Lab 04 – Scripting in Bash

Raymond Ng: JQG999
IS 1003 Spring 2021
April 15, 2021

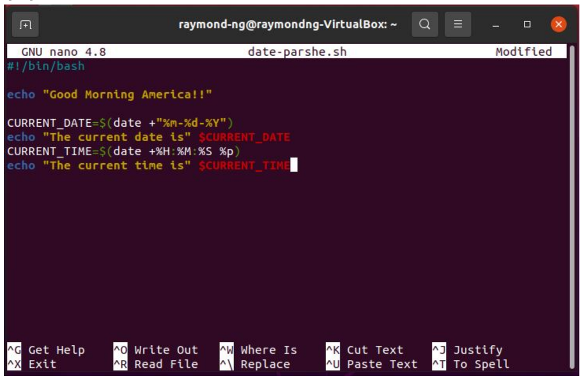
Introduction

In this lab I attempted to meet the following the objectives:

- Used a text editor, mainly Nano, in Linux
- Created basic scripts in a Bash shell.
- Differentiate between Bash keywords/functions and in my programmer-defined variables/values

Process

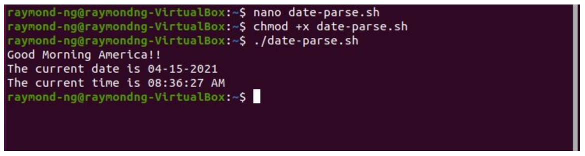
(1) Nano Editor Interface



```
raymond-ng@raymondng-VirtualBox: ~
  GNU nano 4.8                  date-parshe.sh                    Modified
#!/bin/bash

echo "Good Morning America!!"

CURRENT_DATE=$(date +"%m-%d-%Y")
echo "The current date is" $CURRENT_DATE
CURRENT_TIME=$(date +%H:%M:%S %p)
echo "The current time is" $CURRENT_TIME







^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell
```

Per the initial block of instructions I used the command `nano date-parse.sh` at the command line to open up the nano editor interface. In my script, I replaced the "*Rise and Shine! Carpe Diem!!*" comment with "*Good Morning America!*" Additionally, I changed the date format to month-day-year rather than the day-month-year format from the example. I left the time format the same, hour-minute-second.

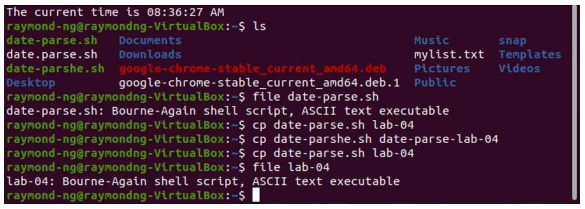
(2) Executing Script



```
raymond-ng@raymondng-VirtualBox:~$ nano date-parse.sh
raymond-ng@raymondng-VirtualBox:~$ chmod +x date-parse.sh
raymond-ng@raymondng-VirtualBox:~$ ./date-parse.sh
Good Morning America!!
The current date is 04-15-2021
The current time is 08:36:27 AM
raymond-ng@raymondng-VirtualBox:~$ 
```

In order to run the script, I had to use "change mode" (`chmod`) command and also add "execute" (`x`) permissions on the file by using `chmod +x date-parse.sh`. The script was executed by typing `./date-parse.sh`.

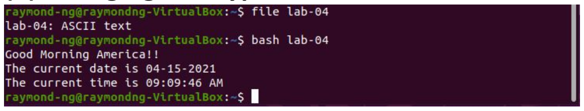
(3) Using `file` Command



```
The current time is 08:36:27 AM
raymond-ng@raymondng-VirtualBox:~$ ls
date-parse.sh   Documents                                Music       snap
date.parse.sh   Downloads                                mylist.txt  Templates
date-parshe.sh  google-chrome-stable_current_amd64.deb   Pictures    Videos
Desktop         google-chrome-stable_current_amd64.deb.1 Public
raymond-ng@raymondng-VirtualBox:~$ file date-parse.sh
date-parse.sh: Bourne-Again shell script, ASCII text executable
raymond-ng@raymondng-VirtualBox:~$ cp date-parse.sh lab-04
raymond-ng@raymondng-VirtualBox:~$ cp date-parshe.sh date-parse-lab-04
raymond-ng@raymondng-VirtualBox:~$ cp date-parse.sh lab-04
raymond-ng@raymondng-VirtualBox:~$ file lab-04
lab-04: Bourne-Again shell script, ASCII text executable
raymond-ng@raymondng-VirtualBox:~$ 
```

