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) ?

Shells have a set of meta-characters, one of which are whitespaces. When whitespaces are encountered the shell recognises a new word, signifying a new type of command.

1. What is globbing ?

Globbing characters are passed to the shell and used by programs to generate filenames.

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

Using backwards quotes are hard to distinguish.

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

Single quotes do not allow the shell to recognise special characters, doubles quotes maintain this ability. If using double quotes, a backslash can be used prior to a special character to disable its function.

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

This uses the ls command, but processes it with the preferences set by the last time that command was used.

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** p\*

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

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

iv) Begin with either **e** or **g** or **m**

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

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

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

viii) Contain exactly 4 characters ????

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

1. What will the   
   **echo \* \*** ls  
   command display? Run it, and comment. ***Every file and directory in etc.***
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. 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).

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**?

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.

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**).
2. Create a new variable, called **here** and assign to it the terminal being used for the current session. Hint use the **tty** command.
3. Test the assignment (check the value of **here**).
4. 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.

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