Laboratory Report

LECTURER	John ORaw
STUDENT NAME	Edmund Connolly
STUDENT NUMBER	L00194727
PROGRAMME	MSc Cloud
MODULE CODE	IaC
ASSIGNMENT TITLE	Week 6 - Linux
SUBMISSION DATE	9/11/2025

Student Signature	**Date**
Edmund Connolly	9/11/2025

Contents

Table of figures

Description

In this laboratory report we explore the Bourne-Again SHell (BASH) scripting language used in Linux. BASH Scripting allows system administrators to automate repetitive tasks thereby increasing efficiency for system administrators. These scripts can be scheduled to run at regular intervals or at certain times.

Aims

BASH files can be used to automate processes on Linux virtual machines, thus we should become familiar with them, their scripting language and experiment with them by executing them and observing the results.

The primary aim of this work is:

1. To become familiar with Linux terminal and file structure.
2. Learn the BASH scripting language, such as printing to the screen, loops and if statements.
3. Learn how arguments can be read and processed in a BASH script
4. Write and test Linux BASH script files.
5. Learn how to change file permissions, to make a BASH script executable.
6. Executing BASH script files.
7. Observe the output from executing the BASH files.

Method

In this section we will guide you through creating a directory to store our BASH scripts and then create the bash scripts in the nano text editor, saving the script. Then to execute a script we need to change the file permissions, and then finally execute the script and observe the output

A Linux virtual machine was used to execute the bash scripts. First a directory named Linux was created, this directory would be used to store all the BASH scripts.

Script1.sh

The Nano text editor was used to input the code for all the BASH scripts see figure 1. To open nano we type nano script1.sh into the terminal. The following code was entered into Nano:

```
# By: Edmund Connolly
# Date: 30 OCT 20205
# Function: show the users entry in the password file
# Script: script1
grep $USER /etc/passwd
exit 0
```

To save the script, we press CTRL + O and then press Enter.

Then to exit CTRL + X

The code uses grep to find the current user which is stored in the $USER variable in passwords file called passwd located in the etc directory. It then exits with return value 0 which is used to indicate the script executed successfully.

Then to make the script executable in linux we need to change the file permissions to allow the user to execute the script this is done by typing chmod u+rx scriptname. In figure 1, we can see that script1.sh is now in green and the file permissions has x for user portion both these indicate that the script is now executable.

To execute the script we use ./scriptname, ./ says look in current working directory and execute the script with the name scriptname.

Script2.sh

The following code was used for the script2 code,

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Show users who are logged on
# Script: script2

echo "Welcome" $USER
echo "The following users are currently logged on"
who
exit 0
```

it prints to the screen a welcome message to the current user using $USER environment variable, then it prints out all currently logged in users using the WHO command.

Script3.sh

The following code was used for the script3 code,

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Show the process tree
# Script: script3
echo " The following processes are running"
ps -f
read -p "Press return to continue"
echo "This script will show the process tree for" $HOSTNAME
read -p "Press return to continue"
pstree | less
exit 0
```

The code prints the message “The following processes are running” to the screen, it then lists all running processes using the ps command with the -f switch for full format to show detailed info. Read -p is used to make the system pause and it uses the -p switch to prompt the message “Press return to continue”.

The script then displays the message “This script will show the process tree for” $HOSTNAME, $HOSTNAME is a environment variable that stores the computers name. It then pauses and waits for user input before continuing.

Using pstree command we can see a textual hierarchically tree of all the processes, less allows scrolling of the tree.

Script4.sh

The code for script4.sh is shown below.

```
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Demonstrate the use of variables
# Script: script4
MY_MESSAGE="Good Morning!"
echo $MY_MESSAGE
exit 0
```

The code stores the text sting “Good Morning!” in a variable called MY_MESSAGE, it then prints the contents of the MY_MESSAGE variable to the screen.

Script5.sh

The code for script5.sh is shown below.

```
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Demonstrate the use of variables
# Script: script5
a=5.66
b=8.67
c=`echo $a + $b | bc`
echo "$a + $b = $c"
exit 0
```

The code stores 5.66 in variable a, and 8.67 in variable b. it then adds these 2 values together using the bc command, and outputs the result 14.33 to the screen. The bc command is a calculator it performs math calculation in the terminal.

Script6.sh

The code for script6.sh is shown below

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Show users who are logged on
# Script: script6

for i in 1 2 3 4 5
```

```
do
    echo "Looping ... number $i"
done
exit 0
```

The code performs a for loop, looping through the values 1,2,3,4,5 in order. Do marks the beginning of the loop and done marks the end of the loop. The code between these is done for each loop iteration. For our code it prints out the current loop value of i.

Script7.sh

Script7.sh code is shown below.

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: List all the files in home directory
# Script: script7

for file in ~
do
    ls -l
done
exit 0
```

The code performs a for loop, that loops through all files in a home directory represented as ~ in the code, so the loop runs 1 time for the home directory.

Script8.sh

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: While loop example
# Script: script8

INPUT_STRING=hello
while [ "$INPUT_STRING" != "bye" ]
do
    echo "Please type something in (bye to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
exit 0
```

The code initialises INPUT_STRING variable equal to hello, we then enter inside the while loop because the INPUT_VARIABLE is “hello” which isn’t equal to “bye, we then print out a message saying “please type something in (bye to quit). Then we get input from the user and store the input in the INPUT_STRING VARIABLE, we print out what was typed, now we loop back to the start of the loop and check what the INPUT_STRING value is, if it is not bye we perform another iteration of the loop otherwise if it is bye we exit out of the loop and the script finishes.

Script9.sh

The code for script9.sh is below.

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Check language for hello greeting
# Script: script9

echo "press [ctrl][c] to exit"

while read f
do
case $f in
    Hello)                echo English ;;
    "Dia Duit")           echo Irish ;;
    Ciao)                 echo Italian ;;
    *)                    echo Unknown: $f;;
esac
done
exit 0
```

The code begins by printing out press ctrl c to exit. It then enters a while loop where it reads in from the keyboard and places the value in a variable f. In the while loop there is a case statement

If the variable f is “Hello” it prints out English
If the variable f is “Dia Duit” it prints out Irish
If the variable f is “Ciao” it prints out Italian
Otherwise it will print out Unknown, this is defined by using *.

The script will keep looping and repeat until the user presses ctrl c which causes the loop to break and the code to exit

Script10.sh

The code for script10.sh is below.

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Simple number check (takes input from user)
# Script: script10_modified

# Ask user for input
read -p "Enter a number: " X

# Check if less than zero
if [ "$X" -lt 0 ]
then
    echo "X is less than zero"
    exit 0
```

```
else
    echo "X is greater than zero"
    exit 0
fi
```

The code prompts the user for a value and stores the entered value into a variable X. We then Check if it less than 0 using a for loop, if it less than 0, we print X is less than zero and then exit. If X is 0 or greater than 0, it prints out X is greater than zero and then exits.

Script11.sh

The code for script11.sh is below.

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Simple number check
# Script: script11

for indexnumber in `seq 10`
do
    touch TextFile$indexnumber.txt
done
```

The for-loop loops through number in sequence from 1,2,3 … up to 10. In the for loop we create an empty file called TextFile with the current$indexnumber appened to it and a .txt extension. Thus, on the first iteration of the loop we create a file called TextFile1.txt, on 2nd iteration TextFile2.txt and so on until TextFile10.txt

Script12.sh

The code snippet below is for Script12.sh

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Command Line Arguments
# Script: script12

echo "This script was called with $# arguments"
echo "This script was called by $0"
echo "The first parameter passed was $1, the second was $2"
exit 0
```

The code snippet prints out the various command line arguments

The variable $# - is the number of command line arguments
The variable $0 – is the command typed to call the script
The variable $1 – is the 1st parameter passed, $2 is the 2nd and so on

Script13.sh

The code snippet below is for Script13.sh

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Command Line Arguments
# Script: script13

echo "The parameters are arg1=$1   arg2=$2   arg3=$3"
shift
echo "The parameters are arg1=$1   arg2=$2   arg3=$3"
shift
echo "The parameters are arg1=$1   arg2=$2   arg3=$3"
shift
echo "The parameters are arg1=$1   arg2=$2   arg3=$3"
shift
exit 0
```

shift moves the argument value, from argument n to argument n-1. Argument 1 will be shifted out and replaced with argument 2's value.

Script14.sh

The code snippet below is for Script14.sh

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Command Line Arguments
# Script: script14

set $(date)
echo $*
echo
echo "It is $1"
echo "The date is $3 $2"

exit 0
```

Script 14 sets the command line arguments to the current date, it then prints out all the command line arguments using $* and finally it prints out 1st argument and then the 3rd and 2nd.

Script15.sh

The code snippet below is for Script15.sh is below

```
#!/bin/bash

# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Calculations
# Script: script15

echo "This script adds two numbers together"

# Check to see if we have two arguments
if [ $# -ne 2 ]
then
  echo "Usage - $0 x y"
  echo "Where x and y are summed"
  exit 1
else
  echo "$1 plus $2 is `expr $1 + $2`"
fi

exit 0
```

This script first checks to see if there are not 2 arguments with the line of code `if [ $# -ne 2 ]` -ne stands for not equal it checks that the number of arguments is not equal to 2, if there are not 2 arguments it prints usage instructions and exits with 1 to signal script didn't run successfully. If there are 2 arguments it adds and prints the result.

Script16.sh

The code snippet below is for Script16.sh is below

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Calculations
# Script: script16

if [ $# -ne 1 ]
then
  echo "Usage - $0 x"
  echo "Where x will be tested"
  exit 1
fi

echo "This script checks one argument"

if [ $1 -lt 0 ]
then
  echo "$1 is less than zero"
elif [ $1 -eq 0 ]
then
  echo "$1 is zero!"
elif [ $1 -gt 0 ]
then
  echo "$1 is greater than zero"
```

```
fi
exit 0
```

Script16.sh checks that 1 argument was passed to the script it does this in the line `if [ $# -ne 1 ]` -ne tells BASH to evaluate for not equal, if this is not true it prints usage instructions to the screen and exits with code 1 to say script didn't run successfully. If 1 argument is passed it checks if this arguments value is less than 0 with line `if [ $1 -lt 0 ]` -lt stands for less than, it checks argument 1 is less than 0 if true it prints a message saying its less than 0. It also checks if it is equal to 0 with line `elif [ $1 -eq 0 ]`, eq stands for equal and if true prints a message saying it is 0. Finally at line `elif [ $1 -gt 0 ]` it checks if its value greater than 0 if true it displays a message saying its greater than 0

Script17.sh

The code snippet below is for Script17.sh is below

```
#!/bin/bash
# By: Edmund Connolly
# Date: 30 OCT 2025
# Function: Test for File and Directory
# Script: script17

echo "This script checks a file and directory to see if they exist"

# Check to see if we have two arguments
if [ $# -ne 2 ]
then
  echo "Usage - $0 x y"
  echo "Where x and y are FileName and Directory"
  exit 1
fi

if [ -f $2/$1 ]
then
  echo "This filename [$1] exists"
elif [ -d $1 ]
then
  echo "This dirname [$2] exists"
else
  echo "Neither [$2] or [$1] exist"
fi

exit 0
```

It first checks to see that 2 arguments were passed, it does this by line `if [ $# -ne 2 ]`
Here we check the number of arguments is "not equal" to 2 $1 is the filename and $2 id the directory. If 2 arguments were not passed it displays usage instructions. If 2 parameters were passed

it checks the filename and directory exist and prints the appropriate message. We check to se if a file exists by specifying -f `if [ -f $2/$1 ]`, we check for a directory by using -d `elif [ -d $1 ]`

Results and Testing

The BASH scripts were tested by executing them and observing their output, screenshots of the results from execution can be found in the appendix.

Script1.sh Results

The results of script1.sh can be seen in figure 2. We can see there are 2 result entries.

Saned – a system account used for the *SANE* (Scanner Access Now Easy) daemon, not a real person. It has /usr/sbin/nologin, meaning it cannot log in.

Ed – My user account It has /bin/bash as a shell, meaning you can log in and run commands. The Ed account is the one specified in the $USER variable

Script2.sh Results

The results of script2.sh can be seen in figure 3. It prints a welcome message to the current user and then it lists all the users are logged in, In our results there is 1 user logged in “ed”, :1 means the user is logged into a GUI, next is the date and time that the user logged in.

Script3.sh Results

In figure 4 we can see the list of processes running.