Using the file command, I checked the file type by typing in file *date-parse.sh* in the command terminal. Then I copied the contents of my "*date-parse*" to a new file I labeled as "*Lab-04*" using the copy (`cp`) command; typed in `cp date-parse.sh lab-04` in the terminal. In this screen shot you'll notice I was debating on what to name the new file. I was juggling between "*date-parse-lab-04*" and "*lab-04*". I thought that "*date-parse-lab-04*" was too long of file name so I went with "*lab-04*" as the new file name.

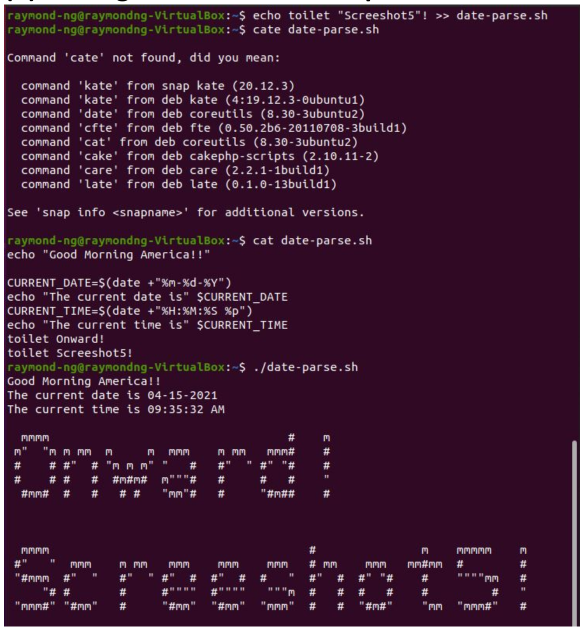
(4) Changing File Types



```
raymond-ng@raymondng-VirtualBox:~$ file lab-04
lab-04: ASCII text
raymond-ng@raymondng-VirtualBox:~$ bash lab-04
Good Morning America!!
The current date is 04-15-2021
The current time is 09:09:46 AM
raymond-ng@raymondng-VirtualBox:~$
```

I modified my newly crated "*Lab-04*" file by eliminating the shebang (`!/bin/bash`) and executed `file lab-04` to check the type. It was no longer identified as a script and merely a text file. In order to run the script again I used `bash lab-04` to call on the bash interpreter. The script executed without the shebang opener, by explicitly calling the bash interpreter.

