PHYSICS 2T

C Programming Under Linux

Linux Lab 4

Advanced Bash Scripting

INTRODUCTION

Lab 3 introduced the fundamentals of bash scripting: conditionals, loops and variables. These 3 things alone make up the overwhelming majority of what you need to know to write any bash script. This lab will show you how to get input from the user, read files and write functions in bash as well as give you more experience writing bash scripts.

GETTING STARTED

Start by opening the Terminal application. Verify that you are in your home directory, you can do this by typing **pwd** (Print Working Directory) into the terminal. It should be similar to **/home/0800890**.

To obtain all the files required for this lab, please enter the following into your BASH shell:

git clone https://bitbucket.org/glaphysp2t/linux-lab04.git

Confirm you have the necessary files by entering the command: **ls linux-lab04**

GETTING USER INPUT

Reading user input from the command line is done using the **read** command. The read command reads in a line of text and stores it in a variable. **read name**, for example, reads in a line of text and stores it in a variable called **name**.

```
#!/bin/bash
echo "What's your name?"
read name
echo "Hello, ${name}!"
```

The read command does not have a man page however you can find out more using help-read.txt found in the **linux-lab04** directory. In the examples directory input.sh may also help.

QUESTIONS

1. Write a script **(Task1.sh)** which finds individual files in the current directory with a size of 0, ask for confirmation, and deletes them. Hint: The **-s** flag will test if a file is zero size in your conditional statement.

READING FILES

There are several different ways to read a file line-by-line, however easiest and shortest methods use the **cat** command to pipe the contents of the file to a loop containing the read command.

Here is an example:

```
#!/bin/bash
cat file.txt | while read line;
do
        echo ${line}
done
```

Take a look at **readfile.sh** in the **Lab04** directory for more examples.

QUESTIONS

2. Write a script **(Task2.sh)** which prints the word count, using the **wc** command, of each line in **help-read.txt**.

FUNCTIONS

Bash has the ability to use functions, just like C. Creating a function is as simple as:

```
# Create the function
function_name() {
        function_code
}

# Run the function, it do not require parenthesis ()
function_name
```

Just like scripts, functions can have parameters too. They work in exactly the same way as passing parameters to a script however **they are completely separate excluding the $0 variable.** That means that the variable **$1** is different from **$1** in the rest of the script. **$0**, however, will always be the name of the script itself.

```
#!/bin/bash

func() {
        echo "${0}, ${1}"
}
```

```
func james
echo “${0}, ${1}”
```

```
james@x110e$ ./param.sh different
./param.sh, james
./param.sh, different
```

Take a look at **functions.sh** in the **Lab04** directory for more examples.

QUESTIONS

3. Write a function **(Task3.sh)** which takes a directory name as a parameter and:
    a) Checks if the directory already exists
    b) Create it if it does not exist
    c) Moves into the directory
4. Write a function **(Task4.sh)** which, for every file name passed as a parameter, prints if it is a file, directory or does not exist. Hint: You should be able to reuse the script you wrote in Lab3, task 5 for much of this script.

PUTTING IT ALL TOGETHER

Using all that you've learned in Linux Lab 01 to 04 along with the material presented in the Linux Lectures answer the following question.

5. Write a script **(Task5.sh)** which takes the encoded data in the **data** directory of the **linux-lab04** directory and decodes the **first** and the **last** line of each file, printing the decoded values to the screen. Your script should accept a list of filenames to decode as command line arguments and meet all the criteria below. The Linux command **base64** takes encoded data and converts it back into its non-encoded form. For instance, if you were given the string "**SGVsbG8K**" you would convert it back into plain text by doing the following:

   [brutha]$ **echo "SGVsbG8K" | base64 -d**
   Hello

Your Bash script must:

- Include the correct interpreter directive.
- Check the number of arguments and exit with 1 if none are supplied.
- Loop over each argument passed to the script.
- Check that each argument is a file, exit with 2 if it is not.
- Write a function **process** which decodes a string using **base64** as above and print the decoded string to screen.
- For each file, call the function **process** you have written on the **first** line of the file.
- For each file, call the function **process** you have written on the **last** line of the file.
- At the end of the script exit with a successful status.
- Correctly comment your code.