# Mandatory Assignment 2

## Introduction to Bash Scripting

IN3110 12+3 Bonus Points / IN4110 15 Points

University of Oslo - IN3110/IN4110

Fall 2021

Your solutions to this mandatory assignment should be placed in a directory named `assignment2` in your Github repository. You need to create a `README.md` file in your `assignment2` directory that documents the usage of the scripts created including, but not limited to how to run the script and which functionalities it covers in general. We added a simple example README.md to the assignment2 documents.

Comment your code, so that people that where not involved in the coding process understand what is happening in your script.

Since some compatibility issues might arise, when solving these tasks on different systems, we recommend you to checkout the remote login to the Ifi machines [1], when running into issues on your own machine. All correctors will have access to the ifi machines as well.

### 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`

[1] `https://www.mn.uio.no/ifi/tjenester/it/hjelp/it-vakten/laptophjelp/laptophjelp-guide/tilgang-til-uio-hjemmeomrade-og-ifi-linux-terminal/index.html`

- you want to assign the commandline arguments to a variable (Hint: Access commandline arguments via `$0,$1`)
- you should check that the `src` directory exists and return an error message to the user if the `src` directory does not exist
- you should check that the `dst` directory exists and return an error message to the user if the `dst` directory does not exist
- you should check that there are at least two commandline arguments passed (Hint: Check length of an array `$#`)
- you should return an error message to the user, if the number of commandline arguments was wrong. The message should point the user to the right usage of your script.
- The user should be able to enter both the relative path and the full path without the script producing an error.

**Since a directory is a type of file they should also be moved from the source directory to the destination.**

**REQUIRED FOR IN4110 & OPTIONAL FOR IN3110 (3 bonus points):** *Advance the Moving Script*

(a) Implement an additional functionality of the script to specify the type e.g. `*.txt`-files, `*.dat`-files of the files to be moved

(b) Implement that, if the `dst` directory does not exist already, the directory gets created automatically (using the name passed as commandline argument by the user), if the user want (Hint: Ask the user for a yes/y/Yes/Y or no/No/n/N)

(c) Implement that the user can choose the directory created to have the current date and time in the format `YYYY-MM-DD-hh-mm` (Hint: This could be implemented as an alternative naming option for the directory.)

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 a permanent 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 already 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 in the terminal to tell the user how the program works.
- If another task is running already, a helpful message should be displayed in the terminal.
- If no task is running but the user is trying to stop tracking, a helpful message should be displayed in the terminal.
- The actual log file should be created by the script and located in ∼`/.local/share/`

**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:** The `LOGFILE` variable would hence point to the location of your `.timer_logfile`

**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`.

**Hint:** As you are not sure, if the variable `LOGFILE` is already set in the test environment or `.bashrc` or `.bash_profile`, you could use `grep` as one option in your script to check if it is set already, before adding it, if it is not set already.

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
```

If you are not sure, if the variable is set in your environment or `.bashrc`, you could use `grep` to check, if it is set already (see your first bash lecture).

**Hint:** Make sure you check, that tasks exceeding 24 hours are still tracked correctly.