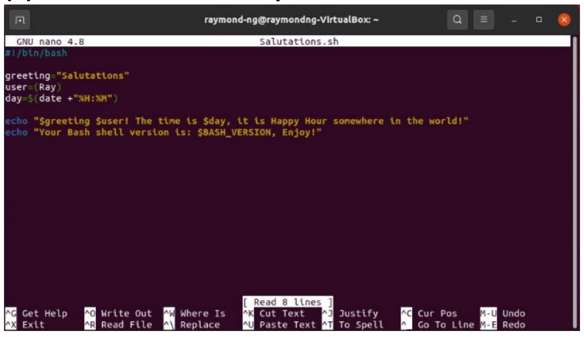
(5) Adding to a Bash Shell Script



```
raymond-ng@raymondng-VirtualBox:~$ echo toilet "Screeshot5"! >> date-parse.sh
raymond-ng@raymondng-VirtualBox:~$ cate date-parse.sh

Command 'cate' not found, did you mean:

  command 'kate' from snap kate (20.12.3)
  command 'kate' from deb kate (4:19.12.3-0ubuntu1)
  command 'date' from deb coreutils (8.30-3ubuntu2)
  command 'cfte' from deb fte (0.50.2b6-20110708-3build1)
  command 'cat' from deb coreutils (8.30-3ubuntu2)
  command 'cake' from deb cakephp-scripts (2.10.11-2)
  command 'care' from deb care (2.2.1-1build1)
  command 'late' from deb late (0.1.0-13build1)

See 'snap info <snapname>' for additional versions.

raymond-ng@raymondng-VirtualBox:~$ cat date-parse.sh
echo "Good Morning America!!"

CURRENT_DATE=$(date +"%m-%d-%Y")
echo "The current date is" $CURRENT_DATE
CURRENT_TIME=$(date +"%H:%M:%S %p")
echo "The current time is" $CURRENT_TIME
toilet Onward!
toilet Screeshot5!
raymond-ng@raymondng-VirtualBox:~$ ./date-parse.sh
Good Morning America!!
The current date is 04-15-2021
The current time is 09:35:32 AM

  mmmm                                 #    m
 m"  "m m mm  m     m  mmm    m mm   mmm#    #
 #    # #"  # "m m m" "   #   #"  " #" "#    #
 #    # #   #  #m#m#  m"""#   #     #   #    "
  #mm#  #   #   # #   "mm"#   #     "#m##    #



  mmmm                              #               m    mmmmm    m
 #"   "  mmm    m mm   mmm    mmm    mmm  # mm    mmm   mm#mm  #        #
 "#mmm  #"  "   #"  " #"  #  #"  #  #   " #"  #  #" "#    #    """"mm   #
     "# #       #     #""""  #""""   """m #   #  #   #    #         #   "
 "mmm#" "#mm"   #     "#mm"  "#mm"  "mmm" #   #  "#m#"    "mm  "mmm#"   #
```

Here, I followed the instructions to append my existing shell script with a redirection (`>>`) command. To append the ending message to my *date-parse.sh* file I used the Toilet tool. I knew I had to install it first because I have never used this tool before. Typed in `echo toilet "Onward"! >> date-parse.sh` to append the file. Next, I used the concatenate (`cat`) command to read the file, writing it to standard output; typed in `cat date-parse.sh` (you will notice I accidentally entered "cate" in stead of "cat" by accident in the initial execution of this command). To execute the script, I typed in `./date-parse.sh`. I wanted to add a personalized touch to this step so I went over the sequence of steps for this screenshot again and added "*Screenshot5*" (I know I misspelled "Screenshot" in this step).

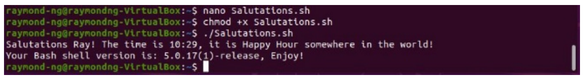
(6) Variables in Bash Script via Nano



```
raymond-ng@raymondng-VirtualBox: ~
  GNU nano 4.8                         Salutations.sh
#!/bin/bash

greeting="Salutations"
user=(Ray)
day=$(date +"%H:%M")

echo "$greeting $user! The time is $day, it is Happy Hour somewhere in the world!"
echo "Your Bash shell version is: $BASH_VERSION, Enjoy!"

                                 [ Read 8 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^  Go To Line  M-E Redo
```

Here I created a new script and labeled it *Salutations.sh*. First, I had to declare a variable greeting and assign a string value "*Salutations*" to it. The next variable "*user*" contained a value of a user name running the shell session implemented via command substitution. Here, instead of *whoami*, I went with my name *Ray*. Lastly, instead of using the actual day of the week for the variable day, I chose to go with the current time.

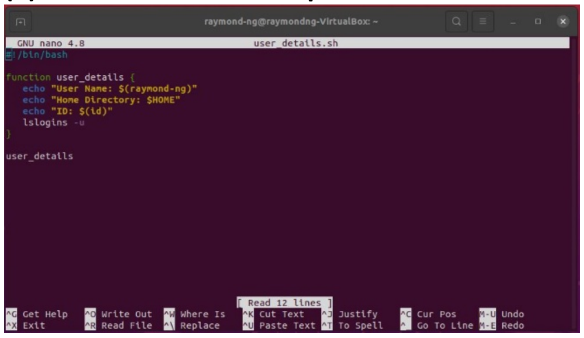
(7) Executing Variable in Bash Script



```
raymond-ng@raymondng-VirtualBox:~$ nano Salutations.sh
raymond-ng@raymondng-VirtualBox:~$ chmod +x Salutations.sh
raymond-ng@raymondng-VirtualBox:~$ ./Salutations.sh
Salutations Ray! The time is 10:29, it is Happy Hour somewhere in the world!
Your Bash shell version is: 5.0.17(1)-release, Enjoy!
raymond-ng@raymondng-VirtualBox:~$
```

