ITC313 - Git & GitHub

Fundamentals of programming: Introduction to C++/C

Dominique Ginhac

Tutorial

Git & Github

Students are introduced to the source code management with git & github.

https://git-scm.com

Git is an Open Source Distributed Version Control System.

1. Git is used to store content (typically code).
2. Git tracks document versions and keeps a history of changes
3. Git is distributed: the code is not just stored in a central server, but the full copy of the code is present in all the developers' computers.

Git concepts: Snapshots

A Snapshot keeps track of your code history.

A Snapshot records what your files look like at a given point in time.

You decide when to take a snapshot and what files are included.

You have the ability to go back to visit any snapshot.

A Snapshot is called a Commit. A project is made up of a bunch of commits.

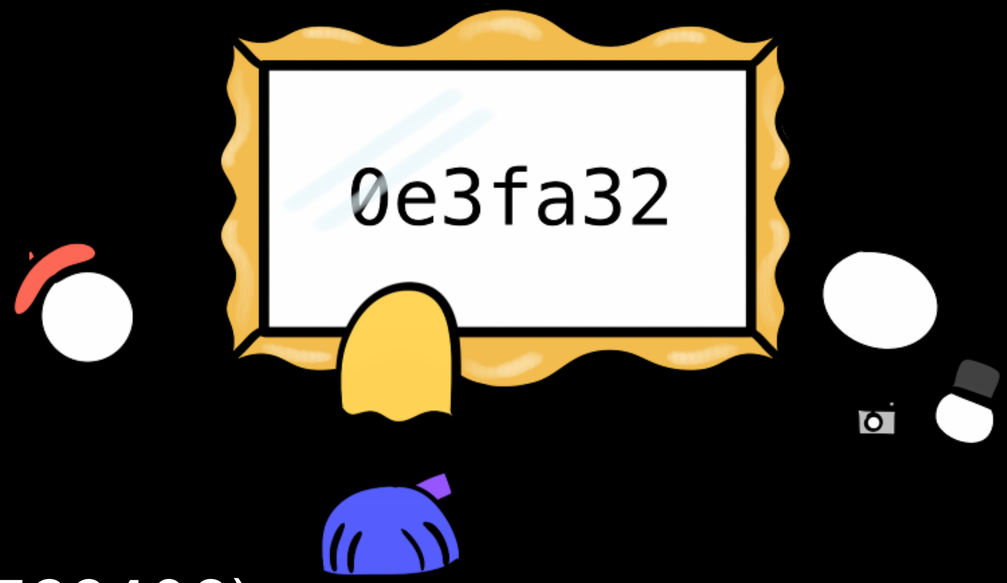
Commits

Commits contain 3 pieces of information

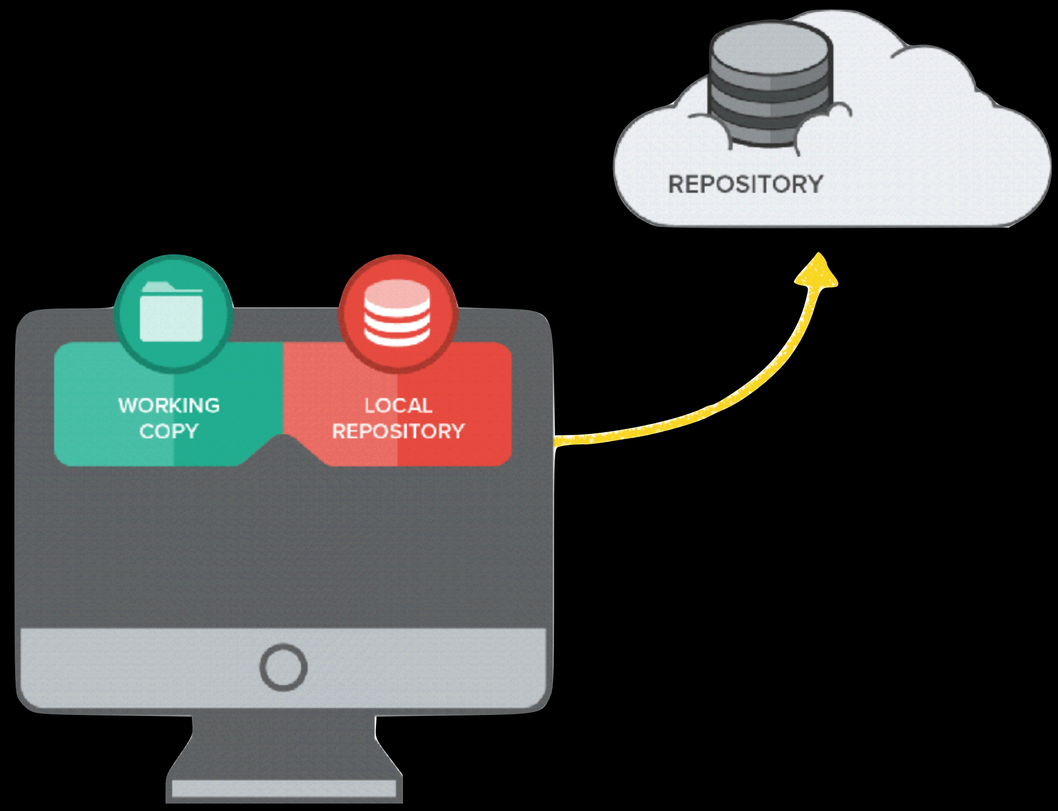
1. Information about how the files changed from the previous commits (diff).

2. A reference to the previous commit (the parent commit).

3. A hash code name (something like fe6f7ffa8562fc9b9ef8f033a8523decdd532406)



Repositories



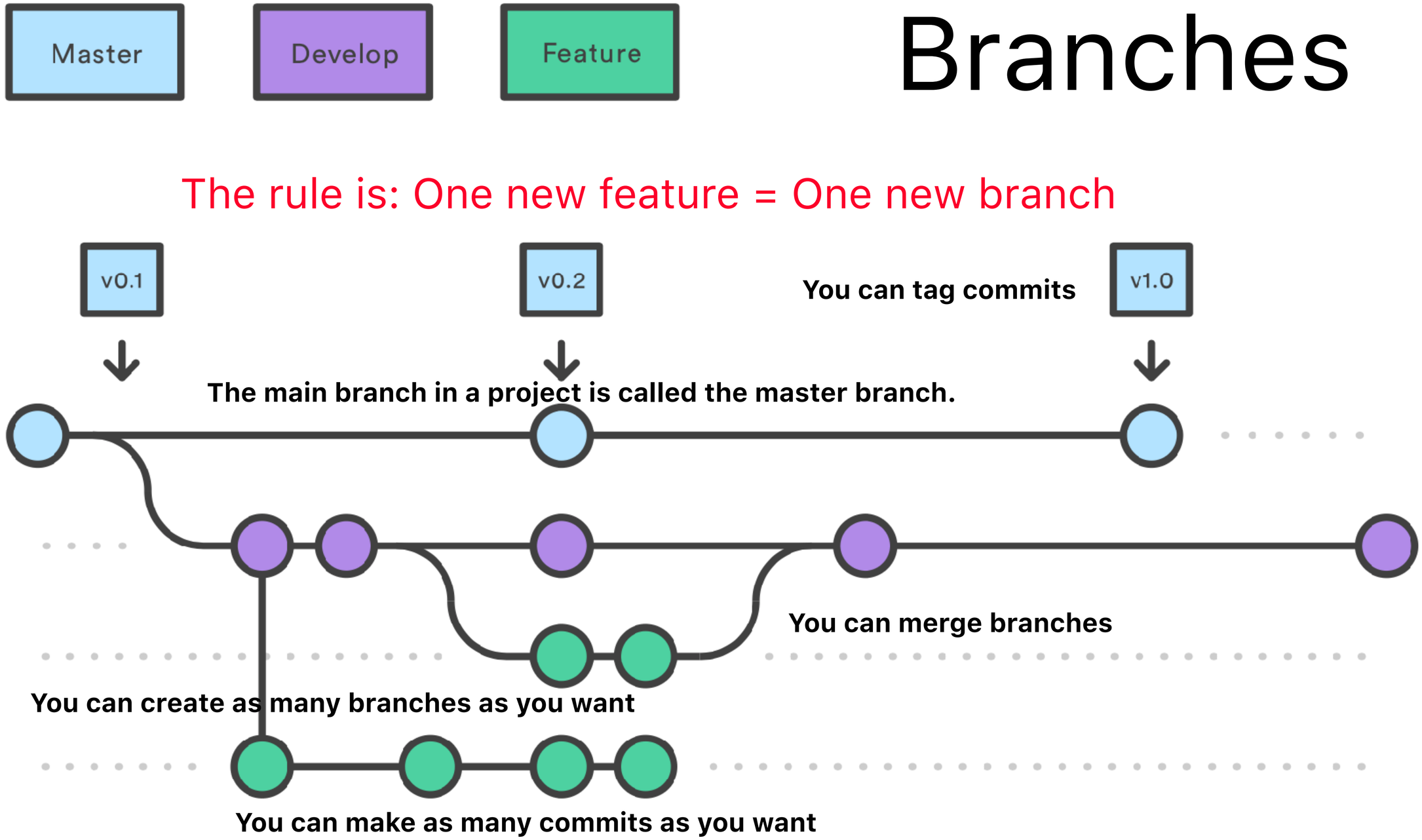
Often shortened to ‘repo’.

