UNIX LAB

CSC 2500

Department of Computer Science
Tennessee Tech University

LAB 5: More Bash Scripting with if, Variables, Command Substitution, and for

Objective:

- Shell variables
- Simple `if` decision statements
- The `read` command
- File redirection
- Command substitution
- Looping with `for`

Home Work

First create `classlist.txt file` with the following entries:

```
browndon

davidso2

desjardi

goldberg

kinneysu

kruegerp

leibljon

maurymor

mentzers

mitchell

naglejea

nixonmar

padgetta

rickersg
```

```
rosechri

starkmol

timmonsl

whittenb
```

For this lab, you will write a shell script that uses some things that we have already practiced, along with some new things. **You will write a script called `lab5.sh` that takes two parameters as arguments.** The first parameter is a file that contains a list of user names, one per line. The usernames contain only alphabetic characters and no white space. The second parameter is a user name.

Your script should:

1. Determine that the correct number of arguments was received, or print a usage message and exit if not.
2. Determine that the file with the name given as the first parameter exists, or print an error message and exit if not.
3. Determine that the username is already in the file, and then either

    - Print a message stating that the name already existed,

      or,

    - If the name was not in the file, then add the name to the end of the file.

Following is an example run of the script.

```
$ # First show what running with the wrong parameters does

$ ./lab5.sh

usage: ./lab5.sh file user

Where file is the file to search

  and user is the user to find

$ ./lab5.sh classlist.txt

usage: ./lab5.sh file user

Where file is the file to search
```

```
  and user is the user to find
$
$ # Next show that the script finds an existing user name
$ ./lab5.sh classlist.txt  mentzers
mentzers found on line: 8
$
$ # Finally show that the script can add a missing user name
$ ./lab5.sh classlist.txt  johndoe
johndoe not found
Do you wish to add the name to the end of the list (Y/N)?
y
$
$ # And just to show that it worked...
$ cat classlist.txt
browndon
davidso2
desjardi
goldberg
kinneysu
kruegerp
leibljon
maurymor
mentzers
mitchell
naglejea
nixonmar
padgetta
rickersg
```

```
rosechri

starkmol

timmonsl

whittenb

johndoe

$
```

Details

Step 1

To accomplish step 1 above, you must be able to determine the number of parameters passed on the command line when the script is invoked. Bash provides you with a special variable called `$#` that holds the number of parameters passed. So, all you must do is put the check in an if statement. The syntax will look like the following:

```
if [ ! $# -eq 2 ]; then
 # you figure this out
fi
```

Remember the spaces after the [and before the]. Notice that the `if` statement uses the `-eq` option instead of the equal sign. Use the equal sign when you want to compare strings, and use `-eq` when you want to compare numbers.

In the example run above, the script prints out the script name in the usage statement (`./lab5.sh`). Bash has a variable that holds the name of the current script. The variable is `$0`. Using this variable is handy in your echo statement for printing the proper usage of the script. This variable guarantees that even if the command is renamed, the appropriate usage statement is still printed.

Step 2

To determine if the file exists, you will need another `if` statement. The `-e` option to bracket expressions returns true if a given file exists and false otherwise. Also the `!` stands for negation. How do you get the name of the file? Remember that the file name was passed as a parameter. Bash stores all parameters in special variables. The first parameter passed is stored in variable

`$1`, the second in `$2`, and so on. So, putting all this information together, the proper expression between the brackets of the if statement would look like the following:

```
! -e $1
```

Now you should be able to write the rest of the if statement yourself.

Step 3

To determine whether the user name is in the file, you will iterate over each line in the file and compare the line to the second parameter passed on the command line that is in variable $2. You will use a Bash `for` loop. Bash `for` loops iterate over a list of items. Bash treats any white-space separated string as a list of items. In this case, the file of user names is one user name per line. In other words, it is a white-space separated list of user names.

However, how do you get the list of user names from the file to the program script? Bash lets you capture the output of any command by surrounding the command with back ticks (i.e. the ` that is above the tab key on your keyboard). So, if you can list the file to the screen, you can capture that output using back ticks. Remember that you can list any file to the screen using the `cat` command. So, your `for` loop should look like the following:

```
for line in `cat $1`; do
 # determine if $line is equal to $2 here
done
```

The variable `line` is the loop variable (you can call it anything you like). So, the above code will run `cat` on the file whose name was passed as the first parameter and then capture the output. The `for` loop will then iterate over each line in the file, and at each iteration, set the variable `line` to the contents of the next line.

Now you must figure out what goes in the body of the loop. You will need an `if` statement to determine if the `line` is equal to the user name and, if it is, print that the user name was found and the line number, and then exit. To exit, just use the exit statement: `exit 0`. How do you know the line number? Count the lines yourself in the code. Before the `for` loop you can set a variable called `count` to 0. You should know how to set variables because you have done so in previous labs. Then, inside the loop, increment the `count` by one. You can do math in a shell script by using the `expr` command. So, the following will increment `count` by 1:

```
count=`expr $count + 1`
```

Make sure that you put a space before and after the +, or the shell will think that $count+1 is the whole variable name.

Finally, after the `for` loop, you can add the line to the file. Because you know that the only way to get passed the `for` loop is if the user name was not found (otherwise, the `exit` statement would have stopped the script). You should know how to append a string to a file from the previous lab (use `echo` and >> to append the user name). However, don't forget to ask the user if they want to add the user name to the file. Only add the user name if the response is Y or y. For this, you need to convert a string from lower case to upper case. You can use the following trick in your script so that you do not have to check for either Y or y. Given that you read the user response into a variable called answer, you can convert its contents to upper case like so:

```
answer=`echo $answer | tr [a-z] [A-Z]`
```

The `tr` command takes two parameters. The first is a sequence of characters to be converted. The second is a sequence of characters to which you want the previous sequence converted. `tr` reads from `stdin`, finds each occurrence of characters described by the first parameter, turns those characters into the corresponding characters described by the second parameter, and then prints the results.

Submission

Submission Site: iLearn (a Dropbox folder named “Lab 05”)

Submission Content: Submit your **lab5.sh**.