Lab *03 – Scripting in Bash*

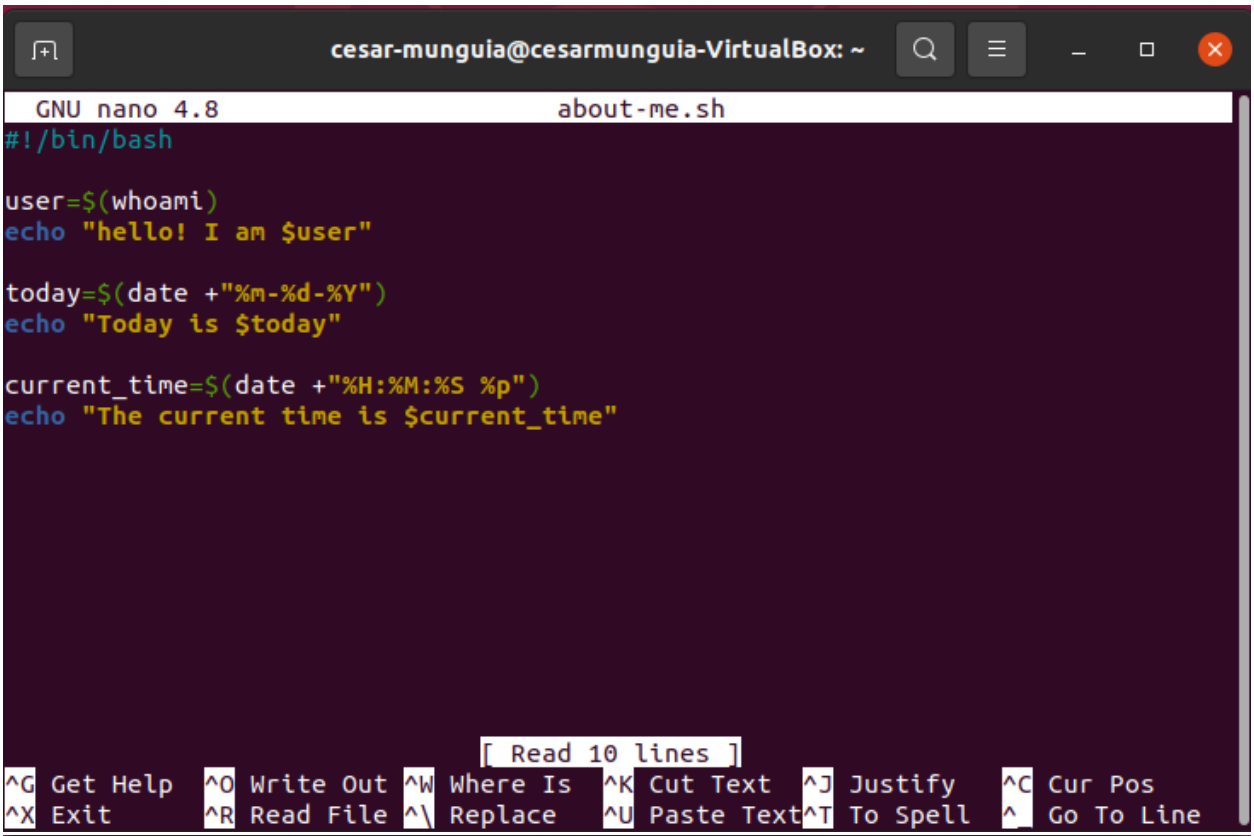
Author: Cesar Munguia
Course Section: IS-1003-002
Date: 04/07/2022

