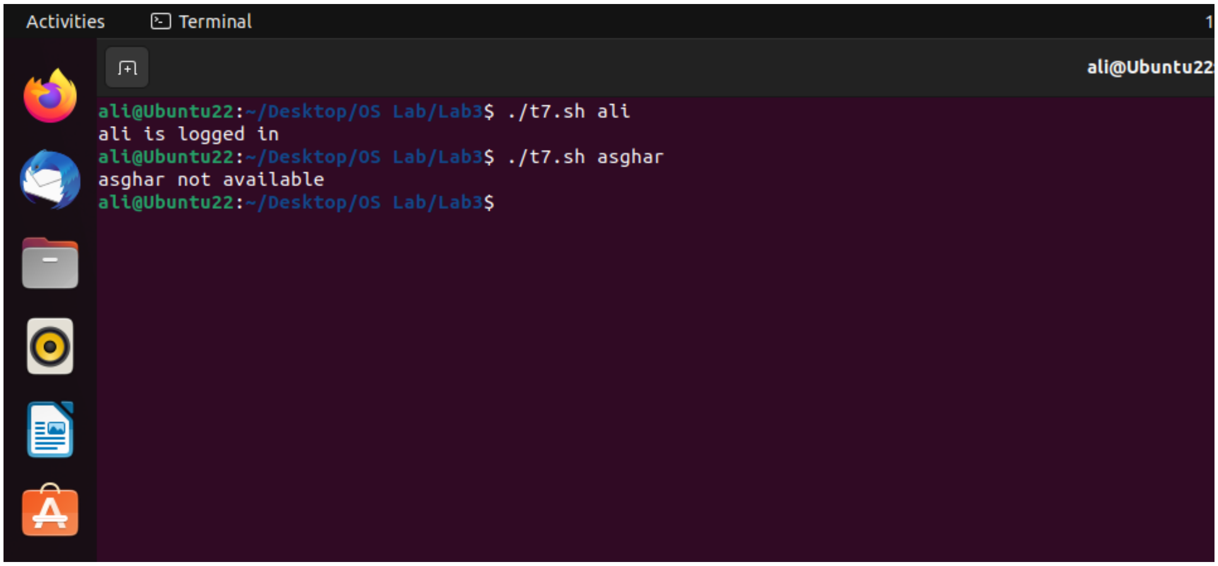
Figure 6-6: Output of Example 6

EXAMPLE 7:



```
for i in $*
do
        if who | grep $i > /dev/null
        then
                echo $i is logged in
        else
                echo $i not available
        fi
done
~
~
~
~
~
~
```

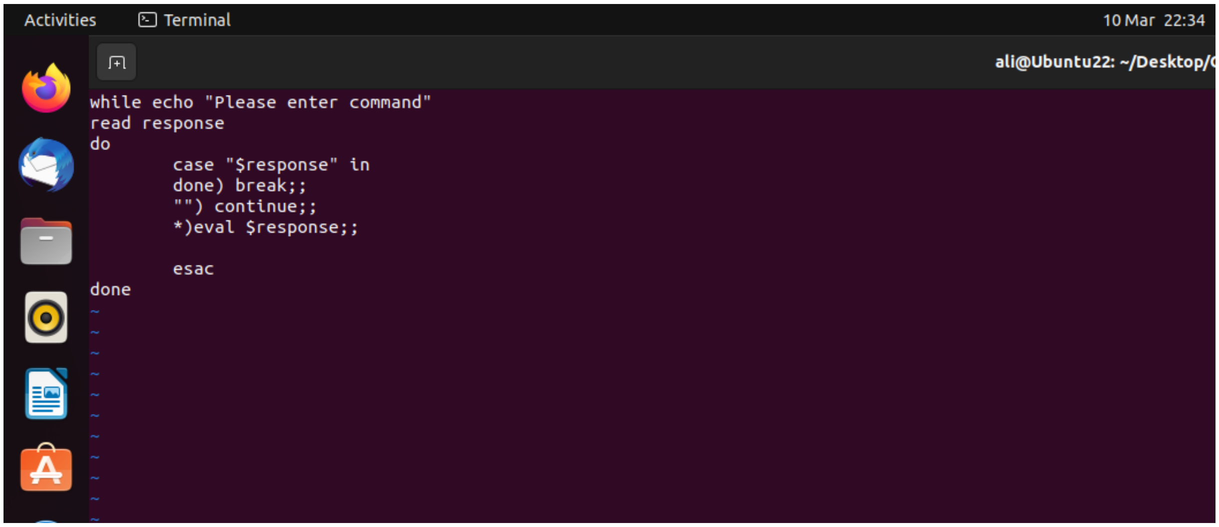
Figure 7-1: Code of Example 7



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t7.sh ali
ali is logged in
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t7.sh asghar
asghar not available
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

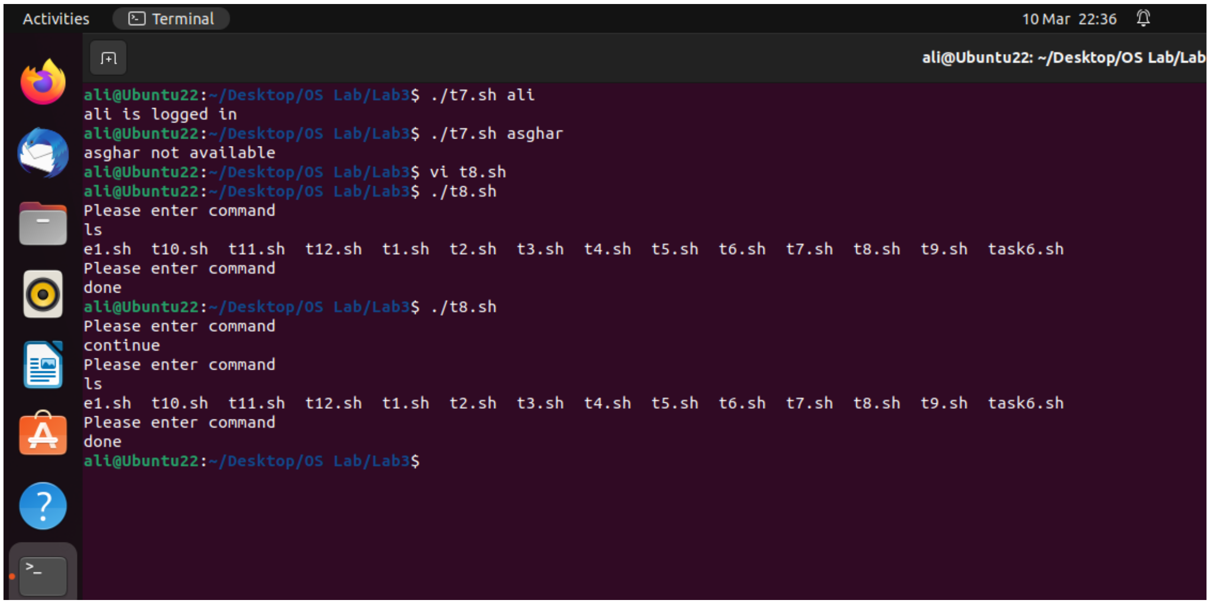
Figure 7-2: Output of Example 7

EXAMPLE 8:



```
while echo "Please enter command"
read response
do
        case "$response" in
        done) break;;
        "") continue;;
        *)eval $response;;

        esac
done
~
~
~
~
~
~
~
~
~
~
~
```

