**04 – Shell (e.g. bash) Scripting**

**Activities**

COMP190 – Tools and Techniques for Software Development

Dickinson College

Fall 2020

Prof. Grant Braught

In A03 you gained experience with many of the Unix tools and filters. The Unix design philosophy is that these small, focused, purpose-built programs each do one thing well. They are then combined with piping and redirection to accomplish more complex tasks. In A03 you did a variety of tasks from the command line combining tools with pipes and redirection. Frequently, you may want to perform the same operation repeatedly, or on many files, or only on some of a larger group of files. In these situations, combining the Unix tools with more traditional programming constructs such as user input, variables, conditionals (if) and loops (for, while) is a powerful technique. To support this, most shells in addition to providing a command line interface (CLI) also act as interpreters for programs written in their own scripting languages. These programs are called shell scripts. In this set of activities, you will gain experience writing shell scripts for the bash shell (i.e. bash scripts).

As with A04, you will be following along with another of Ryan’s Tutorials, this one specifically on Bash Scripting. Each of the sections below points you to the relevant tutorial sections and I have included headers within each section to connect them to the sections of the tutorial. Use these to clue you in to the sections of the tutorial where the material relevant to the question is. That doesn’t mean the other sections are not important or can be skipped. You should work each tutorial from the start to get a full picture of how bash scripting works. As with A02 and A03 I highly recommend that you try each of the things in the tutorial as you read through it, play around and try some other things and then answer the questions that I pose.

**Permissions!**

Before we get into writing and running (i.e. executing) shell scripts it will be useful to understand a little about how Unix/Linux controls which actions that which users are permitted to perform on a file or directory.

Open the “Permissions!” section of the tutorial:

* <https://ryanstutorials.net/linuxtutorial/permissions.php>

*So what are they?*

1. What are the three different permissions that exist for files and directories?

2. What are the three different sets of users that have their own permissions for files and directories?

*View permissions:*

3. Use a text editor to create a new file named myfile.txt in your home directory and type a few lines of text into the file:

a. Who is the owner of the myfile.txt file?

b. What permissions does that owner have by default?

c. What permissions do other users in the owner’s group have by default?

d. What permissions does everyone else have by default?

4. The command ./myfile.txt will try to execute the myfile.txt file in the current working directory as a program. What happens when you try to execute myfile.txt? Why?

Note: The ./ is required here. Recall that it is a relative path to the current working directory. Here it just tells the shell to look in the current working directory for the file you are trying to execute.

5. Recall that many of the Unix commands are just little programs that you execute when you type their name on the command line. Answer the following questions for the program ls. Hint: Use which to find the full path to the ls file.

a. Who is the owner of the ls file?

b. What permissions does the owner of the ls file have?

c. You are in the everyone else category for this file. What permissions do you have for the ls file?

*Change permissions:*

6. Give a command that grants write permission to your myfile.txt file to everyone.

7. Give a command that removes write permission to your myfile.txt from yourself.

8. If you do not have write permission to myfile.txt what happens if:

a. You try to open, modify and save myfile.txt in a text editor?

b. You try to delete (rm) myfile.txt?

9. Even though myfile.txt is not a program, Unix will let you set the execute permission. Give a command that gives you all of read, write and execute permission to myfile.txt.

10. What happens now if you try to execute myfile.txt? Don’t forget the ./. What happens should make sense because even though you are allowed to try to execute the file, because the execute permission is set, the file does not actually contain a program. However, as we will see, shell scripts are simply text files with commands in them. So, while setting the execute permission on a text file may seem silly now, you’ll see soon that it is what allows us to execute the scripts that we write.

*Permissions for Directories:*

11. What does execute permission mean for a directory?

12. Give a command that will display the permissions on your home directory (e.g. /user/comp190 or whatever you named your user).

13. Will other users be permitted to cd into your home directory and see what files are there? How can you tell?

14. By default all users have access to most directories on a Unix machine. You used this feature a lot in A03 when looking at the files in the /etc directory.

a. Who owns the /etc directory?

b. What permissions do you have to the /etc directory?

15. There are however a few places other than your home directory on Unix systems where you will be able to create and modify files. These locations are usually used to create temporary files that are needed by programs for short periods of time.

a. Who owns the /tmp directory?

b. What permissions do you have to the /tmp directory?

16. A *drop folder* is a directory that you can use to let other users give files to you. Other users should be allowed to place files into a drop folder, but they should not be able to see what is in the folder. Give a list of commands that will create a new drop folder named DropFolder in your home directory. Note: This is a little trickier than it seems at first. Play with this using your own permissions. You should be able to use touch to create a file in the DropFolder, but then you should not be able to use ls to see it or any of the other files that might be there,

**Bash Scripting!**

Now that you know a little about file permissions, we’ll move onto bash scripting. As mentioned earlier and explained in the video, a bash script is just a plain old text file that happens to contain commands that the bash shell program can interpret. This part of the activity introduces you to the basics of bash shell scripts.

