Exercise 4 – Bash - Your Interpreter

1. Review questions
2. How does the shell tokenise the command line (i.e. splits the command line into words) ?

Splits the command into tokens that are separated by the fixed set of metacharacters.

1. What is globbing ?

Characters used by programs to generate file names.

1. Why is the **$(cmd)** syntax recommended over the **`cmd`** notation ?

It is less confusing – backquotes are usally very similar to ordinary quotes and can be mistaken for them.

Some keyboards do not have backquote keys.

1. What is the difference (when used on the command line) between the single and double quotes ?

Single Quotes disable recognition of all special characters.

Double quotes protect most special characters

1. In Bash, what does the **!ls<**Enter**>** command achieve ?!l

ls– help

1. Experiment with globbing

In this exercise we will use the filename generation ("wildcards") features of the shell to find all files and directories under ***/etc*** whose names match the criteria given below.

To test your patterns, use one of the commands:

**echo <***pattern>* *OR*  **ls -d <***pattern>*where *<pattern>* stands for the search string you will type, for example, for filenames beginning with **m**, type:

**echo m\*** *OR* **ls –d m\***

1. Change to the **/etc** directory, and display files whose names:

i) Begin with **p** **echo p\***

ii) End with **y** **echo \*y**

iii) Begin with **m** and end with **d echo m\*d**

iv) Begin with either **e** or **g** or **m echo [egm]\***

v) Contain an **o** followed (not necessarily immediately) by a **p echo \*o?p**

vi) Contain the string **conf echo \*conf\***

vii) Begin with **s** and contain an **n echo s\*n\***

viii) Contain exactly 4 characters **echo ????**

ix) Contain a digit anywhere in the filename **echo \*[0-9]**

1. What will the   
   **echo \* \***   
   command display? Run it, and comment. Displays all files
2. Creating and manipulating user-defined variables

Simple variable assignment is not difficult. However, using quotes to correctly create (and expand) variables that contain non-alphanumerics as their values is a bit more involving. This is what we will practice in this exercise.

1. Create a new variable, called **name** and assign to it the value: **Joe**
2. name=Joe
3. echo $name
4. Create a new variable, called **address** and assign to it the value:

*<4 spaces>***22 Acacia Avenue** (i.e. address line to be indented by four spaces).

Address = ‘ 22 Acacia Avenue’

echo “$address”

1. Check the values of the two new variables you have created above.

Hint: How are you going to force the shell to show the spaces belonging to **address**?

nowecho $name

echo “$address”

1. Using command substitution

Command substitution is invaluable in scripts as well as in log file generating.

We will practice the technique here, using mostly the $(cmd) notation as one that provides more readable characters and better nesting capability.

1. Create a new variable, called **now** and assign to it the current date and time.

now = “$(date)”

Hint: **date +%c** formats the output of date to display date and time only. Now that we are substituting a command into the variable, we will need to use brackets and a dollar sign.

1. Test the assignment (check the value of **now**).

echo $now

1. Create a new variable, called **here** and assign to it the terminal being used for the current session. Hint use the **tty** command.

here=”$(tty)”

1. Test the assignment (check the value of **here**).

echo $here

1. Incorporate the two variables into a single **echo** command line, just as a program or a script might do to create a meaningful log file entry.   
     
   Your resulting line should look something like this:

This line was created on Wed Nov 4 18:02:41 2009, whilst working on the /dev/pts/1 screen.

echo ”This line was created on $now, whilst working on the $here”

Output: This line was created on Wed 2 Jan 20:52:16 GMT 2019, whilst working on dev/pts/17

If time allows

1. Practice using BASH ability to recall and edit command line
2. First place the commands shown below into the history file by executing them.   
   Note: the only significance in typing the commands shown below is to ensure a known content of the history file for us to explore in a moment.

$ **cd**

$ **w**

$ **pwd**

$ **ls –l**

$ **less helloworld.txt** (press q to exit)

$ **less /etc/passwd** (press q to exit)

$ **wc –l helloworld.txt /etc/passwd**

$ **history**

1. Now enter the following commands and observe the results:

$ **!ls**

$ **!wc**

$ **!less** Which **less** command was run?

$ **!ls:s/-l/../** What has this done?

$ **!!**

$ **!***n* For *n*, enter any number from history