**## Git and GitHub ##**

**\*Git\***

→ **Installing Git :-**

from gitforwindows.org

**#Configure Git**

① user name:-

→ git config -- global user.name “Name”

②Email :-

→ git config --global user. email "Email"

=> Email → should be the same as used

in "github."

3. default editor:-

→ git config --global core.editor "editor Name”

\* Editor Name → Code or notepad or Atom .. etc

=> to test the editor :

Code test.txt

\* To review and Confirm all the changes :-

→ git config --list

Show all the changes in config

→ git config --global --edit

To edit in VSCode.

**#Basic Commands**

→ git help “command”

\* To gett all info about this command

→ git init

\* To initialize git on the selected directory

→ touch filename.filetype

\* To create new file inside the selected directory

→ rm filename.filetype

\* To remove a file inside the selected directory

→ Is & ls -A

\* list items

→ echo "path/to/yourfile.ext" >> .gitignore

**Ex.** echo .env >> .gitignore

\* to stop Git from track the file in the future

→ git status

\* to show the status of repo.

→ git add filename

→ git add \*

\* To add file **or** all files to the staging state

→ echo "path/to/yourfile.ext" >> .gitignore

**Ex.** echo .env >> .gitignore

\* to stop Git from track the file in the future

→ git rm --cached path/to/yourfile.ext

**Ex.** git rm --cached .env

\* --cached tells Git to remove the file only from the index (staging area), not from your working directory (so it stays on disk).

→ git commit -m "Commit"

\* It means that I will leave a message with this commit

→ git commit -am "Delete files"

commit the deletions .. After using git rm or deleting the files manually, you must commit

(If you used git rm, this stages the deletions. If you deleted via your file explorer, use git add -u to stage deleted files before committing.)

→ git log

\* show all the commits & their unique IDs. (Use “q” button to exit the log)

→ git diff

\* show the difference between the last 2 commits

→ git log -1

\* The most recent commit

→ git diff --staged

\* the difference between what are in the staged and previous Commit.

→ git diff “ID” “Commit or File”

\* ID :first 7 letters of the required Commit

**Important Note :**

→ In Commits log from command git log :-

Commit “Unique ID1” (HEAD→Master)

↑ This is “HEAD~0” which means This is the head of all the changes that we've made.

Commit “Unique ID2”

↑ This is “HEAD~1” after “HEAD~0”

Commit “Unique ID3”

↑ This is “HEAD~2” from “HEAD~0”

.

.

etc.

لذلك للرجوع لأي Commit فيهم يكون كالآتي :

→ Checkout HEAD~2 filename.filetype

يعود بالملف لـ Commit الموجود في HEAD~2

→ Checkout HEAD filename.filetype

Return to the latest version of the file.

In other words : return to HEAD~0

**#What If we made a mistake ?!**

1. If we put some wrong modifications

of some files into staging state . We use the Command :

→ git reset filename.filetype

To have unstaged those Changes

2. If we committed the wrong staging files . We use the Command :

→ git reset HEAD~1 --one of the following

• soft OR • mixed OR • hard

And the differences will be as the following :

• soft

- Removes the last Commit

- don't change what happened in the staging.

• mixed

- Removes the last Commit

- makes the Changes unstaged

• hard

- Removes the last Commit

- Removes the changes we've added to the file

**#Branching**

→ git branch “branch\_name”

\* Create a new branch with the selected name.

→ git branch

\* show all the branches exists

→ git checkout “branch\_name”

\* change the branch and select the requested branch.

→ git branch -d branch\_name

→ git branch -D branch\_name

\* Use -d to delete the branch only if it has been fully merged.

\* Use -D (capital D) to force delete, even if it hasn’t been merged

→ git push origin --delete branch\_name

→ git push origin :branch\_name (older syntax)

\* Delete a remote branch.

**#Merging**

→ git merge “branch\_name”

\* To merge the selected “branch\_name” with the master branch.

\* We must be in the master brunch before we enter this command.

**\*GitHub\***

After we create GitHub repo we took the remote repo’s url and run the following command :

→ git branch -M main

\* To make all the commit on “main” branch

→ git remote add origin “Remote\_URL”

\* To link the local repo with the alias origin that directly refer to the remote URL

→ git remote -v

\* To get more info.

→ git push -u origin main

\* To send “push” the local repo into "origin"

alias from “main” brunch

→ git push origin master

\* To send “push” the local repo into "origin"

alias from master brunch

**\*GitClone\***

→ git clone “Remote\_URL”

\* To download the entire remote repo from

URL into the selected directory.