To run the script, I had to use "change mode" (`chmod`) command and also add "execute" (`x`) permissions on the file by using `chmod +x Salutations.sh`. The script was executed successfully by typing `./Salutations.sh` command.

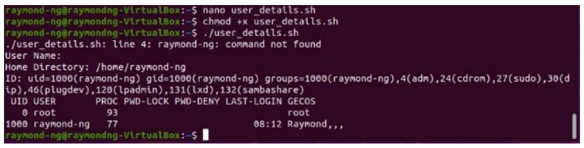
(8) Functions in Bash Script via Nano



```
  GNU nano 4.8                        user_details.sh
#!/bin/bash

function user_details {
   echo "User Name: $(raymond-ng)"
   echo "Home Directory: $HOME"
   echo "ID: $(id)"
   lslogins -u
}

user_details

                                 [ Read 12 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^_ Go To Line M-E Redo
```

In this script, I explored the use of functions. This specific function is *user_details.* I kept all variables and strings the same as the example except for the *User Name*. I used my user name "*raymond-ng*" instead of "*whoami*".

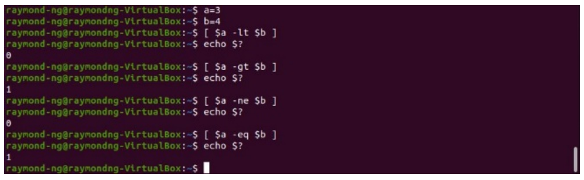
(9) Executing Functions in Bash Script



```
raymond-ng@raymondng-VirtualBox:~$ nano user_details.sh
raymond-ng@raymondng-VirtualBox:~$ chmod +x user_details.sh
raymond-ng@raymondng-VirtualBox:~$ ./user_details.sh
./user_details.sh: line 4: raymond-ng: command not found
User Name:
Home Directory: /home/raymond-ng
ID: uid=1000(raymond-ng) gid=1000(raymond-ng) groups=1000(raymond-ng),4(adm),24(cdrom),27(sudo),30(d
ip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
 UID USER       PROC PWD-LOCK PWD-DENY LAST-LOGIN GECOS
   0 root         93                              root
1000 raymond-ng   77                        08:12 Raymond,,,
raymond-ng@raymondng-VirtualBox:~$
```

Here, I executed the script and the output is depicted in the screenshot. Moreover, I learned that `lslogins` command display information about known users in the system (Kerrisk, 2021). The `id` command displays the user and group names and numeric IDs, of the calling process (Gite, 2012).

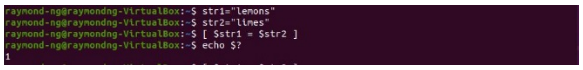
(10) Numeric and String Comparisons



```
raymond-ng@raymondng-VirtualBox:~$ a=3
raymond-ng@raymondng-VirtualBox:~$ b=4
raymond-ng@raymondng-VirtualBox:~$ [ $a -lt $b ]
raymond-ng@raymondng-VirtualBox:~$ echo $?
0
raymond-ng@raymondng-VirtualBox:~$ [ $a -gt $b ]
raymond-ng@raymondng-VirtualBox:~$ echo $?
1
raymond-ng@raymondng-VirtualBox:~$ [ $a -ne $b ]
raymond-ng@raymondng-VirtualBox:~$ echo $?
0
raymond-ng@raymondng-VirtualBox:~$ [ $a -eq $b ]
raymond-ng@raymondng-VirtualBox:~$ echo $?
1
raymond-ng@raymondng-VirtualBox:~$
```

In this step, I compared integer numbers as variables. I defined two variables `$a` and `$b`; `$a` `=` 3 and `$b` = 4. I used square brackers and numeric operators, specifically `-lt`, `-gt`, `-eq`, and `-le` to perform the actual evaluations. Using the `echo $?` command, I checked the return values of each executed evaluation respectively rending either a *true* or *false* output in *0*'s or *1*'s.

