PROJECT GIT & GITHUB

Dictionary

- Github is a backup of my timeline !
- Git is my magic timeline, each timeline is refering to one project
- Git can track files in sub-folders, there it is not needed
- Initializing it will create a git repository (timeline) inside of another Git repository

Starting my timeline

with *$ git init*, you initiate an *empty timeline*

git **add** new_file.txt adds a file to a timeline

4 conceptual areas

- Developing area : folder (First_Git_project)
- Local repository : timeline of *The Godfather* subfolder
- Staging area : place we put to organize things BEFORE they go to our timeline
- ???

Saving a point in time

We should commit information and be descriptive about the changes (why it was changed, how this addresses the issue, what are the effects and limitations of this change)

- In the staging area we organize our files before commiting.

  From *developing area*, we **ADD** files to the *staging area*, then from there we **COMMIT** to the *local repository*.

==> It's very helpful to keep track of your changes, especially when you are working on several projects, you will easily forget them !

- What if I don't use -m "nice message" ?

  --> You get an error !

COMMANDS

- *git init*
- *git add*
- *git commit -m* "Meaning full message to explain the changes (why, how, effect(s), limitation(s)"
- *git status* : tells you the status of your git
- *git log* : shows you the history of you timeline (how has been committing, how was it committed, when, ...)

  --> It's useful to have the name and email address of the person who committed, collaboration-wise. This way, you can easily discuss with them to understand and discussed what they did for example.
- *git diff* HEAD^ HEAD -- path : shows you the differences between your current commit and the previous one

  ==> gives the difference **even if the git is not committed yet !**
- *git show* (even simpler, and you don't even need to now the IDs of your commits, if you do flat it will show you the difference between the current one and the previous one, but of course if you know the ID of the commits you want to compare you can choose which commits you want to compare then !)
- *git push* : send to the remote repository
- *git pull* : retrieve from the remote repository to our local

Where is my content ?

To check what is where, we can check the git status : tool that will give you all this information

git status : command to check the status of your project

! you should alway check before commiting !

Vi basic commands

i: to insert text

q: to exit (and save)

q!: to exit without saving

w: to write

Why staging before saving ?

The best way to do it is to organize your files and commit them where they need to be !

--> If you stage all documents at once, you may have insufficient or confuse history

If you stage only one document, you have a repetitive history, easier to understand

--> This way, you don't have too many messages to read

README.TXT

Detailed description of your project and tool usage

.gitignore

List of files that should not be added to repository

- Data files
- Backup files
- Intermediate files

You can use whatever text editor you want btw.

--> Several templates are suggested by github

--> It's also possible to use **regular expressions**. For example, ignore all .csv files, but except one, or make comments, ...

How do we connect to a remote repository ?

By using the command *git remote add*

--> This creates a bridge between our computer and a remote repository, but nothing is shared YET

Once we have commited to **OUR** local repository, we need to **push** to the other repository on GitHub