CSC 209 UNIX Tools

|  |  |  |
| --- | --- | --- |
|  | **Assignment 9** |  |

**Objectives: Shell Script Programming**

**Note**: **Read** the entire assignment carefully and carry out the following tasks one by one. For some steps, I may provide the required UNIX command. For others, identify the UNIX commands you need. **UNIX is case sensitive**. Create folders and files exactly as indicated below, paying special attention to case. Complete the report as indicated below and submit the same. I do not want to see copy-and-paste of your computer output in your report. By requiring you to write I am hoping that you will remember the UNIX commands used and the results observed.

1. Login to holly2 server using your Webmail Net-ID and password.

Take a note of how the command prompt placed by the shell appears and report below:

Shell prompt: [71] [etay1@courses2016:~/csc209]$

Type an appropriate command to print the working directory.

Command to print the working directory: pwd

Your login directory: /home/etay1

Use an appropriate command to switch to the **csc209** folder which you created in Assignment # 1.

Command to switch from login directory to **csc209**: [75] [etay1@courses2016:~/csc209]$

1. Complete the following steps:

Determine the current working directory.

Command to print the working directory: pwd

Your current working directory: /home/etay1/csc209

You should currently be located in **csc209**. Create a new folder named **asgn09** under **csc209**. Set the access permissions for **asgn09** as **rwx --- ---** by typing a command:

**chmod 700 asgn09**

Command to create the **asgn09** folder: mkdir asgn09

Command to set the access permissions: chmod 700

Move to **asgn09** and determine the current working directory.

Command to switch from **csc209** directory to **asgn09**: cd asgn09

Command to print the working directory: pwd

Your current working directory: /home/etay1/csc209/asgn09

Shell prompt: [80] [etay1@courses2016:~/csc209/asgn09]$

**From now on, unless explicitly permitted, you should not move away from this asgn09 directory.**

**If you happen to log out in the middle of the exercise, be sure to use appropriate command(s)**

**and move to asgn09 before you continue.**

2. (6 points) In this exercise, we will understand how to create, test and debug a Bash script program code. The purpose of the code is to print out a suitable greeting, based on the time of day. Specifically, if **hour** is not provided, the program determines the current time and prints an appropriate greeting. The greeting must be “Good Morning!” till 12 Noon. After that, i.e., at 12 Noon and after, the greeting must be “Good Afternoon!” till 6 PM. Beyond that, i.e., at 6 PM and after, the greeting must be “Good Evening!”

Use **pico** editor to create a file named **greet**

Command to create the file: pico greet

Enter “The time of day greeting” code given below. Replace the author name with your name.

The code contains some obvious errors. Do not fix them now. We will fix them later.

While typing the code, ensure that there are no spaces around the “equal to” symbol.

Save the file as **greet**

Control sequence to save the file: Control O to overwrite, then enter to confirm name of file “greet” then control x to exit

**greet: Time of day greeting code**

#!/bin/bash

#

# Time of day greeting (Author: Jane Doe, Date: 11/04/2014)

#

# usage: ./greet [hour between 0-23]

#

if [[ $# -eq 0 ]]

then

hour='date +%H' # command output substitution

else

hour=$1

fi

#

if [[ $hour < 12 ]]

then

echo "Good Morning!"

else

if [[ $hour -lt 18 ]]

then

echo "Good Afternoon!"

else

echo "Good Evening!"

fi

fi

#

exit 0

#

Make the file executable for you.

Command: chmod 700 greet

The script program is invoked as follows: type the path name of the command followed by an hour, e.g., type: **./greet 1**

Note that the **hour** is optional. If not provided, the system should determine time of the day and display the appropriate greeting.

Test the program by typing **./greet 1** and write the output.

Good Morning!

Test the program systematically for several values of **hour** for the following boundary cases: 0, 11, 12, 13, 17, 18, 19, and 23. Write the output.

|  |  |
| --- | --- |
| **hour** | **Greeting** |
| 0 | Good Morning! |
| 11 | Good Morning! |
| 12 | Good Afternoon! |
| 13 | Good Afternoon! |
| 17 | Good Afternoon! |
| 18 | Good Evening! |
| 19 | Good Evening! |
| 23 | Good Evening! |

Did you find any errors? Are you ready to certify that the program is correct?