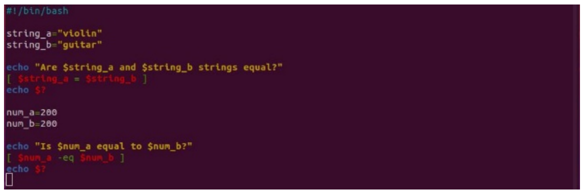
(11) String Comparisons



```
raymond-ng@raymondng-VirtualBox:~$ str1="lemons"
raymond-ng@raymondng-VirtualBox:~$ str2="limes"
raymond-ng@raymondng-VirtualBox:~$ [ $str1 = $str2 ]
raymond-ng@raymondng-VirtualBox:~$ echo $?
1
```

Here, I simply compared two distinct strings to see whether they are equal using the string comparison operator =. I assigned Str1 `=` "*lemons*" and Str2 `=` "*limes*". Using the `echo $?` command I checked the return values of the executed evaluation, which yielded a *1* indicating *False*.

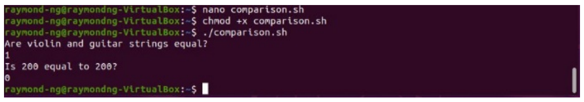
(12) Comparison in Bash Script via Nano



```
#!/bin/bash

string_a="violin"
string_b="guitar"

echo "Are $string_a and $string_b strings equal?"
[ $string_a = $string_b ]
echo $?

num_a=200
num_b=200

echo "Is $num_a equal to $num_b?"
[ $num_a -eq $num_b ]
echo $?
```

Per the instructions, I created a new script using `nano comparison.sh` command. In my script I will be comparing strings (words) and integer numbers as values and/or variables. Using the string comparison operator `=` I will be comparing *string_a* to *string_b* and $num_a to $num_b". Per the script, `echo $?` Command will evaluate each comparison respectively, which should yield a *0* or a *1*, indicating *True* or *False*.

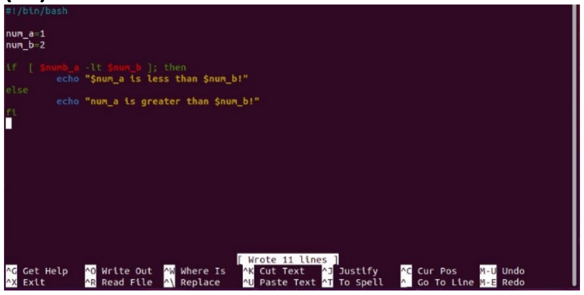
(13) Evaluating Comparisons



```
raymond-ng@raymondng-VirtualBox:~$ nano comparison.sh
raymond-ng@raymondng-VirtualBox:~$ chmod +x comparison.sh
raymond-ng@raymondng-VirtualBox:~$ ./comparison.sh
Are violin and guitar strings equal?
1
Is 200 equal to 200?
0
raymond-ng@raymondng-VirtualBox:~$
```

Executed the script by using the `chmod +x comparison.sh` to change permission and `./comparison.sh` to execute the script. The script ran successfully, yielded valid *True*/*False* results.

(14) Conditional Statements via Nano

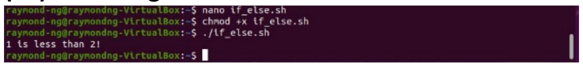


```
#!/bin/bash

num_a=1
num_b=2

if  [ $numb_a -lt $num_b ]; then
        echo "$num_a is less than $num_b!"
else
        echo "num_a is greater than $num_b!"
fi
```

```
                              [ Wrote 11 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^_ Go To Line  M-E Redo
```

This script created a two-way conditional using *if-else* statements. In this conditional statement, I am evaluating if the assigned value to *num_a* is less than the value assigned to *num_b*.

