Shell Scripts

Introduction and/or Background
Shell scripts are a series of commands listed in a computer program for execution by the shell's command line interpreter. Most shell scripts are interactive (accept commands as input from users and execute them). System administrators use scripts often to avoid repetitive work, to conduct routine backups, or control/monitor system processes.

Objectives
In this project/lab the student will:

- Use a text editor to create, execute, and examine basic programming constructs by developing small bash scripts.

Equipment/Supplies Needed
- As specified in Lab 0.0.1.

Procedure

Note to Instructor/Student: It will be necessary in these scripts to replace the " with a real " in nano. A cut and paste will not work. Google Docs INSISTS on inserting the typographical " and not the ASCII one!

Perform the steps in this lab in the order they are presented to you. Answer all questions and record the requested information. Use the Linux Virtual Machine to perform lab activities as directed. Unless otherwise stated, all tasks done as a non-root user. If root access is needed use the sudo command.

Assignment
Lab Submissions Proof: Provide screenshots as indicated in the lab; upload your proof to Canvas for grading.

Bash Script

From the home directory, create a subdirectory named ***My_Scripts***.

1. mkdir *My_Scripts*

Move to the ***My_Scripts*** directory. Create a script using the nano editor. Name the file ***MyFirstScript.sh*** (where .sh will indicate this as a script file)

2. nano *MyFirstScript.sh*

Edit the empty file.

Since there are several shells available, it is good practice to define which one is being used. Define the script's interpreter as Bash, by defining it at the top of the script, entering the prefix (shebang) **#!**, and defining the bash shell (/bin/bash) in the editor.

To add comments, use the #(hash) character before any statements. Comments can be used to document details about the script. Lines starting with the # sign, except shebang, will not be interpreted by bash, are used to document a script. At a minimum, common practice is to define the name of the script, its purpose, date created, date last edited, and the owner's name and contact information.

```
#! /bin/bash

# Script Name: MyFirstScript.sh

# Purpose: Practice script, to learn the basics of shell scripting and the echo command.

# Date: [ENTER TODAY'S DATE HERE]

# Owner: [ENTER YOUR NAME HERE]

# Email: [ENTER TSTC EMAIL ADDRESS]
```

Add an echo command (send message to terminal window) to get the script started.

```
echo "Welcome to the world of script files."

echo "I hear shell scripting is fun! I am here to find out"

echo -n "The current time and date is now "

date
```

Save the script file and exit the editor.

The script will require permissions to execute. Use the **chmod** command to make

sure it will run. Permissions can be written as **a+x** or numerically as **755**.

3. chmod 755 *MyFirstScript.sh*

Use **./** to execute the command.

4. *./MyFirstScript.sh*

Troubleshoot -- If the script does not run, go back through the instructions and look for typos, or anything missing in the syntax. Make sure you have the execute bit set. Check by doing ls -la MyFirstScript.sh. You should see "- rwx" in the first 4 columns associated with your script. Have someone else review your script, or ask your Instructor for help.

Record a screenshot of successful script execution.

Variables

Variables are a key component of any program. They allow a script to accept and store data from STDIN for later use/processing.

Open a new file. Name it *Variable.sh*. Add the file information at the top as before. Adjust the name and purpose statement.

Store the output of the system hostname command in a variable. Display to the terminal window.

```
# use a variable to hold information, and echo to send it STDOUT.
GuestPC=$(hostname)
echo "I see you are using the Virtual Machine, $HOSTNAME"
```

Create an interactive variable using the **read** and **echo** commands to bring information into the script (STDIN).

Add the following lines to the script.

```
#use a variable to collect information (STDIN) and send to STDOUT.
echo "What is your name?"
read username
echo " Hello $username. I see you want to learn Linux scripts too."
echo " ${username}, I am practicing, so I will create a file for you just for fun"
touch $username
```

Save the script and exit the editor. Execute the script to test it. Make you have set the file permissions. Troubleshoot -- If the script does not run, go back through the instructions and look for typos, or anything missing in the syntax. Have someone else review your script, or ask your Instructor for help.

Verify/Record that the file with your name was created. Record a screenshot of successful script execution.

If, Case Statements

If and **Case** statements enable decision making in scripts. Whether an action is going to take place or not depends on conditions met/not met.

Create an **If** statement to perform some action, based on meeting or not meeting a test or condition. If performs a set of commands if the test is true.

5. Open a new file. Name it *If.sh* add the file information at the top as before. Adjust the name and purpose statement.