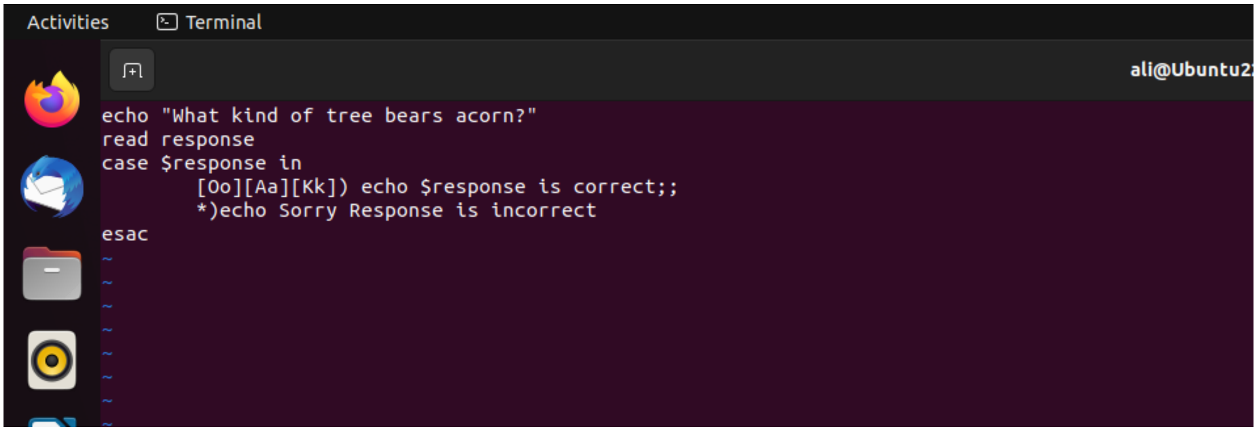
Figure 8-1: Code of Example 8



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t7.sh ali
ali is logged in
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t7.sh asghar
asghar not available
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t8.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t8.sh
Please enter command
ls
e1.sh  t10.sh  t11.sh  t12.sh  t1.sh  t2.sh  t3.sh  t4.sh  t5.sh  t6.sh  t7.sh  t8.sh  t9.sh  task6.sh
Please enter command
done
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t8.sh
Please enter command
continue
Please enter command
ls
e1.sh  t10.sh  t11.sh  t12.sh  t1.sh  t2.sh  t3.sh  t4.sh  t5.sh  t6.sh  t7.sh  t8.sh  t9.sh  task6.sh
Please enter command
done
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

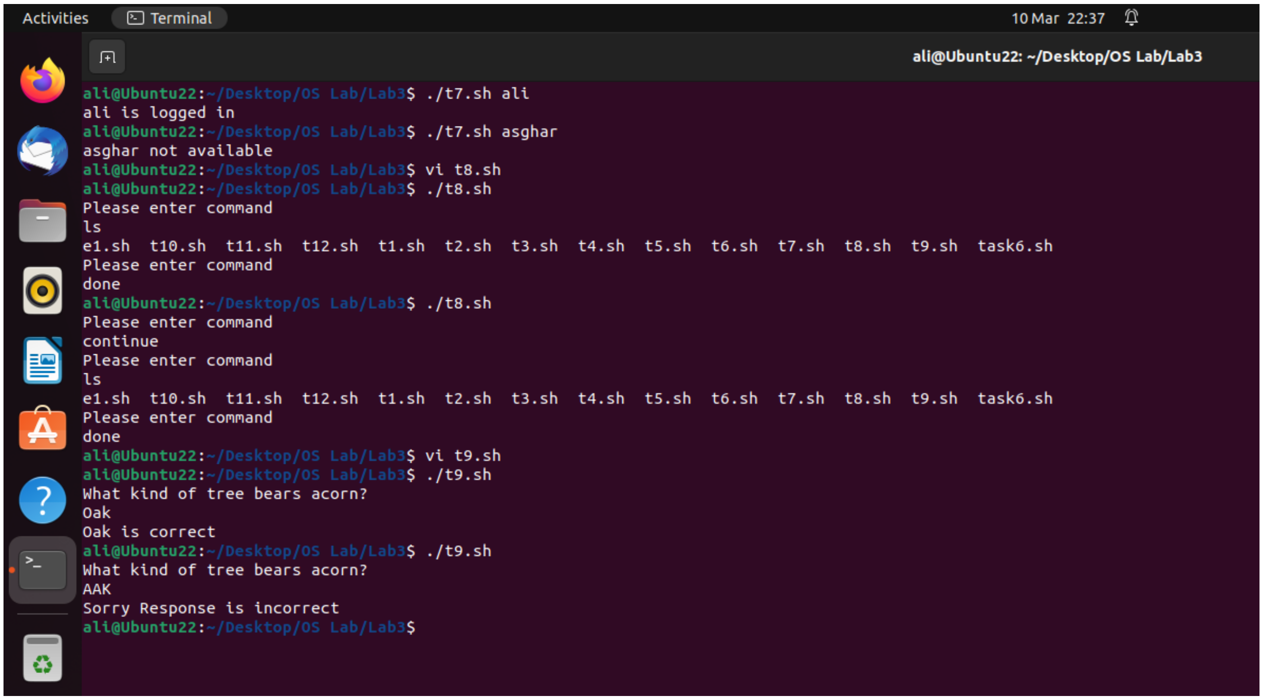
Figure 8-2: Output of Example 8

EXAMPLE 9:



```
echo "What kind of tree bears acorn?"
read response
case $response in
        [Oo][Aa][Kk]) echo $response is correct;;
        *)echo Sorry Response is incorrect
esac
~
~
~
~
~
~
~
~
```

