INTRO:

→ Git : git is a version control tool. (software)

→ Github : github is a hosting service of git repository.
- it is very usefull for developers to collabration in projects.

→ Repository : repository is a folder with additional capability to track all the file (to watch the folder) with the help of git tool.

→ AFTER Installation :
commands in ☐ and file name ☐

`git --version`	to check whether git installed or not and it shows git's version

```
git config --global user.name "github username"
git config --global user.email "regi. emailid"
```

```
git config --list
```

`pwd` to see current directory

`cd folder` change directory (to change path)

`ls` to list folder's items

`git status` it shows current git repo's status

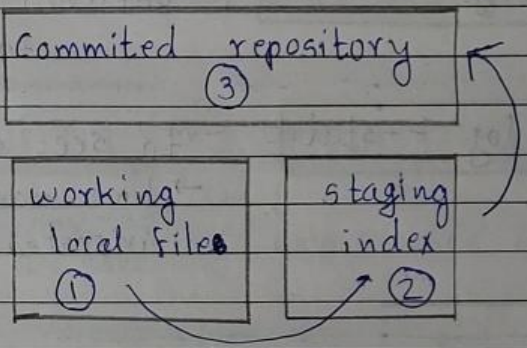
`git init` to convert a normal folder to git repository
↳ a hidden .git folder will insert in folder to track files.

→ git cloning : means github's repo in our local workspace.

`git clone` `github's repo link` `folder name (we want)`

`git diff` to see what we have changed

→ LIFE CYCLE of a CHANGE :



`git log` show all commits from top (latest) to bottom.
→ we can `enter` to see more...
→ and `q` to quit.

`git log -n` to see latest n commits
→ n must be digit (int)

`git log -p` it show commits with git diff.

`git log --oneline` — oneliner breif log inform. to quick access.

`git log --stat` — to see statistic of lines → how many line of code inserted and deleted.

`git show [SHA id]` , or you can say commit id
↓
to see changes in particular commit.

→ COMMITING THE FILES :-

`mkdir [folder name]` — to make a new folder

`git init` — making folder into git repo

`git status` — to see git repo's status

now,
↳ make file in that folder (repo)

`git status` → again, to see that file.

`git add [file]` → to let file go in stagging index.
file name that we gave
you can see git status.

`git commit -m "[message]"` → to commit the file.
but, we haven't done any changes

now,
↳ let's do changes in file. then

`git status` → to see modified or not.

`git diff [file]` → it shows what changes has done.

`git add [file]` → ① if file is new (fist time code) add file to track
↳ ② if file is old (modified code) → stagging index

`git add file` → then see, `git status`

↓ then

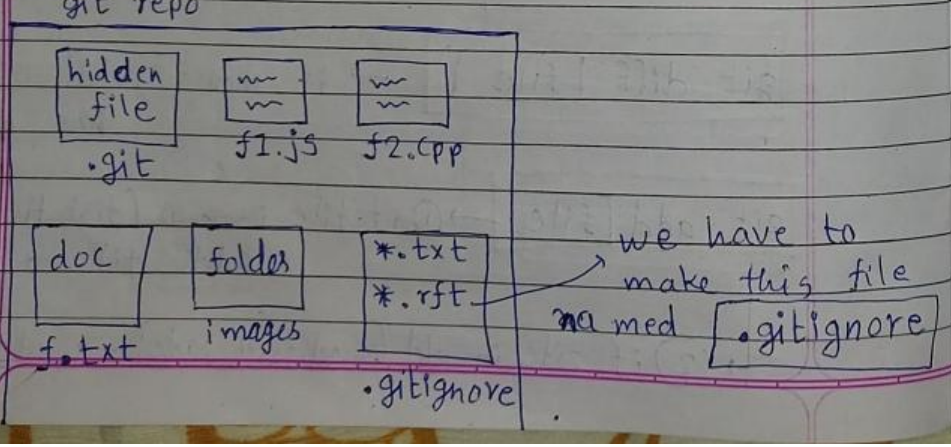
`git commit -m "message"`

↳ file changes has been commited

`git log` to see both 2 commits. our

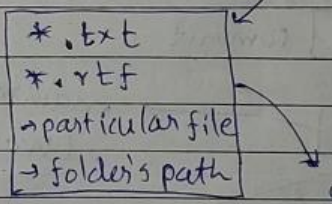
→ if I have done changes by mistake (unwanted) and haven't commited then,

`git restore file` put changes to lastest commit.