Open the “Bash Scripting!” section of the tutorial:

* <https://ryanstutorials.net/linuxtutorial/scripting.php>

*So what are they?*

17. What types of tasks are shell scripts most useful for?

18. Briefly describe something you have had to do frequently or repeatedly on the computer where being able to bundle a bunch of steps into a single command would have made it a lot more convenient or less error prone?

*A Simple Example:*

19. Enter the text of the example script (lines 2-7) into a text file named myscript.sh in your home directory.

20. Try to run the script using the command ./myscript.sh

a. What happens? Why?

b. What command can you use to fix this error? Be sure that the command ./myscript.sh works correctly before going on.

21. Modify your script file so that instead of displaying the contents of the current directory it displays the names of all of the users on the system in alphabetical order. Hint: Recall the following:

* Any command that you can type at the command line can appear in a script.
* The /etc/passwd file contains information about the users on the system.
* Unix/Linux has filters for cut and sort that can be combined using a pipe.

Give a screen shot of your modified script in an editor as your answer to this question.

*Important Points:*

22. What is a shabang (or shebang)? What is its purpose?

23. In the Windows OS file extensions are used to indicate the content of a file (e.g. a .exe extension indicates an executable file, .bat or .ps1 indicates a Windows shell script). Unix does not use file extensions in this way. Thus, scripts are not required to be named with an extension of .sh. But they can be. What advantage might there be to consistently naming shell scripts with a .sh extension?

24. What character is used to start a comment in a bash shell script?

25. What is the purpose of the PATH variable in a Unix system?

26. What directories are in the PATH variable on your Linux Lite system? What command did you use to find this?

* Note: People will often say things like /bin is “on the path” or ./ “is not on the path” to mean that a particular directory is or is not included in the PATH variable.

27. You have actually been making use of the PATH all along without really knowing it. This question sheds a little light on that. Recall that in A03 you entered the command xeyes. This command ran the program named xeyes, which popped up some cartoon eyeballs on the screen that would follow your mouse pointer.

a. What is the absolute path to the xeyes program? What command did you use to find this?

b. What is the absolute path to the directory containing the xeyes program? Hint: This is just part of your answer to a.

c. Is the directory that contains xeyes on the path?

d. Imagine that the directory from b was not on the path. What command would you need to enter in order to run the xeyes program?

*Variables:*

28. Considering the morevariables.sh example in the tutorial, what output would be generated if the following command is entered:

./morevariables.sh Ashir Becky Diego Xia

29. Create the morevariables.sh script in your home directory. Modify it so that instead of displaying all of the command line arguments on a single line it will display the first five of them as follows:

1. Ashir

2. Becky

3. Diego

4. Xia

5.

Note that if there are fewer than 5 command line arguments the number will be printed but the rest of the line will be blank.

Give a screen shot of your modified script in an editor as your answer to this question.

30. The “back ticks” are one of the most useful bits of bash script syntax for automating tedious and repetitive tasks. They provide the ability to put the result of a command into a variable and to use that variable later in other commands. This process of placing the output of a command into a variable is also called *Command Substitution*.

Using the back ticks, write a little script named filecount.sh that displays the number of files (including hidden files) that exist in the working directory. This script will be very similar to the back ticks example in the tutorial. Give a screen shot of your filecount.sh script in an editor as your answer to this question.

*If Statements:*

Like in other programming languages the if statement in a shell script allows the program to make decisions. This section of the tutorial gives an example that contains several examples of if statements. In studying that example, it may help to know that the general forms for an if statement in a bash script are:

if [ *<test>* ]

then

*<code>*

fi

if [ *<test>* ]

then

*<code>*

else

*<code>*

fi

The syntax of these statements is very particular. The space after the [ and before the ] must be present. The then, else and fi must be on their own lines. So be sure to pay close attention when writing if statements in your bash scripts.

The sample programs in this section show a few of the different types of <test> that can appear in if statements. There are many more. A complete list can be found using man test. A partial list of the most useful of these includes the following:

A screenshot of a cell phone

Description automatically generated

31. Write a little script named birthday.sh that accepts one command line parameter that will be either -y or -n. If the parameter is -y then the script displays:

You say it's your birthday  
It's my birthday too, yeah

*(If you don’t get the reference:* [*https://www.youtube.com/watch?v=MjF1bG5LUcs*](https://www.youtube.com/watch?v=MjF1bG5LUcs)*).*

If the parameter is -n then the script displays:

Oh well, maybe tomorrow.

