**Lab 3 – Part 1**

Oakland University / CSI3660

Fall 2018

This is a 2-part lab. Part 1 is what you \*should\* be able to accomplish in class. Part 2 extends the work and is intended to be homework.

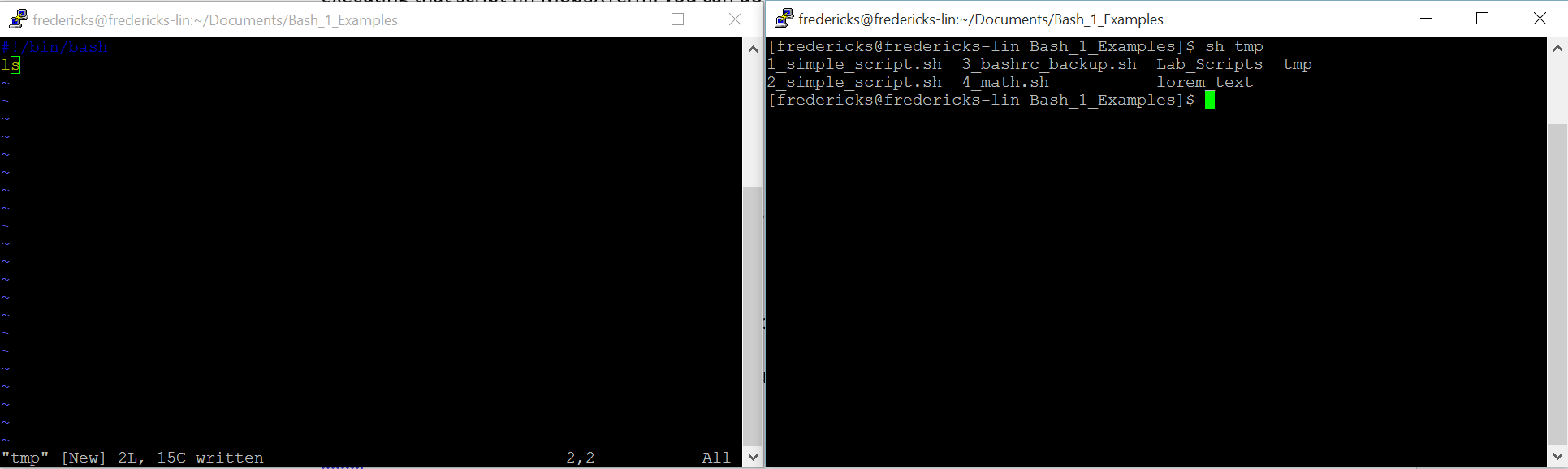
**Parts 1 and 2 due by October 12th @ 11:55pm**

**100 points**

This lab is intended to get you familiar with bash scripting.

Configuration and Setup

Open up 2 terminals side by side. One will be your text editor and another will be your terminal for executing that script (in MobaXTerm, you can do tabs instead if you prefer). Note, anything that looks like <….>, you need to replace the <...> with either a command or some text.



That’s it. Onto the actual work!

**LAB PROCEDURE**

We are going to do 3 sample problems of increasing difficulty during this hour.

**Script 1**

Let’s do some basic scripting. Open up vim/nano/emacs/gedit to start creating your first script.

The first line needs to define which shell will be used to execute the program. Therefore, make the first line read:

#!/bin/bash

In this case,#! is a special set of characters that defines the bash shell. Any other instances of # will be treated as a comment (i.e., not parsed by the interpreter).

Next, we are going to do some simple string manipulations. This will be a simple program that accepts as input a string, and then simply outputs that string to the terminal.

Command line arguments are straightforward. $1 is the first, $2 is the second, and so on.

For instance:

#!/bin/bash

echo "Hello there $1, how are you doing today?"

would output "Hello there Erik, how are you doing today?" if I called this script as:

[user]$ bash sample\_script.sh Erik

You can also pass a script a command as a parameter. For instance, if a script looked like:

#!/bin/bash

$1

and was called with:

[user]$ bash sample\_script.sh pwd

The output (for me) would be: /home/fredericks

**For Script 1, you need to do the following:**

1. Create a bash script that takes in 3 command line parameters. The first parameter is your first name, the second parameter is your username for your system, and the third parameter is the command used to list the contents of a directory.