→ git repo has various files
- ↳ .git (hidden file that track all files)
- ↳ main codes to track
  - ↳ .js
  - ↳ .cpp
  - ↳ .py
- ↳ some file that we don't want to track but imp. for documentation purpose.
  - ↳ .txt , .rtf
  - ↳ images

↳ .gitignore file we make to avoid that unwanted shit that we don't want to track



git will ignore extantion file that mentioned in this and particular file and folder that are in this .gitignore file

↳ commands to do this is as followed:

→ make .gitignore file in repo

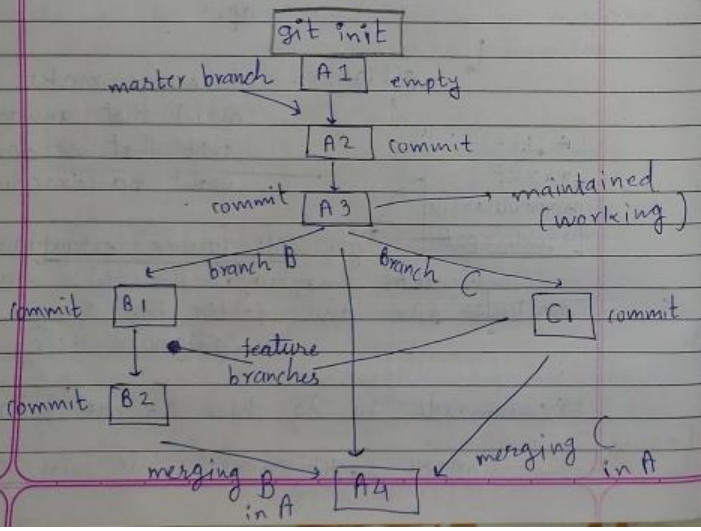
`git add .gitignore`

`git commit -m "adding .gitignore"`

`git status` } will not show (won't track)
`git log` } unwanted files.

and move HEAD to latest commit.

→ BRANCHING and MERGING:



`git branch` to see which (how many) branches are there and which is my current branch.

`git branch feature 1` to create new branch.

`git checkout feature 1` move to others (feature1) branch (current → others)

`git branch` to see current branch

now ↳ let's add feature 1 in file. (modify code)

`git status` → to see just sanity.

now,

`git add file` stagging index

`git commit -m "feature 1 added"` final commit

`git log` → to see changes
HEAD is moved to feature 1 brand from master.

`git checkout master` switch to master

`git checkout -b feature 2`

this will create new branch (feature 2) and shift it to current branch togather.

`git log` you can see new branch.

now, ↳ let add feature 2 code in file.

process to commit :-
status → add file (stge) → commit chang → log

`git status`

`git add file`

`git commit -m "feature 2 added"` final commit

`git branch` → * feature 2 (active)
master
feature 1

→ let merge feature 2

`git checkout master` switching to master

`git merge feature 2` branch to merge with master

`git log` you can see feature 2 added

↳ after this we can delet feature 2 branch

`git branch -d feature 2` → branch name that we want to delet

`git branch` to see whether it is deleted or not.

→ let merge feature 1

`git status`

`git merge feature 1` Automatic merge failed. because, same line of code conflicts.

`git status` → it will show merge conflicts.

→ now, we need to resolve conflict in our file code (do edit file)

`git status`

`git commit -m "message"`

again, throw error because, we haven't tell that we have resolved

→ we can tell git that it's resolved by moving it to stagging index.

`git add file`

`git status` to see changes to be commited

`git commit -m "feature 1 added in master"`

merged to master (by final commit)

`git log` → to see merge ids

→ merged all branches

→ TAGGING :

→ tagging: we can tag a particular SHA id (commit id) to beta release.

`git tag -a betaV1.0 SHA id -m "message"`

`git log`

↓

it will show commit : ID (HEAD → master, tag : betaV1.0)

now → let, add new feature feature 2 in file (edit code)

then,

`git status`

process: add ↓ commit

`git commit -am "feature 3 added"`

to add (move to stagging index) and commit togethes

`git log` show that feature 3 added

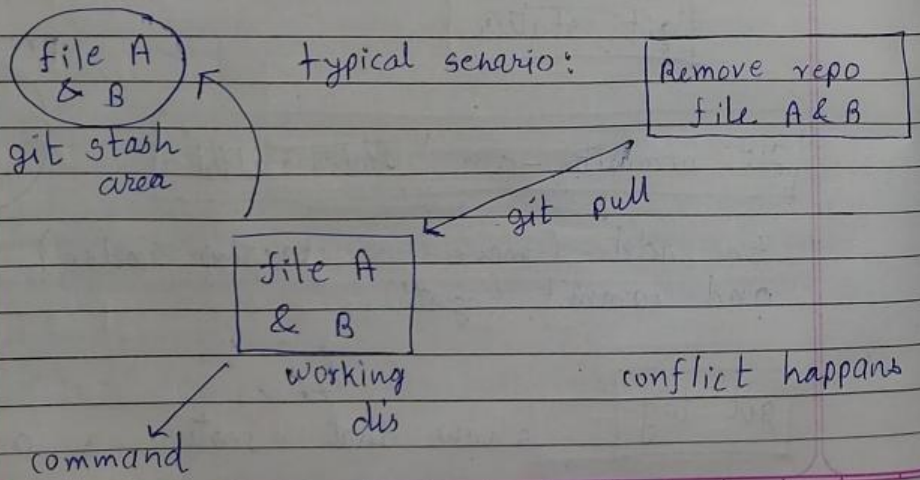
→ master moved to latest and betaV1.0 tag remain on that SHA id.