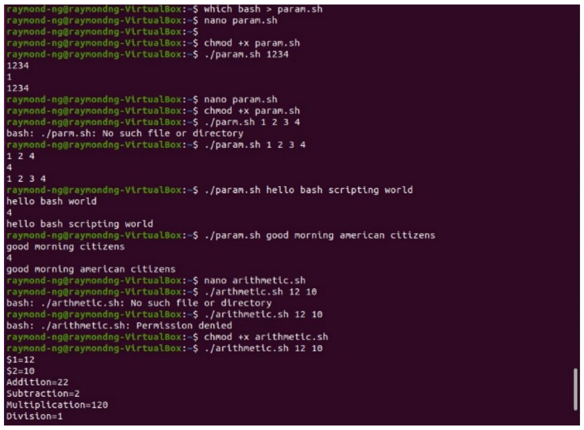
(15) Executing Conditional Statement



```
raymond-ng@raymondng-VirtualBox:~$ nano if_else.sh
raymond-ng@raymondng-VirtualBox:~$ chmod +x if_else.sh
raymond-ng@raymondng-VirtualBox:~$ ./if_else.sh
1 is less than 2!
raymond-ng@raymondng-VirtualBox:~$
```

The script executed successfully, output is correct, validating the conditional statement in the script.

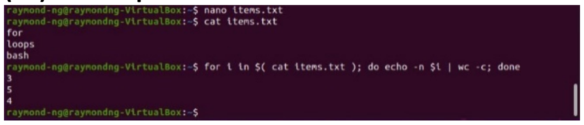
(16) Positional Parameters



```
raymond-ng@raymondng-VirtualBox:~$ which bash > param.sh
raymond-ng@raymondng-VirtualBox:~$ nano param.sh
raymond-ng@raymondng-VirtualBox:~$
raymond-ng@raymondng-VirtualBox:~$ chmod +x param.sh
raymond-ng@raymondng-VirtualBox:~$ ./param.sh 1234
1234
1
1234
raymond-ng@raymondng-VirtualBox:~$ nano param.sh
raymond-ng@raymondng-VirtualBox:~$ chmod +x param.sh
raymond-ng@raymondng-VirtualBox:~$ ./parm.sh 1 2 3 4
bash: ./parm.sh: No such file or directory
raymond-ng@raymondng-VirtualBox:~$ ./param.sh 1 2 3 4
1 2 4
4
1 2 3 4
raymond-ng@raymondng-VirtualBox:~$ ./param.sh hello bash scripting world
hello bash world
4
hello bash scripting world
raymond-ng@raymondng-VirtualBox:~$ ./param.sh good morning american citizens
good morning citizens
4
good morning american citizens
raymond-ng@raymondng-VirtualBox:~$ nano arithmetic.sh
raymond-ng@raymondng-VirtualBox:~$ ./arthmetic.sh 12 10
bash: ./arthmetic.sh: No such file or directory
raymond-ng@raymondng-VirtualBox:~$ ./arithmetic.sh 12 10
bash: ./arithmetic.sh: Permission denied
raymond-ng@raymondng-VirtualBox:~$ chmod +x arithmetic.sh
raymond-ng@raymondng-VirtualBox:~$ ./arithmetic.sh 12 10
$1=12
$2=10
Addition=22
Subtraction=2
Multiplication=120
Division=1
```

The example provided in instructions for this lab regarding positional parameters confused me, but I executed the script provided in the example to try it out and wanted to test out the `which` command to create a bash file. The script executed successfully; however, I wanted to define positional parameters a little better for my personal understanding, so I conducted some research on my end. I came across an example via TheGeekStuff.com that broke down positional parameters for me with a little more clarity (Sasikala, 2010). I now understand that a positional parameter is a parameter denoted by one or more digits, other than the single digit 0. Positional parameters are assigned from the shell's arguments when it is invoked and may be reassigned using the set buit-in command. In the screenshot you'll see that there are two arguments, and the script provides arithmetic operations result between two integers. In my output *$1* has the value of *12* and *$2* has the value of *10*. The script executes the basic mathematics operations on the given parameters.

