**Git**

**N.B :**

Index = Staging area

Git configuration :

After installing Git we need to make some configurations : In fact, there are 3 levels of configuration of Git :

* System level : All users
* Global : All repositories of the current user
* Local : The current repository

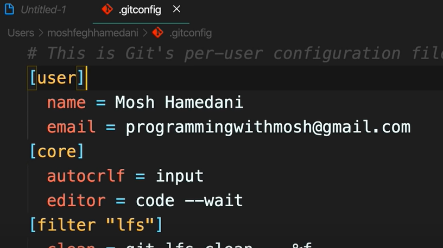
git config –-global user.name “elie saba”

git config –-global user.email [elie.bs.15@hotmail.com](mailto:elie.bs.15@hotmail.com)

git config –-global core.editor “code -–wait”

So the last command will set an editor that we can use to edit some config so here we are saying that our editor will be vs code and then we can launch vs code by using ‘-e’

git config –-global -e



git config –-global core.autocrlf input

Here we are handling the end of lines so for linux we use the value ‘input’

Using git :

mkdir moon

cd moon

git init

So here we created a hidden Git repository inside of our directory ‘moon’. Then we make some modifications in our actual directory, after that we make a snapchot of changed things so these modifications will be added to our git directory and it is done with the ‘commit’ command. However, there is something called staging area which is the intermediaire between really saving our work to our git repository and our actual work (It’s done with the’git add’ command)

**N.B :**

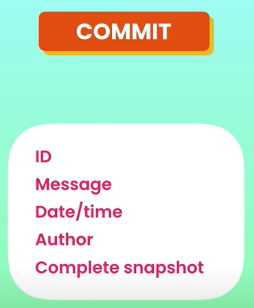
The staging area will not be empty after commiting so if we make chanes for a specific file and we have commited something before so the staging area is not empty, we need to execute the command ‘add file1’ so that we can replace the older version file in the staging area with the new one.

Let’s say we want to delete a file (file2 for example) from our working directory, we delete it but it will still exist in our staging area. So to delete this file we need to execute the command ‘git add file2’. Here git will not find ‘file2’ in our working directory so he will delete it from the staging area.

git status

This command will show us status about our staging area and our commit history.

**What does commit actually stores :**

****

So the commit saves the complete content and not only what was changed. With this it can easily restores the project from a snapchot if something happens.

**How git stores all these contents :**

Git is so efficient because it compress the content and does not store duplicate content.

**Add some files to our git :**

git add \*.txt

Will add all our txt files to git staging area

git add .

Will add everything in our working directory to the staging area

**N.B :**

The staging area will trace our files so for example if we add a file to the staging area and then we modify that file in our working directory and we execute ‘git status’ we will see that there is a file that exists in our staging area but have been modified in our actual working directory.

**Why use a staging area :**

So a staging area is used so that we can review our modifications before actually submitting them to our git repo.

**See staging area present files :**

git ls-files

**Delete file from both actual directory and staging area:**

git rm file1.txt

**Rename a file in our directory and change it also in the staging area:**

git mv file1.txt newfile1.txt

Ignoring files :

While working with git, we do not want to save everything from our project to git because it can be unuseful and storage costy, for that we use .gitignore where we will put the things that we wont save.

.gitignore

**Delete file only from staging area :**

git rm -r –-cached bin/

So here we are deleting bin directory from the staging area but not from our actual directory, if we did not add the parameter ‘cached’ it would also delete it from our working directory.

Review staged area content :

In order to view the content in our staged area before commiting them to the git repo :

git diff –-staged

Link git to github :

mkdir Git

git clone https://github.com/eliesaba/Git.git

git add .

git commit -m 'first commit'

git remote add origin <https://github.com/eliesaba/Git.git>

git branch -M main

git push -u origin main

git push -u origin main