GIT AND GITHUB

Git (Version control System) → It tracks the history of changes as people and team collaborated on project together. Any version of the code can recover anytime.

↳ which, who, when, why changes were made

Github → It is a web platform which is used to manage the projects. All the features of git were there in github.

Git Commands

- **pwd** :- It shows the current path of the terminal.
- **Cd (change Directory)** :- It is used to change the path of the terminal.
- **Ls** :- It list down all the files and directory in the folder.
- **mkdir <folder name>** :- It is used to make directory in the folder.
- **Ls -al** :- It will show all the hidden files and directory which was previously not shown by normal Ls command.
- **git status** :- It shows the status of changes as untracked modified or staged.
- **git init** :- It make a brand new git respository by simply converting a normal folder into git repo. It adds hidden (.git) file which stores all the meta data of git repo.
- **clear** :- It is used to clear the terminal.
- **cd..** :- It is used to move backward to previous folder.
- **git clone <address> <new folder name>** :- By this we can clone a git repository of anyone and save that git repo to our local machine.

  Ex → git clone https://github.com/[illegible] Tanish new
  address / Link — new folder name.

* If I do any file changes then run **git status** then it will display all the files in which the changes were made.
* If I want to see the changes then I can use **git diff**
  - **git diff** :- It will display all the changes at once.

    Ex → git diff — It display changes of all the files latest.

    or git diff index.html — display changes of a particular file.

    [Red one → old code
    Green one → new code]

- **git add** :- Before commiting file you have to stage the change. So git add helps us to stage the changes.
  - → new file → add file to track
  - → old file → staging index

  Ex → git add <file_name>
  To add particular file.
  OR
  git add --all
  To add all the file.

- **git commit** :- It saves the snapshot of project history and completes the change tracking process.

  Ex → git commit -m "message"

* If you want to see all the changes history of the git repo.

- **git log** :- It will display all the commits. (9,)

  Ex → git log
  It display all the commits.

  or git log -3
  It display last 3 commits.

  or git log -3 -p
  It display the changes of last 3 commits,

  git log --oneline
  It only display the commit ID and commit msg only.

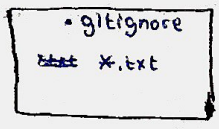
  git log --stat
  It will display all the file name in which the changes were made.

  git show <commit id>
  It will show the changes in a particular id.

* If you want to restore the code to previous version. but the code was restore before commiting it. then,

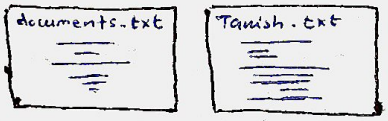
- **git restore <file_name>** :- It will restore the code ~~the~~ to older version. But the code can restore ~~if the~~ before commiting

* If you want to ignore any type of file. like docx, txt, ... etc. Then simply create a file name `.gitignore`. And in .gitignore file write `*.txt`. By this all the .txt files in the folder were not were the part of commits.



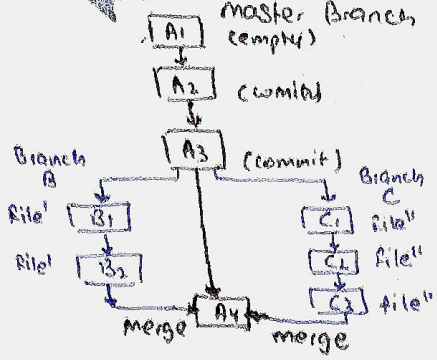
*.txt
To ignore all the txt files

Tanish.txt
To ignore a specific file



* After that you have to add and commit the .gitignore file also.

Branching →



- git branch → It shows the branch on which we are currently in.
- git branch <branch_name> → By this we can create a new branch.
- git checkout <branch_name> → By this we can switch the Branch.

* In different branches you have to do commit and add things also.
* After the feature was developed on the branch. Then merge the branch.

- git merge <branch-name> :- By this we can merge the branch with the master branch. But the catch was that if you want to merge 'Branch B' with master branch then the terminal be on the master branch and then run the command "git merge Branch B".

- git branch -d <branch_name> :- It is used to delete the branch when the branch is not longer needed or after merging the branch with master.

* If conflict was there after adding them simply solve in the master branch and add and commit the master branch.

TAGGING ⇒ It is mainly use for tagging the beta.

- git tag -a <name> <lastest master id> -m "<message>"
  betaV1.0 d285ae6285ed.... my beta Release.

⇒ By this command we can tag any commit.

- git tag -d <name> :- By this we can remove the tag name.
  betaV1.0