Assignment — Count the Zoo

CS 265 Advanced Programming Techniques

Introduction

The purpose of this assignment is for students to become writing Bash scripts and functions, traversing the filesystem, and using Unix filters, pipes, and redirection to solve a text processing problem.

Problem

For this assignment, you will write a Bash script that visits every subdirectory in a folder (provided via command line argument). In each sub-folder (including the top-level folder), you might find a metrics file. You will summarise data across all subdirectories found in these files.

To Do

The Metric File

By default, the file will be called **`.KS_Dir`** . Whatever its name is, it will be consistent in all subdirectories.

The metrics file is a 2-column file. The first column is the name of the measurement, and the 2nd is an integer, some scalar value. The columns are speparated by whitespace.

There are 5 possible measurements which might be reported on, in no particular order:

- `bear`
- `dromedary`
- `aardvark`
- `genet`
- `lemur`

Here is a sample metric file, looks like this:

```
dromedary 5
lemur 33
bear  11
```

Command-line Arguments

```
assn1 [-f filename] [start_directory] measurement
```

- There is one required argument, the measurement. It will appear last.
- Optionally, it might be preceded by a starting directory. Current directory is the default, if missing
- There is an option, `-f`, to set the name of the metric file. Defaults to **`.KS_Dir`**
- If arguments are wrong (wrong number, unknown option, etc.), print a usage message and quit
- If you can't access the starting directory, print an error message and quit

See Examples, below.

Output

Just print the measurement, followed by the sum of the selected measurement, on the same line:

```
genet 842
```

Other Requirements

- You may not use the `find` utility. You must search for subdirectories yourself
    - Likewise, you can't use something like `ls -R`
    - Do the file testing and recursion yourself
- Your script **must be written in Bash.**
    - It *may* contain some AWK
- Call your script **`assn1`** (that's a one, not an "ell").
- We'll run your top-level script directly
    - So get the sha-bangs right
    - We'll set the execution bit
    - Your script is not guaranteed to be in the PATH

Examples

Get a total of genet, rooted in the current directory:

```
$ ./assn1 genet
```

Get a total of lemur, rooted in **`../dev/submit`** :

```
$ ./assn1 ../dev/submit lemur
```

Get a total of dromedary, rooted in the current directory. The metric files will be name **`dir_met`** :

```
$ ./assn1 -f dir_met dromedary
```

Get a total of bear, rooted in **`../dev/submit`** . The metric files will be name **`dir_met`** :

```
$ ./assn1 -f dir_met ../dev/submit bear
```

Hints

- **Make sure you understand the problem before you begin to code.** It's difficult to solve a problem that you don't understand.
- Use functions
    - Finding the location of your script can be messy. Functions can easily be called
    - Iterate over files, recursively call a function on directories that you find
- Do **not** write a bunch of lines of code, then wonder why it doesn't work
    - Break your task into logical pieces
    - Implement them using functions, where it makes sense
    - E.g., start by pulling the value you need from the file, if the file exists, and if the metric is in the file
    - Then, process just a single directory, accessing the metric file, and identifying subdirectories
    - Put the above together. Not all at once, but, gradually. Make it easy to test

- Create your own test directory that you can test your script on as you write it. Your test should contain all of the edge cases you can think of (missing READMEs, missing index/required entries, empty folders, etc.).
    - Start it very simple
    - As you get a feature working, add more test data, and test cases
- See **`Labs/Bash/getopts.bash`** to help you parse the option argument (maybe)
- Use `grep` and `cut` to pull values out of the metric file
- The return value of a Bash function is different than in your CS1 language
    - It's usually just a 7 or 8 bit number
    - So it's really just a status, just like the exit value of a script
    - If you need a number out of a function, have the function print (echo) the value, and use command substitution to capture it

Submission

Do **not** submit temporary files, nor your test data. Only submit the required scripts, and any helper files you wrote.

What to hand in

- **`assn1`** — your top-level script, the entry point, the script we will call with the directory to be processed
- All the rest of your source code
- **`README`** (*optional*) — anything you want to tell us before we grade.

Again, **`README`** is not **`README.txt`** , and **`assn1`** is not **`assn1.sh`** .