1. Creating a simple shell script

1.1 General

By now we have looked at enough shell commands to be able to write some small programs at the shell level. A program written in the shell is known as a shell script program.

1.2 A simple shell script program

We will now create a very simple shell script program. The example program is kept deliberately simple so that we can concentrate on the steps involved in editing and executing the program.

Using the editor (any plain text editor) create a file called **easy_text** with this content:

easy_text

```
#! /bin/bash

 echo   "Hello world!"
 echo   "The University of Limerick is situated"
 echo   "down by the banks of the Shannon river,"
 echo   "about three miles from Limerick city."
```

The **easy_text** file is a simple shell script file. It contains four **echo** commands. Now we need to make the file executable. The first line of all the script examples contains **#! /bin/bash** to specify to the system that the **/bin/bash** file is to be used to execute the script file. In other words, we need to tell the system to use the Bash shell for this script program.

To run this shell script program, type:

bash easy_text

This command will execute the **easy_text** program. However, the **bash** command in the command line will start a new copy of the shell program. The new copy of the shell program executes the **easy_text** file and then the shell quits passing control back to the calling shell. Another way of running the script program is to make the shell script file directly executable.

1.3 Make the shell script program executable

Now that you have created the file **easy_text** you can make this file executable. List the file using the following command:

ls -l easy_text

You will get a report something like the following:

-rw-rw-r-- 1 hristo hristo 197 Feb 6 12:55 easy_text

You will see from this line that the file is not executable (i.e. no **x** in the user permission field). To make the file executable you could enter the following commands:

hristo@hristo-lubuntu18:~/EE5012$ chmod u+x easy_text

The **ls -l easy_text** command will now list the file, something like as follows, showing that the file is now executable:

-rwxrw-r-- 1 hristo hristo 197 Feb 6 12:55 easy_text

The file is now executable by the user as we can see by inspecting the above line. To execute the script file **easy_text** you can simply type the file name.
Usually you will need to specify its path (e.g. **./**) here also, as shown:

hristo@hristo-lubuntu18:~/EE5012$ ./easy_text

"Hello world!"
"The University of Limerick is situated"
"down by the banks of the Shannon river,"
"about three miles from Limerick city."

1.4 Using comments

Just like any programming language you can include comments in your script files to make the program more readable. The hash (#) symbol is used to indicate a comment. Add some comments to the **easy_text** script program and run the program again. You will see that the comments do not affect the operation of the program. Please note that the **#! /bin/bash** line is not a comment, since the #! character pair forms a special combination to define which shell is to be used to execute the script.

Example comments in **easy_text**:

easy_text

```
#! /bin/bash
# This is a simple script file.
# It prints some text and then exits.
# Author: Me
# ID: 11111111111111

 echo   "Hello world!"
 echo   "The University of Limerick is situated"        # the premier university!
 echo   "down by the banks of the Shannon river,"       # longest river in Ireland
 echo   "about three miles from Limerick city."
```

2. Practical Exercises

2.1 Test Questions and Exercises

Please state what each one of the following bash shell commands does:

1) ls –l ./ ../ ../ (is it different to ls –l ./ ../../ ?)

2) cat ~/games/playfile | more

3) cat titlefile verse_1 verse_2 verse_3 >> full_poem

4) cp ./* ../

5) rm ../?est

6) chmod o+rwx test_file

7) find . -name "verse*" -print

8) grep "And" verse_2

9) grep "^d" temp_file

10) uniq ../testfile > ./ testfile1 ; wc –l testfile1

11) ls num[xyz]test

12) mv verse_temp verse_demo

13) awk ' /Science/ { print $1, $3 } ' students

14) x=$(ls –l | wc -l); echo $x

15) x=5 ; echo $((x * x + 5))

16) x=7; y=$((x ++)) ; echo “$x $y”

17) x=7; y=$((++x)) ; echo “$x $y”

18) x=$(echo $x | sed ' s/%/ / ')

2.2 Some sample questions

The list below shows one-line command exercises, with solutions. It will be important that you know how to do such exercises, as you will use similar style commands and utilities for later exercises and assignments. So, if you are not confident with these type of exercises, please study and practice them.

Example single line set of commands as solutions to the following problems:

Q1 Find the **largest file** in the **home** directory and put the result in a file called **longFile** in your **home** directory

A solution:

ls -l ~ > temp ; sort –g –r –k5 temp > temp1 ; head –1 temp1 > ~/longFile

A solution using pipes:
ls -l ~ | sort –g –r –k5 | head –1 > ~/ longFile

Q2 Find how many files are in the **parent** directory of your **home** directory and put the answer in a file called **countFile**, in your **home** directory.

A solution:
ls -l ~/.. > temp ; wc –l temp > ~/countFile

A solution using pipes:
ls -l ~ /.. | wc –l > ~/countFile

Q3 **Find** how many **directory files** are in your **home** directory, and put the answer in a file called **countDir**, in your **home** directory.

A solution:
ls -l ~ > temp ; grep "^d" temp > temp1 ; wc –l temp1 > ~/countDir

A solution using pipes:
ls -l ~ | grep "^d" | wc –l > ~/countDir

Q4 Find out how many times the **mkdir** command been used in the past 100 history lines and list the result in a file called **histCount** in your **home** directory.

A solution:
history 100 > temp ; grep "mkdir" temp > temp1 ; wc –l temp1 > ~/ histCount

A solution using pipes:

history 100 | grep "mkdir" | wc –l > ~/histCount

*NB: In Q3 above, the **ls -l** command output includes an extra line to indicate total size of the directory in blocks. Thus one extra line needs to be subtracted from the line count to get a more accurate number of files. This correction can be ignored for now.*

DISCUSSION

In what way does the use of pipes give an improved solution in the examples above?

The use of pipes results in a shorter command sequence, and more importantly pipes allow us to avoid the use of temporary files; as the use of such files slows down the command execution time and clutters up disk space.

What if you wanted to use a variable to represent a result for some of the above commands?

The **‘command substitution’** feature of the bash shell will allow us to do this, using the following syntax: **$(command)**. As an example, for **Q3** above, we could use a variable called **dirNum** to contain the number of directory files, as follows:

```
dirNum=$( ls –l  ~  |  grep “^d”  | wc –l  )
```

Q5 Find the **largest file** in the **home** directory and put the result in a variable called **bigFile**.

The following line will display the size of the largest file as a number on the terminal:

```
ls  -l  ~  |  sort -g -r -k5  |  head -1  |  awk '{print $5}'
```

We might prefer to try the **cut** command and do the same thing:

```
ls  -l  ~  |  sort -g -r -k5  |  head -1  |  cut  -d  ' '  -f5
```

Now, we were asked to get this value as a variable, so the following line will take the result of the commands and put that value in the variable **bigFile**, as was requested in the question:

```
bigFile=$( ls  -l  ~  |  sort -g -r -k5  |  head -1  |  cut  -d  ' '  -f5  )
```

Simply, type **echo $bigFile** to see the value of the variable on the screen**.**

Q6 Assume that some variable **x** has the value **25%.** Write a command to eliminate the **%** character so that the variable can be used as an integer.

A solution:

```
x=$( echo $x  |  sed  ' s/%/ / ' )
```