Mandatory Assignment 2

Introduction to Bash Scripting

IN3110 12+3 Bonus Points / IN4110 15 Points

University of Oslo - IN3110/IN4110

Fall 2020

Your solutions to this mandatory assignment should be placed in a directory named `assignment2` in your Github repository.

2.1: Moving Files to a Destination (4 points)

When you want to connect several scripts, bash is often used as the glue connecting single scripts. One of `bash‘s` strengths is file handling. Often you need to move files from one location to another since the output files of one program in your project might be the input files of another script in your project. Therefore, it might be neat to have a function that does that for you!
In this task you will write a bash function `move`, which moves all of the files in a directory to another directory you define.

What must the function be able to do? - Supported Functionality

- take two commandline arguments: the source, where the files you want to move are located, let's call it `src`, and the destination, where the files should be moved to, let's call it `dst`
- you want to assign the commandline arguments to a variable (Hint: Access commandline arguments via `$0,$1`)
- you should check that the `src` directory exists
- you should check that the `dst` directory exists
- you should check that there are actually to commandline arguments passed (Hint: Check length of an array `$#`)
- you want to loop through all the files in the `src` directory and move them to the `dst` directory

REQUIRED FOR IN4110 & OPTIONAL FOR IN3110 (3 bonus points): *Make these script less rigid* in order

(a) that you are not moving all files, but only files of one type, e.g. `*.txt`-files ?

(b) that you check that the directory your files are moved to actually exists, and if not, it throws an error or just creates the directory auromatically ?

(c) that you create a directory in the new destination that has the name of the current date and time in the format `YYYY-MM-DD-hh-mm`?

Great, now you can move files like a pro!

2.2: A Simple Time Tracker (5 points)

Curious about how much time you are spending on a task? Let's create a small program tracking the time spent on various tasks. To make things easier, we are going to assume that we are only working on one task at a time. When the timer is stopped, we cannot start it again without creating a new task. The time tracking data should be stored in a file whose name is specified by an environment variable called `LOGFILE`.

What does the function need to be able to do? - Supported Functionality

- `track start [label]`: Starts a new task with a label. Should print an error message if a task is aready running.
- `track stop`: Stops the current task, if there is one running.
- `track status`: Tells us what task we are currently tracking, or if we don't have an active task.
- If any other arguments are given, a helpful message should be displayed to tell the user how the program works.

The Logfile
This is how the logfile could look like:

```
START Fri Aug 24 15:31:59 CEST 2020
LABEL This is task 1
END Fri Aug 24 15:32:18 CEST 2020

START Fri Aug 24 15:31:59 CEST 2020
LABEL This is task 2
END Fri Aug 24 15:32:18 CEST 2020
```

Hint about Accesibility: To be able to manipulate your current bash session, you **cannot run this as a script** (i.e. with `bash track.sh`).

Instead, you need to source the file containing the function before you run it. i.e. `source track.sh`. Then you can use the function in the command line. To enable the function to work whenever you open a new terminal, you can put it in your `.bashrc`.
Congrats, you wrote your first basic time tracker!

2.3 Making The Time Tracker Useful (3 points)

Tracking time isn't really helpful unless we can display the data in a useful way. Extend your script from 2.2 to support a `log` command, that displays the time spent on each task in the format `HH:MM:SS`. Example output:

```
Task 1: 02:30:12
Task 2: 00:03:32
```