For example, it will be run as:

[user]$ sh <lastname>\_Script1.sh <first name> <username> <directory listing command>

1. The script must print the following output, replacing <first name> with your first name, and <contents of user’s home directory> with a listing of the files in that directory (hint, remember the **echo** command):

*Hello <first name>, the contents of my home directory are:*

*<contents of user’s home directory>*

Please note that you cannot directly list these contents, but **must** use the command line argument. If I see you directly listing the contents of your username in your script, you will lose points for this section.

As a reminder for how to save output to a file, you can use the redirect operator (>) to redirect to a file (e.g., bash script.sh > script\_output.txt)

Save the script as <lastname>\_Script1.sh, and the output in a text file named <lastname>\_Script1\_Output.txt.

**Script 2**

This script introduces the concept of a loop and the modulo. For this script, you must perform a loop, over a range from 1 to 15, that outputs the remainder of all numbers when divided by 2 (you must also print the operation). For instance, I expect to see the following:

1 % 2 = 1

2 % 2 = 0

3 % 2 = 1

4 % 2 = 0

…

We can use a **for loop** to iterate over a range of numbers. For instance, if we want to iterate over a range of numbers between 1 and 20, we would use the following:

for i in {1..20}

do

<commands>

done

The first line (for i in {1..20}) tells the script that each time the loop is executed, the variable i will take on a value between 1 and 20. So, the first time the loop runs, i will be 1, the second time i will be 2, and so on until it hits 20.

The second and last line (do / end) specify that the contents in between will be executed repeatedly until the end case is hit (i = 20).

Finally, let’s do a little bit of math. The modulo operator gives us the remainder of a division operation. So, if we divide 10 / 2, the answer is 5 with no remainder. However, 10 / 3 has a remainder of 1. 10/4 has a remainder of 2, and so on.

We won’t go into a basic math lesson, but understand that, if a number is divisible by another number, the remainder will be 0.

To find this remainder, we use the modulo, or %, operator. For instance (either work):

[user]$ expr 10 % 2

0

[user]$ echo $[10 % 2]

0

We can use this information to determine if a number is odd or even. If you take the modulo 2 of a number, a remainder of 1 indicates that the initial number is odd, and a remainder of 0 indicates if it is even.

Again, save your script file as <lastname>\_Script2.sh, and the output as <lastname>\_Script2\_Output.txt.

**Script 3**

Let’s do something practical. We will automate the creation (and deletion) of user accounts on our system.

First, create an array of users (you need to have at least 5 users in your array). The username itself is up to you. Just make sure the names are appropriate for a child under 10 to read.

Here is the syntax for defining an array with two elements:

userlist[0]="user1"

userlist[1]="user2"

Here, userlist is the variable (that is actually an array). It contains data, in this case strings, that can be accessed using an *array index*, where the array index is simply a number. Note that there is **NO SPACE** before or after the equal sign

Another way to declare the array is to:

userlist=("user1" "user2")

Now, if I wanted to output the 2nd user, I would do:

echo ${userlist[1]}

Note that array numbering starts at 0, not 1! The second element of the array is indexed using 1…the third would be index 2, and so on. This is a common paradigm in most programming languages.

Now, recall from a previous assignment that the user creation command is useradd.

**For this portion, you must do the following:**

1. Create an array that contains at least 5 usernames
2. Create a for loop that prints each username to the console
3. Create another for loop that adds each user to the system (note: you can combine (2) and (3) if you want)
4. Demonstrate that the users have been added to the system (hint: you did this in HW2)

*Name this script <lastname>\_Script3\_a.sh, and any output save as <lastname>\_Script3\_a\_Output.txt*

1. Copy the file you just created, but name it <lastname>\_Script3\_b.sh
   1. (use the cp command)
2. Now, instead of adding users, delete the users from your system (hint: pass the –r flag to the appropriate user deletion command – you also did this in HW2)
3. Again, demonstrate that (a) the users are no longer on the system and (b) their user directories are gone

*Name this script <lastname>\_Script3\_b.sh, and any output save as <lastname>\_Script3\_b\_Output.txt*

**LAB REPORT**

Note that Part 2 will be released by tomorrow. For now, you don't need to do anything. Packaging up your scripts and working on the lab report will be written up in part b. If you needed to take any screenshots, though, make sure you have those (like prior to deleting users to show it worked…).

Note that each script must be formatted as follows – points will be taken off if it is not commented approrpiately:

#!/bin/bash

# Your name

# Your email address

# The date this assignment is due

# A short description of what this script does

*Code*

You should also properly comment your code as necessary to ensure that you describe the intent of various code sections.