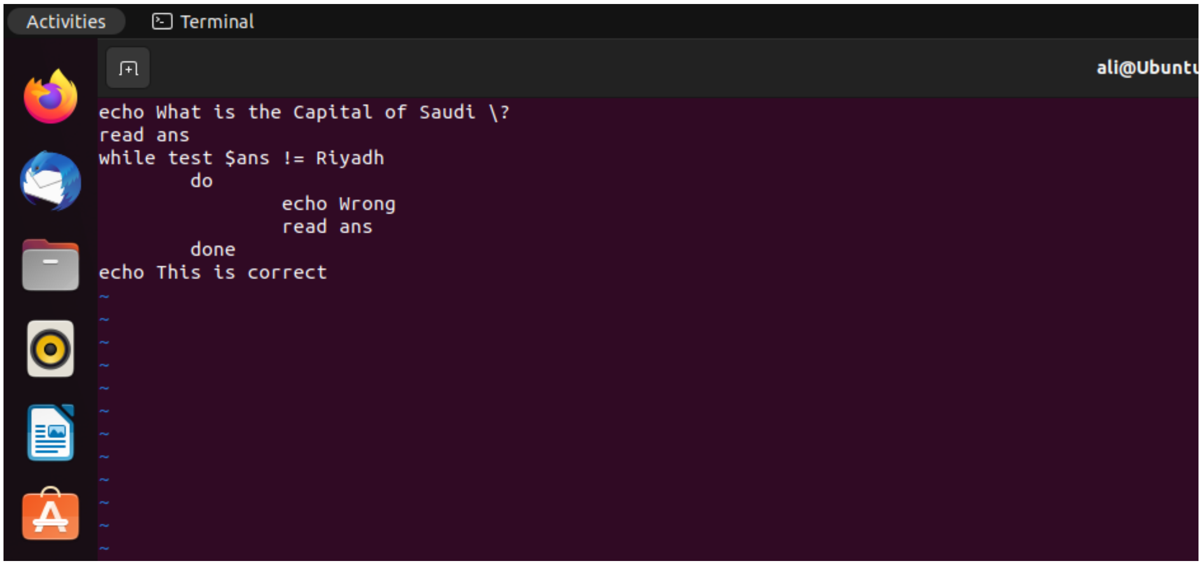
Figure 9-1: Code of Example 9



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t7.sh ali
ali is logged in
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t7.sh asghar
asghar not available
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t8.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t8.sh
Please enter command
ls
e1.sh  t10.sh  t11.sh  t12.sh  t1.sh  t2.sh  t3.sh  t4.sh  t5.sh  t6.sh  t7.sh  t8.sh  t9.sh  task6.sh
Please enter command
done
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t8.sh
Please enter command
continue
Please enter command
ls
e1.sh  t10.sh  t11.sh  t12.sh  t1.sh  t2.sh  t3.sh  t4.sh  t5.sh  t6.sh  t7.sh  t8.sh  t9.sh  task6.sh
Please enter command
done
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t9.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t9.sh
What kind of tree bears acorn?
Oak
Oak is correct
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t9.sh
What kind of tree bears acorn?
AAK
Sorry Response is incorrect
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

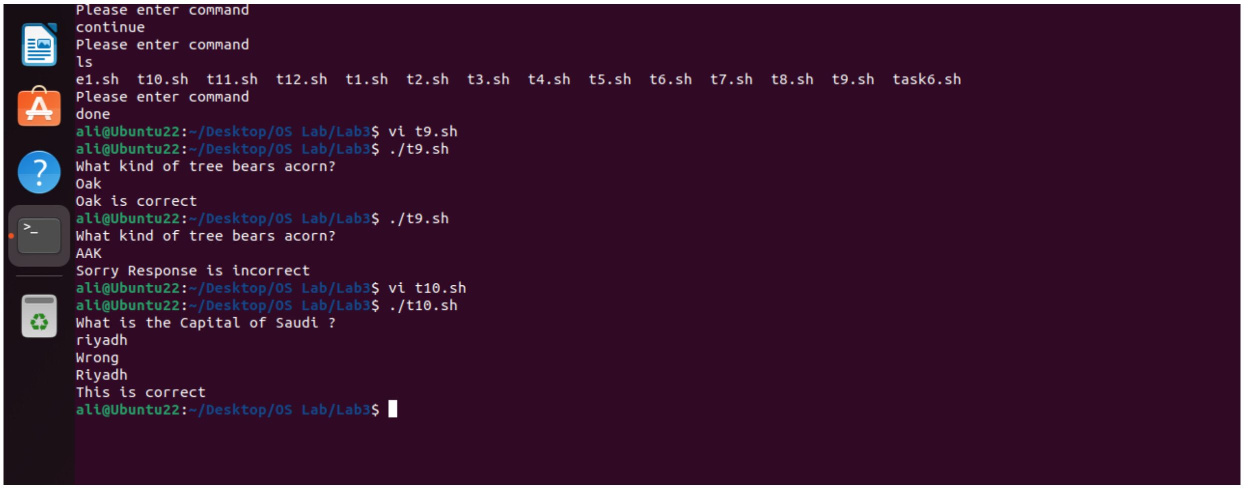
Figure 9-2: Output of Example 9

EXAMPLE 10:



```
echo What is the Capital of Saudi \?
read ans
while test $ans != Riyadh
        do
                echo Wrong
                read ans
        done
echo This is correct
~
~
~
~
~
~
~
~
~
~
~
~
```

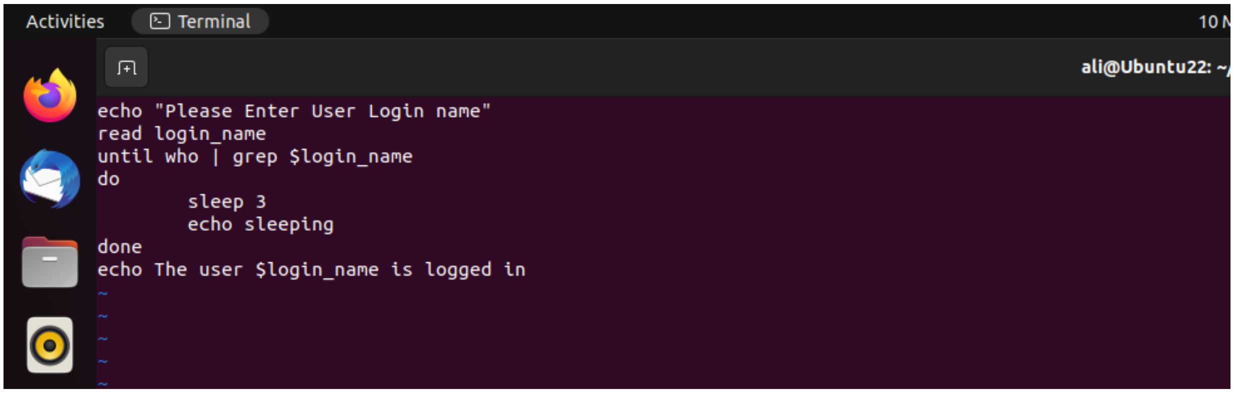
Figure 10-1: Code of Example 10



```
Please enter command
continue
Please enter command
ls
e1.sh  t10.sh  t11.sh  t12.sh  t1.sh  t2.sh  t3.sh  t4.sh  t5.sh  t6.sh  t7.sh  t8.sh  t9.sh  task6.sh
Please enter command
done
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t9.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t9.sh
What kind of tree bears acorn?
Oak
Oak is correct
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t9.sh
What kind of tree bears acorn?
AAK
Sorry Response is incorrect
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t10.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t10.sh
What is the Capital of Saudi ?
riyadh
Wrong
Riyadh
This is correct
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ 
```