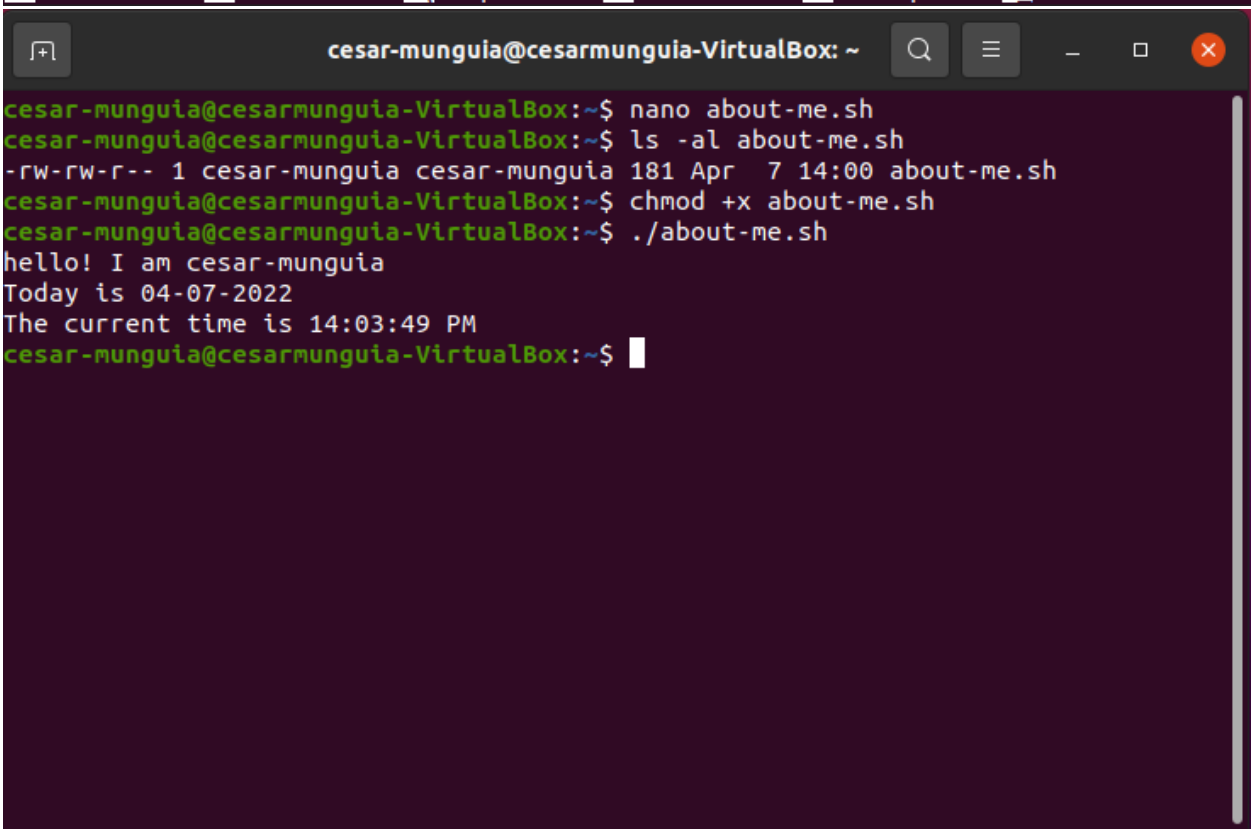
INTRODUCTION

The objectives are to use a text editor, like nano, in Linux, create some scripts in a Bash shell, and differentiate between Bash keywords/functions and my programmer-defined variables/values. This will be the first time I'll be programming inside a virtual machine and also it will be my time programming in the Bash language. I'm also kind of familiar with C and Java.