Identifier	Name	Explanation
UID	User ID	The user who started or owns the process
PID	Process ID	Unique number which identifies the process
PPID	Parent Process ID	PID of parent process which started this one
C	CPU Utilisation	Percentage of CPU recently used by process
STIME	Start Time	Process start time or date
TTY	Controlling Terminal	Terminal device process is attached to
TIME	CPU Time	Total amount of cpu time process has used
CMD	Command	Command used to start the process

In figure 5 we can see the process tree, with parent and child relationships shown with lines.

Script4.sh Results

In figure 6, we can see the result of executing the code, the message Good Morning! Is displayed on the screen because this is the value that was stored in the MY_MESSAGE variable.

Script5.sh Results

In figure 7 we can see the result of adding the variable a and b together.

Script6.sh Results

Figure 8 shows the result of executing script6.sh, we print out the current value of I for each iteration of the for loop. Thus, our value of I loops giving values of 1,2,3,4,5

Script7.sh Results

Figure 9 shows the result of running script7.sh. The output list's all the files in current directory it is similar to just running ls -l, except executable files are coloured in green.

Script8.sh Results

Figure 10 shows the result of running script8.sh. The script keeps looping and printing out "Please type something in (bye to quit)" , until the user types in bye then the loop terminates and the script stops.

Script9.sh Results

Figure 11 shows the result of running script8.sh

When "test" is typed it prints out Unknown because this evaluates as * .
When "Hello" is typed it prints out English
When "Dia Duit" is typed it prints out Irish
When "Ciao" it prints out Italian

Script10.sh Results

Figure 12 shows the result of running script10.sh

We can see when the user enters 5 for x's value we get a message x is greater than 0, and when the user enters -2 for x's value we get a message x is less than 0

Script11.sh Results

Figure 13 shows the result of running script11.sh

The program creates 10 empty files called TextFile1.txt, TextFile2.txt up to TextFile10.txt in the directory the script is located

Script12.sh Results

Figure 14 shows the result of running script12.sh when we issue the command ./script12.sh dog cat. We can see that there are 2 arguments dog and cat. The command to run the script is ./script12.sh, and dog is the first command line parameter and cat is the 2nd parameter.

Script13.sh Results

Figure 15 shows the result of running script13.sh when we issue the command ./script12.sh dog cat mouse. We can see the effect of shift, shift moves the arguments from higher argument value to a lower argument value it shift argument n to argument n-1. Argument 1 is shifted out and is gone after the shift but its replaced by argument 2's value.

Script14.sh Results

Figure 16 shows the result of running script14.sh, when we set the command line arguments to the current date and time, we can now access all arguments using $* and individually using their position

The script set the arguments to be Sat 01 Nov 2025 14:09:44 GMT, this is 6 arguments we then print out the first which is Sat, and the 3rd and 2nd which are Nov 1

Script15.sh Results

Figure 17 shows the result of running script15.sh, when there are 2 command line arguments the script adds these values and prints the result. When the number of arguments is not equal 2 usage instructions are printed on how to use the script

Script16.sh Results

Figure 18 shows the results of running script16.sh with 3 different values, when -5 is passed we get a message saying its less than 0, when 0 is passed we get message saying the value 0 was passed, when we pass 10 we get message saying value passed was greater. When 2 arguments are passed we get usage instructions on how to use the script printed to the screen

Script17.sh Results

Figure 19 shows the results of running script17.sh, we can see when we enter 1 argument we get usage instructions, and when we enter 2 it checks if the file exists in a directory. If it does it prints a message saying it exists other wise it prints either the directory or filename doesn't exist.

Conclusions

In this assignment all our aims were met, we became familiar with the Linux terminal, its file structure and commands to change file permissions of the scripts and then execute them. For Independent learning I looked at the Linux File Permissions, there are three types of users: owner, group and others. Then foreach user type they have their own permissions read, write and execute [1].

- Read means they can read the file
- Write means they can edit the file
- Execute means they can run the file

We used Linux built in commands such as echo to print to the STDOUT, in this case the screen, we used Linux bash environment variables to see who the current user is. Then I used grep which is a command-line tool used to search for text patterns inside files. It scans through text and print lines that match a given pattern [2]. Combining the use of grep, echo and the $USER environment variable we looked through a password file and found that the user was stored in the password file. This confirmed that I was a user.

I looked at processes in Linux, they are defined as a running instance of a programme, each process is identified by a unique process ID or PID. Linux runs multiple processes at the same time. In the lab I used the built-in ps command [3] to list all processes currently running on ubuntu, their PID and the user who created them, this is very useful for system administrators if they want to identify who is running a process, and using the PID they can "kill" the process if there is a problem with it or they need to reclaim resources. The pstree [4] command displays an ASCII tree view of the processes, using this we can see the relationship between parent, and child processes.

In some Linux scripts we investigated exit codes [5], when a process terminates in Linux it returns an eight-bit code so value 0 to 255. Linux has standardised some exit codes for example it defines if the return value is 0, that it was successful, whereas any other value indicates an error, for example 127 means command not found. Using these exit codes, we can determine what is wrong and using if statements we can take corrective action.

We also experimented with printing output to the screen, creating variables and accessing them, loops to repeat code a number of times, and later looked at if statements to execute certain code based on arguments passed to the script such as a directory and file and tailor the output based on whether they exist or not.

References

[1] R. Linux, "Linux file permissions explained," [Online]. Available: https://www.redhat.com/en/blog/linux-file-permissions-explained. [Accessed 8 Nov 2025].

[2] "Grep(1) - linux manual page," *man7.org*, Dec. 29, 2019. https://man7.org/linux/man-pages/man1/grep.1.html

[3] "ps(1) - Linux manual page," *man7.org*. https://man7.org/linux/man-pages/man1/ps.1.html

[4] "pstree(1) - Linux manual page," *Man7.org*, 2025. https://www.man7.org/linux/man-pages/man1/pstree.1.html (accessed Nov. 19, 2025).

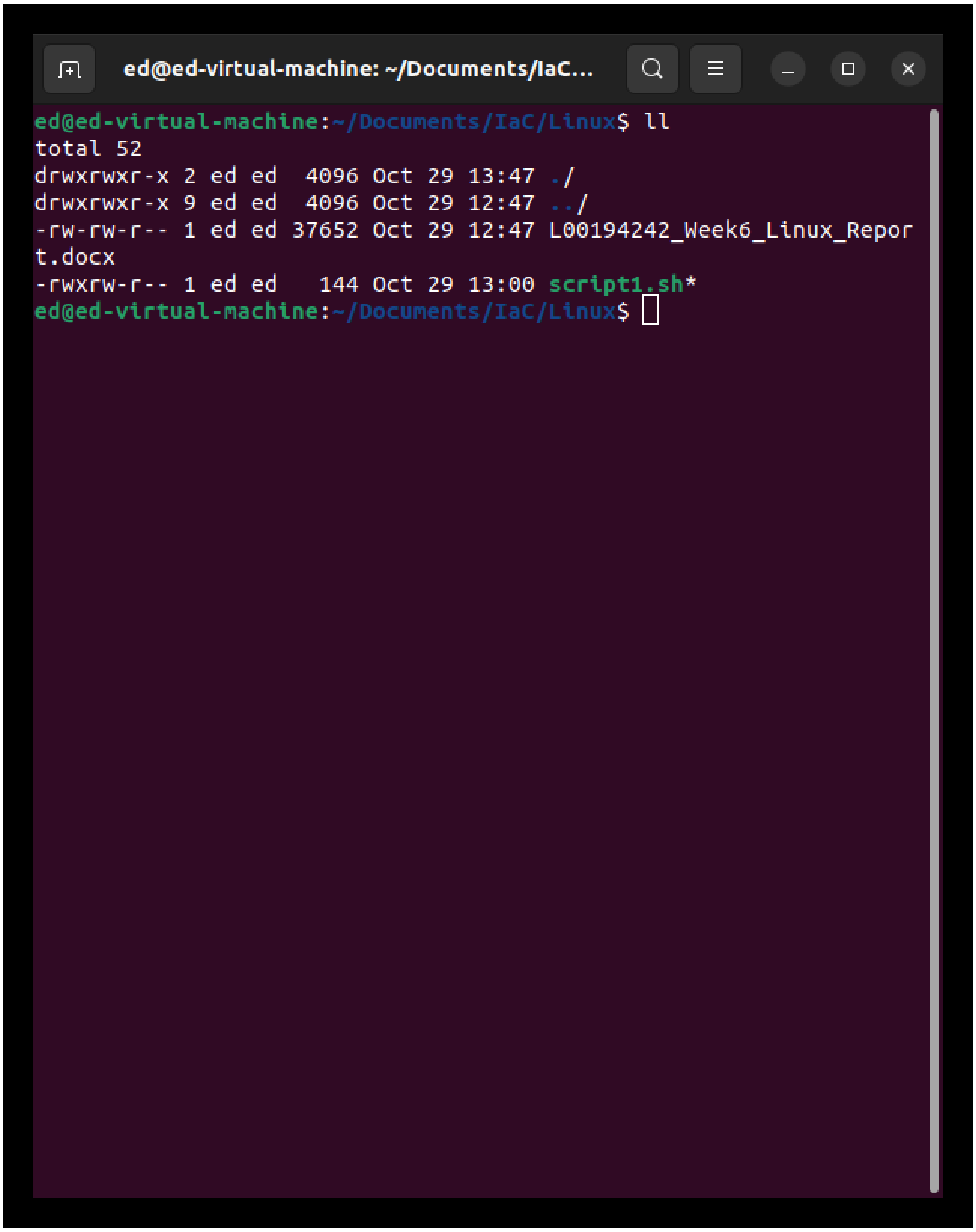
[5] sandra_henrystocker, "Understanding exit codes on Linux," *Network World*, Oct. 04, 2024. https://www.networkworld.com/article/3546937/understanding-exit-codes-on-linux-2.html

GNU Project. *Bash Reference Manual*. [Online]. Available: https://www.gnu.org/software/bash/manual/

DigitalOcean Tutorials. *Introduction to Bash Scripting*. [Online]. Available: https://www.digitalocean.com/community/tutorials

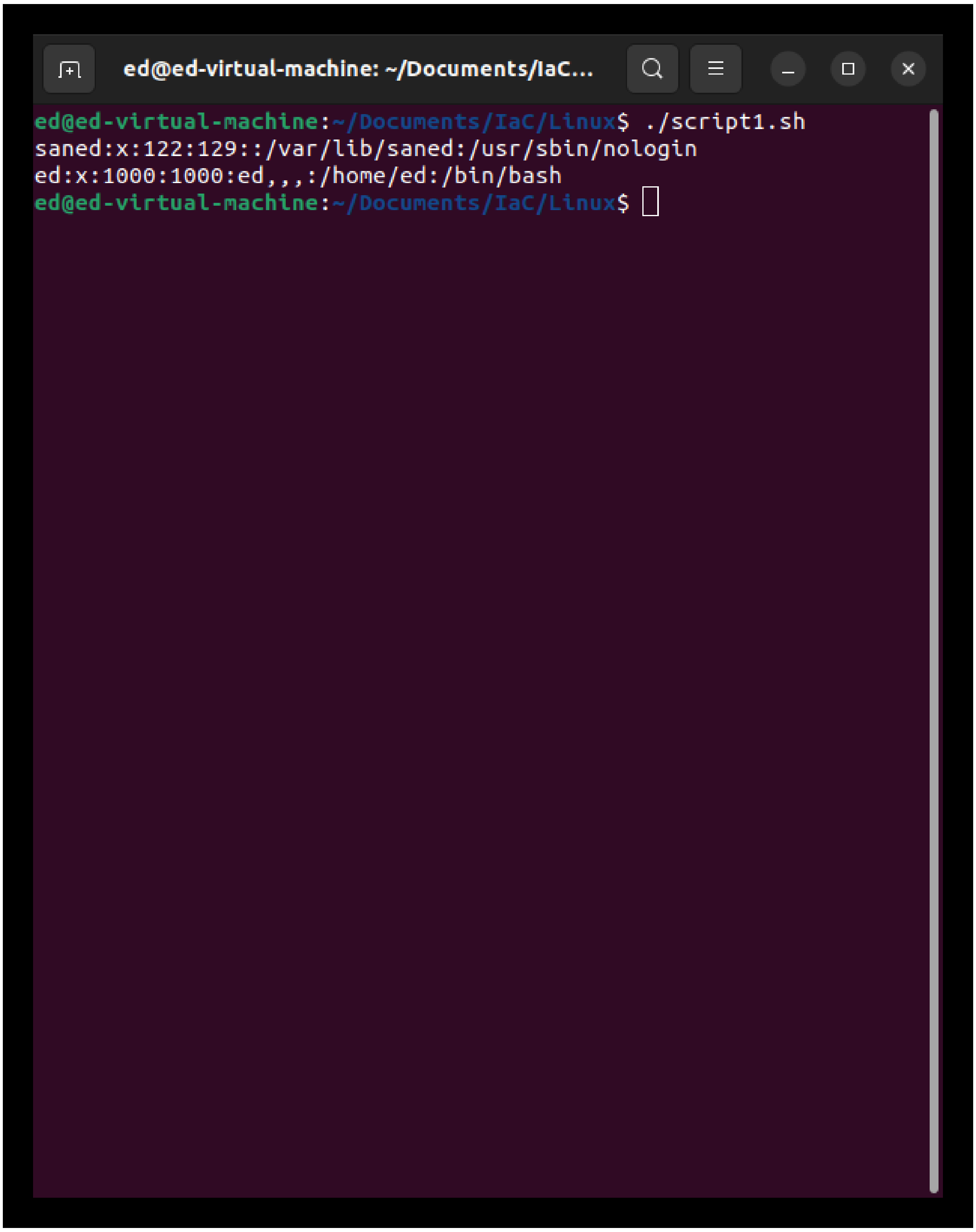
Linuxize. *Bash For Loops Explained with Examples*. [Online]. Available: https://linuxize.com/