If the parameter is neither -y nor -n then an error message should be displayed.

Give a screen shot of your birthday.sh script in an editor as your answer to this question.

**Writing a Useful Script:**

Okay, enough playing around… let’s do something that can actually demonstrate the usefulness of scripts. Imagine a large collection of files, 1000’s or even 10’s of thousands of them, and you have been asked to put a copyright notice (e.g. (c) 2020) at the bottom of each file. Now, you could sit and open every one of them, one-by-one, and paste in the line. Or… you could create a script for this!

The best way to write a script is to build it bit by bit, checking at each step that it is working correctly before adding more code. You will often find it helpful to test and debug individual commands on the command line and then add them to your script. Then you can test them within your script by adding echo commands, which you can then remove later. This section will walk you through this process of writing and testing a script.

31. We’ll be creating a bunch of files and scripts so making a sub-directory in which to work rather than working in your home directory will keep things tidier. Create a new directory copyrightex and change into it. What commands did you use?

32. In order to test our script for adding copyrights we’ll need a bunch of files that we can add the copyright notice to. We could create them all one-by-one by hand… that’s fine if there are 1 or even 5, but not if we want hundreds or thousands of them… So let’s create a script!

a. Open a text editor and enter or copy and paste the following script:

﻿ ﻿#!/bin/bash

rm File\*.txt # Delete the old test files.

# Create 100 new test files.

for NUM in {1..100}

do

echo "This is file "$NUM > File$NUM.txt

done

Notice that this script uses a for loop. That was not in the tutorials but it shouldn’t be too hard to figure out what this one done.

b. Save this script as maketestfiles.sh in your copyrightex directory.

c. Change the permissions on maketestfiles.sh so that you can execute the script. What command did you use?

d. Run the maketestfiles.sh script. What command did you use?

e. What text appears in the File1.txt file?

33. Let’s get started by using a text editor to creat a new script named addcopyright.sh in your copyrightex directory. Be sure to add the shebang at the top and to change the permissions on the file.

34. The first step will be to get a list of all of the files to which the copyright should be added.

a. Experiment on the command line to develop a command that lists only files with a .txt extension. What command did you use?

b. Add a line to the script that puts this list of files into a variable. Hint: Use back ticks.

c. Add a line to the script that displays the contents of your variable. Hint: use echo and don’t forget the $.

d. Run the script and ensure that it prints out the names of all of the .txt files. Notice that the names are all run together. This is a quirk of the command substitution, it removes all of the newlines in the output. Luckily this won’t affect us. When that works paste a screen shot of your script in the editor below.

35. Next we will need to go through each of those files one-by-one so that we can paste in the copyright notice.

a. Comment out or remove the echo command from question 34.

b. If the variable you used in question 34 was named files, then the following for loop would go through and display the name of every file.

for FILE in $files

do

echo $FILE

done

c. Add this code to your script, being sure to adapt it for the variable name that you used for the list of files.

d. Run the script and ensure that it now prints out the names of all of the .txt files one per line. When that works paste a screen shot of your script in the editor below.

36. Now we need to figure out how to paste in the copyright notice at the end of each file.

a. Comment out or remove the echo command from question 35.

b. Experiment on the command line to find a command that will append the string

(c) 2020 to the end of File1.txt. Be sure that the notice appears after any text that was already in the file. Hint: Use redirection. You can then use cat to display the file and check that the notice was added.

c. Put the command you found in part b into the for loop and adapt it to use the $FILE variable so that the copyright is redirected into each successive file.

d. Run the script and ensure that each of the .txt files now contains the copyright notice. If things don’t work and the test files get messed up, just run the maketestfiles.sh script again. When this part works paste a screen shot of your script in the editor below.

**Extra Challenges:**

The following are optional but will significantly enhance the functionality of the script:

37. Woooo hoooo it works… but wait…

a. Run your script again and check the contents of one of the files. What happened?

b. Add an if statement in the for loop that does not append the notice if one already exists. Hint: Use egrep and experiment on the command line with some files that have the notice and some that don’t until you can tell the two apart. Then use the back ticks and an if to put it into your script.

38. Now what about next year? And the year after? We would have to update this script every year so that it puts the proper notice in. Wouldn’t it be much nicer if it just knew what year it was? Modify the script so that it automatically inserts the correct year. Hint: Use the date and cut commands.

**Learning More:**

There is a lot more to bash scripting. We’ve just scratched the surface. Ryan has a full tutorial on bash scripting that dives into it in much more detail if you are interested:

* <https://ryanstutorials.net/bash-scripting-tutorial/>