git

GitHub

Git and GitHub

Hello there! Git

Git is a VCS to track the changes in the source code.

[Version Control System]

Github

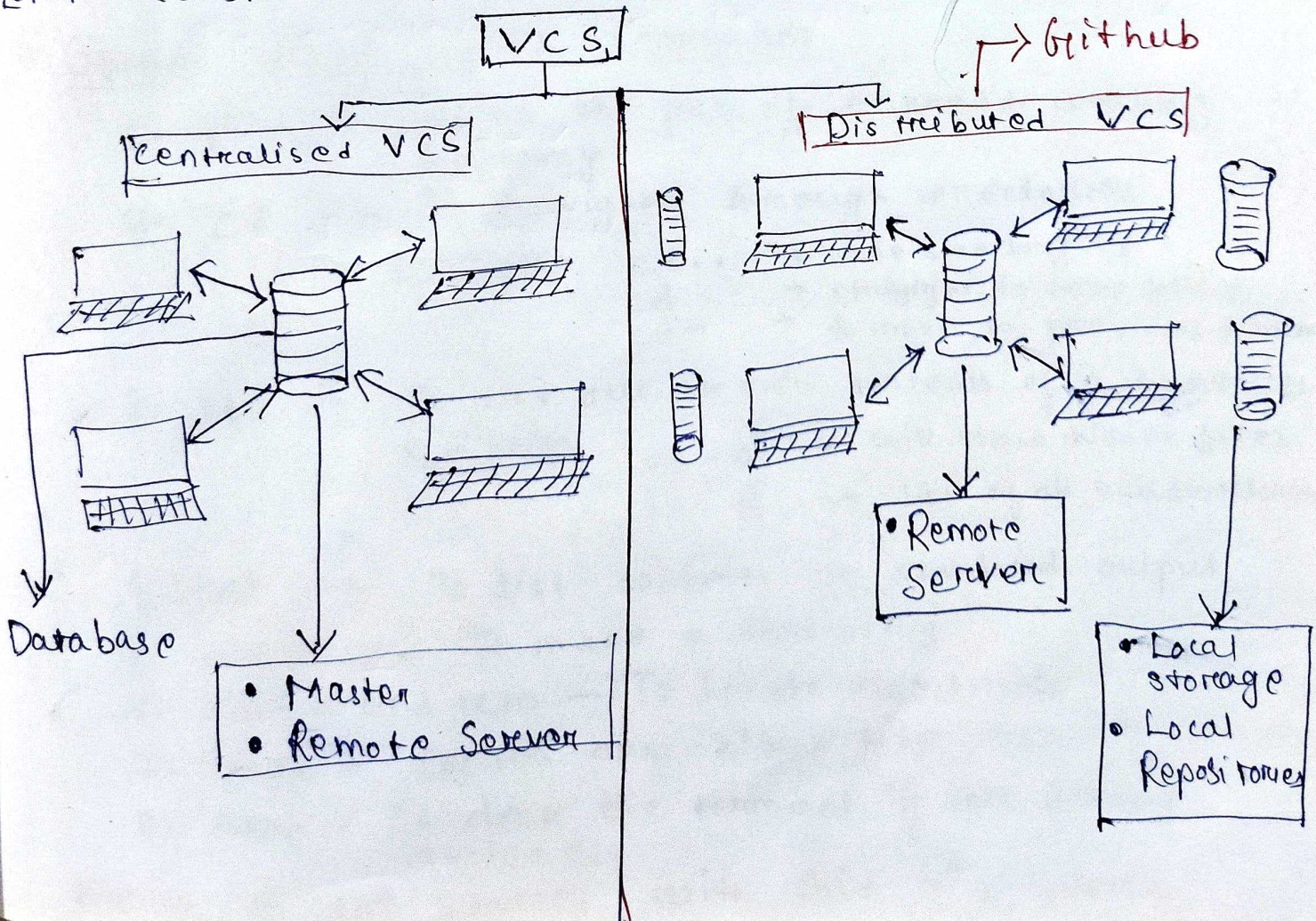
Github is a web-based cloud service to host your source code (Git repositories).

(a folder where all the changes are saved)

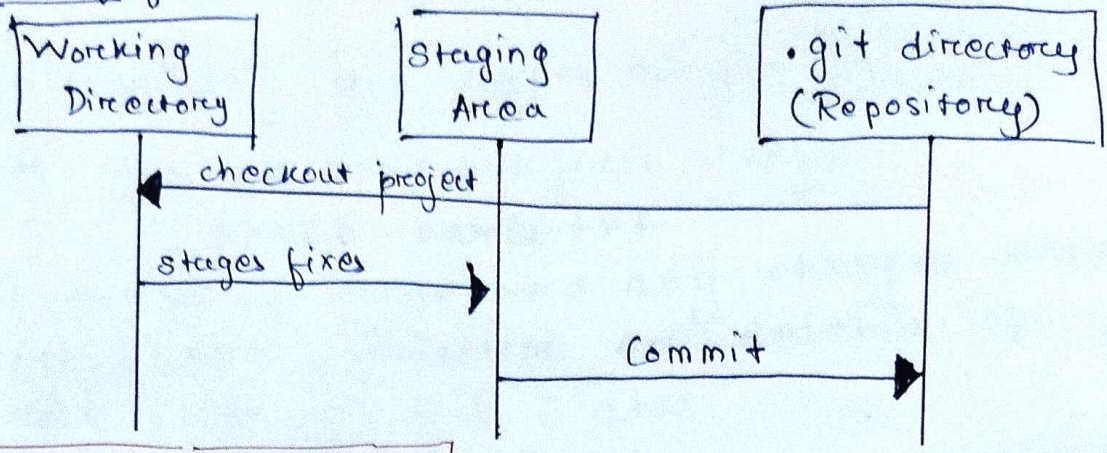
So, basically, there are so many other online platforms that allow us to host these folders, repositories or our projects online. So that other people from around the world can 'share', look and contribute.