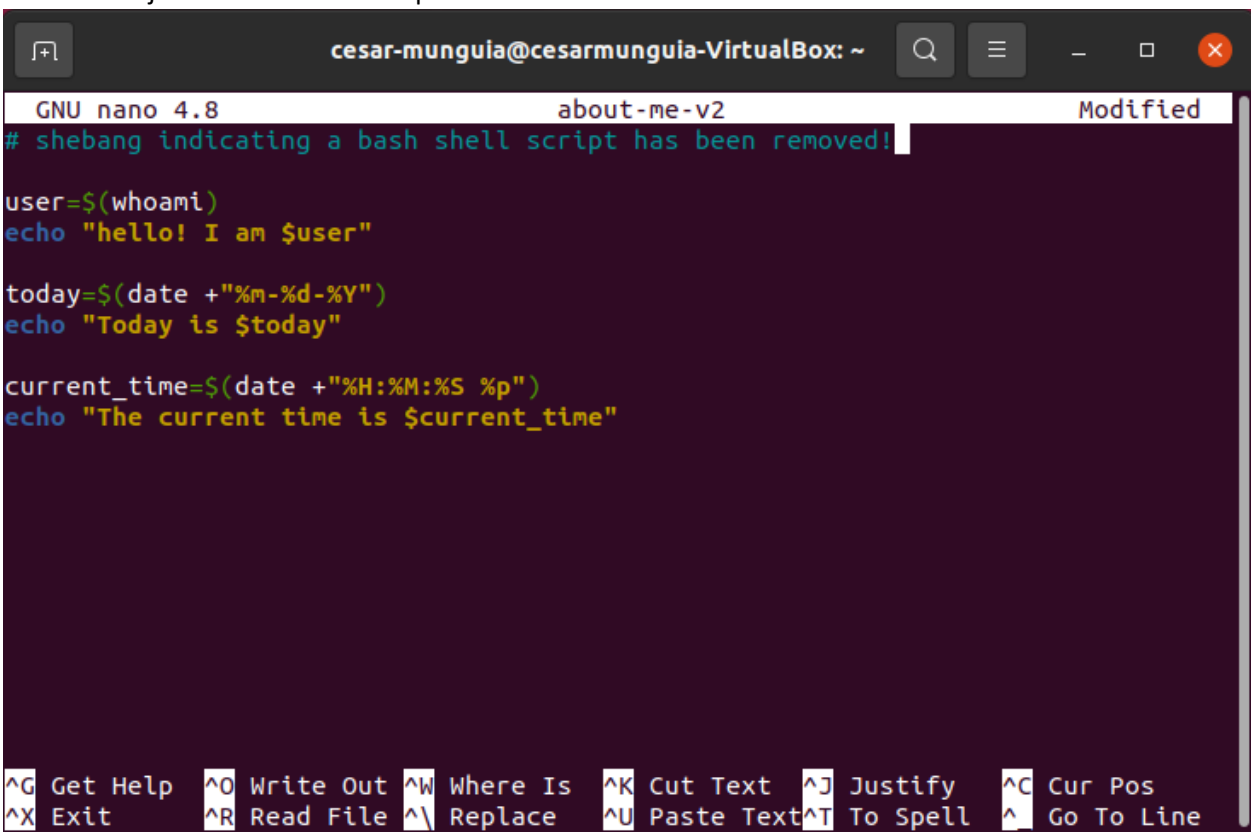
PROCESS

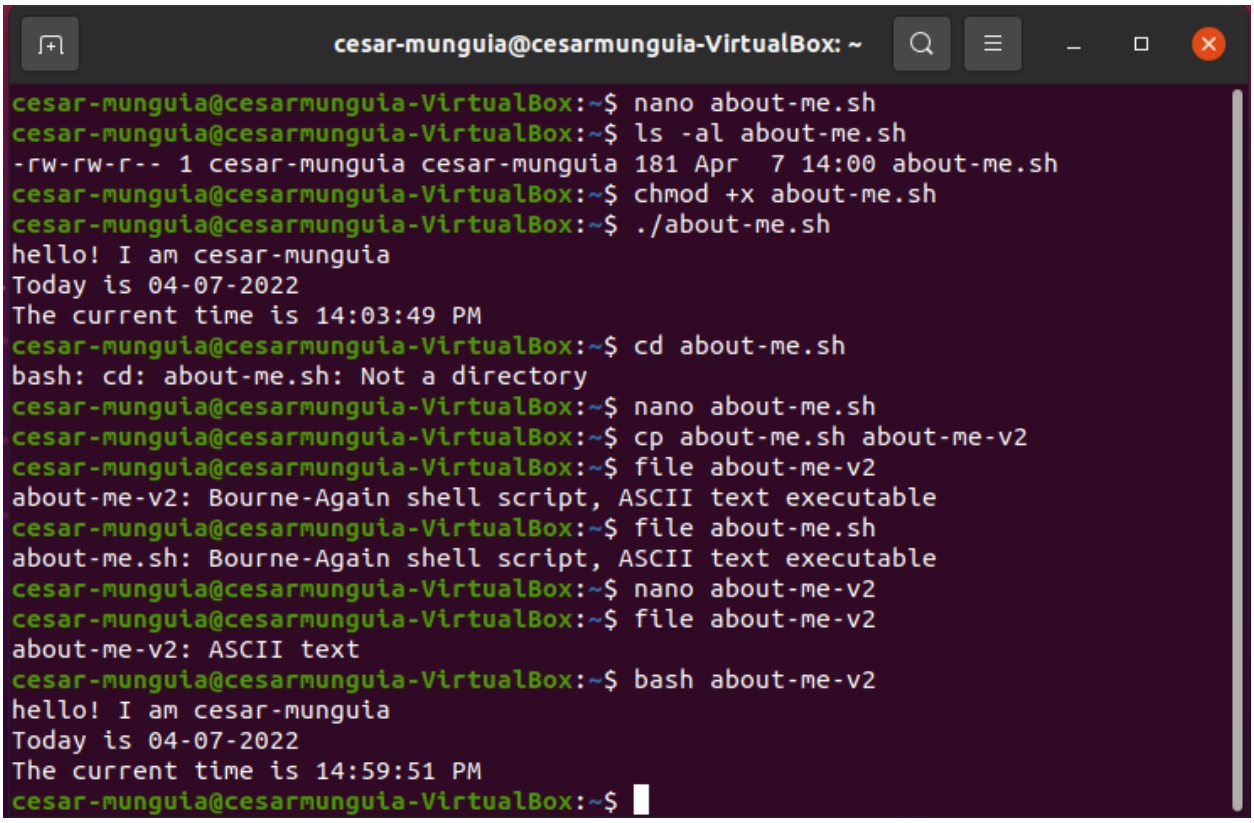
1. In phase 0 I was asked to open my terminal on my Linux machine and create a nano file with a specified filename named "about-me.sh". This filename could be changed inside the nano interface. I was also given a table full of common keyboard shortcuts used in the nano editor. I created the file and was prompted in the nano editor interface.
2. In phase 1 I was asked to write some code. The code was given to me. The code, or could be better called the script, is a set of commands to execute specific tasks. In this case, user was assigned a specific command which is whoami. The next line of code displays a text to the user with the variable user as an input. The same thing can be said for the variable today and current_time. Below is a picture of the given code with my name and with the date and time it was executed.
   The dollar sign (%) is the command substitution. In this case, whatever it is inputted in the variable it will be transferred to the output.


cesar-munguia@cesarmunguia-VirtualBox: ~
GNU nano 4.8 about-me.sh
#!/bin/bash
user=$(whoami)
echo "hello! I am $user"
today=$(date +"%m-%d-%Y")
echo "Today is $today"
current_time=$(date +"%H:%M:%S %p")
echo "The current time is $current_time"
[Read 10 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line