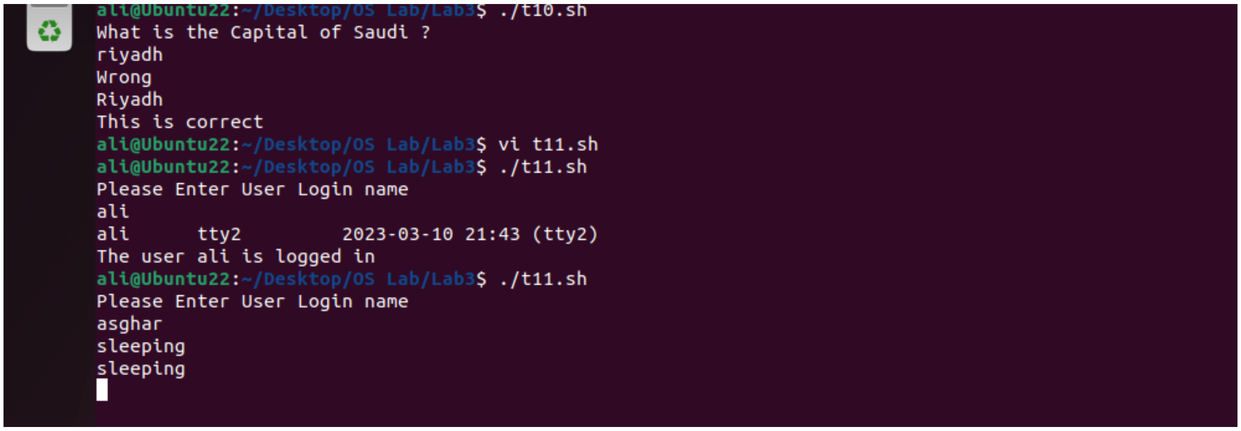
Figure 10-2: Output of Example 10

EXAMPLE 11:



```
echo "Please Enter User Login name"
read login_name
until who | grep $login_name
do
        sleep 3
        echo sleeping
done
echo The user $login_name is logged in
~
~
~
~
~
```

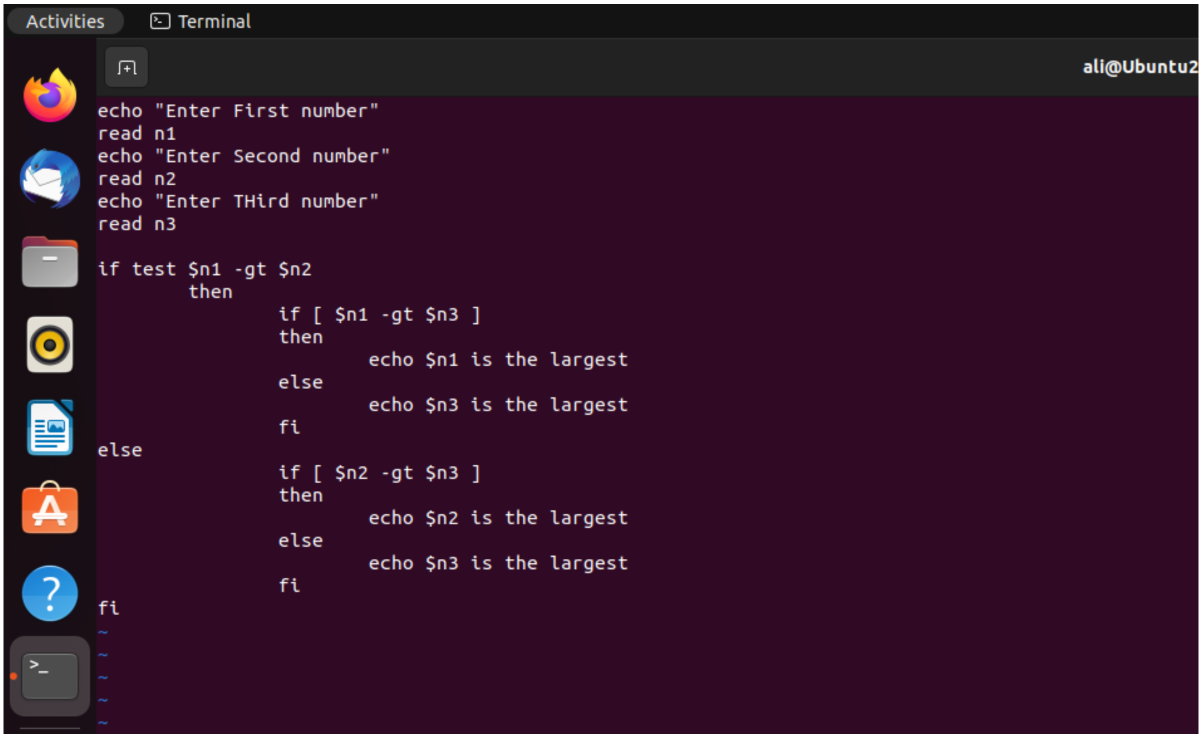
Figure 11-1: Code of Example 11



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t10.sh
What is the Capital of Saudi ?
riyadh
Wrong
Riyadh
This is correct
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t11.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t11.sh
Please Enter User Login name
ali
ali      tty2         2023-03-10 21:43 (tty2)
The user ali is logged in
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t11.sh
Please Enter User Login name
asghar
sleeping
sleeping
```

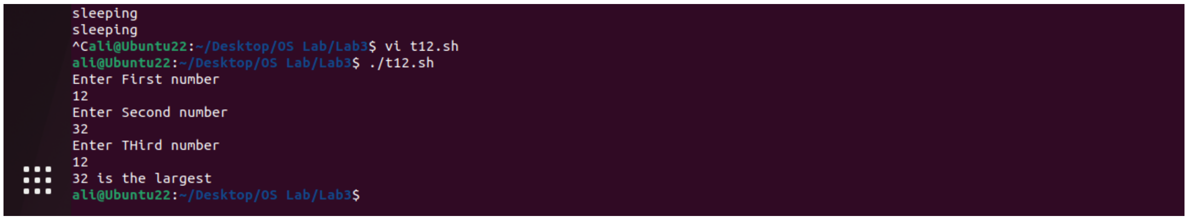
Figure 11-2: Output of Example 11

EXAMPLE 12:



