# LAB 5-1 – Log Your Work

Use the logging via PuTTY to log your terminal session.

Alternatively, use the command **script** with a filename to log your session:

**script lab5-2021-yourname.log**

Use the command **exit** when fini­­shed with the log file.

# LAB 5-2 – Environment Variables

Run the **printenv** command and use a pipe ( | ) character to send the output into less.

**printenv | less**

Use the up and down arrows to look at the output. You can use this to view the values of all of your environment variables.

You can also use **printenv** to show specific environment variables as well. To show the values of the USER, SHELL, and PATH variables run the following command:

**printenv USER SHELL PATH**

# LAB 5-3 – Arithmetic Operations

Using the **let** command you can tell the shell to process math operations.

**let ANSWER=5+2\*4**

**echo $ANSWER**

This should display a value of **13**.

**ANSWER=5+2\*4**

**echo $ANSWER**

Without the **let** command it interprets the value as a string and displays text only.

# LAB 5-4 – Exporting Variables

Use the text editor of your choice to create a new script named **firstscript.sh**.

Using **vi** you would use the command

**vi firstscript.sh**

Add the following line to your script file:

**echo $VAR1**

Save your changes and exit the text editor.

Make the script executable by granting it the execute permission.

**chmod +x firstscript.sh**

Give $VAR1 a value using the command:

**VAR1="I'm a value!"**

Print the value of **VAR1** to the terminal with an **echo** command.

**echo $VAR1**

You should see the text **I’m a value!** displayed.

Run your script which has the same command.

**./firstscript.sh**

Despite being the same command, nothing is displayed. Export the variable, then try running the script again.

**export VAR1**

**./firstscript.sh**

The text **I’m a value!** should be displayed properly.

# LAB 5-5 – Editing the Path

In 4-4 you had to use **./** before the name of your script to run it. In this lab we will edit the path to always include your current directory.

Attempt to run the script without the ./

**firstscript.sh**

Echo the **$PATH** variable to the terminal to view it.

**echo $PATH**

Notice that the directories are separated by colons (:).

Edit the **PATH** variable to include the current directory and verify your changes by echoing the **PATH** again.

**PATH=$PATH:.**

Now test that you can run **firstscript.sh** without the ./

You can now run scripts in your present working directory without the leading ./ and it is amazing. Tell your friends!

*Note: The PATH variable will be reset when you end your current terminal session.*

*Another Note: The PATH variable is being modified using string concatenation. No special characters are needed to concatenate strings using bash. You can see this easily by entering the command* ***echo $VAR1$VAR1***

# LAB 5-6 – Simple Shell Script

Using a text editor of your choice create a script called **addingnumbers.sh** and add the following lines to the file:

**let NUM1=2  
 let NUM2=4  
 let NUM3=6  
 let TOTAL=NUM1+NUM2+NUM3  
 echo $TOTAL**

This script will be run differently. Instead of using **chmod** to add the execute permission, launch a new bash shell and give the name of the script as a parameter.

**bash addingnumbers.sh**

# LAB 5-7 – If Statements

This lab will create a script using a simple if statement.

Create a script called **faveclass.sh** and add the following lines:

**echo -n "What is your favourite class this term? "  
read class\_choice  
if [ "$class\_choice" == "Linux" ]  
then  
 echo "Good answer! Good answer!"  
else  
 echo "Are you sure about that?"  
fi**

*Note the spacing around the quotes and braces as they are important.*

Run your script (using any method) and verify both conditions of the if statement can be executed.

Next, modify the script to include a nested if statement.

**echo -n "What is your favourite class this term? "  
read class\_choice  
if [ "$class\_choice" == "Linux" ]  
then  
 echo "Good answer! Good answer!"  
else  
 if [ "$class\_choice" == "Mechatronics" ]  
 then  
 echo "Yeah, Mechatronics is pretty great!"  
 else  
 echo "Are you sure about that?"  
 fi  
fi**

Run the script again and verify all three conditions of the if statement can be executed.

# LAB 5-8 – For Loop Statements

This lab will display a simple for loop.

Create a script file called **students.sh** and add the following lines to it:

**for STUDENTS in Jeff Britta Abed Shirley Annie Troy Pierce  
do  
 echo $STUDENTS  
done**

Execute the script and verify it works correctly, listing the name of each student.

You can also execute for loops directly into your terminal without the use of a script. **Try typing the for loop directly into the terminal**, using the same line breaks.

# LAB 5-9 – For Loop Statements and Wildcards

Create four files named chap1, chap2, etc.:

**touch chap1 chap2 chap3 chap4**

Modify the text of each file to read **This is chapter X** (replace X with the number for the file).

Create a script named **showchapters.sh** and add the following lines to it:

**for BOOK in chap[1234]; do  
 cat $BOOK  
done**

Execute the script and test that you can read the output from each of the **chap** files.

