OS1 Lab 07 - Linux Shell Scripting & Environment Variables

Sample Solutions

Dr. Martin O'Connor

Task 1

Run the following examples in the Linux command line terminal and write the output displayed as your answer.

1. Exercise 1

```
$ X="hello world"
$ echo "printing $X to screen"
```

Answer:
printing hello world to screen

2. Exercise 2

```
$ echo \$X
```

Answer:
$X

3. Exercise 3

```
$ echo 'Single quotes "protect" double quotes'
```

Answer:
Single quotes "protect" double quotes

4. Exercise 4

```
$ echo "Well, isn't that \"special\"?"
```

Answer:
Well, isn't that "special"?

5. Exercise 5

```
$ echo "You have the following files in `pwd`"
```

Answer:
You have the following files in /home/mfocitt2016/lab07

6. Exercise 6

```
$ X=100
$ echo "The value of \$X is $X"
```

Answer:
The value of $X is 100

Task 2

What is the outcome of the following commands?

1. Exercise 1 (note: single quotes)

```
$ Greeting='Good Morning $USER'
$ echo $Greeting
```

Answer:
Good Morning $USER

2. Exercise 2 (note: double quotes)

```
$ Greeting="Good Morning $USER"
$ echo $Greeting
```

Answer:
Good Morning mfocitt2016

Task 3

The output of the following commands demonstrates an error. What is the error and how do you fix it?

```
$ cost="Price is $5.00"
$ echo $cost
```

Answer:
After executing the commands above, the following output is displayed:

```
Price is .00
```

The $5 is not displayed. The error occurred because the dollar symbol is interpreted as a special character by the bash shell. The bash shell considers $5 to be the name of a variable. In order for the dollar symbol to be interpreted literally, there are two possible solutions.

Solution 1: Escape the dollar symbol using the backslash

```
$ cost="Price is \$5.00"
$ echo $cost
```

Solution 2: Enclose the string in single quotes instead of double quotes.

```
$ cost='Price is $5.00'
$ echo $cost
```

Bash Script Variables

The following variables are available and accessible from inside a bash script.

- $0 - The name of the Bash script.
- $1 - $9 - The first 9 arguments to the Bash script.
- $# - The number of arguments passed to the Bash script when it is run.
- $@ - All the arguments supplied to the Bash script.
- $$ - The process ID of the current script.

Task 4

Write a script that takes one command line argument - your first name - and displays the message (assume the name Joe was given):

```
Hi Joe, don't you just love this module!
```

Answer:

Create a script called `task4.sh` with the following contents:

```
#!/bin/bash
echo "Hi $1, don't you just love this module!"
```

Then, grant execute permission to the owner of the script.

```
$ chmod u+x task4.sh
```

Finally, run the script passing your name as an argument to the script.

```
$ ./task4.sh Martin
```

The following output will be displayed:

```
Hi Martin, don't you just love this module!
```

Task 5

Write a script that takes two command line arguments, your first name and your favourite colour, and displays the message (assume the name Joe and the colour red were given):

```
Joe prefers the colour red.
```

Answer:

Create a script called `task5.sh` with the following contents:

```
#!/bin/bash
echo "$1 prefers the colour $2"
```

Then, grant execute permission to the owner of the script.

```
$ chmod u+x task5.sh
```

Finally, run the script passing your name and favourite colour as arguments to the script.

```
$ ./task5.sh Martin blue
```

The following output will be displayed:

```
Martin prefers the colour blue
```

Task 6

Write a script that may receive an arbitrary number of arguments (between 0 and 9 inclusive). The script must do the following:

- print the name of the bash script itself.
- print the username of the user running the script.
- print the number of arguments received.
- print all the arguments received.
- print the process ID of the running bash script.
- print the current working folder.

Answer:

Create a script called `task6.sh` with the following contents:

```
#!/bin/bash
echo "The name of the current script is: $0"
echo "The username running this script is: $(whoami)"
echo "The number of arguments received is: $#"
echo "The arguments received are: $@"
echo "The process ID of the current running script is: $$"
echo "The current working folder is: $(pwd)"
```

Then, grant execute permission to the owner of the script.

```
$ chmod u+x task6.sh
```

Finally, to test the script, run the script passing three arguments.

```
$ ./task6.sh arg1 arg2 arg3
```

The output displayed should be similar to the following:

```
The name of the current script is: ./task6.sh
The username running this script is: mfocitt2016
The number of arguments received is: 3
The arguments received are: arg1 arg2 arg3
The process ID of the current running script is: 896
The current working folder is: /home/mfocitt2016/lab07
```

Bonus Exercise

Create a shell script called `linecount.sh` which outputs your login name and the number of files in your current directory, followed by the name of your current directory.

Hint:

- Store the number of files in the current directory in a variable called *FILECOUNT*.
- To determine the number of files in the current directory, you may use redirection in combination with the commands `wc`, `ls`, and `grep`. OR
- To determine the number of files in the current directory, you may use redirection in combination with the commands `wc` and `find`.
- To learn more about the commands `wc`, `ls`, `grep`, and `find`, look up the system help by using the man command

Answer:

Create a script called `linecount.sh` with the following contents (two solutions provided):

Solution 1:

```
#!/bin/bash
echo $USER
FILECOUNT=`find . -maxdepth 1 -type f | wc -l`
echo The number of files in the current directory is $FILECOUNT
echo The name of the current folder is $PWD
```

Solution 2:

```
#!/bin/bash
echo $USER
FILECOUNT=`ls -al | grep ^- | wc -l`
echo The number of files in the current directory is $FILECOUNT
echo The name of the current folder is $PWD
```

Then, grant execute permission to the owner of the script.

```
$ chmod u+x linecount.sh
```

Finally, run the script.

```
$ ./linecount.sh
```

Sample output generated by both sample solutions when run in a folder with 7 files (including hidden files) is as follows:

```
mfocitt2016
The number of files in the current directory is 7
The name of the current folder is /home/mfocitt2016/lab07
```

Recall that hidden files are any files with filenames that begin with a period. For example: *.test*