```
echo "Enter First number"
read n1
echo "Enter Second number"
read n2
echo "Enter THird number"
read n3

if test $n1 -gt $n2
        then
                if [ $n1 -gt $n3 ]
                then
                        echo $n1 is the largest
                else
                        echo $n3 is the largest
                fi
else
                if [ $n2 -gt $n3 ]
                then
                        echo $n2 is the largest
                else
                        echo $n3 is the largest
                fi
fi
~
~
~
~
~
```

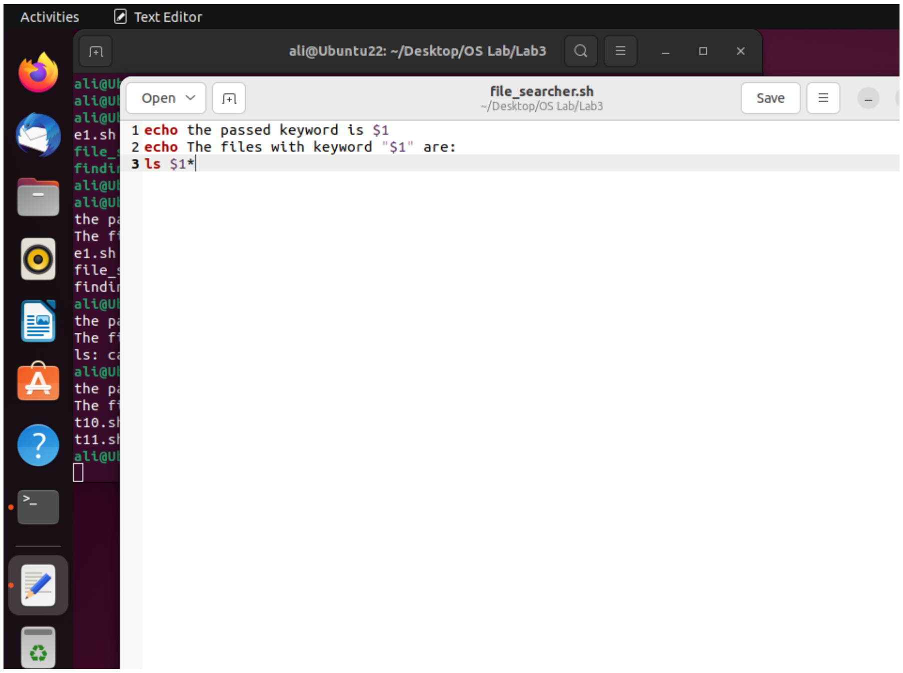
Figure 12-1: Code of Example 12



```
sleeping
sleeping
^Cali@Ubuntu22:~/Desktop/OS Lab/Lab3$ vi t12.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./t12.sh
Enter First number
12
Enter Second number
32
Enter THird number
12
32 is the largest
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

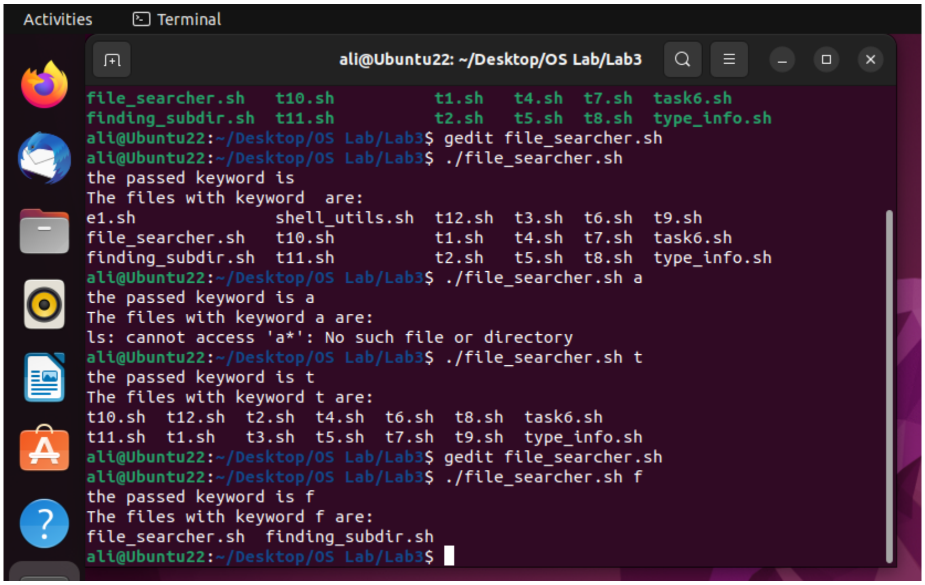
Figure 12-2: Output of Example 12

2. Write a shell script that takes a keyword as a command line argument and lists the filenames containing the keyword.



```
echo the passed keyword is $1
echo The files with keyword "$1" are:
ls $1*
```

