CSCB09 - Lab 2: More Shell Programming

Introduction

In this lab, you will write six shell programs. These will be very small programs with just about a handful of lines each.

Remember, the TAs are there to help you. You should work in pairs and are welcome to ask other students for help too.

Script 1: Hello world

Start by writing a very simple script named hello.sh that has only one command which prints "Hello world". You will need to use an editor to create the script. Try gedit, scite, nedit, or nano if you haven't begun to learn an editor on Unix/Linux yet. Remember to make your file executable before trying to run it (using `chmod`).

Script 2: For loops

In lecture we talked about the "`if` statement" (which works somewhat differently from the `if` statement in language such as Python or Java). Fortunately, the `for` loop works almost the same way it does in Python. For example the following `for` loop will print the numbers 1 to 3 followed by the values of the variables foo and bar:

```
for i in 1 2 3 $foo $bar
do
   echo $i
done
```

The only important thing to keep in mind is that before iterating through the elements following the `for ... in` the shell will perform any of the substitutions we talked about, including variable expansion (as in the example above), command substitution, and file expansion.

Write a script named for.sh that uses a `for` loop to print the following four things: the value of the environmental PATH variable, the number of command-line arguments passed to your script, the path to your home directory and the output of the command `pwd`.

Script 3: Counting characters in strings

Write a shell script named `count.sh` with a `for` loop that iterates over the command-line arguments passed to your script (recall our discussion in lecture of positional parameters to figure out a variable that will produce a list of the command-line arguments to iterate over) and outputs the number of characters in each of the command line arguments. (Hint: you can use the `wc` shell command from last week's lab to help you.)

Script 4: Test for files and directories

Write a script named `file.sh` that checks for each of its command-line arguments whether there exists a file or directory with this name in the current working directory, i.e. for each input it either outputs "`[name] is a` `file or directory`" or "`[name] is not a file or directory`", where `[name]` is the value of the command-line argument.

Script 5: Organising files

Shell scripts are often used to automate operations on a number of files in a directory. In this exercise you will take the files in a directory called `tests`, create for each file a new directory and copy the file into the new directory.

First some preparation. Copy the tests directory to your account using the following command.

```
 cp -r /courses/courses/cscb09w19/nizamnau/labs/lab2/tests .
```

Now create a directory called `actual`. Running `ls` should show both directories, `actual` and `tests`.

Write a shell program named `org_files.sh` that creates for each of the files in `tests` a new directory inside `actual` and copies the file into this new directory. The name of the new directory should be `actual_[filename]`, where `[filename]` is the name of the file you are copying. So after running your script the output of `ls actual/*/*` should be:

```
actual/actual_faculty1/faculty1  actual/actual_nobel1/nobel1  actual/actual_nobel4/nobel4
actual/actual_faculty2/faculty2  actual/actual_nobel2/nobel2  actual/actual_nums1/nums1
actual/actual_faculty3/faculty3  actual/actual_nobel3/nobel3  actual/actual_nums2/nums2
```

Optional

If you still want more practice after completing the scripts above, below is another sample exercise you can use to get some practice in shell programming.

There is a program called `which` that takes a program name as an argument and looks through the directories in `PATH` to find it. It prints out the absolute path to the first instance of the program name that it finds. For example, if I run `which python` in my account, the output is:

```
    /usr/bin/python
```

You will write a shell program called `whicha` that prints the absolute path of all occurrences of the program name given as an argument. For example, in my account, `whicha python` gives the following output:

```
    /usr/local/bin/python
    /usr/bin/python
```

Recall the elements of the PATH variable are separated by a colon. The program `tr` can be used to replace all occurrences of one character with another, reading from standard input. So `tr ":" " "` will replace all instances of a colon with a space when reading from standard input.

Your program will only print the absolute path if the file is executable.