Appendices



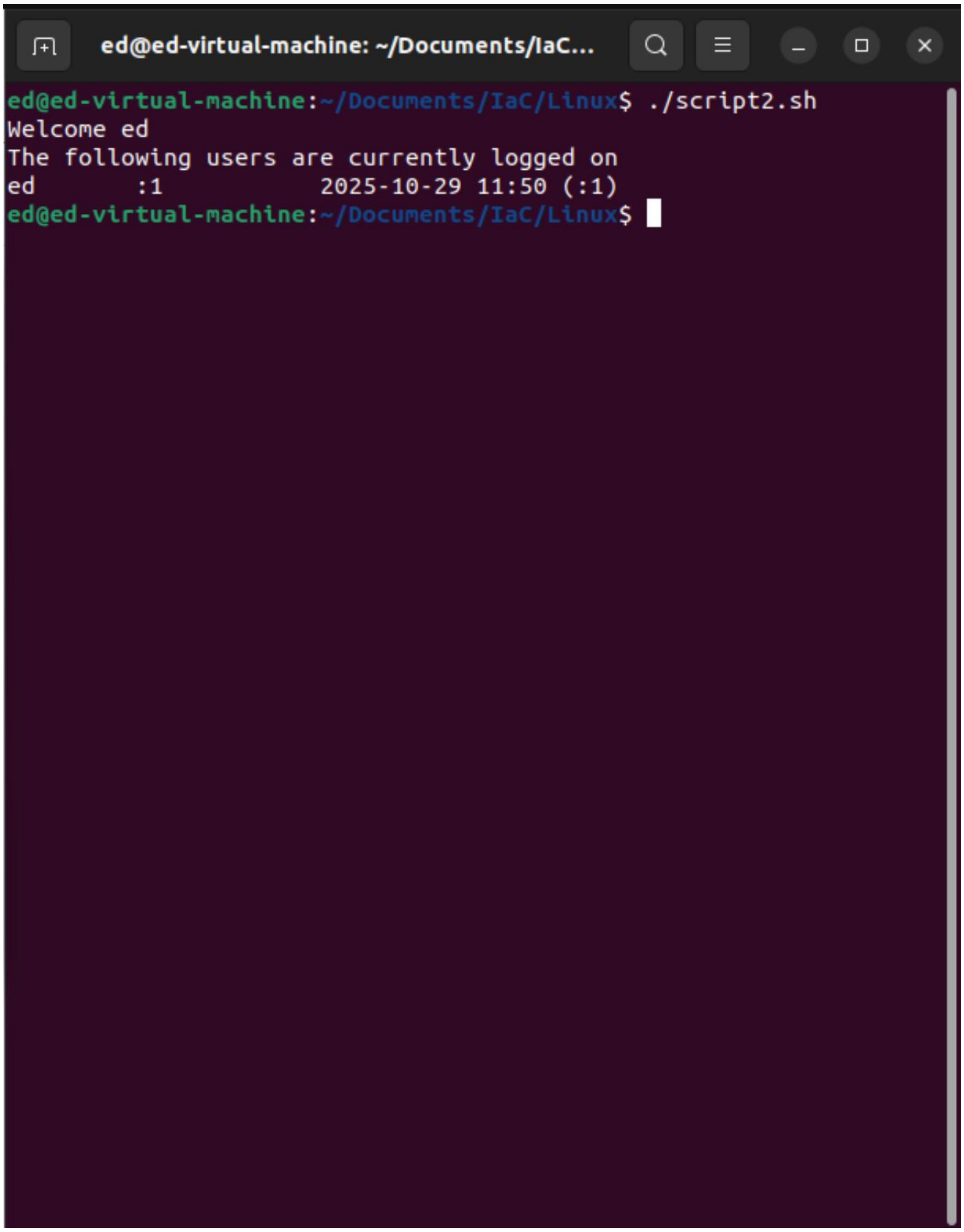
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ll
total 52
drwxrwxr-x 2 ed ed  4096 Oct 29 13:47 ./
drwxrwxr-x 9 ed ed  4096 Oct 29 12:47 ../
-rw-rw-r-- 1 ed ed 37652 Oct 29 12:47 L00194242_Week6_Linux_Repor
t.docx
-rwxrw-r-- 1 ed ed   144 Oct 29 13:00 script1.sh*
ed@ed-virtual-machine:~/Documents/IaC/Linux$ 
```

Figure 1 - Result of executing chmod on script1.sh



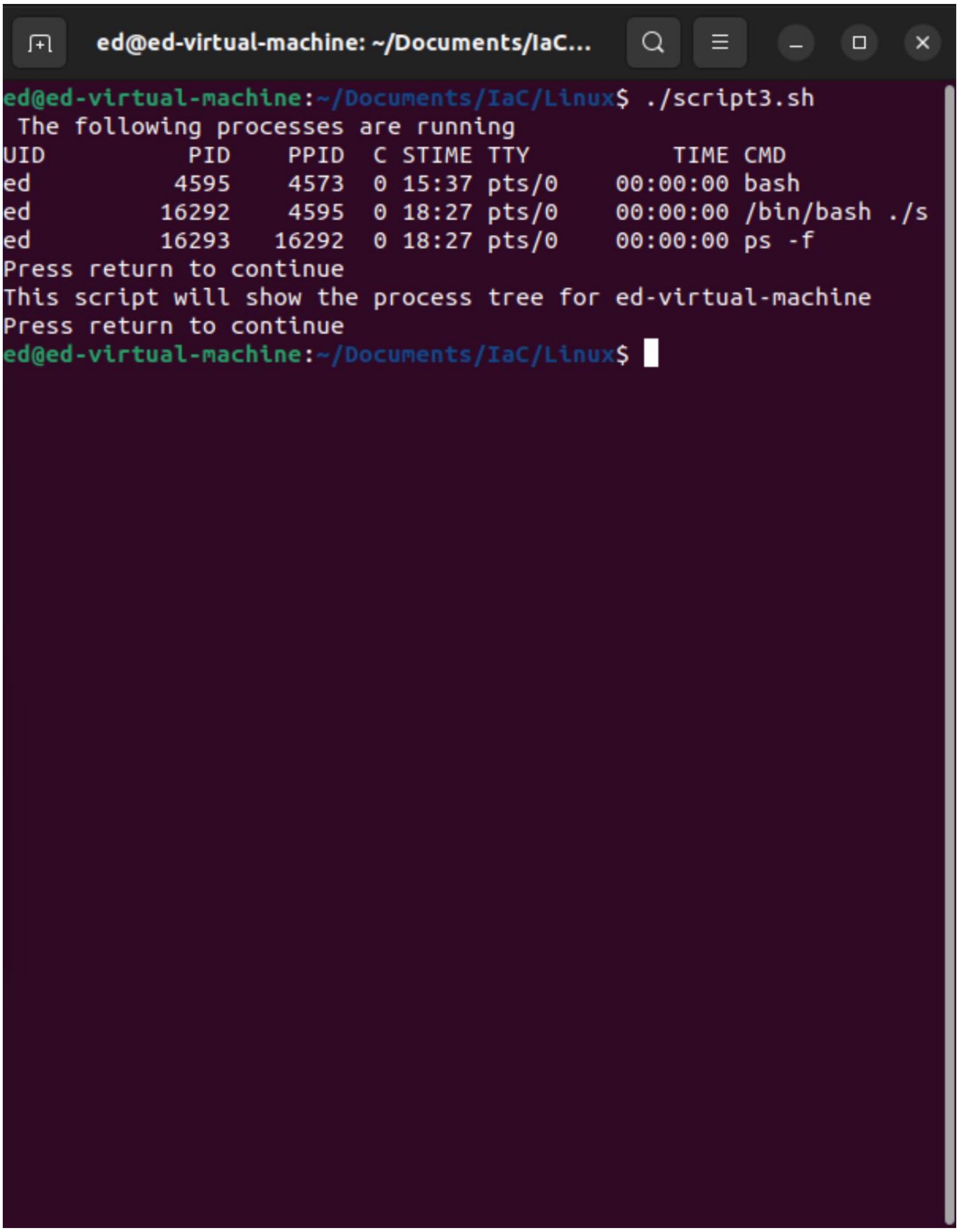
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script1.sh
saned:x:122:129::/var/lib/saned:/usr/sbin/nologin
ed:x:1000:1000:ed,,,:/home/ed:/bin/bash
ed@ed-virtual-machine:~/Documents/IaC/Linux$ 
```

