Student’s Name:

# Small Projects Using Commands, Variables and Shell Scripting

## Project 1

Before setting one or more environment variables, it is a good idea to view their current configurations. In this project, you use the ***printenv***command to view a list of your environment variables.

To see a list of your environment variables:

1. Your list of environment variable might be longer than the default screen or terminal window size, so it can be helpful to pipe the output into the more command. Type ***printenv* | *more*** and press Enter.
2. Record some examples of environment variables. Press the spacebar to advance through the listing one screen at a time. Record your observation:
3. Type ***clear*** and press Enter to clear the screen.
4. Next, use the ***printenv*** command to view the contents of two variables: SHELL and PATH. Type

***printenv SHELL PATH*** and press Enter. Record your observation:

1. Type ***clear*** and press Enter to clear the screen.

## Project 2

The next project enables you to use the defining and evaluating operators to learn how they work. You begin by assigning a value to a variable and then view the contents of the variable you assigned. You then learn how to assign a variable that contains spaces, and you compare using single and double quotation marks to evaluate the contents of a variable. Finally, you use the back quote marks to execute a command and store the result in a variable.

To create a variable, and assign it a value:

1. Type ***DOG=Shepherd*** and press Enter.
2. You’ve created the variable DOG and set its value to Shepherd.

To see the contents of a variable:

1. Type ***echo DOG*** and press Enter. What do you see?
2. To see the contents of DOG variable, you must precede the name of the variable with a $ operator. Type ***echo $DOG*** and press Enter. What do you see?

To use double quotation marks to set a variable to a string of characters containing spaces:

1. Type ***MEMO= "Meeting will be at noon today"*** and press Enter.
2. Type ***echo $MEMO*** and press Enter. What do you see?

To demonstrate how double quotation marks do not suppress the viewing of a variable’s contents, but single quotation mark do suppress the viewing.

1. Type echo ***‘$HOME’*** and press Enter. What do you see?
2. Type ***echo “$HOME”*** and press Enter. What do you see?

To demonstrate the back quote operator for executing a command:

1. Type ***TODAY = ‘date’*** and press Enter. This command creates the variable TODAY, executes the date command, and stores the output of the date command in the variable TODAY.
2. Type ***echo $TODAY*** and press Enter. You see the output of the date command that was executed in Step 1.
3. Type ***clear*** and press Enter to clear the screen.

## Project 3

In this project, you employ the ***let***command to practice using arithmetic operators to set the contents of a shell variable. First, you use an expression with constants (no variables), and then you use an expression containing a variable.

To practice using the arithmetic operators:

1. Type ***let X=10+2\*7*** and press Enter.
2. Type ***echo $X*** and press Enter. What do you see?
3. Type ***let Y=X+2\*4*** and press Enter.
4. Type ***echo $Y*** and press Enter. What do you see?
5. Type ***clear*** and press Enter to clear the screen.

## Project 4

In this project, you export a shell variable to make it globally recognized.

To demonstrate the use of the export command:

1. Type ***cat > testscript*** and press Enter.
2. Type ***echo $MY\_VAR*** and press Enter.
3. Type ***Ctrl+d***. You have created a simple shell script named testscript. Its only function is to display the value of the MY\_VAR variable.
4. To make the script executable, type ***chmod ugo+x testscript***, and press Enter.
5. Type ***MY\_VAR=2***, and press Enter.
6. Type ***echo $MY\_VAR*** and press Enter to confirm the preceding operation. What do you see?
7. Next look at the list of environment variables. Type ***printenv* | *more*** and press Enter.

Look carefully as you scroll through the output of the printenv command. You do not see the MY\_VAR variable.

1. Type ***clear*** and press Enter to clear the screen.
2. Execute the shell script by typing ***./testscript*** and press Enter. The script displays a blank line. This is because it does not have access to the shell variable MY\_VAR.
3. Make the variable available to the script by typing ***export MY\_VAR*** and pressing Enter.
4. Execute the script again by typing ***./testscript*** and press Enter. What do you see?
5. Now look at your list of environment variables by typing ***printenv | more*** and pressing Enter. Again, look carefully as you scroll through the list. What do you see?
6. Type ***clear*** and press Enter to clear the screen.

## Project 5

In previous projects, you had to use ./ before testscript because your current working directory is not in your PATH environment variable. In this project, you view the contents of the PATH variable. Next, you add the current working directory to the PATH variable and run testscript without using ./ characters.

To see the contents of the PATH variable:

1. Type ***echo $PATH*** and press Enter.

You see a list of directories. Notice that the path names are separated by colons (:).

To add the current working directory to the PATH variable:

1. Type ***PATH=$PATH:.*** and press Enter.
2. Type ***echo $PATH*** and press Enter. The dot (.) is now appended to the list.
3. You can now run scripts in your current working directory without typing the ./ characters before their names. Test this by typing ***testscript*** and pressing Enter. What is your output?

