Lab9. Shell Scripts (I) – Introduction

Objectives

After completing this lab, the student should be able to:

- Create and execute simple shell scripts.
- Use positional parameters and shifting to pass command line arguments to scripts.

Introduction to Shell Scripts

One of the most powerful tools in Linux is the ability to group several commands into scripts to automate manual tasks. Scripts are also used as configuration and setup files in different areas of the Linux File System. Let us start by writing and executing our first simple script.
To create and setup a script you need to do the following:

1- Using the `vi` editor open a new file (`myfirst`) and write your script as follows:

```
echo this is my first Linux script
echo I like it
echo bye
```

2- You now need to add the **x** (*execute*) permission to your script to run it. This is done only the first time you create your script. If you try to run your script and get the error "**permission denied**" then the reason would most likely be that you did not do this step. To add the **x** permission run the following command:

```
chmod +x myfirst
```

3- To make the script globally accessible in your Ubuntu environment, we need to modify the PATH variable that will let the shell find your script as follows:

   a. Let us assume that your script is located under directory (`~/Desktop/comp311/lab9`), then you need to add the following line to the end of your (.bashrc) file located into your home directory:

   ```
   PATH=$PATH:$HOME/Desktop/comp311/lab9
   ```

   b. Note that `$HOME` variable will point to your home directory, or you can use (~). For the changes to take affect permanently so that every new shell will see the modification to the `PATH` variable, we need to execute the following line:

   ```
   . .bashrc
   ```

   Without the last step, if you try to close the shell, your modifications will be gone, and your `PATH` environment variable will be reset to default.

4- Now you are ready to run your first script by typing its name on the command line:

```
myfirst
```

If this does not work, then use: `./myfirst`

What was the result of running the script? ________________________________

As you can see the echo command is used to print output messages from your script to the output device (currently set to screen). We can redirect the output to a file.

5- Let us now create another script (`greetings`) that has both input and output as follows:

```
echo What is your name
read name
echo hello $name
```

Then add the execute permission using:

```
chmod +x greetings
```

Finally, execute your scripts using:

```
greetings
```

What do you think is the purpose of the read command? ___________________________

Notice that when you read a value in your variable you just put the name of the variable (`name`) while when you print the value, you need to put the `$` sign at the beginning (`$name`). By default, *shell scripts treat all variables as strings*.

6- Now let us write our own script (`delete`) for deleting a file:

```
echo Enter file name:
read filename
rm $filename
echo File $filename has been deleted
```

Now run your script. Did you forget to add the (x) permission? ______________________

7- Now it is your turn to write a complete script and run it. Write a script called (`copy`) that asks the user to enter a source filename and a destination filename and then copies the source to the destination. Your script should work as follows:

```
Enter source file name:
one
Enter destination file name:
two
File one is copied to file two
```

Your script:

```

```

Try to run your copy script. Did it work? ____

8- You probably realize that programs like `delete` and `copy` would behave more like similar commands if they *took their input from the command line* instead of asking the user to enter those after running the program. To do this we need to use positional parameters. Let us write a simple script (`params`) to understand how those are used:

```
echo $1
echo $3 $2
echo $#
echo $0
echo $5
echo $*
```

Now run the script `params` as follows (use `./params one two 3 four 5 6 bye`):

```
params one two 3 four 5 6 bye
```

What was the output? ____

Now what do you think are the values of each of the following variables:

`$1:` ____

`$3:` ____

`$*:` ____

`$#:` ____

`$0:` ____

9- Now that you understand how positional parameters work, ***rewrite the*** **`delete`** ***scripts above to run as follows:***

```
delete thefile
thefile has been deleted
```

Your script:

```

```

Rewrite the **`copy`** ***script above to run as follows:***

```
copy file1 file2
File file1 has been copied to file2
```

Your Script:

```

```

Now try your new delete and copy scripts? Did they work? ________________________

Notice that since you already had the ***x*** permission on the previous scripts (`delete` and `copy`), you did not have to do that step again to run them.

10- Write a script (`whoisuser`) that takes the login name of a user as a parameter and then uses the `/etc/passwd` file to get and print the full name of that user as follows:

```
whoisuser u1122334
u1122334 = Ahmad Hamdan
```

Answer (*hint*: use variable and command substitution**):**

```

```

Shifting parameters

To shift script command line parameters to the left, we use the `shift` command as follows:

- `shift 2` shifts the command line parameters to the left twice.
- `shift` shifts the command line parameters to the left once by default.

To understand how shift works, let us rewrite and run the `params` script above as follows:

```
echo $1
shift 2
echo $2 $3
echo $#
shift
echo $0
shift 3
echo $1
echo $*
```

Now run the script as follows and notice the effects of shifting:

```
./params one two three 4 5 6 seven 8 9 ten bye
```

Which parameter is not affected by the shift command? ______________________________

Comments

You can add comments to your scripts by using the # sign followed by the comment anywhere in your script. Lines that start with (#) are **interpreted as comments** except in one case where shells have (`#!`) followed by the name of a shell as the first line of a script. In that case that line is interpreted as the **name of the shell to be used for executing that script**. For example, if your script starts with the line: `#!/bin/bash`, then the script is meant to be executed using the `/bin/bash` shell.

- **Check out the system scripts:** `/etc/rc.sysinit` and `/etc/rc.local`
- **What is the first line in those scripts?** ____
- **What is the difference between the first line and the few lines that come after it?**