Figure 2 - Result of executing script1.sh



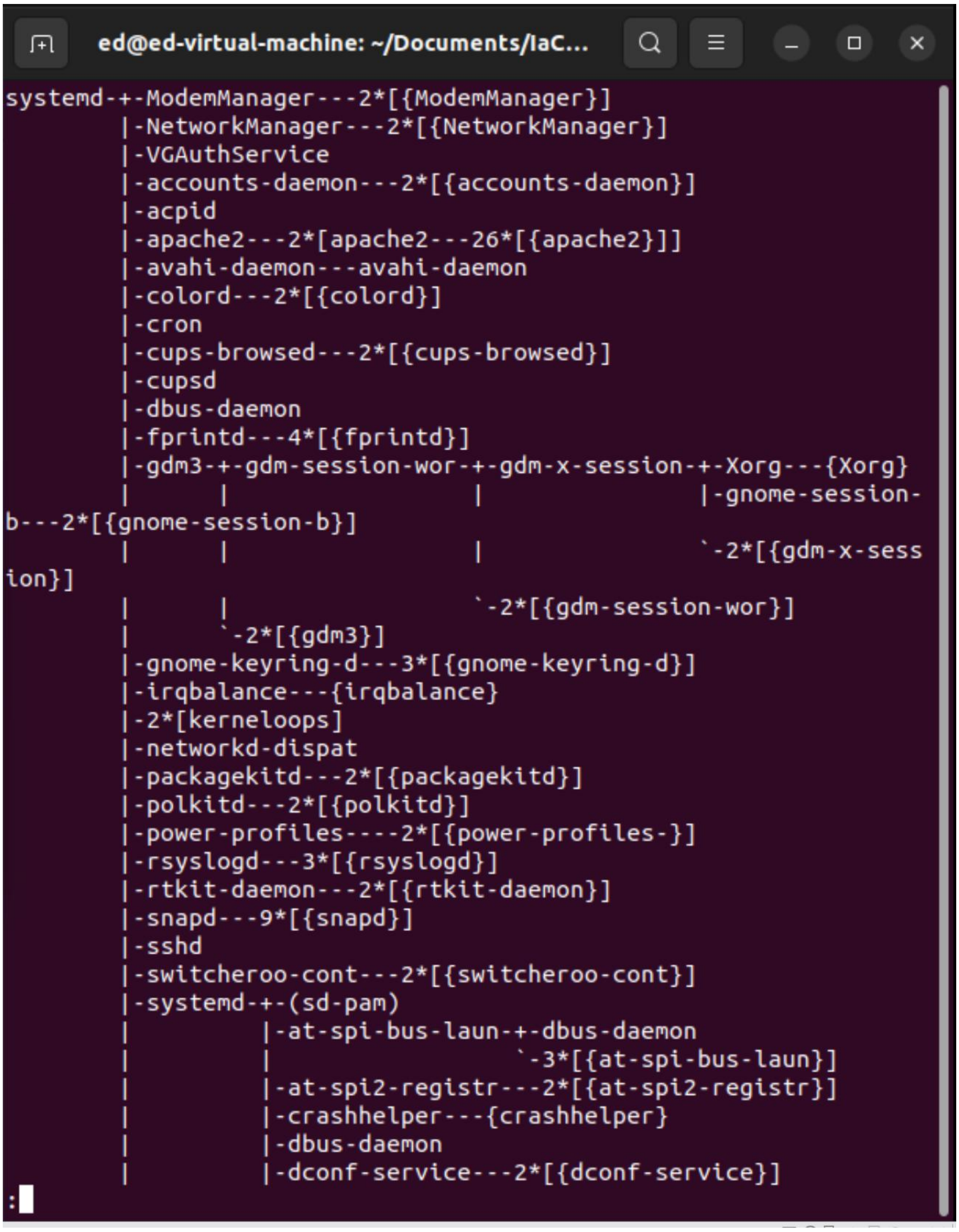
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script2.sh
Welcome ed
The following users are currently logged on
ed       :1           2025-10-29 11:50 (:1)
ed@ed-virtual-machine:~/Documents/IaC/Linux$
```

Figure 3 - Results of script2.sh execution


ed@ed-virtual-machine: ~/Documents/IaC...

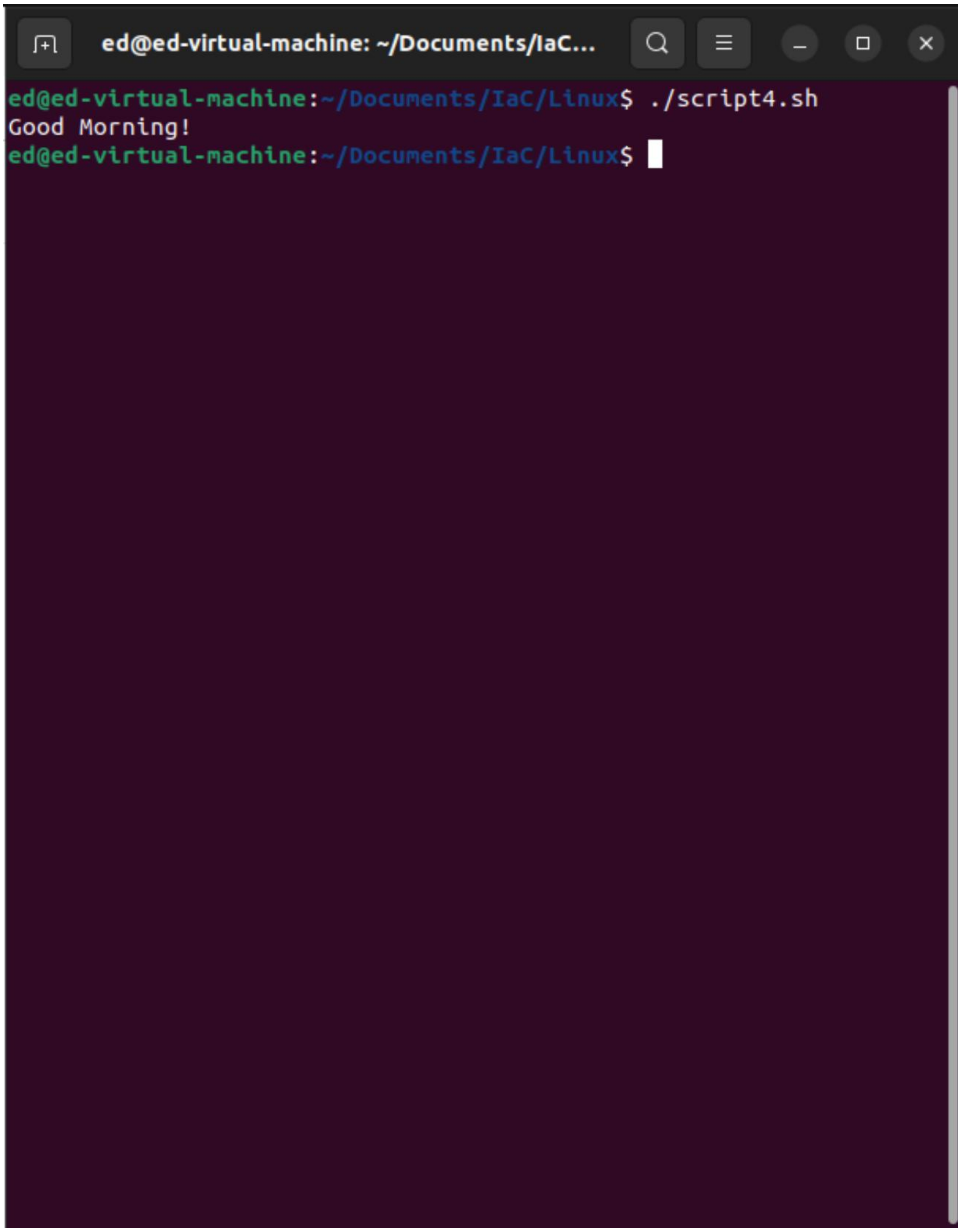
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script3.sh
 The following processes are running
UID          PID    PPID  C STIME TTY          TIME CMD
ed          4595    4573  0 15:37 pts/0    00:00:00 bash
ed         16292    4595  0 18:27 pts/0    00:00:00 /bin/bash ./s
ed         16293   16292  0 18:27 pts/0    00:00:00 ps -f
Press return to continue
This script will show the process tree for ed-virtual-machine
Press return to continue
ed@ed-virtual-machine:~/Documents/IaC/Linux$ 
```

Figure 4 - Script3.sh Results – processes



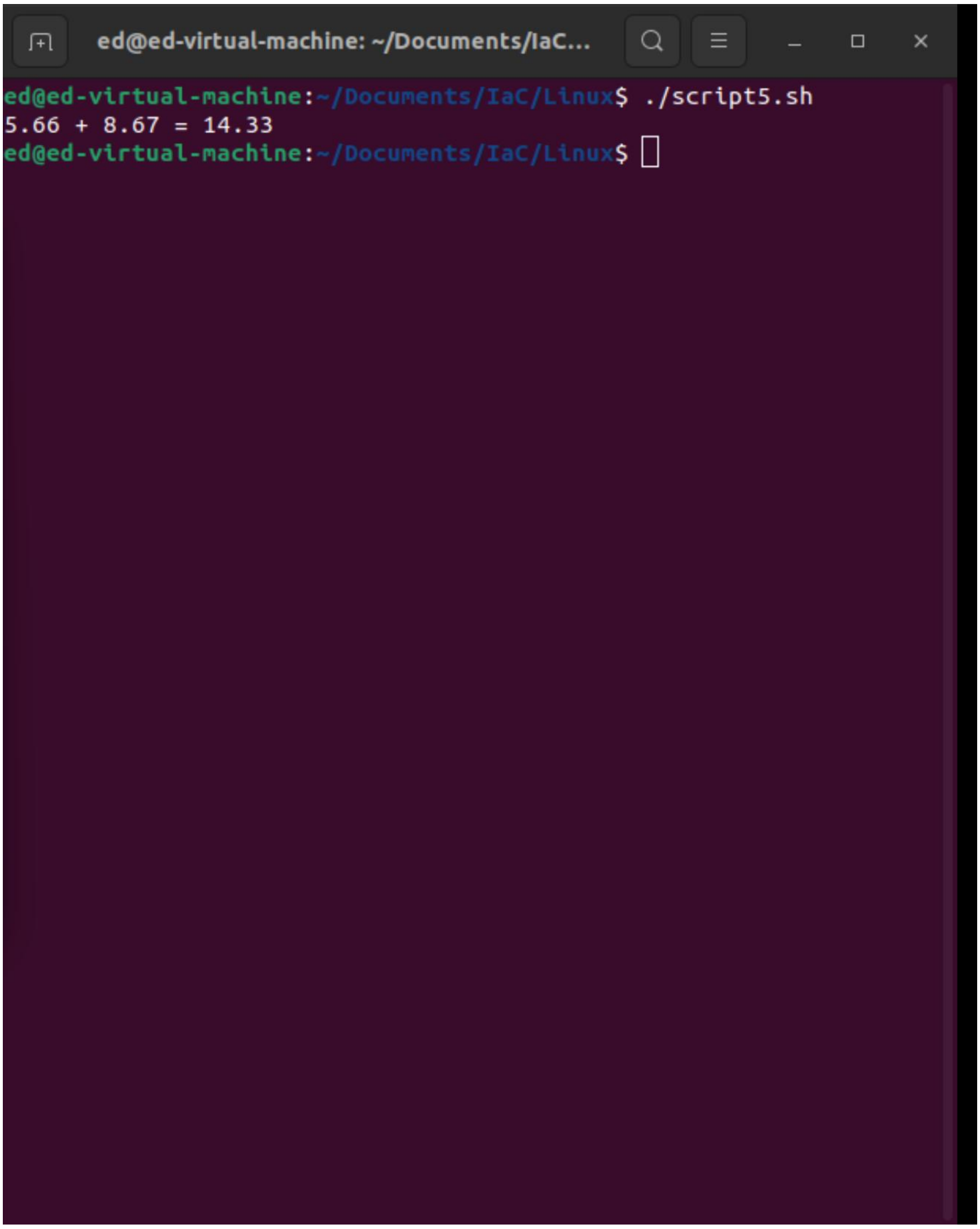
```
systemd-+-ModemManager---2*[{ModemManager}]
        |-NetworkManager---2*[{NetworkManager}]
        |-VGAuthService
        |-accounts-daemon---2*[{accounts-daemon}]
        |-acpid
        |-apache2---2*[apache2---26*[{apache2}]]
        |-avahi-daemon---avahi-daemon
        |-colord---2*[{colord}]
        |-cron
        |-cups-browsed---2*[{cups-browsed}]
        |-cupsd
        |-dbus-daemon
        |-fprintd---4*[{fprintd}]
        |-gdm3-+-gdm-session-wor-+-gdm-x-session-+-Xorg---{Xorg}
        |      |                 |               |-gnome-session-
b---2*[{gnome-session-b}]
        |      |                 |               `-2*[{gdm-x-sess
ion}]
        |      |                 `-2*[{gdm-session-wor}]
        |      `-2*[{gdm3}]
        |-gnome-keyring-d---3*[{gnome-keyring-d}]
        |-irqbalance---{irqbalance}
        |-2*[kerneloops]
        |-networkd-dispat
        |-packagekitd---2*[{packagekitd}]
        |-polkitd---2*[{polkitd}]
        |-power-profiles----2*[{power-profiles-}]
        |-rsyslogd---3*[{rsyslogd}]
        |-rtkit-daemon---2*[{rtkit-daemon}]
        |-snapd---9*[{snapd}]
        |-sshd
        |-switcheroo-cont---2*[{switcheroo-cont}]
        |-systemd-+-(sd-pam)
        |         |-at-spi-bus-laun-+-dbus-daemon
        |         |                 `-3*[{at-spi-bus-laun}]
        |         |-at-spi2-registr---2*[{at-spi2-registr}]
        |         |-crashhelper---{crashhelper}
        |         |-dbus-daemon
        |         |-dconf-service---2*[{dconf-service}]
:
```