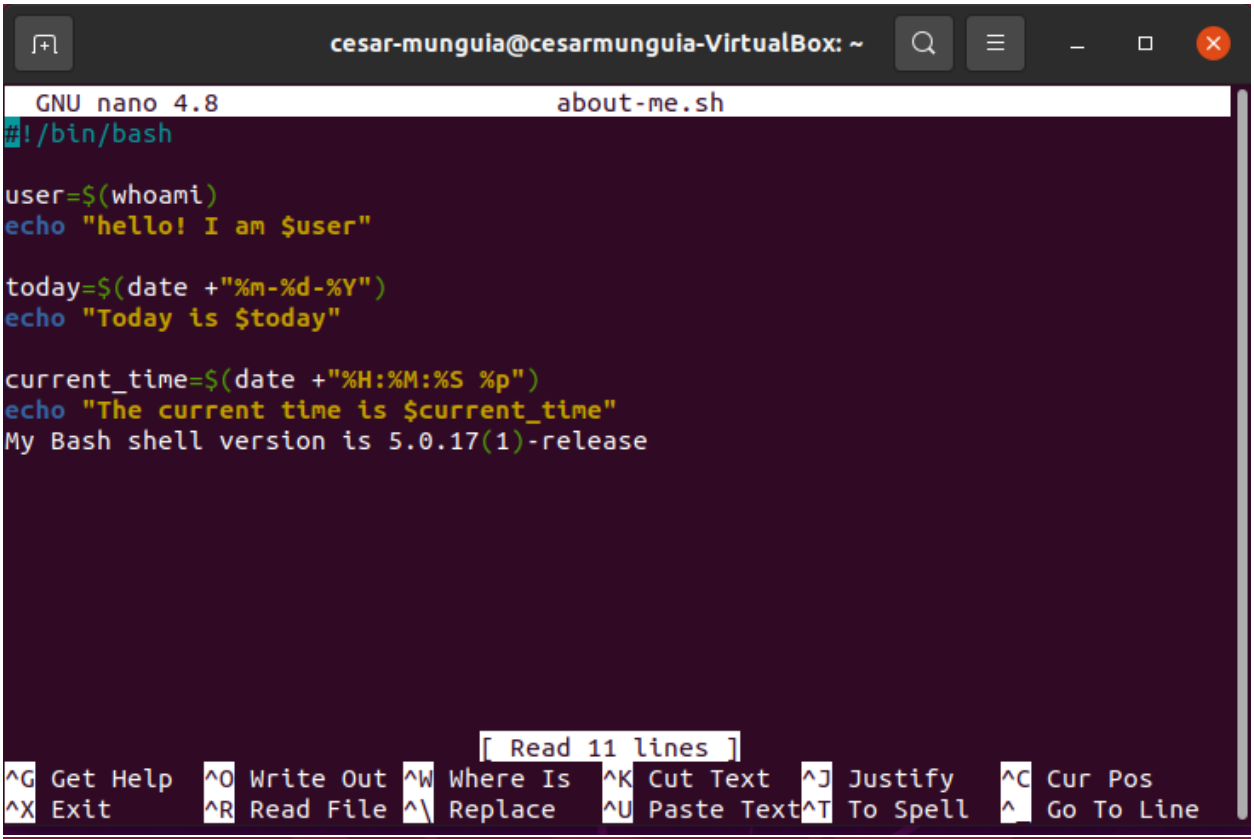

cesar-munguia@cesarmunguia-VirtualBox: ~
cesar-munguia@cesarmunguia-VirtualBox:~$ nano about-me.sh
cesar-munguia@cesarmunguia-VirtualBox:~$ ls -al about-me.sh
-rw-rw-r-- 1 cesar-munguia cesar-munguia 181 Apr 7 14:00 about-me.sh
cesar-munguia@cesarmunguia-VirtualBox:~$ chmod +x about-me.sh
cesar-munguia@cesarmunguia-VirtualBox:~$ ./about-me.sh
hello! I am cesar-munguia
Today is 04-07-2022
The current time is 14:03:49 PM
cesar-munguia@cesarmunguia-VirtualBox:~$

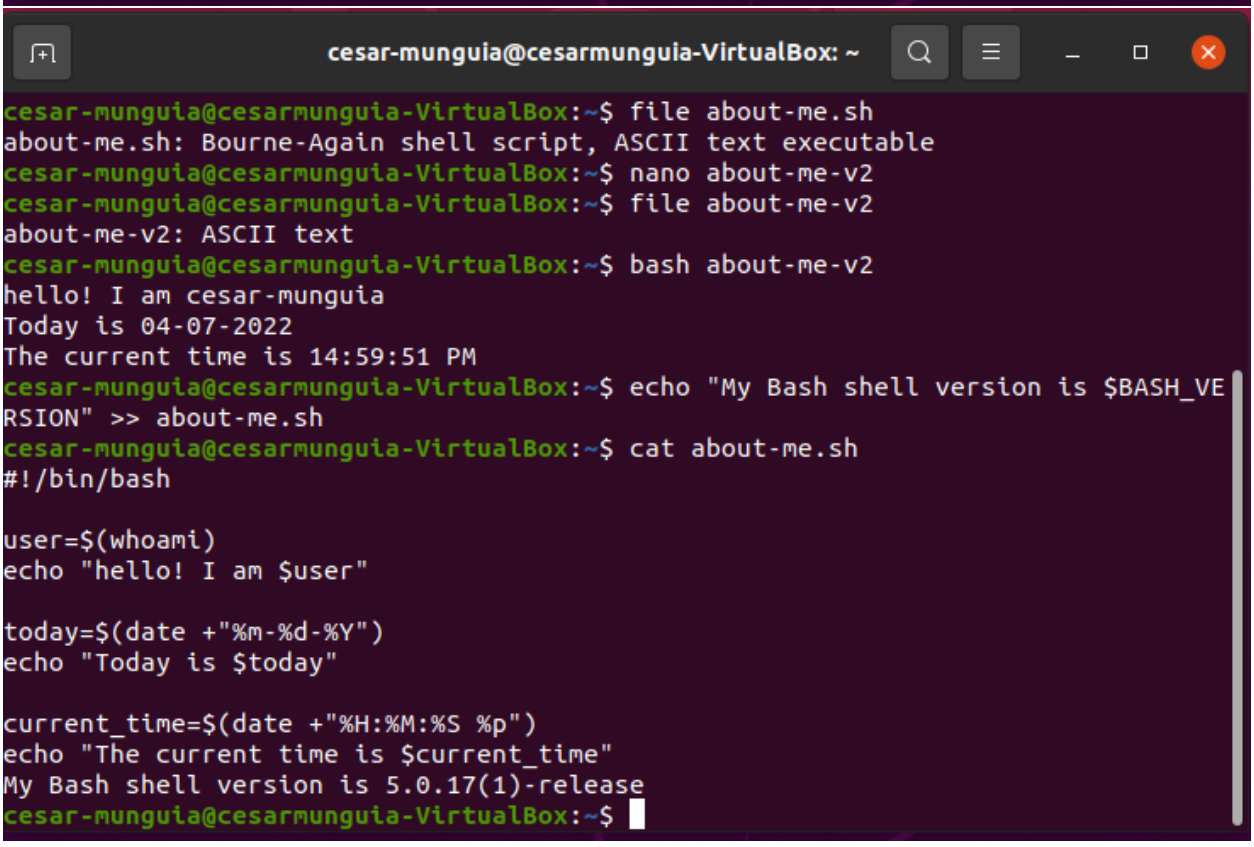
3. In phase 2 I was asked to copy the about-me.sh file I have previously created and paste it into another file named about-me-v2. The command I used for this execution was **cp about-me.sh about-me-v2**. The file was successfully copied. I then went ahead and checked for the file type with the command **file about-me-v2** for the new file I had just created. The file type displayed was a shell script. Then, I went into the nano editor interface of the new file about-me-v2 and erased the shebang. Saved it and exited. I checked for the file type again and this time it showed that it was a simple ASCII text. Finally, I executed the file using the command **bash about-me-v2** and the file was successfully executed.
   The term *executable* means a file that contains specific commands or code meant for performing specific tasks. A new file needs to be created aside from the original one. I know this because back in my programming class you would always have a file where the code is, and another file just to execute or compile the code.