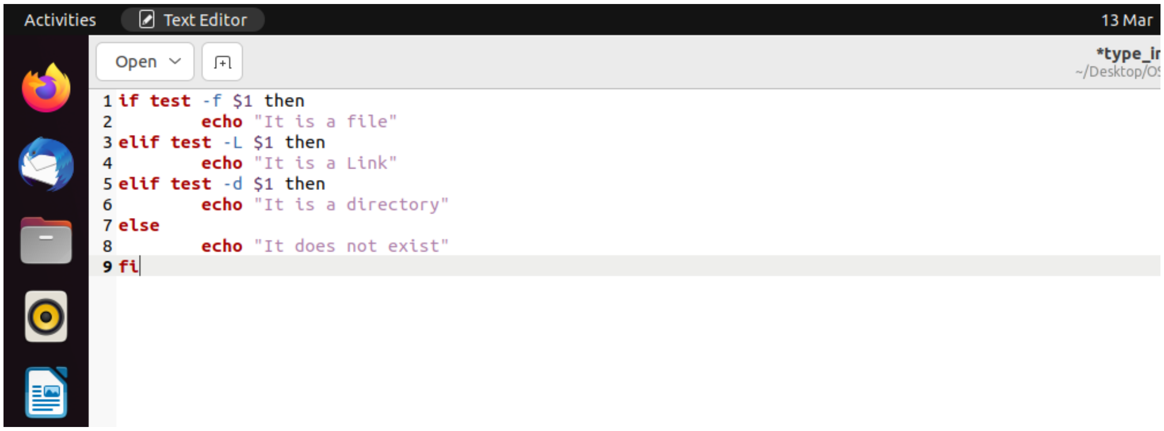
Figure 13-1: Code of file_searcher.sh



```
file_searcher.sh   t10.sh          t1.sh   t4.sh  t7.sh  task6.sh
finding_subdir.sh  t11.sh          t2.sh   t5.sh  t8.sh  type_info.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ gedit file_searcher.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./file_searcher.sh
the passed keyword is
The files with keyword  are:
e1.sh              shell_utils.sh  t12.sh  t3.sh  t6.sh  t9.sh
file_searcher.sh   t10.sh          t1.sh   t4.sh  t7.sh  task6.sh
finding_subdir.sh  t11.sh          t2.sh   t5.sh  t8.sh  type_info.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./file_searcher.sh a
the passed keyword is a
The files with keyword a are:
ls: cannot access 'a*': No such file or directory
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./file_searcher.sh t
the passed keyword is t
The files with keyword t are:
t10.sh  t12.sh  t2.sh  t4.sh  t6.sh  t8.sh  task6.sh
t11.sh  t1.sh   t3.sh  t5.sh  t7.sh  t9.sh  type_info.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ gedit file_searcher.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./file_searcher.sh f
the passed keyword is f
The files with keyword f are:
file_searcher.sh  finding_subdir.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

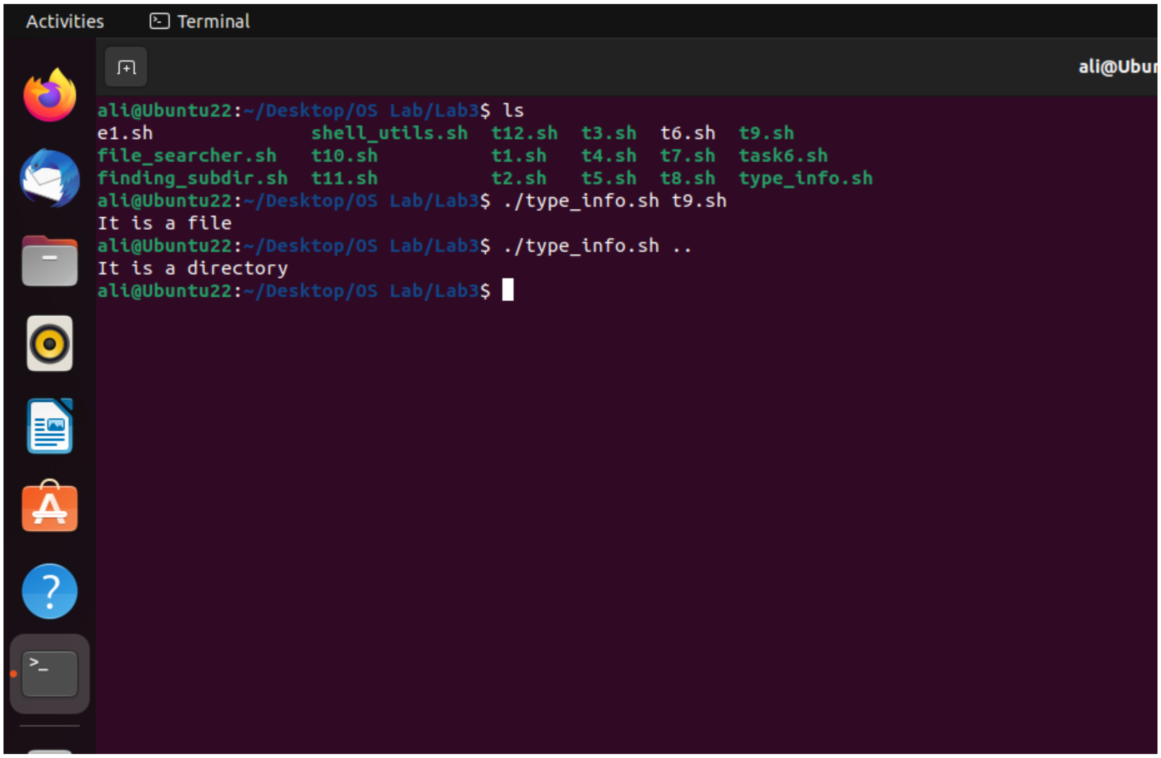
Figure 13-2: Output of file_searcher.sh

3. Write a shell script that takes a command line argument and reports whether it is a directory, or a file or a link.



```
if test -f $1 then
        echo "It is a file"
elif test -L $1 then
        echo "It is a Link"
elif test -d $1 then
        echo "It is a directory"
else
        echo "It does not exist"
fi
```

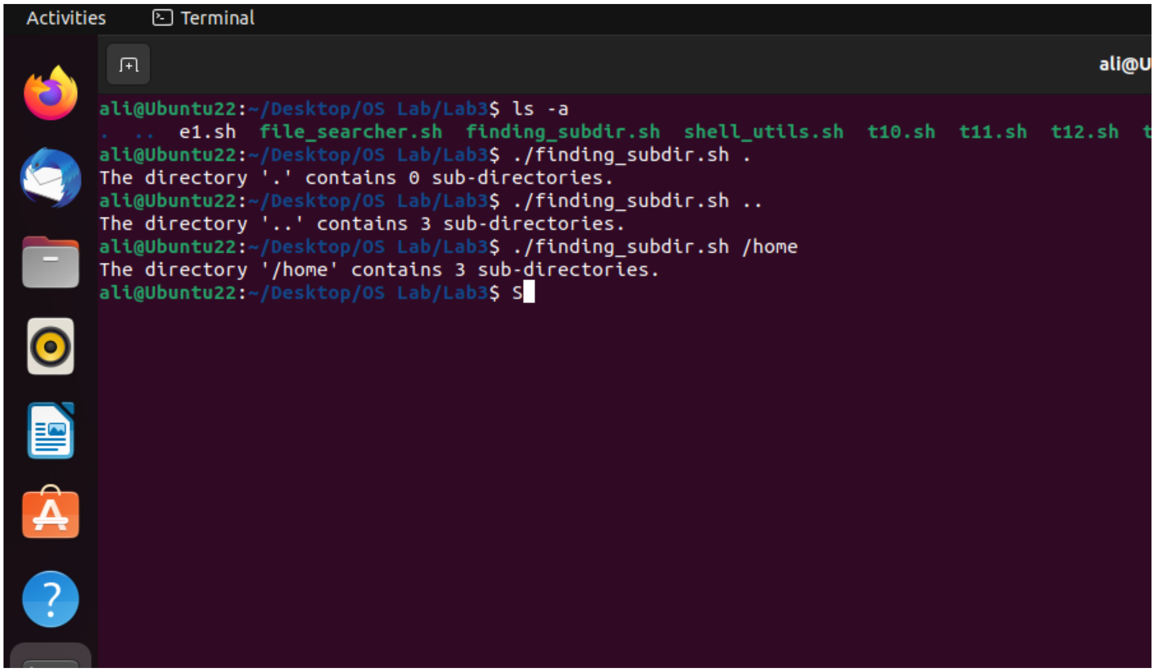
Figure 14-1: Code of type_info.sh



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ls
e1.sh              shell_utils.sh  t12.sh  t3.sh  t6.sh  t9.sh
file_searcher.sh   t10.sh          t1.sh   t4.sh  t7.sh  task6.sh
finding_subdir.sh  t11.sh          t2.sh   t5.sh  t8.sh  type_info.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./type_info.sh t9.sh
It is a file
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./type_info.sh ..
It is a directory
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ 
```