## Project 6

In this project, you gain future experience in writing a very simple shell script using sequential logic. In these steps, you create the shell script, seqtotal.

To demonstrate sequential logic:

1. Create script using vi, nano, or other editor and save as **seqtotal. let a=1**

**let b=2**

**let c=3**

**let total=a+b+c**

**echo $total**

1. Type ***bash seqtotal*** and press Enter. What is your output?

## Project 7

This project provides your first introduction to using an ***if statement*** in a shell script and demonstrate decision logic. In the first set of steps, you create a script using a basic ***if statement***. Then, in the second set of steps, you modify your script to include an ***if statement*** nested within an ***if statement***.

To demonstrate the if statement as well as to implement decision logic:

1. Create script using vi, nano, or other editor and save as ***vegchoice***. You will need to make somce corrections to the script in order for it to work properly.

#!/bin/bash

**echo “What is your favorite vegetable?”**

**read veg\_name**

**if [ “$veg\_name” = ”broccoli” ]; then**

**echo “broccoli is a healthy choice.”**

**else**

**echo “do not forget to eat your broccoli also.”**

**Fi**

What changes did you make? List changes here:

1. Make the script executable by typing ***chmod ugo+x vegchoice*** and pressing Enter. Next, run the script by typing ***./veg*choice** and pressing Enter.
2. When asked to enter the name of your favorite vegetable, enter ***broccoli***. Record your output:
3. Run the script again and enter ***corn or some other vegetable name***. Record your output:

To practice writing a nested ***if statement***:

1. Open the **veg\_choice** file in vi, nano, or other editor.
2. Edit the file so it contains the following lines:

**echo "What is your favorite vegetable?"**

**read veg\_name**

**if [ “$veg\_name” = ”broccoli” ]; then**

**echo “Broccoli is a healthy choice.”**

**else**

**if [ “$veg\_name” = ”carrots” ]; then**

**echo “Carrots are great for you.”**

**else**

**echo "Don’t forget to eat your broccoli also."**

**fi**

**fi**

1. Execute the script and respond with ***carrots*** when asked for your favorite vegetable. What response do you see?
2. Type ***clear*** and press Enter to clear the screen.

## Project 8

In this project, you learn to use a **for loop** in a shell script and on the command line, both demonstrating how looping logic works.

To demonstrate looping logic in a shell script:

1. Create the file ***ourusers*** with vi, nano, or other editor.
2. Type the following lines into the file:

***for USERS in john ellen tom becky eli jill***

***do***

***echo $USERS***

***done***

1. Save the file and exit the editor.
2. ***Give the file execute permissions***, and run it. Record your output:

To demonstrate entering the same for loop at the command line:

1. At the command line, enter ***for USERS in john ellen tom becky eli jill*** and press Enter.
2. At the > prompt, type ***do*** and press Enter.
3. Type ***echo $USERS*** and press Enter.
4. Type ***done*** and press Enter. What do you see on the screen?
5. Type ***clear*** and press Enter to clear the screen.

## Project 9

In this project, you create a **for loop** and use the brackets wildcard format to loop through each element in a for statement, which consists of simulated book chapters. You first create the files: chap1 through chap4. Next you create a script that displays the contents of each file using the more command.

To create the sample chapter file and use wildcards in a ***for loop***:

1. Type ***cat > chap1*** and press Enter.
2. Type ***This is chapter 1*** and press Enter.
3. Type ***Ctrl+d.*** The file chap1 is created.
4. Type ***cat > chap2*** and press Enter.
5. Type ***This is chapter 2*** and press Enter.
6. Type ***Ctrl+d.*** The file chap2 is created.
7. Type ***cat > chap3*** and press Enter.
8. Type ***This is chapter 3*** and press Enter.
9. Type ***Ctrl+d.*** The file chap3 is created.
10. Type ***cat > chap4*** and press Enter.
11. Type ***This is chapter 4*** and press Enter.
12. Type ***Ctrl+d.*** The file chap4 is created.
13. Use the vi, nano, or other editor to create the shell script, chapters.

The script should have these lines:

**for file in chap[1234];**

**do more $file**

**done**

1. Save the file and exit the editor.
2. Give the file execute permission, and test it. Record your output:

## Project 10

The ***while statement*** is another example of looping logic in addition to the ***for statement***. In this project, you first create a shell program that contains a basic ***while statement***. Next, you create a shell program for an onscreen data input form to store name and address information in a flat data file.