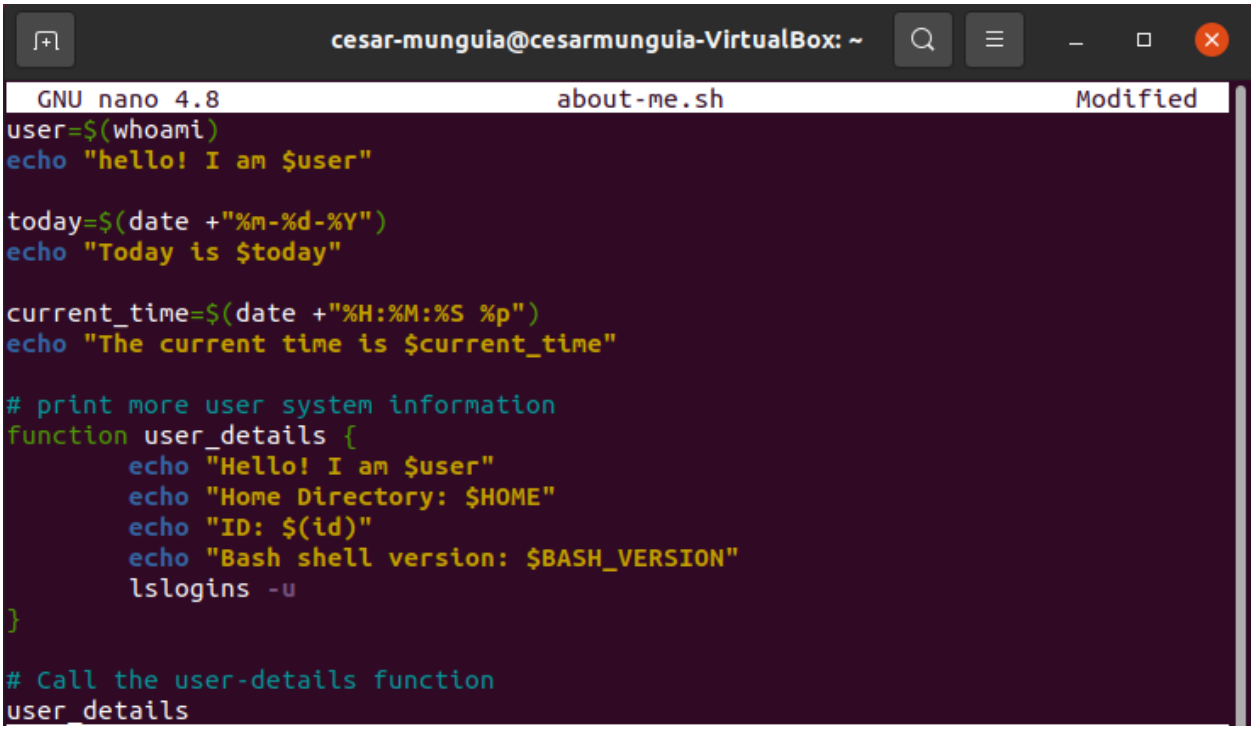
4. In phase 3 I was asked to append to a bash shell script. Appending means simply to add to the current file without having to enter the interface and add it yourself. I did this by using the command **echo "My Bash shell version is $BASH_VERSION" >> about-me.sh**. I clarified my work (or the appending) by using the command **cat about-me.sh**. Below is a picture that explains what I am saying.
   What is actually on the script for the Bash shell version is that the "My Bash shell version is $BASH_VERSION" was added at the end of my code. In addition, for some reason whenever I tried to execute the file by using command **bash about-me.sh** it would tell me there was a syntax error. I believe this is because of the variable, the command $BASH_VERSION was not declared first so for that reason it was showing me the syntax error.

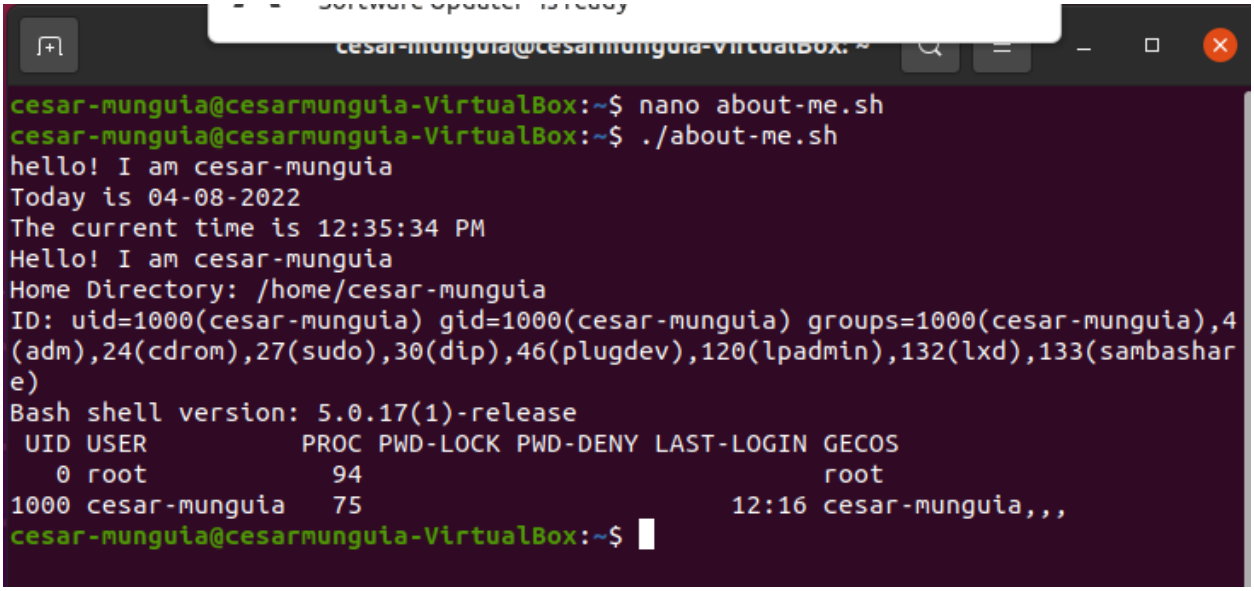

cesar-munguia@cesarmunguia-VirtualBox: ~
GNU nano 4.8 about-me.sh
#!/bin/bash
user=$(whoami)
echo "hello! I am $user"
today=$(date +"%m-%d-%Y")
echo "Today is $today"
current_time=$(date +"%H:%M:%S %p")
echo "The current time is $current_time"
My Bash shell version is 5.0.17(1)-release
[Read 11 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line


