Git & GitHub

**Basic Steps Handling local repos**

1. Create a local repo in the folder
2. Stage files in it
3. Commit files

***git init***

to initialize the git at the current directory

creates a hidden .git folder that will keep track of you files, changes, edits.

the first stage is always to initialize an empty git repository

***git status***

to check the commit status of the repo

it will show the changed, unstaged files

***git add <name> || git add –a || git add .***

add files to staging area

only staged items can be committed, when you do ‘git commit’ all the staged files get committed

\*you might wonder why do I need to stage first? Direct commit kem no karai? Wont it be easier if git worked like that? While that can be done, its not advisable

Suppose you’re working on a app, you need to add a feature and improve the overall UI, you successfully upgrade the UI but the feature you’ve been trying to add is not happening from you, now you think since the UI is already done you might as well push it to everyone’s devices but if there was a direct commit how would you choose specific files to be pushed, you can say one could’ve specified all the files for commit, but what if you had hundreds or thousands of files, thankfully git thought of that you can just stage the UI file and commit it alone, leaving the incomplete feature file for later.\*

***\*committing files directly without staging them\****

git commit -a -m “message”

-a = all files, -m = message

Note: only tracked files can be committed this way

***touch (Linux command actually)***

to create files in current directory

e.g. touch hello.txt

touch 27May.log

***.gitignore***

Used to ignore certain files in git, prevent them from popping up in unstaged status

\*For eg. In a program there might me certain log files that are generated during execution, these files are of no use to be tracked and being large if done so will consume resources, we also don’t want to show up as unstaged every time we do git status, this is where .gitignore comes in.\*

.gitignore is a file rather than command, you can create a .gitignore file using the touch command and inside that file write whatever you want to ignore like “\*.log” or “Hello.txt” or “lib/res/”. Here all the files with .log extension, Hello.txt and everything in lib/res will be ignored.

***git rm <filename> (Linux command without the git in begining)***

to remove a file

use with git at beginning to remove file with help of git (if you do this git will automatically stage It, you won’t need to explicitly stage the changes)

***git mv <source> <Destination>(Linux command without the git in beginning)***

the move command is the same as rename in Linux, if you do

*$ git mv test.txt lib/*

It will move the test.txt to lib folder of same working directory

But if you do

*$ git mv test.txt hello.txt*

It will rename the text.txt to hello.txt

(in some ways what it did was cut the test.txt file and pasted it into hello.txt)

\*Important case study : let’s say you are tracking all the files in a folder, your tree is up to date and there’s nothing to merge, now if you add a .gitignore and add a hello.txt file to it, later change its contents on late git status you’ll find that it will show up as modified, this might not make any sense as you just added it to git ignore, avu etle thyu karanke pellathi e track to thatij ti, pachi apde gitignore kari but e file to pan track thatij rese jya sudhi apde explicitly ene git na tracking system mathi nai kadiye, so how to do that…\*

***Removing a file from git tracking system***

git rm –cached <filename>

this command will remove the file from tracking and will no longer conflict in .gitignore

***git clone***

to clone the given repository at the current directory

***git diff <filename> || git diff || git diff --staged***

to check what changes occurred in a file

to see difference between working area and staging area

to see difference between last commit and staged area (all 3 respectively)

***git log***

to see all the changes made

*git log -5*

to see last five changes

***git branch navibranchnunam***

creates new branch named navibranchnunam

***git checkout -b navibranchnunam***

creates new branch named navibranchnunam and navigates to it \* -b jose eni vagar nai thai \*

***git checkout master***

switches to master branch

\*important: je branch ma apde already hoi tyathi te branch delete no thai biji branch ma javu pade for example to hu navibranchnunam ma hoi to ty thi hu 'git checkout -b navibranchnunam' karin e branch delete no kari saku, delete karva mate mare master athva biji koi branch ma pela navigate thaviu pade\*

***git branch -d navibranchnunam***

deletes branch named navibranchnunam

***git commit -m "message"***

commits to current branch with the following comment

\*important jo navi branch ne master branch thi merge karravi hoi to pela navigate to master branch amd merge navibranchnunam, undhu nai chale\*

***git merge navibranchnunam***

merges the current branch with navibranchnunam

query: git reset (hard/soft), git restore, git pull, git push

Uploading local repo to GitHub

Step 1: create new repository

Step 2: copy the url of repo

Step 3: link your local repo to github repo, fire command

*git remote add <Branch\_name> <REMOTE\_URL>*

*i.e. $ git remote add origin https://github.com/octocat/Spoon-Knife.git*

Step 4: push the local repo to github branch now by

*git push <Branch\_name> <Remote\_branch\_name>*

*i.e. $ git push origin master*