Git project bible

General structure for commits messages should be consistent and easy to understand.

```
git commit -m "type: short summary"
# feat: add data preprocessing pipeline for dataset
# refactor: move plotting functions into separate module
# fix: handle incorrect computation of covariance
# docs: update README with correct version information
```

If you made a typo:

```
git commit --amend
```

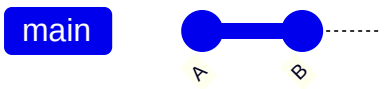
You don't have to push every time you commit. You should commit every time you have made a change that is significant enough for a similar message to the ones above. Basically when you have completed a task.

Remember to pull before you start coding.

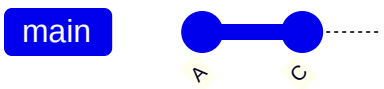
```
git pull
```

If you have committed something to your branch, and someone else has too, if you just pull and merge, it will look something like this.

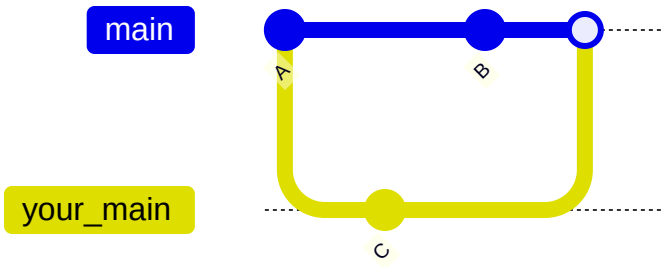
The commit and push of another person:



Your commit:



When you then pull/push, you end up with a merge, even tough the other person made changes to completely different files.



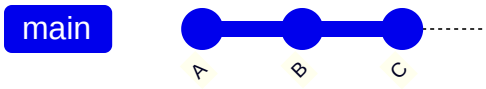
A better way to handle this is to rebase before you push your changes

```
git pull --rebase
```

or you can fetch and rebase instead:

```
git fetch origin
git rebase origin/main
```

This results in a rebase of your **local (unpushed) commits** to the newest version. Essentially moving your commit to the end of the update history.



If there are conflicts, git will tell you. you can resolve the conflicts in the files. then run:

```
git add <file>     # or git add . if all resolved
git rebase --continue
```

If you want to cancel and go back to how things were before the rebase:

```
git rebase --abort
```

When you do a lot of work on one thing, consider creating a branch to work from. This means that pushing wont conflict with others work and the graph is more easy to read when a bunch of commits are mixed together for different parts of the project.

```
git branch branch-name
git switch branch-name
```

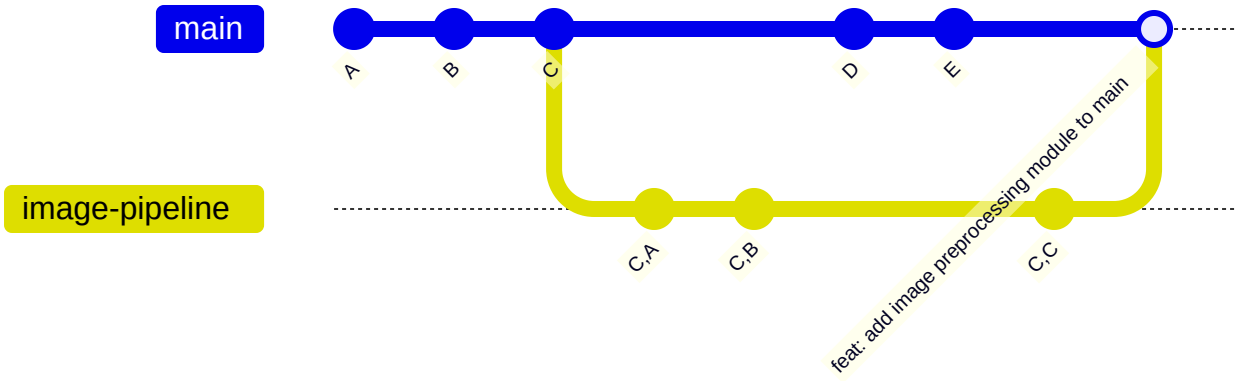
When you are ready to merge:

```
git switch main
git merge branch-name -m "type: summary"
```

For instance, someone makes a branch for image pre-processing. This is maybe done over the span of two days, so someone else has pushed something on main:

```
git branch image-pipeline
git switch image-pipeline # git switch -c image-pipeline (create and jump)
```

```
git switch main
git merge branch-name -m "feat: add image preprocessing module to main"
```



> Do **NOT** `git merge main` from the branch, as this will bring main into the branch.

If you are not ready to commit your ***Work In Progress (WIP)**, but you need something new from origin/main that might conflict, you can stash the changes and pop them to resolve the conflict on local:

```
git stash
git pull # --rebase if you have commits
git stash pop
# then resolve if someone else changed the same lines/files
```

You can also just do the equivalent with 1 line. No reason not to. Do keep in mind that using both rebase and stash at the same time when you pull quickly gets confusing. better to do `git stash` and then pull with rebase, to do one thing at a time.

```
git pull --autostash
```