cesar-munguia@cesarmunguia-VirtualBox: ~
cesar-munguia@cesarmunguia-VirtualBox:~$ file about-me.sh
about-me.sh: Bourne-Again shell script, ASCII text executable
cesar-munguia@cesarmunguia-VirtualBox:~$ nano about-me-v2
cesar-munguia@cesarmunguia-VirtualBox:~$ file about-me-v2
about-me-v2: ASCII text
cesar-munguia@cesarmunguia-VirtualBox:~$ bash about-me-v2
hello! I am cesar-munguia
Today is 04-07-2022
The current time is 14:59:51 PM
cesar-munguia@cesarmunguia-VirtualBox:~$ echo "My Bash shell version is $BASH_VERSION" >> about-me.sh
cesar-munguia@cesarmunguia-VirtualBox:~$ cat about-me.sh
#!/bin/bash
user=$(whoami)
echo "hello! I am $user"
today=$(date +"%m-%d-%Y")
echo "Today is $today"
current_time=$(date +"%H:%M:%S %p")
echo "The current time is $current_time"
My Bash shell version is 5.0.17(1)-release
cesar-munguia@cesarmunguia-VirtualBox:~$

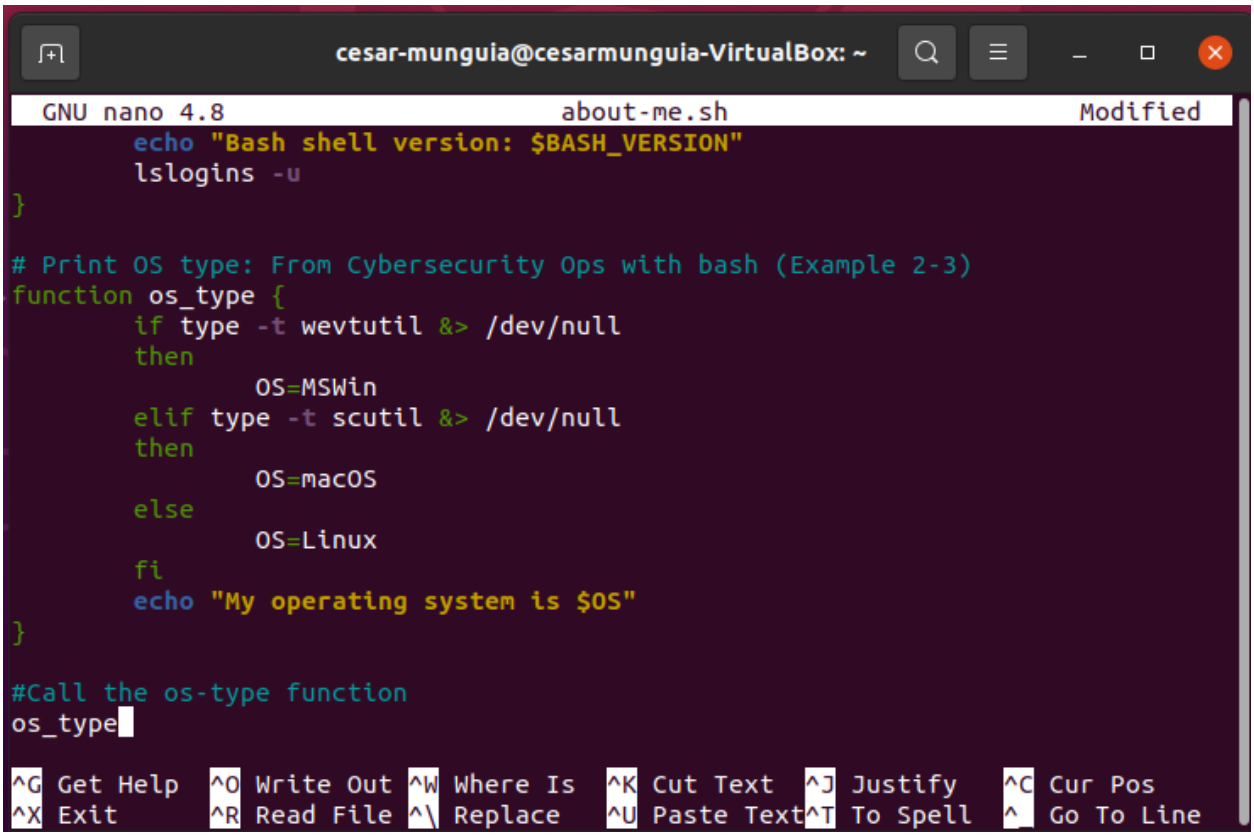
```
cesar-munguia@cesarmunguia-VirtualBox:~$ bash about-me.sh
hello! I am cesar-munguia
Today is 04-07-2022
The current time is 15:51:07 PM
about-me.sh: line 13: syntax error near unexpected token `('
about-me.sh: line 13: `My Bash shell version is 5.0.17(1)-release'
cesar-munguia@cesarmunguia-VirtualBox:~$
```