A repo consists of all your commits.

It is a collection of all the files added
to commits and the history of these files.

Can live on a local machine or on a remote server (gitHub for example)

Branches


Master
Develop
Feature
The rule is: One new feature = One new branch
v0.1
v0.2
You can tag commits
v1.0
The main branch in a project is called the master branch.
You can merge branches
You can create as many branches as you want
You can make as many commits as you want

git

Tutorials

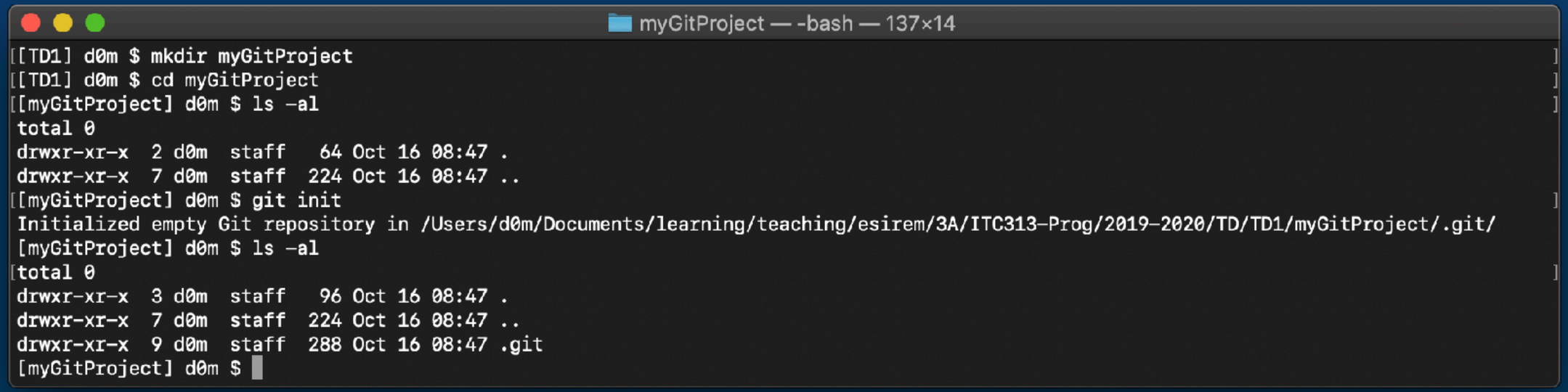
http://rogerdudler.github.io/git-guide/index.html

create a new repository

create a new directory, open it and perform a

`git init`

to create a new git repository.



add & commit

You can propose changes (add it to the **Index**) using

```
git add <filename>
```

```
git add *
```