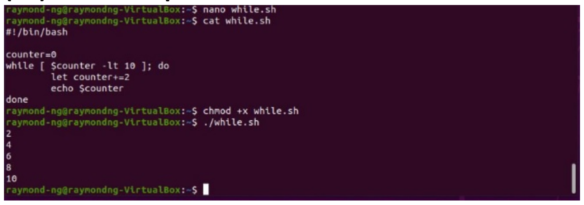
(17) For Loops



```
raymond-ng@raymondng-VirtualBox:~$ nano items.txt
raymond-ng@raymondng-VirtualBox:~$ cat items.txt
for
loops
bash
raymond-ng@raymondng-VirtualBox:~$ for i in $( cat items.txt ); do echo -n $i | wc -c; done
3
5
4
raymond-ng@raymondng-VirtualBox:~$
```

I created my own "*items.txt*" file using nano. Per the example for *for loops*, I typed `for i in $( cat items.txt ); do echo -n $i | wc -c; done` and it counted the characters of each line as my output from the "*items.txt*" file.

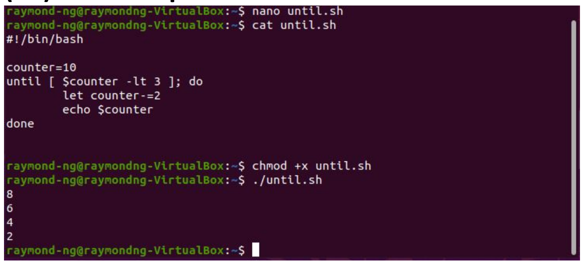
(18) While Loops



```
raymond-ng@raymondng-VirtualBox:~$ nano while.sh
raymond-ng@raymondng-VirtualBox:~$ cat while.sh
#!/bin/bash

counter=0
while [ $counter -lt 10 ]; do
        let counter+=2
        echo $counter
done
raymond-ng@raymondng-VirtualBox:~$ chmod +x while.sh
raymond-ng@raymondng-VirtualBox:~$ ./while.sh
2
4
6
8
10
raymond-ng@raymondng-VirtualBox:~$ 
```

In this while loop, the loop will keep executing the enclosed code only while the counter variable is less than *10*. During each loop iteration the variable is incremented by *two*. Once the variable is equal to *10*, the condition defined becomes false and the while loop execution is terminated.

(19) Until Loops



```
raymond-ng@raymondng-VirtualBox:~$ nano until.sh
raymond-ng@raymondng-VirtualBox:~$ cat until.sh
#!/bin/bash

counter=10
until [ $counter -lt 3 ]; do
        let counter-=2
        echo $counter
done

raymond-ng@raymondng-VirtualBox:~$ chmod +x until.sh
raymond-ng@raymondng-VirtualBox:~$ ./until.sh
8
6
4
2
raymond-ng@raymondng-VirtualBox:~$ 
```

In this until loop, the script begins with the variable counter set to *10*. The condition of the until loop keeps executing the enclosed code (decreasing in increments of *two*) until the condition becomes true.

Conclusion

This lab was excellent way for novice users like me to practice scripting in Bash. From performing this lab, I learned that paying attention to detail is especially important because missing a simple space or inputting a wrong variable or integer can affect your output. Further, I now have a better understanding of some of the syntaxes that we have encounter throughout the duration of this course.

References

Internet Resources

Gite, V. (2012, August 27). *Linux/Unix id Command Examples*. Retrieved from Cyberciti.biz: https://www.cyberciti.biz/faq/unix-linux-id-command-examples-usage-syntax/

Kerrisk, M. (2021, April 1). *lslogins(1) — Linux manual page*. Retrieved from man7.org: https://man7.org/linux/man-pages/man1/lslogins.1.html

Sasikala. (2010, May 10). *Bash Positional Parameters Explained with 2 Example Shell Scripts*. Retrieved from TheGeekStuff.com: https://www.thegeekstuff.com/

Collaboration

No collaboration was conducted with any of my peers or other students. For the most part, I troubleshooted on my own by researching via the internet to debug any issues I may have encountered and obtain clarity on concepts that didn’t make sense to me initially. Additionally, I frequented the chat groups via Slack often to observe issues other students were encountering and applied their lessons learned.