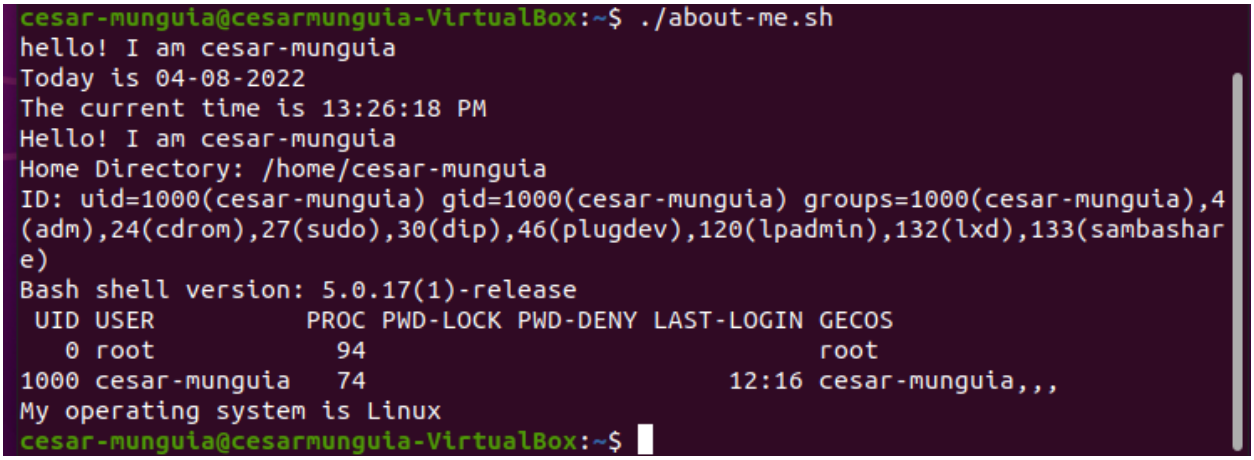
5. In phase 4 I was asked to declare a function and make a function call as well. This function is called **user_details** and is basically a function with several commands that print out more text for the user. The first command prints out my username, followed by the home directory, followed by my ID, and finally my Bash shell version. The order of precedence I think is important here, so for that reason the function declaration is coded first and then the function call. You call the function by simply putting the name of the function.
   My last login was at 12:16 at the time I'm writing this document. My UID is 1000(cesar-munguia) as well as my GID too. Both of them are shown below in the second screenshot.





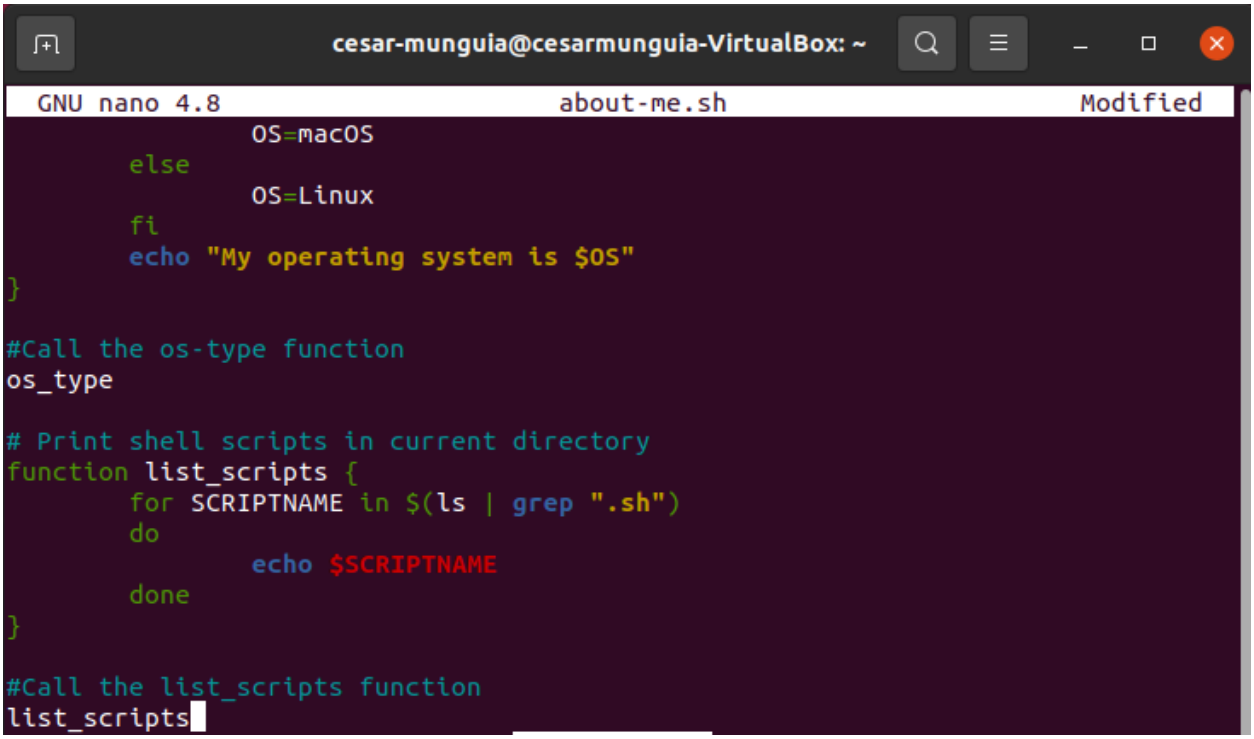
6. Again, in phase 4 I was asked to code a function body that works with conditional statements. A conditional function is a function whose purpose is to check for validation per say. It checks whether the first conditional statement is true, if it is, then it updates the variable OS with a new definition. It then exits the conditional statements (not the function body) and executes the print out.
   If I were to only include the function body and not the function call then the function would never execute. It would not display any syntax error or any type of error it would just not appear on the output. You have to explicitly call the function so that it executes, if not it would just simply be a code block hanging in there with no purpose in the program.