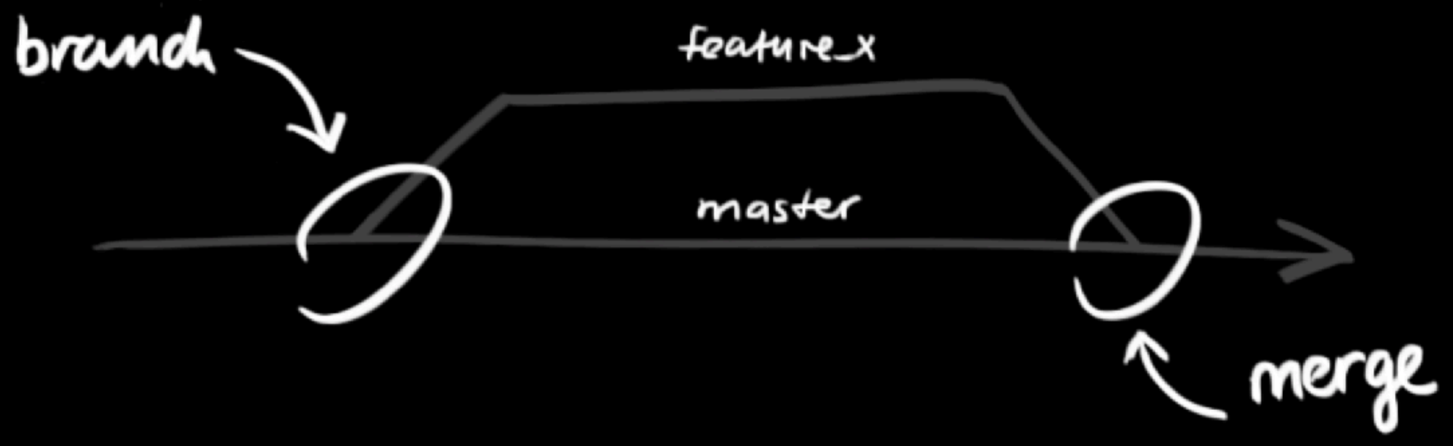
This is the first step in the basic git workflow. To actually commit these changes use

```
git commit -m "Commit message"
```

Now the file is committed to the **HEAD**, but not in your remote repository yet.

branching

Branches are used to develop features isolated from each other. The *master* branch is the "default" branch when you create a repository. Use other branches for development and merge them back to the master branch upon completion.



create a new branch named "feature_x" and switch to it using

```
git checkout -b feature_x
```

switch back to master

```
git checkout master
```

and delete the branch again

```
git branch -d feature_x
```

tagging

it's recommended to create tags for software releases. this is a known concept, which also exists in SVN. You can create a new tag named *1.0.0* by executing

```
git tag 1.0.0 1b2e1d63ff
```

the *1b2e1d63ff* stands for the first 10 characters of the commit id you want to reference with your tag. You can get the commit id by looking at the...

log

in its simplest form, you can study repository history using.. `git log`

You can add a lot of parameters to make the log look like what you want. To see only the commits of a certain author:

```
git log --author=bob
```

To see a very compressed log where each commit is one line:

```
git log --pretty=oneline
```

Or maybe you want to see an ASCII art tree of all the branches, decorated with the names of tags and branches:

```
git log --graph --oneline --decorate --all
```

See only which files have changed:

```
git log --name-status
```

These are just a few of the possible parameters you can use. For more, see `git log --help`

replace local changes

In case you did something wrong, which for sure never happens ;), you can replace local changes using the command

```
git checkout -- <filename>
```

this replaces the changes in your working tree with the last content in HEAD. Changes already added to the index, as well as new files, will be kept.

https://guides.github.com

checkout a repository

create a working copy of a local repository by running the command

```
git clone /path/to/repository
```

when using a remote server, your command will be

```
git clone username@host:/path/to/repository
```

If you have not cloned an existing repository and want to connect your repository to a remote server, you need to add it with

```
git remote add origin <server>
```

Now you are able to push your changes to the selected remote server

```
git remote add origin https://github.com/dginhac/myProjectGit.git
```