# LAB 5-10 – While Loop Statements

Create a script called **guessnum.sh** and add the following lines:

**echo -n "Guess the number I've chosen between 1 and 10: "  
read guess  
while [ "$guess" != "9" ]; do  
 echo "No, that's not it. Try again."; read guess  
done  
echo "You got it!"**

This type of loop can easily be used to create a data entry form.

Create a new script called **dataentry.sh** and add the following lines:

**userdone=n  
while [ "$userdone" == n ]  
do  
 echo -n "Enter Name: "; read name  
 echo -n "Enter Street: "; read street  
 echo -n "Enter City: "; read city  
 echo -n "Enter Province: "; read province  
 echo -n "Enter Postal Code: "; read postal  
 # You could do something with the data here, like output  
 # it to a file or database.  
 echo -n "Have you finished entering data? (y)es or (n)o: "  
 read userdone  
done**

Execute the script, answering “no” at least once to verify the scrip is working.

# LAB 5-11 – Select Case Statements

Create a new script called **zork.sh** that contains the following:

**echo "Welcome to Zork!"  
echo -n "Choose a cardinal direction to move: "; read direction  
case "$direction" in  
 "north") echo "You head north and the air gets colder.";;  
 "east") echo "You head east and find a sword.";;  
 "south") echo "You head south and find a key.";;  
 "west") echo "You have been eaten by a grue.";;  
 \* ) echo "That's not a valid direction.";;  
esac**

Test out the various options using select case.

# LAB 5-12 – TPUT

The **tput** command allows you to initialize the screen as well as position the cursor and text in an appealing way. You will first work directly with tput then create a script with a menu to see the potential of the command.

Start by entering the following command:

**tput clear ; tput cup 10 15 ; echo "Hello" ; tput cup 20 0**

The screen is first cleared.

It then places the cursor 10 lines from the top of the terminal and 15 spaces from the left.

The word “Hello” is printed to the terminal at this position.

Finally, the cursor is placed 20 rows from the top of the terminal and 0 spaces from the left. The prompt is printed at this point.

Next, create a script named **screenposition.sh** with the following:

**tput cup $1 $2 # place cursor on row and col  
tput clear # clear the screen  
bold=`tput smso` # set stand-out mode – bold  
offbold=`tput rmso` # reset screen – turn bold off  
echo $bold # turn bold on  
tput cup 10 20; echo "Type Last Name:" # bold caption  
tput cup 12 20; echo "Type First Name:" # bold caption  
echo $offbold # turn bold off  
tput cup 10 41; read lastname # enter last name  
tput cup 12 41; read firstname # enter first name**

Execute the script and test it.

Note the comments after the commands to understand the flow of the script. The commands to turn bolding on and off are stored in variables for ease of use.

# LAB 5-13 – Shell Script Debugging

Compare the output of **sh -v** and **sh -x** when debugging a script.

Run the following command, using a previous script, and note the output:

**sh -v zork.sh**

Notice that the command lines are printed out.

Run the following command and note the output:

**sh -x zork.sh**

Notice the command lines and arguments are displayed with a plus symbol (+) in front of them.

Create a few errors (at least 3) in the **zork.sh** file and use **sh -x** to execute the script and fix each of the errors.

# LAB 5-14 – Create an Alias

Enter the following commands and note the output:

**ll  
ls -l  
alias ll=** **"ls -l"  
ll**

The command **ll** can now be executed to run the command **ls -l**.

# LAB 5-15 – Create and Search Employee Directory

Make a new directory called **employees**. Create a new file in this directory called **employees.txt** and fill it with the following information:

**219-555-4567:Jake:Peralta:4540:Detective:08-18-1978  
219-555-4587:Rosa:Diaz:4541:Detective:02-10-1981  
219-555-4589:Terry:Jeffords:4544:Det. Sergent:07-30-1968  
219-555-4591:Amy:Santiago:4500:Sergeant:08-19-1982  
219-555-4544:Gina:Linetti:4520:Civ. Administrator:02-20-1978  
219-555-4549:Charles:Boyle:4520:Detective:12-02-1970  
219-555-4501:Raymond:Holt:4501:Captain:07-01-1962**

Use the following command to search **employees.txt**.

**grep Jake employees.txt**

It should return any lines that contain the text **Jake**, like so:

**219-555-4567:Jake:Peralta:4540:Detective:08-18-1978**

# LAB 5-16 – Create an Employee Management Application

Finally, using the skills and commands from this lab, create an application to search and manage and employee list. Name the application **emp-YOURNAME.sh**.

The application should do the following tasks in a main menu:

* View the entire Employee List
* Add Employees
* Search Employees
* Quit

Other requirements to note:

* Employees have the following attributes, which the user will need to enter when adding an employee.
  + Phone Number
  + First Name
  + Last Name
  + Employee Number
  + Position
  + Date of Birth
* Be sure to use **tput** to make the menu and any prompts look good.
  + Add Employee Fields
  + Search Criteria
* Feel free to break the script up smaller scripts, having one main script execute another.
* No input validation is required.
  + We’ll assume if the user is running a terminal application, they know what they’re doing.
* Use the awk command to format the output from any text fields (Search and View All).