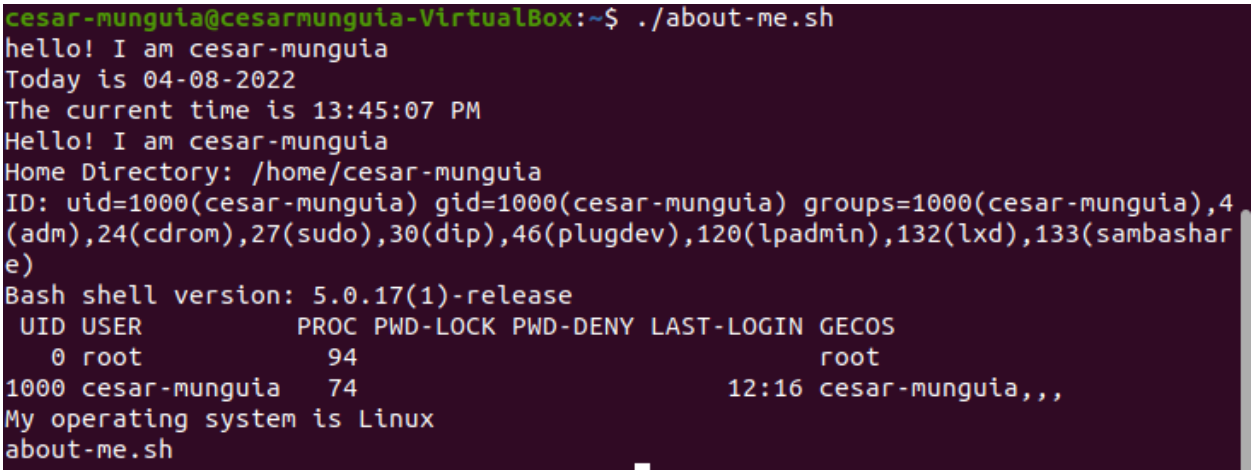




7. In phase 6 I was asked to code a function whose purpose was to print out a list of any file ending with the **.sh** extension. This is done by using a loop. First, we start by declaring the function which I called it **list_scripts**. The next line of code what it does is that it checks for every file in the current directory for a .sh extension. In my case only one was found because that's the only I've created so far. The function call is at the end so that the function executes.

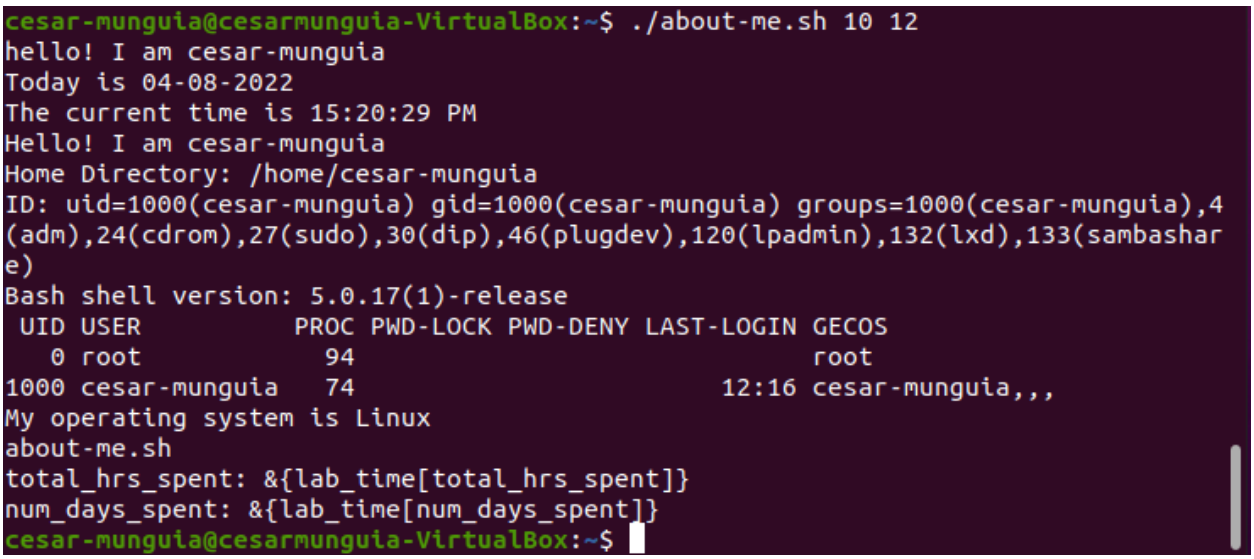
The function of the variable SCRIPTNAME in the code is that it checks for scripts, but not just any scrips, scripts that end with the .sh extension. It basically searches for names, and then based on the function if it comes out to be true then it prints it out. In this case only one was printed out.





8. In this last phase I was asked to create another function which worked with associative arrays and I also worked with positional parameters. In simple words, an associative array is like a list of data elements that can be accessed by specifying a name, or also known as a key. The first block of code is to gather input from the user so that it can be then stored in the associative array and later be printed out. The second block of code is for the actual printing, for this we had to use a for loop to display what the user has inputted first. In this case it was 10 and 12. For some reason I had some problem getting the right output. I wasn't getting what I inputted on the command line. I don't know why this was happening since everything was coded right. At the end we had to call the function **lab_log** to execute it.

The purpose of the at sign (@) is to access full list of positional parameters, in this case **lab_time**.



LIMITATIONS/CONCLUSION

I had one limitation during my work in this lab and it happened at the last phase. For some reason I couldn't get the right output for the positional parameters. I doubled checked everything and the only thing being outputted was "&{lab_time[total_hrs_spent]}". I am unaware of the reason why this was happening. Aside from that I only encountered a couple of syntax errors but was able to debug them easily. The goals of the techniques used in this lab were to be more familiar with the language Bash and with working on the terminal in general. I find this very important for me because this is where all IT workers work during their regular time. Might as well get used to them right now. I thought these

techniques and tools were very effective because it shows the tip of the iceberg of information systems. It shows us the basic concepts that every individual majoring in this should know.

REFERENCES

freeCodeCamp.org. (2021, April 28). *The ultimate linux command line guide - full bash tutorial*. freeCodeCamp.org. Retrieved April 8, 2022, from https://www.freecodecamp.org/news/linux-command-line-bash-tutorial/

COLLABORATION

No collaboration.