Lab 1: Shell Arrays

Introduction

In this Lab, you'll demonstrate your shell scripting knowledge by building a script that uses loops, conditionals, and arrays to pretty print your marks.

Bash Scripting with Loops, Conditionals, and Arrays

In your `Lab1` directory (please create it: `mkdir Lab1`), use `touch` to create a file named `pp_marks.sh`. Then, run `chmod u+x pp_marks.sh` (see lecture slides and/or `man chmod` for more information) to make your script executable. The goal of this section is to turn your new file into a bash script that pretty prints your tutorial and assignment marks.

First, some background. You'll end up having 5 labs (in directories `Lab1` to `Lab5`) and 3 assignments (in directories `a1` to `a3`). When we mark your tutorials and assignments, we will place a file named `feedback.txt` into the directory that contains comments and a mark. The last line of this file will always be in the format `score / max_score`.

Your script will need to loop through your tutorial and assignment directories. In each directory, it will extract your score and the max possible score, display those scores, and update a running tally. After printing all of the tutorial and assignment scores, it'll print totals. If a tutorial or assignment is missing, a placeholder line will be printed. For example, here is the output of my script being run on a folder with some, but not all, of the expected directories:

```
$ cd ~/209_repo
$ ls
a1  a2  t01  t02  t03  t04  t06
$ t02/pp_marks.sh
a1: 25 / 30
a2: - / -
a3: - / -

Lab1: 3 / 3
Lab2: 3 / 3
Lab3: 2 / 3
Lab4: 3 / 3
Lab5: - / -


Assignment Total: 25 / 30
Lab Total: 11 / 12
$
```

In the text above, `$` designates a prompt. Note that the script printed `- / -` if a directory is missing. It prints the same thing if the directory is present but does not contain a `feedback.txt` file. Marks are simply summed. There is no attempt to calculate a percentage or to weight assignments or tutorials.

Tips for the Script

This is a tall order, so how should you start? First, I would extract a grade from a single file. Here is some code to get you started:

```
#!/bin/bash

grade=`cat feedback.txt | tr -d [:blank:]`

IFS='/' read -ra grArray <<< "$grade"
score=${grArray[0]}
max=${grArray[1]}

echo $score
echo $max
```

There are both some familiar features and some new features in the script above. First, check out the first line. That line is necessary, as it tells the OS how to execute the script. Second, look at the last two lines: this script just displays its result using `echo`. You will want to change the output to match the expected output. Next, check out the use of backticks on the line that assigns to `grade`. The backticks mean "execute this command, and capture the output", which we then save as a string into the `grade` variable. Look up `cat` and `tr` as necessary, or try them out to see what they do. Finally, something very new: the middle block introduces the use of bash arrays.

`IFS` is a standard bash variable that stands for "internal file separator". That first line uses it to translate the contents of the `grade` variable into an array variable named `grArray` by splitting the line (which is in the format `grade / max_score`) on the delimiter `/`. The line uses a "here string" to get input from the variable.

The next two lines index into the array to get the first and second items: the score and max score, respectively.

That example does a lot of the work for processing a single file. One thing is missing: you'll need to modify the script to extract the last line of the feedback file. (Hint: look up `tail`.)

Once you get your script working for a single file that you know exists, it's time to make it work for multiple files. You can enclose your previous code in a function, if

you like. In any case, that code will need to be called in a for loop. Here is an example of a loop over an (implied) array of strings:

```
$ for d in Lab1 Lab2 Lab3; do
>     echo $d
> done
Lab1
Lab2
Lab3
$
```

You can use that loop to iterate over a set of directories—but make sure to check that the directories (and the required files in them) exist before you try to access a file in them! You'll also need to keep a running tally of scores, which means that you need bash to have integers. The `declare` statement will be helpful here. For example, the following demonstrates how to declare a variable that is treated like an integer:

```
$ declare -i x=0
$ x=$x+1
$ echo $x
1
$
```

Finally, **please remember that your code is being marked using tests, so it's important that your output matches exactly**. Please check your capitalization, for example, and make sure there are blank lines in between the sections of output.
Submission: submit pp_marks.sh on eclass by Friday, Sep 24, 11:59pm