COMP 0150 Lab 2: Bash Scripting

In this lab, you will write a number of bash shell scripts in order to familiarize yourself with the basic syntax and structure of shell scripts. Each script you write should be copied into your lab submission document along with screenshots demonstrating that the script works as required.

Fire up your VM and log in to get started. You'll need to refer back to the shell scripting and programs slides throughout this lab to look for syntax and examples. For this lab (and most others) you should create a lab directory in your home area to store all your scripts. Recall that you use the `mkdir` command to create a new directory. So, do the following:

1. Make sure you are in your home area by running: `cd`
2. Then create a `lab1` directory: `mkdir lab1`
3. And then change directories in to your `lab1` directory: `cd lab1`

Your first task is to write a simple shell script that prints a list of all processes on the system owned by yourself. The `ps` command is used to show processes on a Linux system. With no arguments it only displays processes started in your current terminal window. Try it out:

1. Run: `ps`
2. You should see a list of processes. Each line is a different process. You probably only have two processes visible, `bash` and `ps`.

There are many, many arguments and options to `ps`. Two of the most useful are `-e` and `-f` that together show a list of all processes running on the system. Do:

1. Run: `ps -ef`
2. Again, you see a list of processes, one per line. If you scroll through the list, you'll see many processes. The first column is the user that owns the process (usually the user who started the process).

We want to narrow down this list to only show processes that you own. How do you narrow down/search for particular values in an input stream? `grep`! So, use a pipe to send the output of the `ps -ef` command to `grep` and search for your username:

1. Run: `ps -ef | grep USERNAME`
2. Obviously, replace `USERNAME` with your actual username.

Now, because that is a useful thing to have handy, let's write a script that does it for us so that we don't have to type all that in every time on the command line:

1. Use your editor of choice and edit a new file named `myps.sh`
2. Add the normal `#!/bin/bash` to the top of the file, then add your `ps/grep` command from above.
3. Save the file and exit your editor.
4. Remember that you have to make a script executable before you can run it for the first time, so run: `chmod u+x myps.sh`
5. Now test your script: `./myps.sh`
6. You should see the same output as you had when you ran the command directly on the command line.

OK, now look at your `myps.sh` script. As it is, it prints out any lines that contain your username anywhere on the line. However, we really only want it to print processes that are actually owned by you. Your username is the very first thing on every line, so how would you modify your `grep` command to ensure that only those lines starting with your username are printed? Go back and look at the `grep` and regular expression slides if you need a reminder. Once you figure it out, update your script, and test it to make sure that it works with the modifications. When you're sure it's working correctly, **COPY AND PASTE THE SCRIPT** into your submission document. Also **TAKE A SCREENSHOT** showing an example run of the script. You don't need to include the entire output, but make sure that the screenshot includes the command line that shows the command you ran.

Although this is a very simple script, you should try to follow this same methodology when you write any script. Namely, make sure you know how to do something on the command line first then incorporate it into your script. Moreover, always take small steps when writing scripts (or any programs). Add one or two lines, stop and test your script. Fix any bugs or syntax errors, then add a few more lines and repeat. If you try to write even simple scripts all in one go you are only making your life harder in the long run.

For the rest of the lab, you will be writing three additional scripts.

First, write a simple calculator script named `calc.sh` that takes three command line arguments: an integer, a string operation, and another integer. The operation should be one of the following: "`plus`", "`minus`", "`times`", "`over`", "`pow`" (`pow` meaning power, as in raise the first number to the second number power). Your script should:

- Check that the number of arguments is correct, and print a usage message if not.
- Check that the operation is one of the five listed above, and print a usage message if not.
- Use either a series of `if` statements or a `case` statement to produce the result (using the correct arithmetic syntax) given the two numbers and the operation.
- Print the result.

Here is an example run:

```
$ ./calc.sh
usage: ./calc.sh num1 op num2
$ ./calc.sh 3 plus 4
3 plus 4 is 7
$ ./calc.sh 3 minus 4
3 minus 4 is -1
$ ./calc.sh 3 times 4
3 times 4 is 12
$ ./calc.sh 3 over 4
3 over 4 is 0
$ ./calc.sh 3 pow 4
3 pow 4 is 81
$ ./calc.sh 3 wop 4
usage: ./calc.sh num1 op num2
$
```

Once you've got it tested and working, **COPY AND PASTE THE SCRIPT** into your submission document and **TAKE A SCREENSHOT** of the output of the script using some example arguments.

Second, write a script named `sum.sh` that reads a series of positive numbers from the user and computes the sum of those numbers. The numbers should be read in using the `read` command, and the script should continue reading numbers until a negative value is entered. So, you'll have to use a `while` loop to read one value at a time and keep looping until a negative number is read from the user. Here is a sample run:

```
$ ./sum.sh
1
2
3
4
5
-1
sum is 15
$
```

Once you've got it tested and working, **COPY AND PASTE THE SCRIPT** into your submission document and **TAKE A SCREENSHOT** of the output of the script for some sample input.

Finally, write a script named `print_lines.sh` that uses `head` and `tail` together to print out a specific set of lines from a file. `head` prints out the first N lines from a file (10 by default, changed with the `-n` option), and `tail` prints out the last N lines from a file (10 by default, changed with the `-n` option). You should get a bit of practice with the `head` and `tail` commands first so you know how they work. Look at the `man` page and output of `--help` for each command for more info, but the basic usage is very simple. For example:

1. To print the first 20 lines of `/etc/passwd`, run: `head -n 20 /etc/passwd`
2. To print the last 4 lines of `/etc/group`, run: `tail -n 4 /etc/group`

Your `print_lines.sh` script should take three arguments: the line number to start at, the line number to stop at, and the file to use. Here's an example run:

```
$ ./print_lines.sh
usage: ./print_lines.sh start_line stop_line file
$ ./print_lines.sh 7 9 /etc/passwd
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
$ ./print_lines.sh 1 1000 /etc/passwd
stop_line must be <= number of lines in the file
$ ./print_lines.sh 3 5 /blah
usage: ./print_lines.sh start_line stop_line file
$ ./print_lines.sh 9 7 /etc/passwd
start_line must be <= to stop_line
$
```

Your script must do error checking. Specifically, you need to check:

- You have the right number of arguments.
- The file specified exists and is a normal file.
- The first line number specified is less than or equal to the last line number specified.
- The actual number of lines in the file is greater than the last line to be printed.

If any of those conditions is not true, you should print an appropriate error message to the user and stop. If they are all met, then you'll need to do a bit of arithmetic and use `head` and `tail` together to print out only the lines requested.

Once you've got it tested and working, **COPY AND PASTE THE SCRIPT** into your submission document and **TAKE A SCREENSHOT** of the output of the script using some example arguments.

Extra Credit: If you want to earn some extra credit, you can do so by modifying the first script from this assignment, `myps.sh`, in two ways.

First, add an optional command line argument to specify the username to use in the `grep` pattern. If the argument is given, you should display all processes owned by that username. If the argument is not given, it should use the username of the current user. You should NOT hard code your username. Instead, you need to find a special environment variable that contains the current user's username and use that instead.

Second, use `awk` to modify the output of the script to print out only the process ID (PID) and command (CMD) columns. You'll first need to figure out which columns these two are in the `ps` output, then send the output of your `ps`/`grep` command to `awk` to print only those columns. Note that you only need to print the name of the command from the CMD column, not the arguments for the command.

Once you've got it tested and working, **COPY AND PASTE THE SCRIPT** into your submission document and **TAKE A SCREENSHOT** of the output of the script using some example arguments.