# GIT & GITHUB

*) Git & Github both are open-source platform in which storing, restoring, transferring of source code occur.

*) Both allows to maintain history of the time/ code of a project.

→ GIT :- is a version control system that lets you manage & keep track of your source group code history.

→ GITHUB :- It is a cloud based hosting service that lets you manage git repositories.

*) Version control, Also known as source control, is the practice of tracking and managing changes to softwar code / source code.

*) Version control systems are software tools that helps software teams manage changes to source code over-time

*) Types of Version control :-

→ Git version control [Centralized repos Ver control]

→ Team foundation version control [TFVC]

↳ Distributed repo version Control

→ In CLI we use "git init" to initialize the repository.
→ The "git init" repo will not be physically present its hidden
→ To check hidden repo "ls -a" → "ls .git" [-a = all]
→ to creat a new file using terminal "touch name.txt"
→ To check the history of the changes that were made is "git status"
→ "git status" shows the untraced/local change made in a repository. Also used to check modified files.
→ To make the changes or add an commit permanently we use "git add ."
→ The "." in the command used to get changes from current directory [adds untraced changes]
→ for a particular file we use "git add names.txt" for a file from untraced history to change permanently.
→ To finalise the permanent untraced history/file that were added we use "git commit -m "added names.txt"
→ "-m" indicates the message with the commit
→ to go into the file "vi name.txt" [vi = visit] allows to make changes in the file
→ To list/view the content of file "cat name.txt"
→ To exit from visit [vi] mode press enter↵ and to save change we write ":wq" or to unsave changes ":q"

→ To add modified file "git status" → untraced/hidden file
"git add ." → permanent commit
"git status" → check status

→ To undo the permanent commit/change we use "git restore --staged 'names.txt' or '.'"

→ To finalise the commit permanently "git commit -m "modified the names.txt"

→ To check entire history of commit/changes in the project "git log"

→ To delete the file "rm -rf names.txt"

→ To delete the history of the commit "git reset ----"

→ '----' indicates the hash key of the commit in which the they you copy except that previous history will be deleted.

→ To get the changes which were not commited / restore/add the new commit without effecting the real change/commit

→ "git stash" to store temporarily or not to lose the changes

→ To check the temporary commit stored commit "git stash pop"

Also to delete the temporary stored commit "git stash clear"

→ To add a repository from the github website/any URL that should be attached to the local project

" git remote ~~origin~~ add origin 'origin URL' "

- remote :- working with URL's indication.
- add :- adding a github repo URL.
- origin :- name of the URL.

→ To shared the changes/commit to the github repo

" git push origin master"

- push :- insert a file or a commit
- origin :- shows the path
- master :- Branch of repo

→ The branches are used to create a separate node or part a repo. There are ~~four~~ no limits to add a branch

→ for example master → The recent finalised branch

root → The origin where it started

test → The testing branch etc. —

→ To add a branch " git branch feature " [feature = name]

→ The " git checkout feature " that means the */head is pointing to feature branch. [head will now come on the specified branch]

→ head is just a pointer that says that any ~~the~~ new commits that you make will be added to the head branch.

→ You should not commit direct changes to the main branch / master because it might have few bugs that were not found.

→ To make changes side by side exit from main branch using "git checkout main" and add new commit

→ To merge the branches "git merge feature-main"
[feature is a branch merged to main branch]

→ To add the file in your local project or to clone
"git clone URL-of the project"

→ Fork :- It is used to get a copy of a project from a repo to your own account. As you are being the origin of the project.

→ To make change in someone else's account we fork the project to our account

→ The changes made in the fork project will not effect the main project until or unless we get approved from the origin project using "pull request"

→ The place were you forked the project that is called upstream by a convention. → [Agreement]

→ To check the original project or to add the upstream URL "git remote add upstream 'URL'"

→ It is also called fetching a project from origin.

→ pull request :- It is the request to the origin project to merge the modified/changed commit from your forked branch to main origin branch.

→ When the pull request is merged and approved than it will reflect on the origin branch which you've been contributor.

→ origin is your own account branch
  ↳ upstream is forked account branch

→ Never commit on main branch because it will not allow you to ~~a~~ crate a new pull request, It will add commits to the branch

→ Create a pull request for every commit that is made.

→ "If a branch is already has a pull request associated with the main branch or any other branch then it will not allow you to create a new pull request. It will just make changes in commit section."

→ In simple terms one branch can only open one pull request. &

→ Any particular new thing that your working on create a new branch in your local folder & then make a pull request from that.

*→ "git push origin rishabh -f" "also add -u"

↳ push = pushing the commit
↳ origin = to branch
↳ rishabh = from branch changes in
↳ -f = force push to make ^ online repo contain a commit that local repo does not

→ commits are interlinked in a branch

→ The changes that you make in your branch can be merged to main branch by using "merge commit"

→ To maintain the commit [changes] done in a main branch to someone else's branch is also the same in your forked branch we use "fetch upstream"

→ To get updates from someone else's main branch to your own forked branch we use fetch upstream and also with few manual commands.