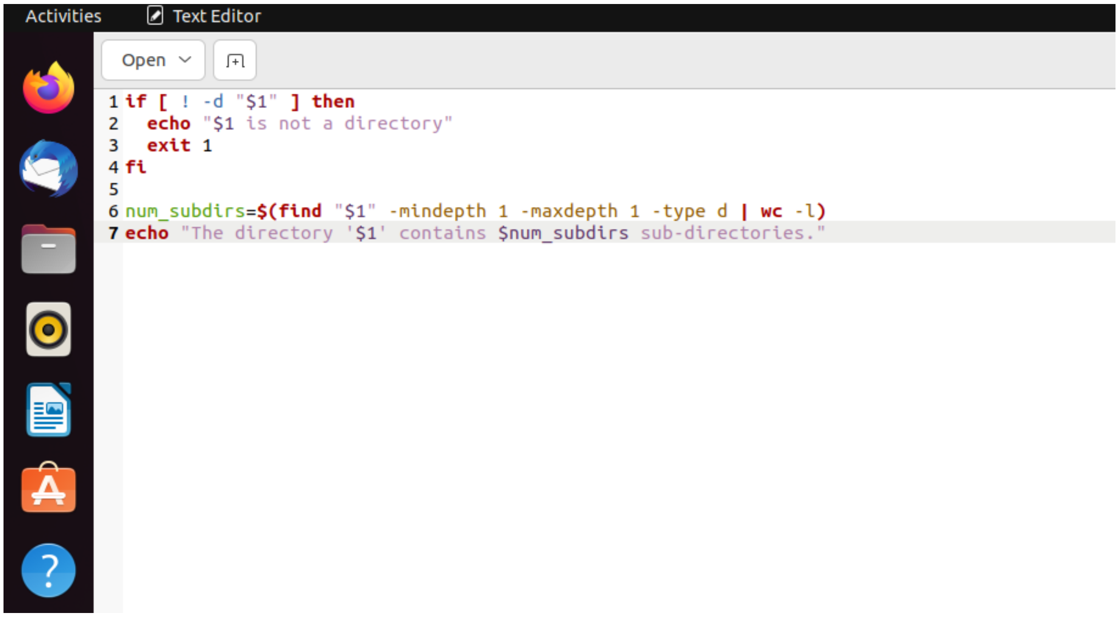
Figure 14-2: Output of type_info.sh

4. Write a script to find the number of sub directories in a given directory.



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ls -a
.  ..  e1.sh  file_searcher.sh  finding_subdir.sh  shell_utils.sh  t10.sh  t11.sh  t12.sh  t
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./finding_subdir.sh .
The directory '.' contains 0 sub-directories.
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./finding_subdir.sh ..
The directory '..' contains 3 sub-directories.
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./finding_subdir.sh /home
The directory '/home' contains 3 sub-directories.
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ S
```

Figure 14-1: Code of type_info.sh



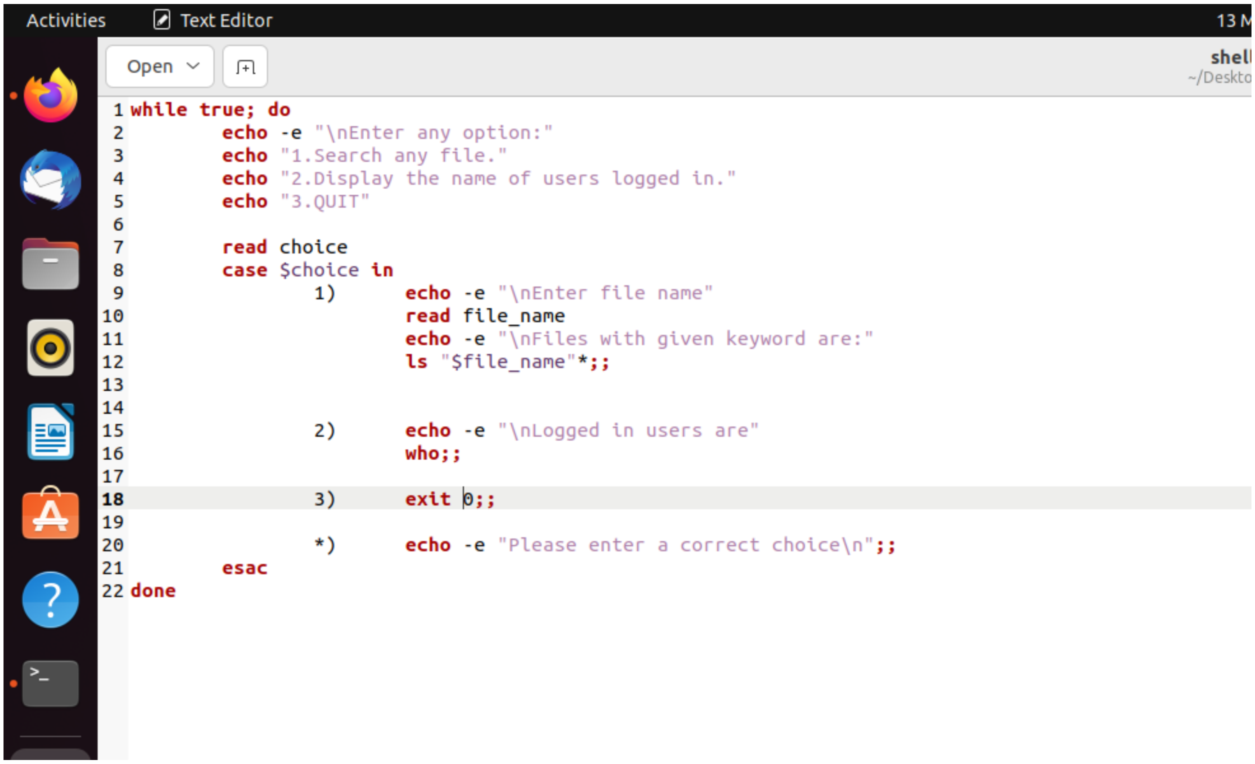
```
if [ ! -d "$1" ] then
  echo "$1 is not a directory"
  exit 1
fi

