PHYSICS 2T C Programming Under Linux

Linux Lab 3 An introduction to bash scripting

INTRODUCTION

This lab will introduce you to writing basic bash scripts and using control structures.

You will find this lab much easier if you read through some of **`man bash`** before starting. It is incredibly detailed and although intimidating, a very useful resource. For each section an example shell script is included in the Lab03 directory, it's important to look at these and try running them.

HELLO WORLD

It's customary for the first example of a new language to print “Hello World”. This lab will be no exception.

1. Create a new file called **`helloworld.sh`** in your **`Lab03`** directory.
2. Add the following and save it:
   ```
   #!/bin/bash
   echo “Hello World”
   ```
3. Use **`chmod`** to make **`helloworld.sh`** executable.
4. Run it using **`./helloworld.sh`**

Congratulations, you've just written and run your first bash script. The process is the same for every bash script you write, the only thing that will changes is be the contents of the file.

<u>Note</u>: To make your life easier, you may want to add the `Lab03` folder to your PATH variable, allowing you to run scripts using only their name (e.g. `helloworld.sh` instead of `./path/to/helloworld.sh`).

QUESTIONS

1. Write a script named **`Task1.sh`** which will create the following directory structure in the current directory:
   ```
   ./data/
   ./data/processed/
   ./docs/
   ```

CONDITIONALS

Conditionals are a core part of a bash scripting, they control the flow of a script and almost all scripts you write will use them.

Conditionals, as the name would imply, allow you to execute certain parts of the script if a condition is met. This is probably best demonstrated with an example.

```
1)  #!/bin/bash
2)
3)  if [ -f “processed” ]; then
4)      echo “There is a file called processed.”
5)  fi
6)
7)  exit 0
```

The code between lines 3 and 5 (**`if[ ]; then`** and **`fi`**) is only executed if the condition between the square brackets on line 3 are met. In this case, line 4 is only executed if a file called **`processed`** exists in the current directory.

`[` is synonymous with the **`test`** command. Line 3 could have been written:

`if test -f “processed”; then`

Of course, there are many more conditionals, the table below details the ones you're most likely to use, however you may want to read **`man bash`** for a full listing.

String Comparisons	**Result**	**File Conditionals**	**Result**
`string1 = string 2`	True if the strings are equal	`-d file`	True if file is a directory
`string1 != string2`	True if the strings are different	`-e file`	True if the file exists
`-n string`	True if the string is not null	`-f file`	True if the the file is regular
`-z string`	True if the string is null	`-r file`	Tue if the file is readable
Arithmetic Comparisons	**Result**	`-s file`	True if the file has a non-zero size
`exp1 -eq exp2`	True if both are equal	`-w files`	True if the file is writeable
`exp1 -ne exp2`	True if both are different	`-x file`	True if the file is executable
`exp1 -gt exp2`	True if exp1 is greater than exp2		
`exp1 -ge exp2`	True if exp1 is greater than or equal to exp2		
`exp1 -lt exp2`	True if exp1 is lesser than exp2		
`exp1 -le exp2`	True if exp1 is lesser than or equal to exp2		
`! exp`	Inverts exp. True if exp is false, false if exp is true		

Much like C, bash also has “else if” and “else” commands named **`elif`** and **`else`** respectively.

```
if [ condition ]; then
    …
elif [ condition2 ]; then
    …
else
    …
fi
```

Take a look at **`conditionals.sh`** in the Lab03 directory for more examples.

QUESTIONS

2. Copy your script from Question 1, **`Task1.sh`** to **`Task2.sh`** and modify **`Task2.sh`** so that it checks if any of the directories already exist. If they do, it should `exit` with an error code and message.

LOOPS

The ability to use loops is what makes bash scripting the perfect tool for repetitive tasks.

FOR LOOPS

For loops do something for every value in a range of values. A basic **`for`** loop looks something like:

```
for value in lots of values
do
      echo $value
done
```

Take a look at **`for.sh`** in the **`Lab03`** directory for examples.

WHILE LOOPS

For loops are useful when looping over a series of strings, however, they are not as useful if you do not know in advance how many times the loop needs to be executed. While loops are much better in this respect.

While loops continue while a condition is true. It is perhaps easiest to think of them as an if statement which will repeat until the condition becomes false.

```
While [ condition ]
do
      …
done
```

Take a loop at **`while.sh`** for some **`while`** loop examples.

QUESTIONS

3. Write a script (**`Task3.sh)`**which checks and reports if the current directory is the end of a branch (i.e. there are no directories inside the current directory).
4. Write a script (**`Task4.sh)`**which generates 100 random numbers and uses redirection to store them in a file called **`random`**.

   Hint: You can generate random numbers using the **`$RANDOM`** variable and can specify a numeric range by using **`{1..10}`**. Use **`>>`** to append to a file.

VARIABLES & COMMAND SUBSTITUTION

Setting and using variables in a bash script is exact the same as using them from the command line.

```
var=”Hello”
echo $var
```

There are, however, a few extra things you should be aware of.

When doing a comparison between two variables you should always surround them in double quotes. This is particularly important when the value of the variable could contain spaces. For example:

```
$test=“this has spaces”
if [ $test = “this has spaces” ]; then
      …
fi
```

will actually result in an error because $test will be replaced by **`this has spaces`** and bash will attempt to compare **`spaces = “this has spaces”`**. This is also helps stop an error if the variable is empty as **`[ = “something ]`** gives an error where as **`[ “” = something ]`** is valid.

There are several important environmental variables which are only used inside scripts. Below is a table of the ones you're most likely to need. **`man bash`** has a full list.

Name	Value
`$0`	The name of the script
`$#`	The number of parameters passed to the script
`$$`	The process ID of the script
`$1, $2, $3...`	The parameters given
`$@`	All of the parameters passed to the script
`$?`	Exit status of last command

Bash also has a feature called Command Substitution. That is, you can set the value of a variable to be the output of a command. There are 2 ways of doing this. The first is to surround the command in backticks (The quote like characters usually inserted by the key above the tab key) like so: **`` `ls- l` ``** or you can used the following structure: **`$(ls -l)`**. The second method is preferred although both are valid.

Take a look at **`variables.sh`** for some examples.

QUESTIONS

5. Write a script (`Task5.sh)`which loops through its arguments and prints if they are a file, directory or do not exist.
6. Write a script (`Task6.sh)`which checks for a **`.bak`** (backup) file for every **`.data`** file in the current directory (i.e. for every **`xxx.data`** file there should also be a **`xxx.data.bak`** file).
   Your script should report if there is a missing **`.bak`** and create one. Run your script in the **`data`** directory to check it works.