Figure 5 - script 3 results process tree



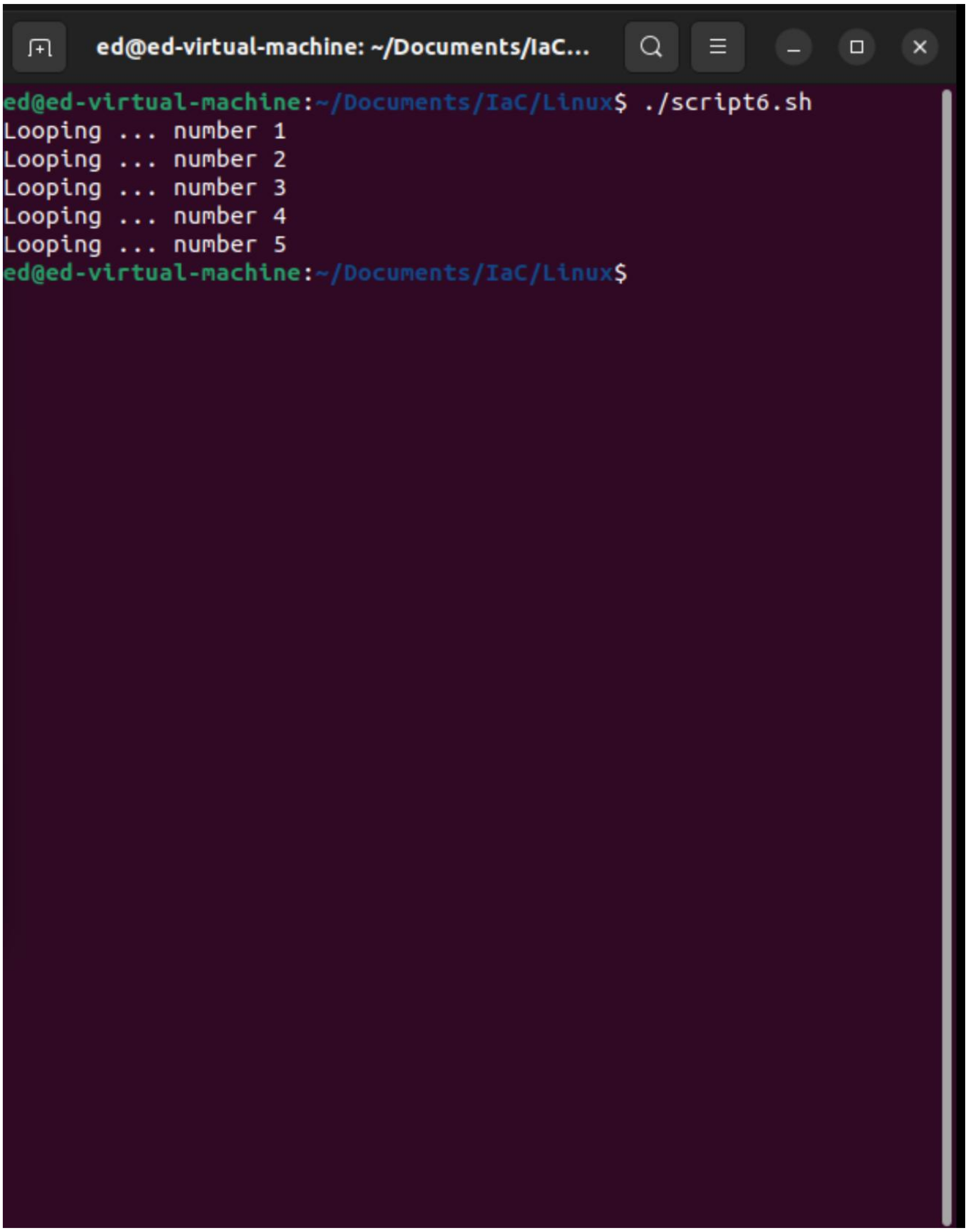
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script4.sh
Good Morning!
ed@ed-virtual-machine:~/Documents/IaC/Linux$ 
```

Figure 6 - Result of executing script4.sh



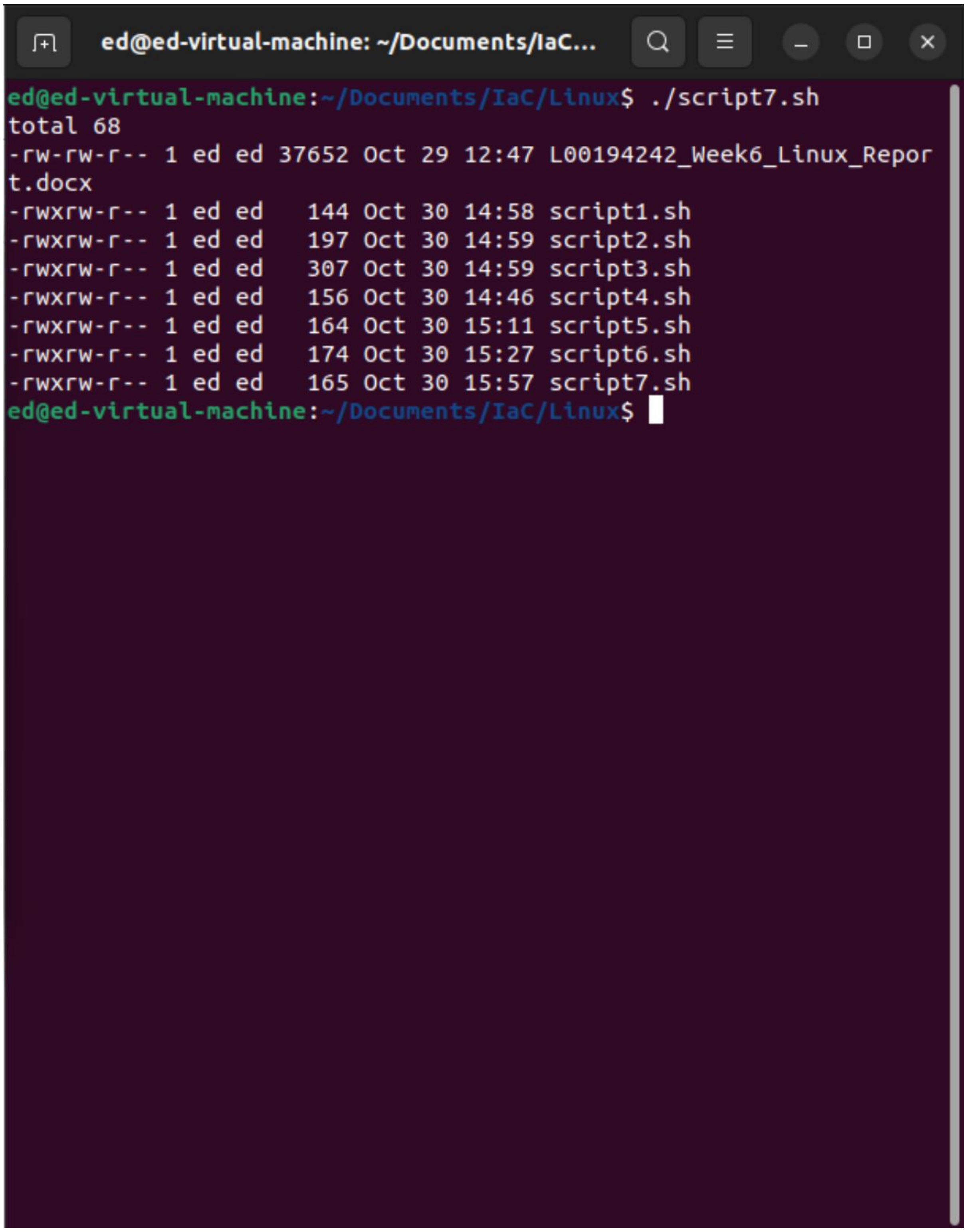
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script5.sh
5.66 + 8.67 = 14.33
ed@ed-virtual-machine:~/Documents/IaC/Linux$ 
```

Figure 7 - Result of execution of script5.sh



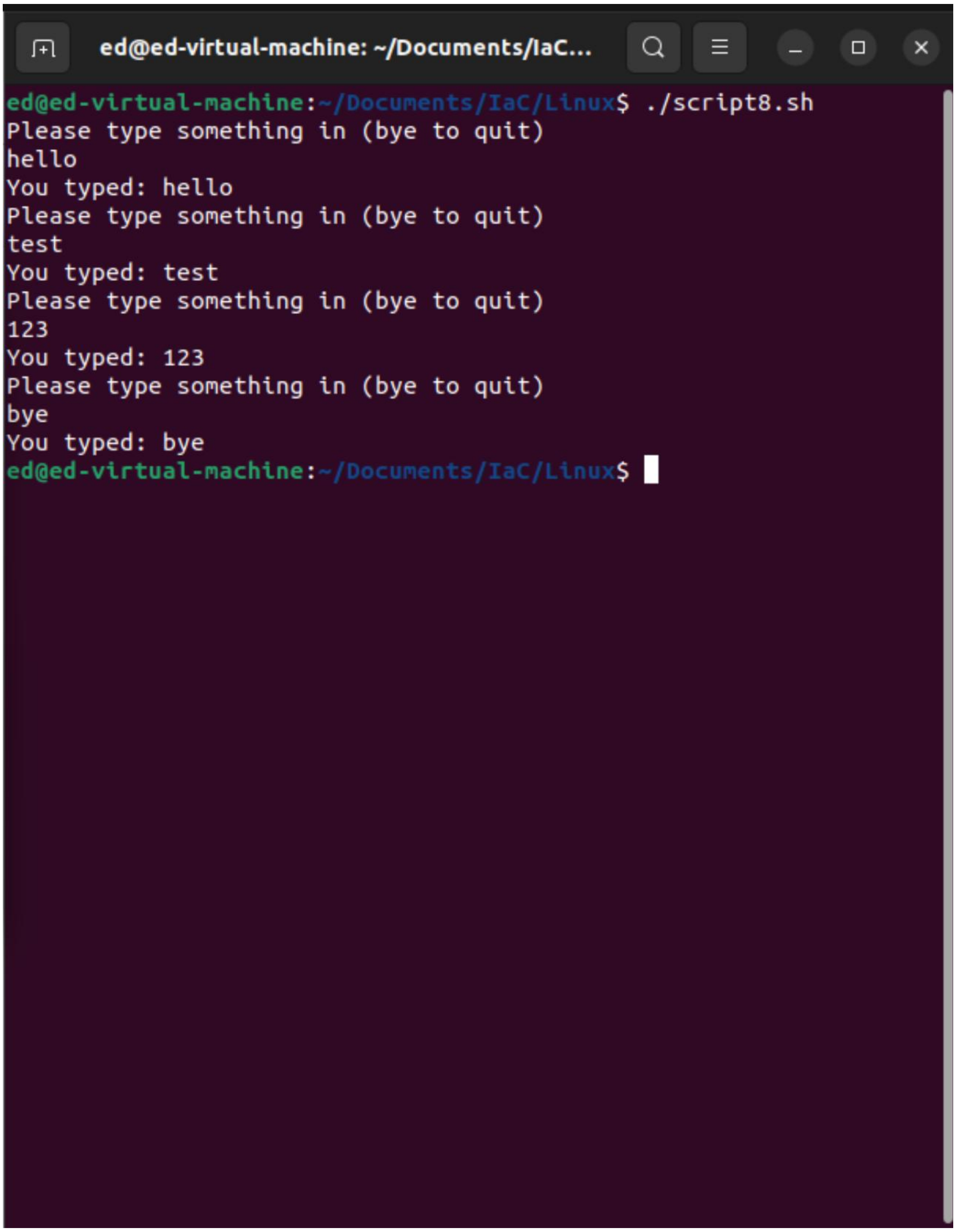
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script6.sh
Looping ... number 1
Looping ... number 2
Looping ... number 3
Looping ... number 4
Looping ... number 5
ed@ed-virtual-machine:~/Documents/IaC/Linux$
```

Figure 8 - Result for script6.sh



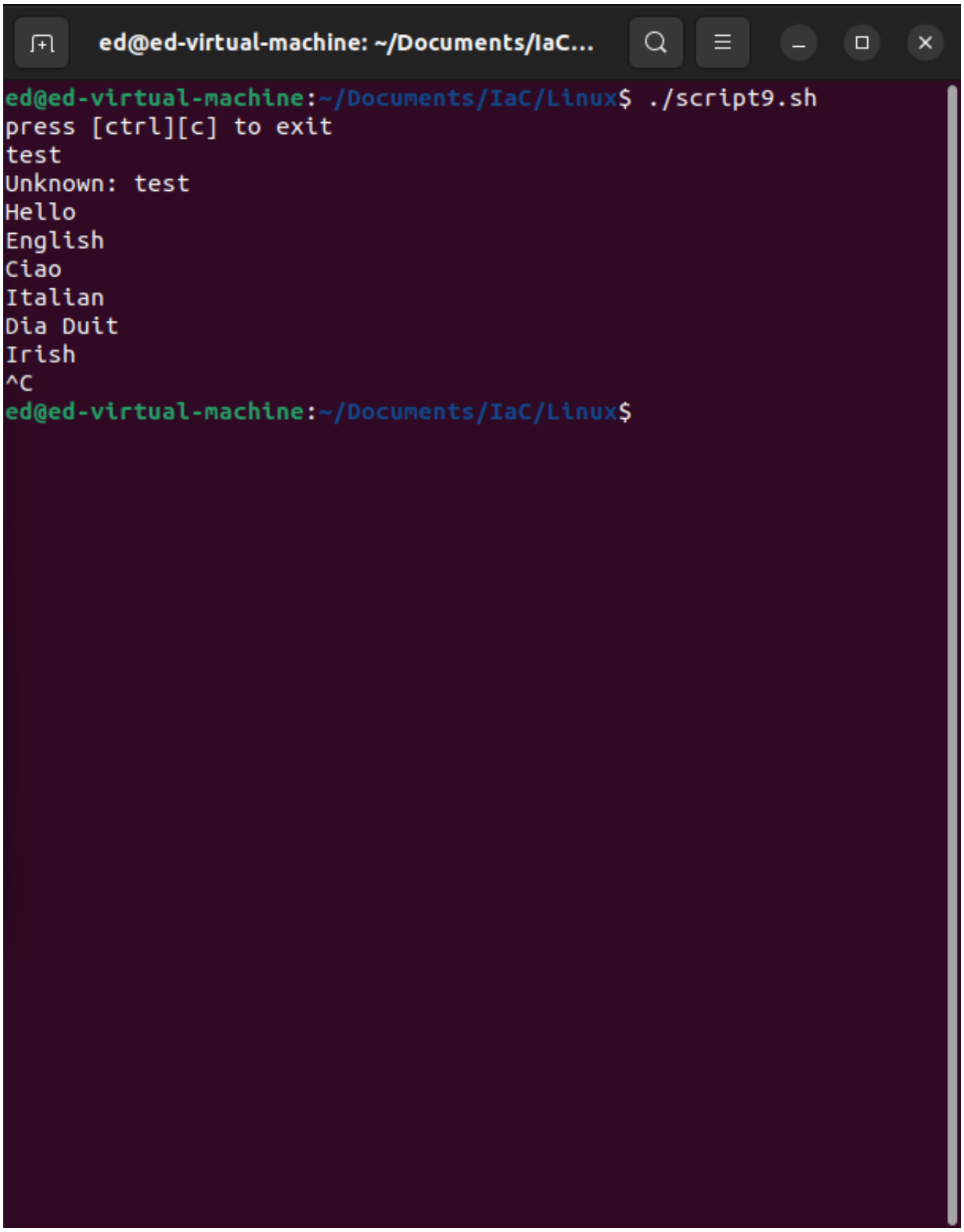
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script7.sh
total 68
-rw-rw-r-- 1 ed ed 37652 Oct 29 12:47 L00194242_Week6_Linux_Repor
t.docx
-rwxrw-r-- 1 ed ed   144 Oct 30 14:58 script1.sh
-rwxrw-r-- 1 ed ed   197 Oct 30 14:59 script2.sh
-rwxrw-r-- 1 ed ed   307 Oct 30 14:59 script3.sh
-rwxrw-r-- 1 ed ed   156 Oct 30 14:46 script4.sh
-rwxrw-r-- 1 ed ed   164 Oct 30 15:11 script5.sh
-rwxrw-r-- 1 ed ed   174 Oct 30 15:27 script6.sh
-rwxrw-r-- 1 ed ed   165 Oct 30 15:57 script7.sh
ed@ed-virtual-machine:~/Documents/IaC/Linux$ 
```