num_subdirs=$(find "$1" -mindepth 1 -maxdepth 1 -type d | wc -l)
echo "The directory '$1' contains $num_subdirs sub-directories."
```

Figure 14-2: Code of type_info.sh

5. Write a menu driven program that has the following options.

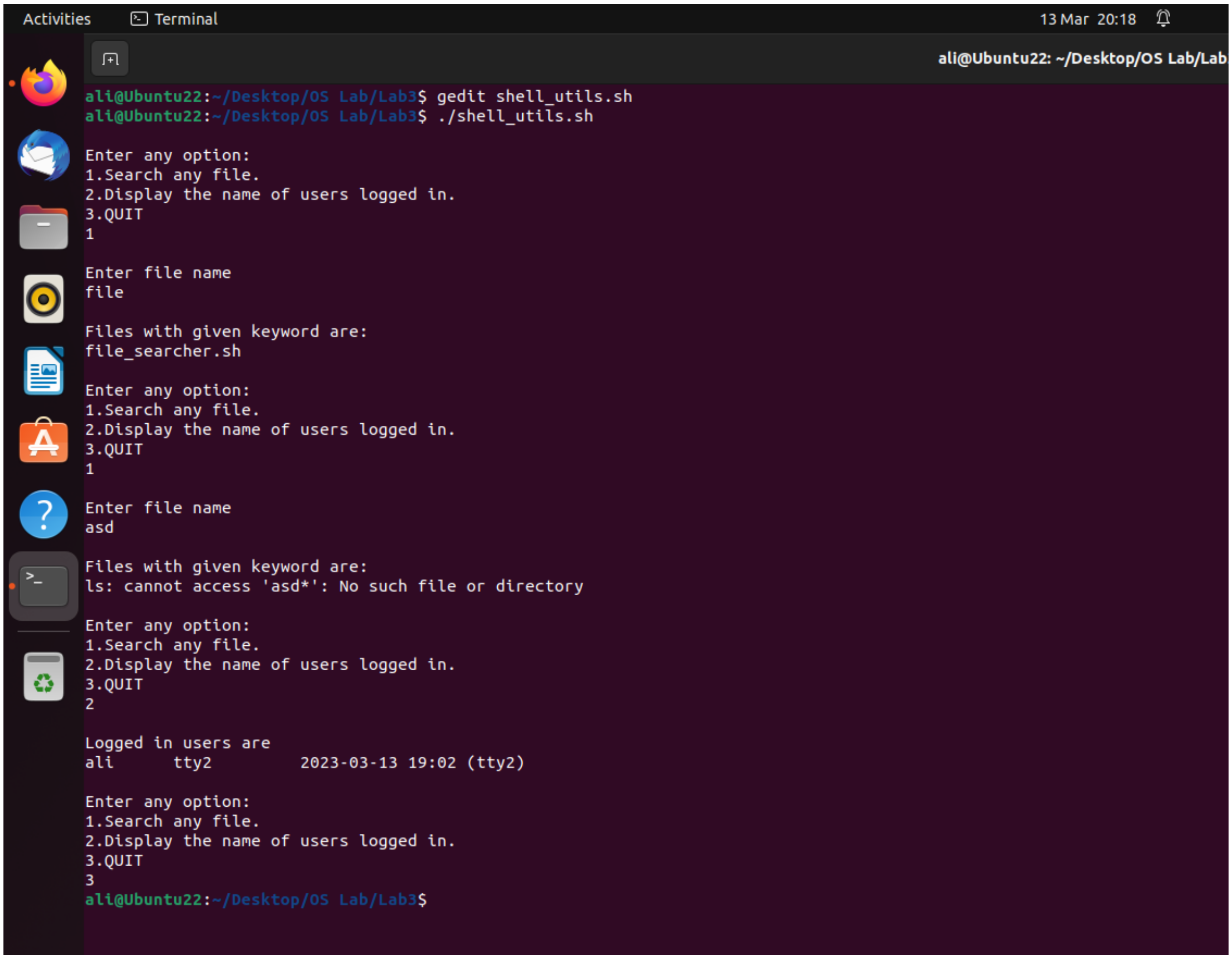
5.1. Search a given file is in the directory or not.

5.2. Display the names of the users logged in.



```
while true; do
        echo -e "\nEnter any option:"
        echo "1.Search any file."
        echo "2.Display the name of users logged in."
        echo "3.QUIT"

        read choice
        case $choice in
                1)      echo -e "\nEnter file name"
                        read file_name
                        echo -e "\nFiles with given keyword are:"
                        ls "$file_name"*;;


                2)      echo -e "\nLogged in users are"
                        who;;

                3)      exit 0;;

                *)      echo -e "Please enter a correct choice\n";;
        esac
done
```

Figure 15-1: Code of shell_utils.sh



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ gedit shell_utils.sh
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$ ./shell_utils.sh

Enter any option:
1.Search any file.
2.Display the name of users logged in.
3.QUIT
1

Enter file name
file

Files with given keyword are:
file_searcher.sh

Enter any option:
1.Search any file.
2.Display the name of users logged in.
3.QUIT
1

Enter file name
asd

Files with given keyword are:
ls: cannot access 'asd*': No such file or directory

Enter any option:
1.Search any file.
2.Display the name of users logged in.
3.QUIT
2

Logged in users are
ali      tty2         2023-03-13 19:02 (tty2)

Enter any option:
1.Search any file.
2.Display the name of users logged in.
3.QUIT
3
ali@Ubuntu22:~/Desktop/OS Lab/Lab3$
```

Figure 15-2: Output of shell_utils.sh

CONCLUSION:

I concluded that I can pass arguments to shell script which can be manipulated in the shell script. Further I learned about using loops and control structures in shell scripting.