6. Add the following lines.

```
#This is an example of using an if statement to read a number, making a decision, and responding accordingly.

echo -n "Let's play a game. Enter a number greater than 10"
read num

if [ $num -gt 10 ]

then
    echo "Number is greater than 10. You got it! You are so smart."

else
  echo " What??!– try again. Are you sure you want to learn linux?"

fi
```

Save and exit the editor. Execute the script to test it. Troubleshoot -- If the script does not run, go back through the instructions and look for typos, or anything missing in the syntax. Have someone else review your script, or ask your Instructor for help.

Record a screenshot of successful script execution.

Use **Case** statements to make a choice, based on multiple options. Case will execute depending on the matching of a specified string.

7. Open a new file. Name it *case.sh*. Add the file information at the top as before. Adjust the name and purpose statement.

8. Add the following lines.

```
# This is an example of the case statement.

echo "Which Linux distro do you like best? "

read DISTR

case $DISTR in

        ubuntu)

            echo "I've heard of it! I think it is debian-based."

            ;;

        debian)

            echo "Hey! That's the one I am using!"

            ;;

        windows)

            echo "Why ARE YOU HERE?"

            ;;

        Kali)

            echo "Yes!.. Do you want to be a Security Guru too?"

            ;;

        *)

            echo "Hmm, Never heard of it...."

            ;;

    esac
```

Save exit the editor. Execute the script to test it. Troubleshoot -- If the script does not run, go back through the instructions and look for typos, or anything missing in the syntax. Have someone else review your script, or ask your Instructor for help.

Record a screenshot of successful script execution.

Loops

Loops are statements that repeat through whatever input it is given, until it runs out. *A For* loop condition is evaluated as the input is presented. In a ***while*** loop, a condition is evaluated before processing a body of the loop.

Use a ***For Loop*** to repeat a task multiple times.

9. Open a new file. Name it *For.sh*. Add the file information at the top as before. Adjust the name and purpose statement.

10. Create a loop that does a countdown from 10 to 1. Use the ***sleep*** *command* to pause the execution on the next *command* for a given time (seconds).

```
#For Loop to do a countdown.
   echo "start the countdown"
   for i in 10 9 8 7 6 5 4 3 2 1
   do
     echo "Countdown through numbers  $i "
   sleep 5
   done
   echo " I am done counting"
```

Save and exit the editor. Execute the script to test it. Troubleshoot -- If the script does not run, go back through the instructions and look for typos, or anything missing in the syntax. Have someone else review your script, or ask your Instructor for help.

Record a screenshot of successful script execution.

A ***While Loop*** checks for a condition before executing a command.

11. Create a while loop.

```
#While loop that keeps responding until condition (bye) is met.
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
echo "So glad you are done – that was fun, wasn't it?"
```

Edit the Name (While.sh) and purpose statement at the top of the file. Save your file as *While.sh* and exit the editor. Execute the script to test it. Troubleshoot -- If the script does not run, go back through the instructions and look for typos, or missing parts of the syntax. Have someone else review your script, or ask your Instructor for help.

Record a screenshot of successful script execution.

Menu Script

Combining knowledge of linux commands and scripts can make an administrator's life easier.

12. Create and execute the following basic script, using the case statement to create a menu system. Record a screenshot of successful script execution.

```
echo -n "Name please? "
read name
echo
echo
echo "$name, what would you like to see?"
echo  "(1) System information"
echo  "(2) Amount of disk space used/available"
echo  "(3) Current memory usage"
echo  "(4) Current running processes"
echo  "(5) Quit "

Echo "Enter a selection (1-5):"

read select

case $select in

        1)

        uname -r

        ;;
```

```
        2)
        df
        ;;
        3)
          free -h
        ;;
        4)
        ps -aux
        ;;
                5)
        echo "Thank You, Good Bye ..."
        ;;
        *)
        echo "Not a valid selection – GoodBye"
        ;;
esac
```

Rubric

Checklist/Single Point Mastery

Concerns Working Towards Proficiency	Criteria Standards for This Competency	Accomplished Evidence of Mastering Competency
	Criteria #1: Recorded screenshot successful execution of bash script (14 points)	
	Criteria #2: Recorded screenshot successful execution of variables script (14 points)	
	Criteria #3: Recorded screenshot successful execution of If statement script (14 points)	
	Criteria #4: Recorded screenshot successful execution of If statement script (14 points)	
	Criteria #5: Recorded screenshot successful execution of For loop script (14 points)	
	Criteria #6: Recorded screenshot successful execution of While loop script (14 points)	
	Criteria #7: Recorded screenshot successful execution of menu script (16 points)	