Figure 9 – Results for script7.sh



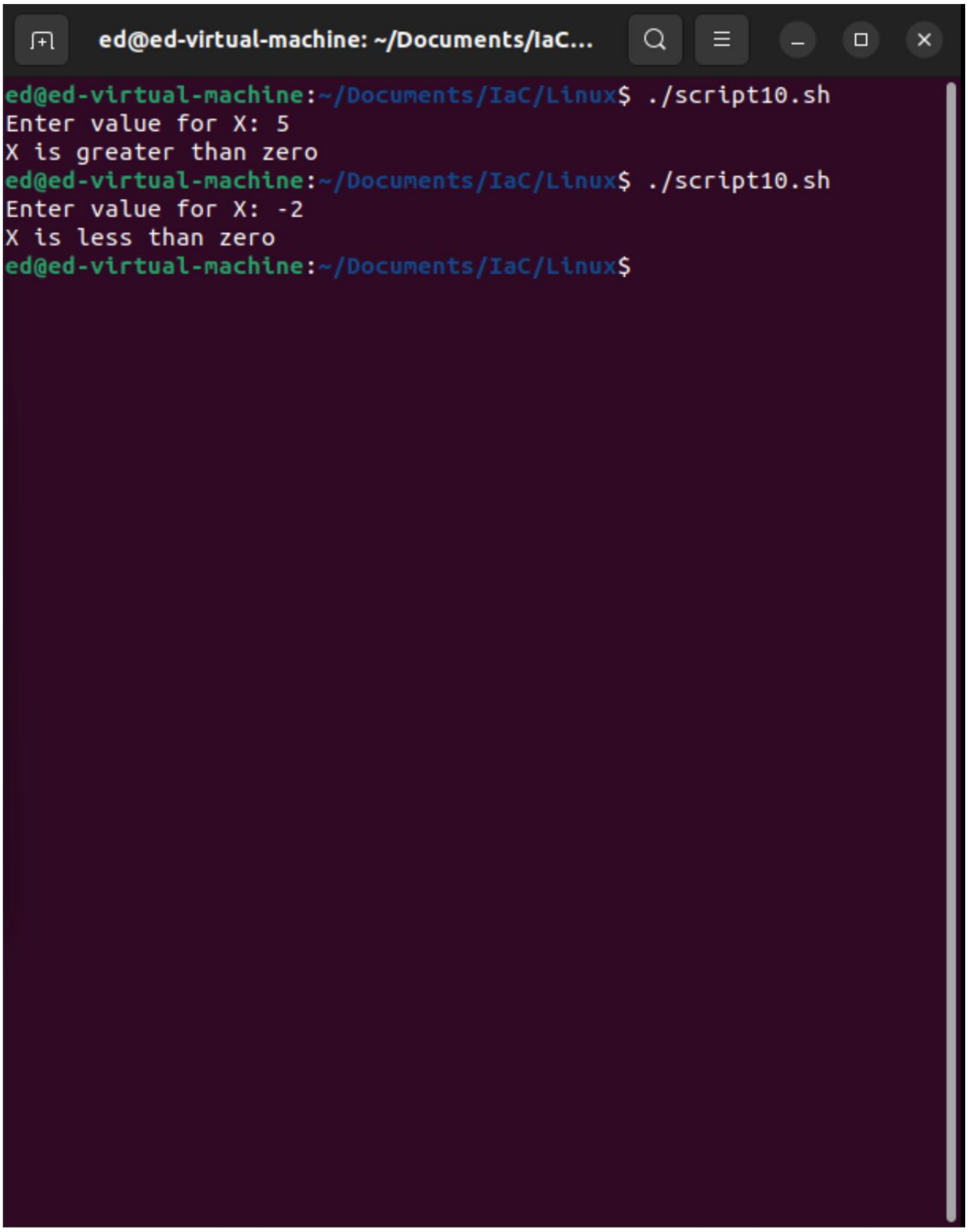
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script8.sh
Please type something in (bye to quit)
hello
You typed: hello
Please type something in (bye to quit)
test
You typed: test
Please type something in (bye to quit)
123
You typed: 123
Please type something in (bye to quit)
bye
You typed: bye
ed@ed-virtual-machine:~/Documents/IaC/Linux$
```

Figure 10 - Results of script8.sh



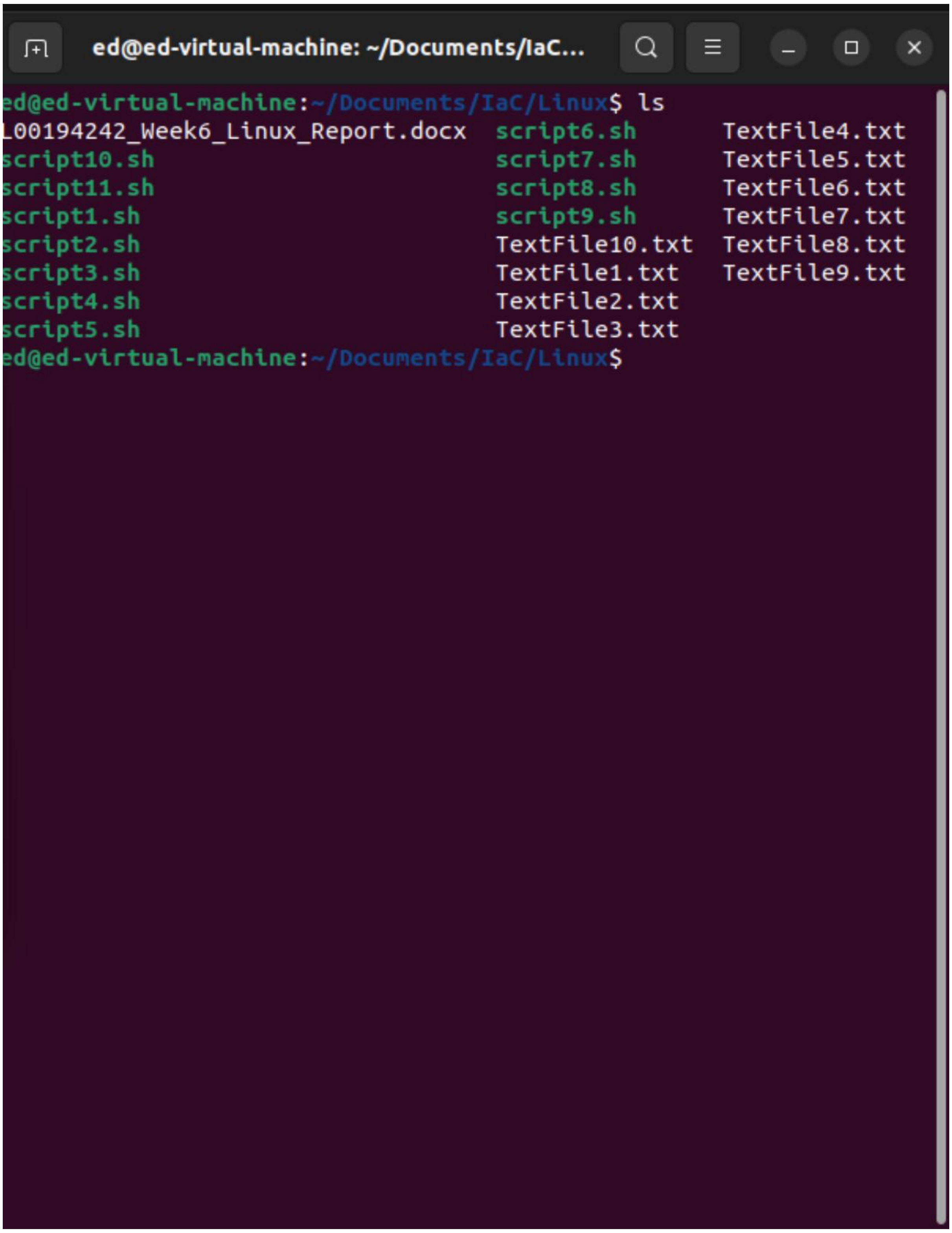
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script9.sh
press [ctrl][c] to exit
test
Unknown: test
Hello
English
Ciao
Italian
Dia Duit
Irish
^C
ed@ed-virtual-machine:~/Documents/IaC/Linux$
```