To use a basic **while statement** in a shell script:

1. use vi, nano, or other editor to create a shell script called **colors**.
2. Enter the following lines of code:

**echo -n "Try to guess my favorite**

**color:" read guess**

**while [ "$guess" != "red" ] ; do**

**echo "No, not that one. Try again."; read**

**guess done**

1. Save the file and exit the editor.
2. Give the file **execute permission**, and test it. Record your output:
3. Type ***clear*** and press Enter to clear the screen.

## Project 11

Case logic is often used when many choices are given through a program or when many responses can be made on the basis of one choice. In this project, you create a shell script that employs case logic to respond to your favorite color.

To demonstrate case logic:

1. Use vi, nano, or other editor to create the **manycolors** shell scrip.
2. Type these lines into the file:

**echo -n “Enter your favorite color: ”;**

**read color**

**case “$color” in**

**“blue”) echo “As in My Blue Heaven.”;;**

**“yellow”) echo “As in the Yellow Sunset.”;;**

**“red”) echo “As in Red Rover, Red Rover.”;;**

**“orange”) echo “As autumn has shades of Orange.”;;**

**\*) echo “Sorry, I do not know that color.”;;**

**esac**

1. Save the file and exit the editor.
2. Give the file **execute permission**, and test it. Record your output:

## Project 12

The **tput** command enables you to initialize the screen and position the cursor and text in an appealing way. This project introduces you to **tput.** First, you enter the command directly from the command line. Next, you create a sample script and menu to understand more about command’s capabilities.

To use tput directly from the command line:

1. Type the following command line sequence, and press Enter:

**tput clear;**

**tput cup 10 15;**

**echo “Hello”;**

**tput cup 20 0;**

in the results of this command sequence, the screen clears; the cursor is positioned at row 10, column 15, on the screen; the word “Hello” is printed; and the prompt’s position is row 20, column 0.

To create a sample input menu in a shell script:

1. Use vi, nano, or other editor to create a screen-management script, **scrmanage**, containing the following lines:

**#!/bin/bash**

**tput cup $1 $2 #place cursor on row and column**

**tput clear #clear the screen**

**#set bold text mode**

**tput smso #start stand-out mode (bold text)**

**tput cup 10 2; echo "Type Last Name:" #bold caption**

**tput cup 12 20; echo "Type First Name:" #bold caption**

**#turn off bold text mode**

**tput rmso #reset stand-out mode (turn off bold)**

**tput cup 10 41; read lastname #enter last name**

**tput cup 12 41; read firstname #enter first name**

1. Save the file and exit the editor.
2. Give the file ***execute* permission**, and then test it. Record your output:

## Project 13

In this project, you first compare the use of the **sh –v** and **sh –x** in terms of the output to the screen. Next, you practice debugging a shell script using **sh –v**.

To compare the results of the sh –v and sh –x options to debug a script:

1. Type ***sh –v*** colors, and press Enter.
2. Type ***green*** and press Enter.
3. Type ***red*** and press Enter. Notice that the command lines are printed.
4. Type ***sh –x colors*** and press Enter.
5. Type ***green*** and press Enter.
6. Type ***red*** and press Enter. Now, the command lines and arguments are displayed with a plus in front of them. Record your output using screenshot or snipping tool.

To practice debugging a shell script:

1. Use vi, nano, or other editor to open the **colors** script for editing.
2. Go to the third line and ***delete the closing (right) bracket (]) after “red”*** and then exit, saving your change.
3. Type ***sh –v*** colors and press Enter.
4. Type ***green*** and press Enter. In the final line of output, you will see a message that shows the closing bracket is missing on line 3 of the colors script. Write down the message:
5. Use the vi, nano, or other editor to open the colors script and put the missing closing bracket back in.
6. Delete the *echo* command on the fourth line of the colors script. Close the editor and save your work.
7. ***Type sh –x*** colors and press Enter.
8. Type **green** and press Enter. Notice in the message that a command is missing on line 4. Write down the message:
9. Type ***red*** and press Enter to exit the script, or press Ctrl+Z to exit.
10. Open the colors script using the vi, nano, or other editor, retype the echo command on line 4, and close and save your work.

## Project 14

In this project you learn how to create an ***alias***.

To create an alias:

1. To create an alias called *ll* for the *ls* command, type ***alias ll = “ls -l”***, and press Enter. Now when you use the new ll alias, the ls –l command executes automatically. Test the alias by ***typing ll*** and pressing Enter. Record your observation:

## Project 15

Include your experiences and answers to all the underlying parts in your report.

I spent hours and minutes to finish this hands-on lab.

I have (percent) finish this lab. I expect a grade of (A, B, C, D or F) on this lab.

This lab helped me understand commands, variables, and shell scripting in the Linux Operating System and environment. Choose a number to indicate how much this lab helped you.

1 2 3 4 5

(less helpful) (more helpful)

\*Submit your script to the Assignment Portal