Errors: no. Sure its correct: No

Not quite. We have to test at least one more case. If no command line argument is given, the program is supposed to check the time on its own and determine the hour of the day.

Test the program with no command line argument, i.e., simply type **./greet**

Output:

./greet: line 18: [[: date +%H: syntax error: operand expected (error token is "%H")

Good Evening!

Obviously there is an error in the program. Display the program code with line numbers.

Command to display file **greet**: cat -n greet

In Line 9, we are using the date command to determine the hour. Let us see if there is any problem there.

Type the command **date** and observe the output.

Output: Wed Nov 30 10:19:55 EST 2022

Type **date +%H** to see if it obtains the hour correctly.

Output: 10

All of this should work correctly. So, what is the problem? Observe the comment that says “command output substitution”. In other words, we want the command **date +%H** to be executed and the output of that command to be substituted in the right hand side of the assignment to the shell variable hour. Given that, fix that statement. If you do not know how to do it, check the textbook on **p.61.**

Corrected form of the line: hour=`date +%H` 🡪 couldn’t find the solution on page 61

Will ask during office hours

Edit the file **greet** and fix the code. Test the program again with no command line argument.

Command: ./greet

./

Output: ./greet: line 9: +%H: command not found

Good Morning!

no

Are you ready to certify that the program as correct?

no

The program is supposed to accept only integers in the range 0-23 and provide an appropriate greeting. It obviously will not work for an hour outside the range. Let us test one random case, say 8 AM in the morning. We expect the program to output “Good Morning”.

Type **./greet 8** and see if the output is correct.

Output: Good Afternoon!

Is this correct? no

Type **./greet 08** and see if the output is any different.

Output: Good Morning!

Is this correct? yes

What is going on?

Im guessing 8 is being read as a string and returning 8 > 12 as false

In order to debug the code in verbose mode, introduce options **–xv** in the first line of code for bash. Edit the program in the file **greet** and change the first line to **#!/bin/bash -xv**

Now run the program for the case of 8 AM, i.e., type **./greet 8**

The debugger shows the various lines of code that were encountered during execution and what the values of the variables were.

In the output, lines with “**+**” are debugger output. The first such line says **+ [[ 1 -eq 0 ]]**

That is, the first thing bash did was to do the comparison **1 –eq 0**

Because we provided one command line argument, the value of **$#** is **1**. That is what we are seeing in the above line. After this line, it encountered the line where the variable **hour** is assigned the value **8**, which is what we would expect.

The next line encountered is where the value of the shell variable **hour** is compared against 12. We see **+ [[ 8 < 12 ]]** as expected. At this stage, we expect the code to take the then clause of the **if** statement and echo the message “Good Morning”. But apparently that is not what the shell did as we see the next debugger output line as **+ [[ 8 -lt 18 ]]**

Why did the shell branch to the else clause?

For a shell, all variables hold string values. They are interpreted numerically when needed.

In the comparison **8 < 12**, the shell really checks if the string **"8"** is alphabetically before the string **"12"**  The answer is false, and hence the shell took the else clause at this branch. What we want is a numerical comparison. How to fix the code? Check **9. Bash Scripts, p. 4.**

Corrected form of the line: [[ $hour -lt 12 ]]

Edit the file **greet** and fix the code. Test the program again: **./greet 8**

Command: ./greet 8

Output: Good Morning!

At this stage, the program works correctly.

Edit the file **greet** and remove the options placed for debugging.

Note that the program does not check that the value for **hour** is between 0-23. Introduce an **if** statement before **if [[ $hour –lt 12 ]]** line. The **if** statement should echo the correct usage of the program and exit with value 1, if the hour is less than 0 or greater than 23. Test the program one more time to ensure that it works for at least one value below 0 (say -1) and one value above 23 (say 24). The five lines of code needed are:

if [[ $hour -lt 0 ]] || [[ $hour -gt 23 ]]

then

echo "Enter hour between 0 and 23"

exit 1

else

couldn’t test but pseudo coded it

The UNIX command **script** is used to record a terminal session in a file and print the file later. It is often convenient to name the file with your last name so it can be identified easily. Then type a series of commands for which the output must be captured. It is very important not to use any control character as every character entered is captured in the file as is, and control characters mess up the display. **So, do not backspace or use arrow keys or open an editor in the script**. Type your commands very carefully. If there are mistakes, hit return and retype. Finally, to close the script process, use the **exit** command.

