# CS108 - Software Systems Lab

Lab 6 - Git

**Student:** Aditya Sanapala, 23b0912@iitb.ac.in
**Lecturer:** Kameswari Chebrolu, chebrolu@cse.iitb.ac.in

Please go through the `README.md` in the `labDirectory` folder before attempting this activity. It describes how to set up the directory and submit your work.

Navigate to the `scripts/` directory and run `bash reset.sh`

**To Submit:** Navigate to the `scripts/` directory and run `bash submit.sh`. Then use the `Git VCS` in VLab to submit, as you have in previous labs. Remember to use the given scripts given before submitting, or if you wish to reset the entire directory and start afresh, otherwise it may cause clashes with the `.git` folder in `labDirectory`. More details on script usage can be found in the `README.md` in the `labDirectory`.

**Please use `reset.sh` carefully, it can reset your whole work with no way of recovering if is hasn't been committed.**

### Problem 1: Add and Commit

In the folder `working_directory`, you have been given 3 files – `README.txt, add.h`, and `main.cpp`. These files, when compiled, generate an executable that takes in 2 integers as input and prints their sum on the screen.
All the git commands have to be run inside the `working_directory` folder (otherwise it may interfere with the parent git folder leading to issues while submitting).

(a) Initialize a new empty git repository inside the `working_directory` folder.

(b) For the first commit, add only the README file, with the commit message `"Added README.txt"`.

(c) For the next commit, add the `add.h` file (in addition to the `README.txt` already committed), with the commit message `"Added add.h"`.

(d) For the next commit, add only the `main.cpp` file (in addition to the `README.txt` and `add.h` already committed), with the commit message `"Added main.cpp"`.

### Problem 2: Branching

Navigate to `working_directory/big_repo/`. All the git commands have to be run inside this directory (otherwise it may interfere with the parent git folder leading to issues while submitting).
The repository `big_repo` has a commit history with a single branch – `master`. Each commit adds a new file with a function implemented in it. However, sometimes, in a commit some previously uploaded files are incorrectly modified.

Your job is to identify the last commit where each of the files – `bfs.cpp`, `binary_search.cpp`, `dfs.cpp`, `matrixmul.cpp` and `sieve.cpp` were functioning correctly.

1. You don't need to test out the functions to check their correctness, assume that when the function was first added it was working correctly.
2. If and when it is modified in any later commit, assume that causes the function to work incorrectly.
3. If any function is only added, and never modified, assume that it works correctly even in the last commit.
4. To check the changes that were done from one commit to another, try the `git diff` command.
5. The `main.cpp` file is modified at each commit but assume that it works correctly at each of them).

After identification of the commits for each of these files, navigate to each of these commits and create a new branch from the commit with the name `correct_<function_name>`. In this branch, (Remember to checkout to the branch first) delete all files except the file that contains the last version of the correct function, add the changes and commit them with the message `Saved <file_name>`.

Finally, you should have 6 branches:

```
master
correct_bfs
correct_binary_search
correct_dfs
correct_matrixmul
correct_sieve
```

Each of these branches (except master) should have exactly one commit not in the master branch with the corresponding message:

```
correct_bfs -> Saved bfs.cpp
correct_binary_search -> Saved binary_search.cpp
correct_dfs -> Saved dfs.cpp
correct_matrixmul -> Saved matrixmul.cpp
correct_sieve -> Saved sieve.cpp
```

The latest commit in each of these would have exactly one file:

```
correct_bfs -> bfs.cpp
correct_binary_search -> binary_search.cpp
correct_dfs -> dfs.cpp
correct_matrixmul -> matrixmul.cpp
correct_sieve -> sieve.cpp
```

Each of these branches has to be created at the right commit which you have to find out. You can manually check the files for changes (which can be tedious) or you can use the `git diff` command (Figure out how one can use this).
Finally, submit this modified repository with the new branches.

**Problem 3: Merging**

For this question, navigate to `merge_repo/`. All the git commands have to be run inside this directory (otherwise it may interfere with the parent git folder leading to issues while submitting).
The repository `merge_repo` has a commit history with two branches – `master` and `development`. The repository had an initial commit where 4 empty files were added - `file_1.cpp`, `file_2.cpp`, `file_3.cpp`, and `file_4.cpp`.
At this commit the development branch was created, which modified these 4 files. In the meantime, these 4 files were also modified in the master branch. This can be seen in the commit history.
Now, we wish to merge these two branches.
Remember to merge the development branch into the master, and not the other way around.
However, since the files were modified differently, there are bound to be some merge conflicts.

Resolve the merge conflicts in the following order:

1. `file_1.cpp` – Modify the file so that it becomes identical to the file in the master branch.
2. `file_2.cpp` - Modify the file so that it becomes identical to the file in the development branch.
3. `file_3.cpp` - Take all the changes in the files in both the branches.
4. `file_4.cpp` - Replace the original file with two files - `file_4_master.cpp` (that contains the file contents of the `master` branch) and `file_4_development.cpp` (that contains the file contents of the development branch).

The above is a basic idea of what is to be done, more specifics on the functionality of the merged file can be found in `README.md`.
Though a basic functionality of the files is outlined in `README.md`, you guys are encouraged to go through the code on a superficial level, and run the programs a few times to get acquainted with it. A basic knowledge of the code will be needed for merging (especially `file_3.cpp`).

Rest assured you don't have to match the file content exactly. As long as the C++ code compiles and behaviour of the code is as expected on a few basic tests, you shall receive full marks for merging.
You may take as many commits as you need for merging, just make sure that the latest commit in the `master` branch contains the expected merged files. Be sure that the commit does not contain any executable or other files. Only five `.cpp` files should be present in the final commit in the `master` branch.
Finally, submit this modified repository with the merged branches.