*) "git fetch --all --prune"

↳ fetch - to get the changes fetched

↳ --all - all the commits that were change

↳ --prune - gets the deleted commits to fetch.

→ To reset the main branch of my origin to the main branch of upstream. we use "git reset --hard upstream/main"

*) "git pull upstream main" It does the ~~the~~ ~~main~~ thingh internally as the • fetch command does

→ To merge all the commits made into on we us

" git rebase -i 'URL from log'"

↳ rebase = merge commits

↳ -i = interactive environment

→ Squash is commit at staged/untraced area will help you to merge it in previous commit. of the pick. [merge with above commit]

→ pick is a commit at staged/untraced area will help you to make a parmanent commit [Taking a commit]

→ The delete the file/commit we use

" git reset --hard 'URL ~~of~~ from log'"

↳ reset - delete but store in local pc

↳ --hard - delete from local pc as well

→ Merge conflict:- It is the situation where the owner of the main branch has to resolve the commit made by two different account in there own branch with a pull request to commit the change

→ Merge conflict has to be resolve manually.

→ To configure the git account in CLI/Bash via username

"git config --global username "rishabh.Sharma""

→ To configure the git account in CLI/Bash via email

"git Config --global user:email "rishab.sharma@---.com""

→ To edit the configuration

"git config --global --edit"

→ If you don't want the file to send it to the git echosystem or you don't want git to track the commit/ file we use gitignore

→ Sensitive files such as Keys/password can be stored in gitignore "touch .gitignore"

→ To show the contents of the branch we use

"git show master"

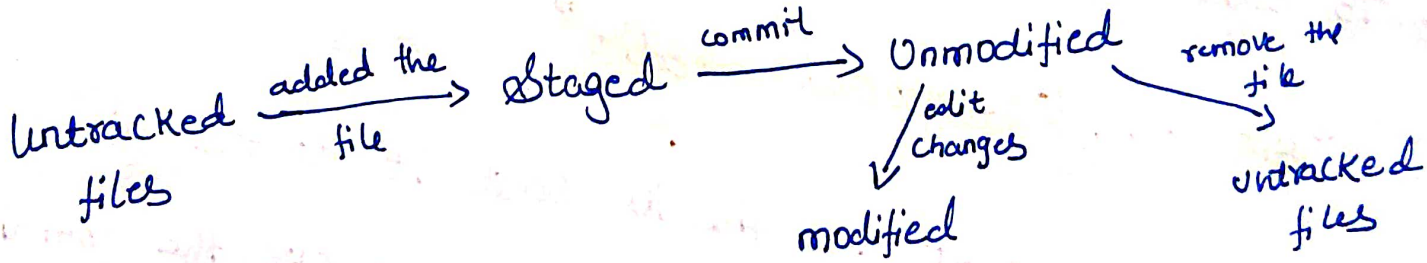
↳ show = view

↳ master = name of the branch.

→ open source contribution steps

1) fork it in your branch
2) clone in your bash/cmd/IDE
3) changes to the file
4) push to the branch from local system
5) pull request to the upstream owner
6) merge owner merges the changes made.

*) git hierichy.




- Untracked → The file are not tracked by git.
- Staged → we directly commit the file.
- Modified → To apply changes to file.
- Unmodified → We are ready to commit

# git command step-by-step.

1y ⟶ $ git init (initialize repo)

2y ⟶ $ git status

3y ⟶ $ git add index.html [file gone to staging area]

4y ⟶ $ git commit -m "added html" [initialized commit]

5y ⟶ $ Vi index.html [vim editor]

6y ⟶ $ esc key → :wq to exit the Vim

7y ⟶ $ git status (working tree clean all changes committed)

8y ⟶ $ touch hi.txt [touch to create empty file]

9y ⟶ $ touch add.Java [.java file created]

10y ⟶ $ touch Hello.HTML [HTML file created]

↳ Now 3 files are untracked, so to track all three files we would use

11y ⟶ $ git add -A [to push all file -a]

12y ⟶ $ git commit -m "added two files"

13y ⟶ $ git status [to check the staged area]

14y ⟶ $ git clear [to clear cmd]

15) → $ git log [to check the updates].

16) → $ git -diff [to know the differences]

17) → $ touch .gitignore [store in unstaged area]

18) → $ git rm hi.txt [to delete a file]

19) → $ git branch [shows all the branch with pointer]

20) → $ git branch test [create a branch]

21) → $ git checkout test [moves to pointer to the branch]

22) → $ git merge demo [merge 2 branch files]

23) → $ git checkout -b demo [create another branch]

24) → $ git clone 'URL' [clone a repo]

25) → $ git remote add origin 'URL'

↳ helps to add repo from local system to github.

26) → $ git push origin master

↳ push the changes from local to master branch

27) → $ git push origin -u test [-f = force push]

↳ push the changes to another branch

** After all these steps you can pull request to the original owner to merge the changes made

28) → $ git reset --hard "URL from log"

↳ deletes the files from local as well.