Figure 11 - Result of script9.sh



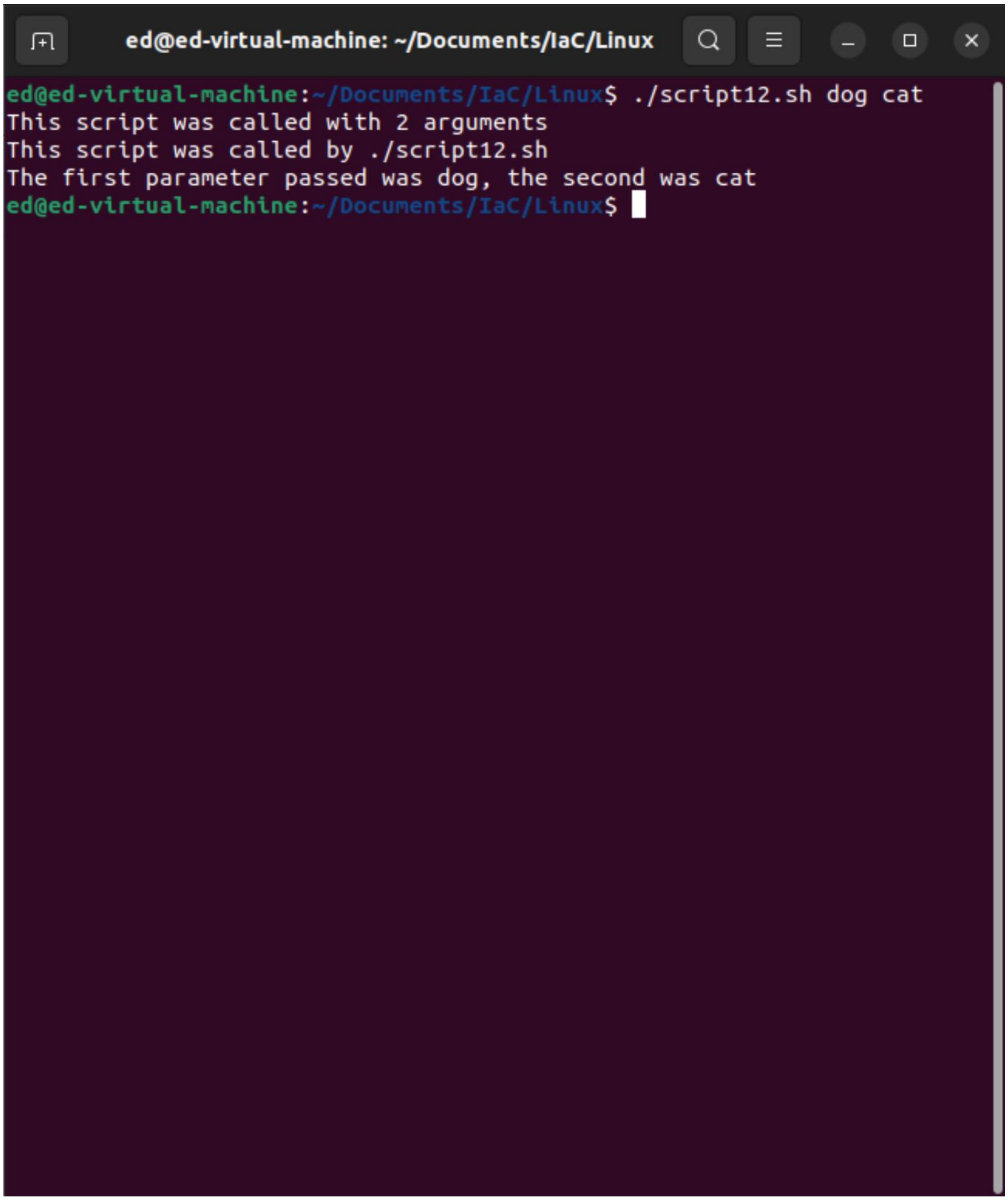
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script10.sh
Enter value for X: 5
X is greater than zero
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script10.sh
Enter value for X: -2
X is less than zero
ed@ed-virtual-machine:~/Documents/IaC/Linux$
```

Figure 12 - Result for Script10.sh



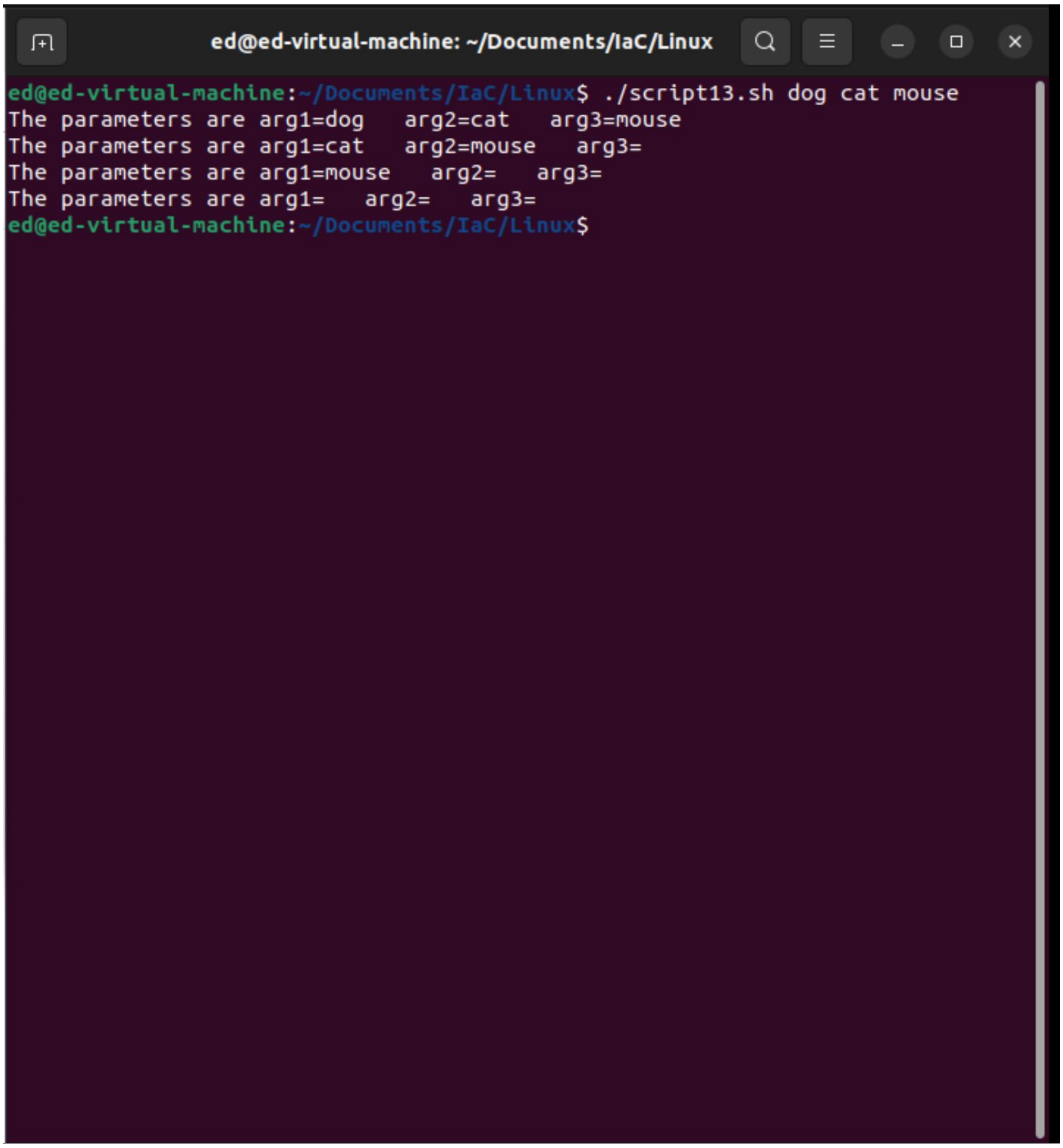
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ls
L00194242_Week6_Linux_Report.docx  script6.sh      TextFile4.txt
script10.sh                        script7.sh      TextFile5.txt
script11.sh                        script8.sh      TextFile6.txt
script1.sh                         script9.sh      TextFile7.txt
script2.sh                         TextFile10.txt  TextFile8.txt
script3.sh                         TextFile1.txt   TextFile9.txt
script4.sh                         TextFile2.txt
script5.sh                         TextFile3.txt
ed@ed-virtual-machine:~/Documents/IaC/Linux$
```

Figure 13- Result for Script11.sh



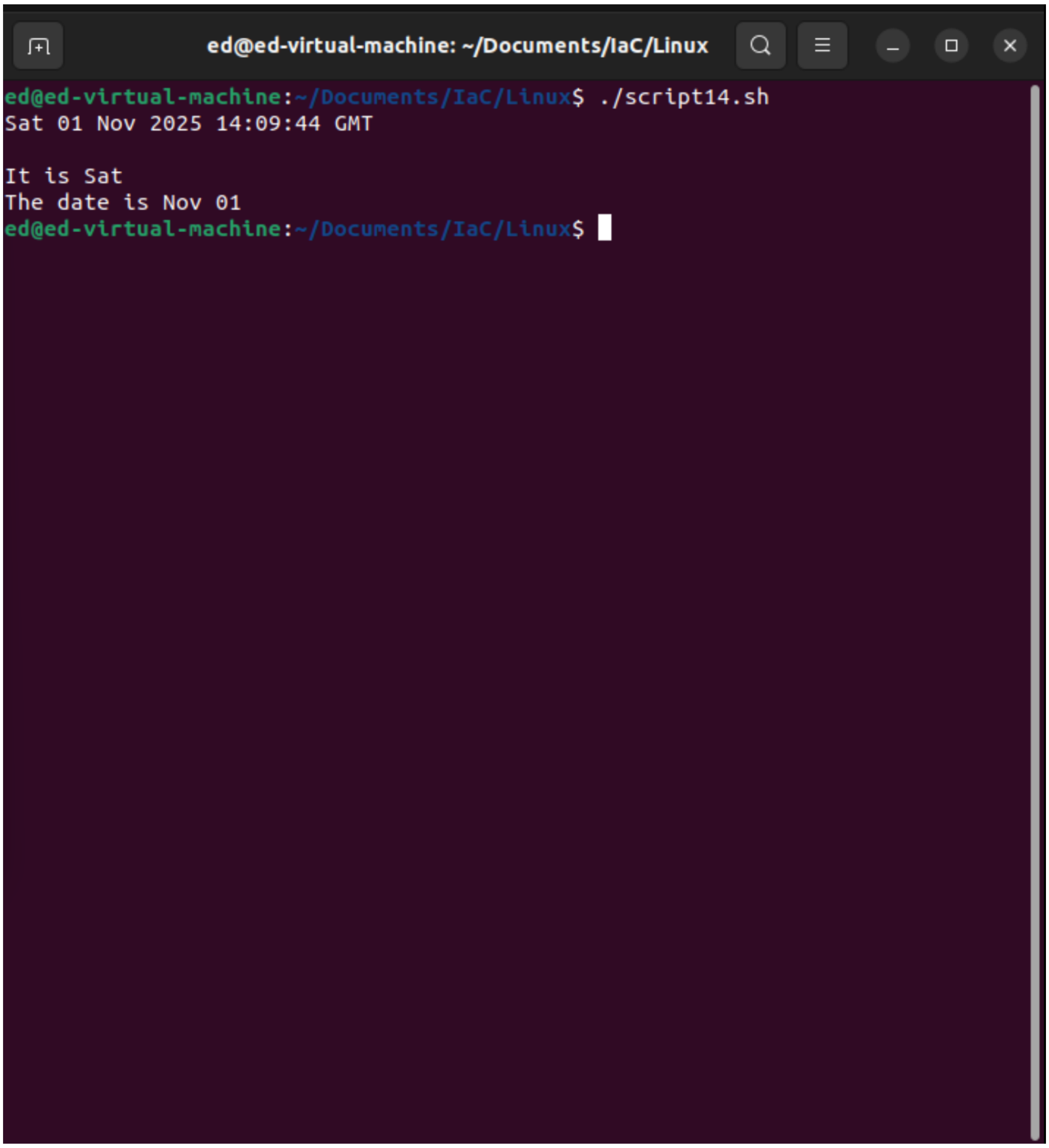
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script12.sh dog cat
This script was called with 2 arguments
This script was called by ./script12.sh
The first parameter passed was dog, the second was cat
ed@ed-virtual-machine:~/Documents/IaC/Linux$ 
```

Figure 14 - Result of Script12.sh



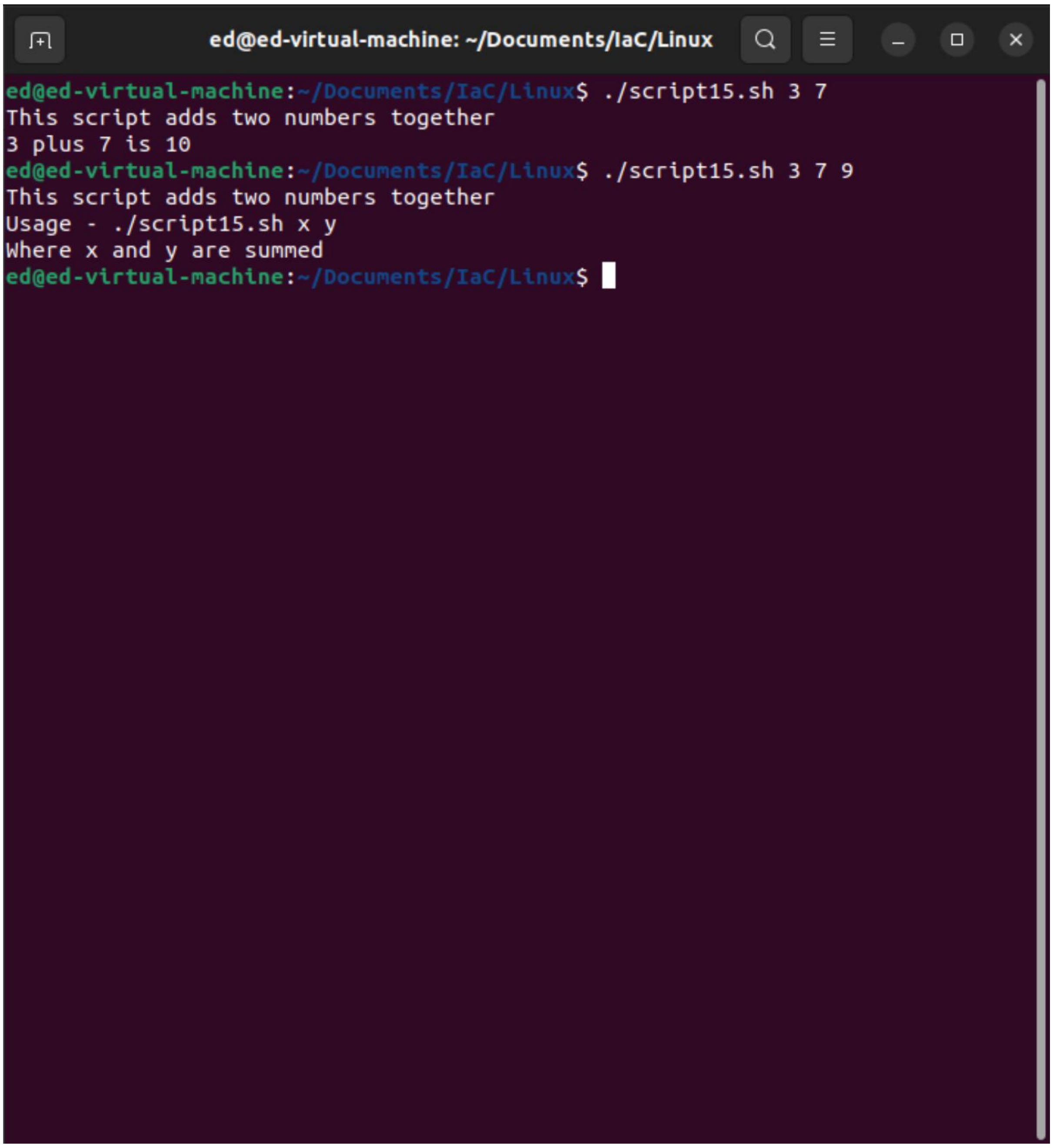
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script13.sh dog cat mouse
The parameters are arg1=dog   arg2=cat   arg3=mouse
The parameters are arg1=cat   arg2=mouse   arg3=
The parameters are arg1=mouse   arg2=   arg3=
The parameters are arg1=   arg2=   arg3=
ed@ed-virtual-machine:~/Documents/IaC/Linux$
```