~~{git tag}~~ `git tag -d betaV1.0`

to remove betaV1.0 tag

`git log` → to see that ~~tag~~ tag is removed

like wise, we can give master branch tag...
(means, we can give tag to master branch also)



→ git stash :-

git pull : update local repo to github repo (latest)

1. clone repo from github to local desktop
2. fellow progsam changed repo on remote (github)
3. `git log` → I don't have this latest change.
4. I changed code on same line (on file)

`git status` I changed

`git pull` won't work because of

② options → my changes → commit for your

follow prog's changes

`git stash` → my changes gone to a stash area

`git stash list` to see SHA id of my changes

`git pull` latest value will be your fellow progsamers

now,

`git stash apply` → to apply my changes

↳ merge conflict happens if same line of code changes.

now, → chang code (modify file)

`git status`

`git add file` stgging index

`git commit -m "message"` commit your code

`git log` to see your commits

`git push -u origin master`

to push changes on github (remote)

↳ it will ask for Authorization username :—
pass :—

Undo commits

- `git reset` → delet commit (dangerous)
- `git revert` → revert given commit
- `git commit --amend` → ammend the most recent (HEAD) commit

`git status`

`git log`

→ do changes in file

`git status`

`git commit -am "message"`

add and commit togathes

`git log` → to see commit

`git revert SHA id` to revert this commit

ESC : wq enter

① ② ③ ④

`git log` undone the latest commit.

`git reset --soft SHA id` → that we want as a latest

`git log` latest commit will be this

working directory changes file 1, 2

--soft → local changes will show as stage
--mixed → as modification
--hard → local changes will discard
↳ avoid

`git log`

→ file changes in code for feature 4

`git status`

`git commit -am "message"` for undo message

added ~~feature 4~~ feature 5 by mistake

`git log`

`git diff ID` or `git diff HEAD` / `git diff HEAD~`

`git commit --ammend`

press (i) for insert

feature 5 → feature 4

ESC : wq enter

(1) (2) (3) (4)

`git log`

→ GUI : Graphical Users Interface (drag & drop)

→ CLI : Command Line Interface (write in shell)

`cd` : to change directory

`mkdir` : to make directory in that folder

`git init` → ~~make~~ convert folder in to git repo

let → make file named README.md → in repo
↳ to write discription

`git add README.md`

`git commit -m "initial commit"`

to map local branch to remote (prime) brance

`git remote add origin` link that github provides

`git status`

`git log`

`git config --global user.name "github username"`

`git config --global user.email "registered email"`

`git config --list` → to see this

flow →

working directory changes → stagging → commits → PUSH → REMOT (github)

`git push -u origin master`

1. github acc.
2. empty repo.
3. local system
   - git config --global user.name "__"
   - " " " user.email "__"
4. git push -u origin master

   ERROR

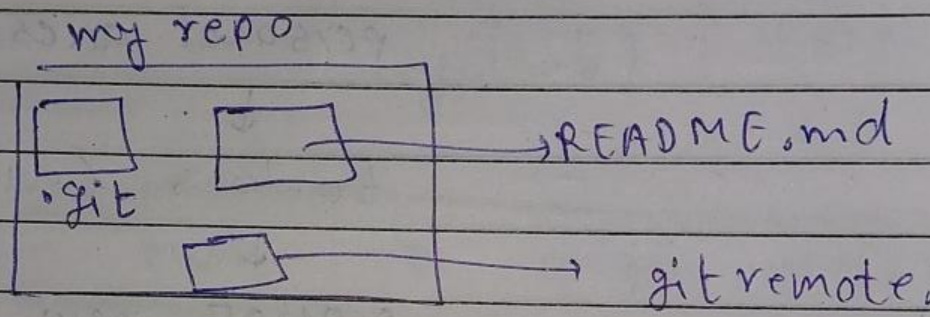
4.1 username ? → your github name

4.2 password ? → github setting
↓
developer setting
↓
personal access token
↓
tokens (classics)
↓
generate new token
↓
generate new token (classic)
↓
notes
↓
access token
↓
no expiration
↓
☑ check all
↓
generate token
↓
copy token (password)

Date
Page

let's do again ex

myrepo git repository



`git status`

`git add file` → gitremote.txt

`git commit -m "message"` commited at local

`git log` to see local commit

`git push -u origin master`

↓

pushed to remote (github)