For example, for a user with the last name **smith**, the session may appear as follows:

**SHELL='/bin/bash' # bash used for script**

**script smith # start recording**

**. . . series of commands**

**exit # stop recording**

At this stage, you should be positioned in the **asgn09** directory. Ensure that you are located in **asgn09** by inspecting the prompt. If not, move in to this directory.

Start the script process with appropriate file name for your case, and complete the following tasks.

Type an **echo** command to display your name.

Type a command to display the file **greet** with line numbers.

Test the program for the following cases: -1, 0, 1, 8, 11, 12, 13, 17, 18, 19, 23, 24.

Test the program for the case with no command line argument.

Stop the script process.

Print the script file that you just created in the last step. To print a file, follow the same approach you used to print a script file for Assignment 8. **The below is another way it used to work.**

* + Open the **WinSCP** program under Internet Utilities
  + In the window that opens up, fill in hostname (holly.acs.brockport.edu), your user name and password, and login. If there are error messages indicating Server’s key was not found …”, continue connecting by choosing “Yes”.
  + A new window will open up showing files on the local machine on the left pane and the files on the remote machine on the right pane.
  + In the right pane, navigate to the directory (folder) of interest to see the script file.
  + Drag and drop the script file from the right pane to the desktop. This will copy the script file on the desktop.
  + Press F10 to terminate the FTP session and close the connection.
  + Use Notepad to open the file on the Desktop. Print the file and attach.

3. (7 points) Write a program that prints terms in the Fibonacci sequence: 1 1 2 3 5 8 ... The first two terms in the sequence are defined as 1 and 1. Each subsequent term is the sum of the previous two terms. Display a message indicating the purpose of the program and then prompt the user for a non-negative integer that determines the number of terms in the sequence to be printed. That is, if the user were to input 5, the first five terms are printed. Test the program for the following cases: -5, -1, 0, 1, 2, 5, 16, and 30. **Be sure** that your output matches the sample output in terms of prompts and messages. See sample output below:

[51] [jdoe1@holly2:~/csc209/asgn09]$ ./fibonacci

This program prints terms in the Fibonacci sequence

Number of terms to be printed (enter a non-negative integer): -5

Sorry, you must enter a non-negative integer!

[52] [jdoe1@holly2:~/csc209/asgn09]$ ./fibonacci

This program prints terms in the Fibonacci sequence

Number of terms to be printed (enter a non-negative integer): 2

1

1

Produce a script command output file that shows the source program code and executions for all of the above cases. Print the file and attach.

4. (7 points) Write a program whose correct usage is as follows:

**add integer1 ...**

That is, the command name is followed by one or more integers (positive or negative). The purpose of the program is to add the integers in the command line and print an appropriate output. See sample output below:

[51] [jdoe1@holly2:~/csc209/asgn09]$ ./add

usage: add integer1 ...

[52] [jdoe1@holly2:~/csc209/asgn09]$ ./add 10

The sum total of the given integers is 10

[53] [jdoe1@holly2:~/csc209/asgn09]$ ./add 10 0 -20

The sum total of the given integers is -10

[54] [jdoe1@holly2:~/csc209/asgn09]$ ./add 0 1 2 3 4 5 6 7 8 9 10

The sum total of the given integers is 55

The program code for add can be structured as follows:

* Check for command line arguments; if any. If the number of command line arguments is zero, report correct usage and exit (See [51] above)
* Write a for loop that allows you to inspect each integer included in the command line argument array
* Add it to the running total maintained
* After the end of the loop, report the total (See [52], [53], and [54] above)

Produce a script command output file that shows the source program code and the executions for all of the above cases. Print the file and attach.