Do not forget to create an empty project on GitHub

pushing changes

Your changes are now in the **HEAD** of your local working copy. To send those changes to your remote repository, execute

```
git push origin master
```

Change *master* to whatever branch you want to push your changes to.

update & merge

to update your local repository to the newest commit, execute

```
git pull
```

in your working directory to *fetch* and *merge* remote changes.

to merge another branch into your active branch (e.g. master), use

```
git merge <branch>
```

in both cases git tries to auto-merge changes. Unfortunately, this is not always possible and results in *conflicts*. You are responsible to merge those *conflicts* manually by editing the files shown by git. After changing, you need to mark them as merged with

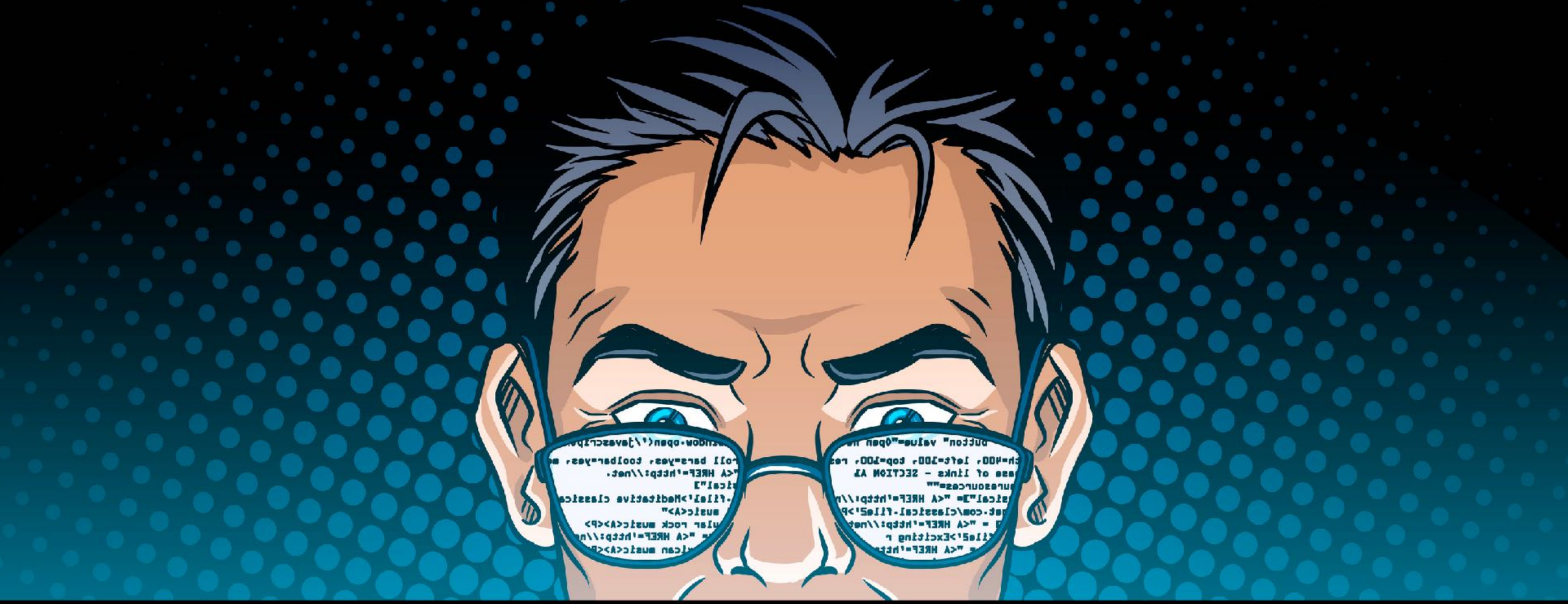
```
git add <filename>
```

before merging changes, you can also preview them by using

```
git diff <source_branch> <target_branch>
```

a branch is *not available to others* unless you push the branch to your remote repository

```
git push origin <branch>
```



So let’s start coding!

(c) 2019/2020 - dginhac@u-bourgogne.fr - @dginhac