Figure 15- Result of script13.sh



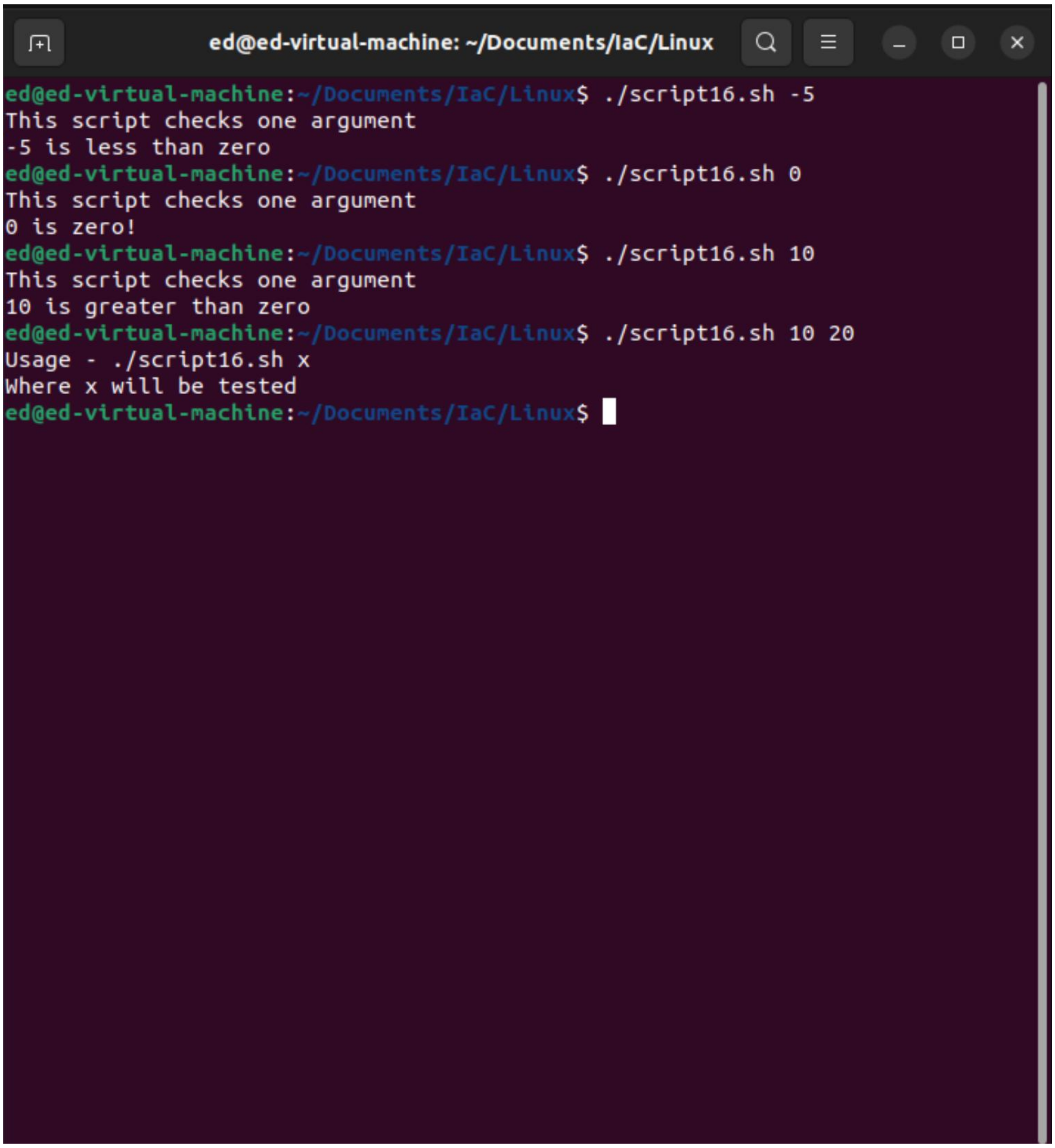
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script14.sh
Sat 01 Nov 2025 14:09:44 GMT

It is Sat
The date is Nov 01
ed@ed-virtual-machine:~/Documents/IaC/Linux$
```

Figure 16 - Result of script14.sh



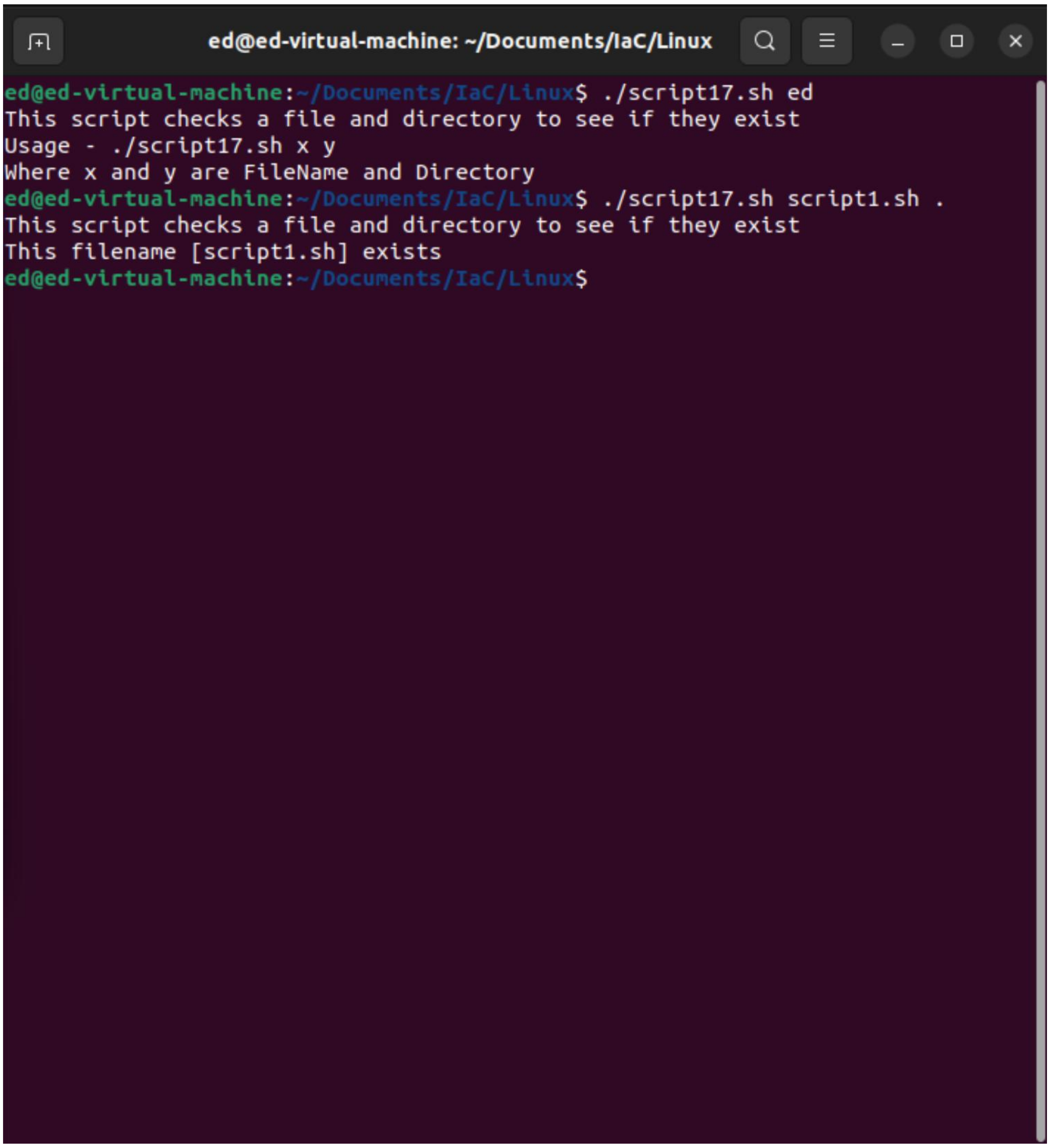
```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script15.sh 3 7
This script adds two numbers together
3 plus 7 is 10
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script15.sh 3 7 9
This script adds two numbers together
Usage - ./script15.sh x y
Where x and y are summed
ed@ed-virtual-machine:~/Documents/IaC/Linux$
```

Figure 17 -Result of script15.sh



```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script16.sh -5
This script checks one argument
-5 is less than zero
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script16.sh 0
This script checks one argument
0 is zero!
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script16.sh 10
This script checks one argument
10 is greater than zero
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script16.sh 10 20
Usage - ./script16.sh x
Where x will be tested
ed@ed-virtual-machine:~/Documents/IaC/Linux$ 
```

Figure 18 - Results of Script16.sh



```
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script17.sh ed
This script checks a file and directory to see if they exist
Usage - ./script17.sh x y
Where x and y are FileName and Directory
ed@ed-virtual-machine:~/Documents/IaC/Linux$ ./script17.sh script1.sh .
This script checks a file and directory to see if they exist
This filename [script1.sh] exists
ed@ed-virtual-machine:~/Documents/IaC/Linux$
```

Figure 19 - script17 result