Eg:- Github, Bit Bucket and GitLab.

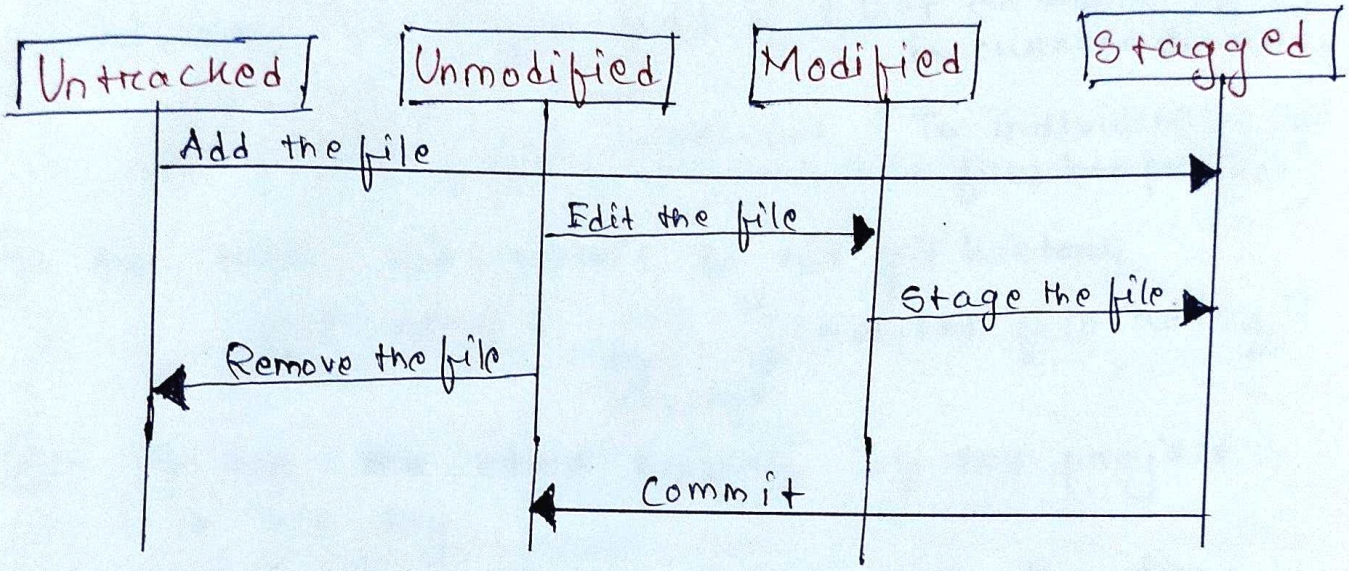
[Git doesn't require GitHub but GitHub requires Git]



3 stages of file in GIT:-



lifecycle of git files:-



Some Basic Linux Commands

1. pwd - To find the path of current working directory
2. cd - To navigate through directories.
   shortcuts - cd.. → one directory up
   cd → straight to home folder.
   cd - → to move to previous directory
3. ls - To list out or view contents of a directory
   variations ls -a → will show hidden files
   ls -R → list of all subdirectories
4. cat - To list contents on standard output.
5. mkdir - To make a directory
6. rmdir and rm - To delete directories
7. touch - Creates new blank file.
8. clear - To clear the terminal if it's getting cluttered.

Enough to get started with Git and GitHub.

Git Commands (In red)

1. Initializing git repository → git init
2. Lets say we make our first change.
   touch . names.txt
   To check if there are any changes like modification, addition or deletion of any files - git status
3. To include these changes in the project history - Staging first change →
   - git add . (Dot means everything in current directory)
   - git add names.txt (To individually add files into project)
4. And then we commit to the git history.
   git commit -m "names.txt file added"
   (-m → message)
5. To see the entire history of the project
   - git log

   This displays the author and the time at which the commit was made.
6. Removing a commit from the history of a project
   Each commit has an id and commits are build on top of each other. we can unstage or remove commits
   - The commit's id which we copy whatever the commits above it will get removed

   git reset [the commit id]

(7) Let's say you are making a project and you are working on a few lines of code and you want to try out something new
- You basically want to save your work somewhere else without making a history in the project and whenever you want to get it back you can.

git stash

↓ Basically putting the work in backstage.

(8) To bring the back stage works to staging area

git stash pop

(9) git stash clear to clear all the back stage works

CONNECTING REMOTE REPOSITORY TO LOCAL REPO

When you create a new repository you get an url for your repository.

↓ Attach that url in your local project ↓

(10) git remote add origin https://github....

(11) git remote -v
This will show all url attached to the folder.

(12) git push origin master

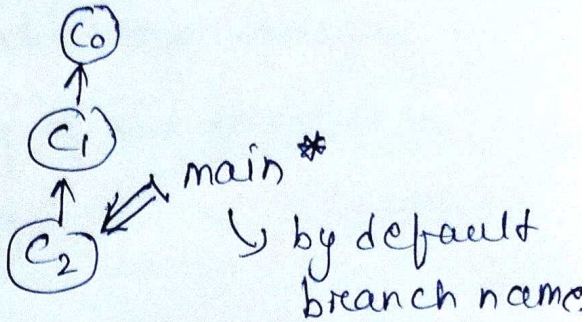
- push the changes
- url to which you want to push
- to which branch you want to push.

Branching to the main

Commits are arrange in a branch structure.

Internally Branches are

Directed Acyclic Graphs

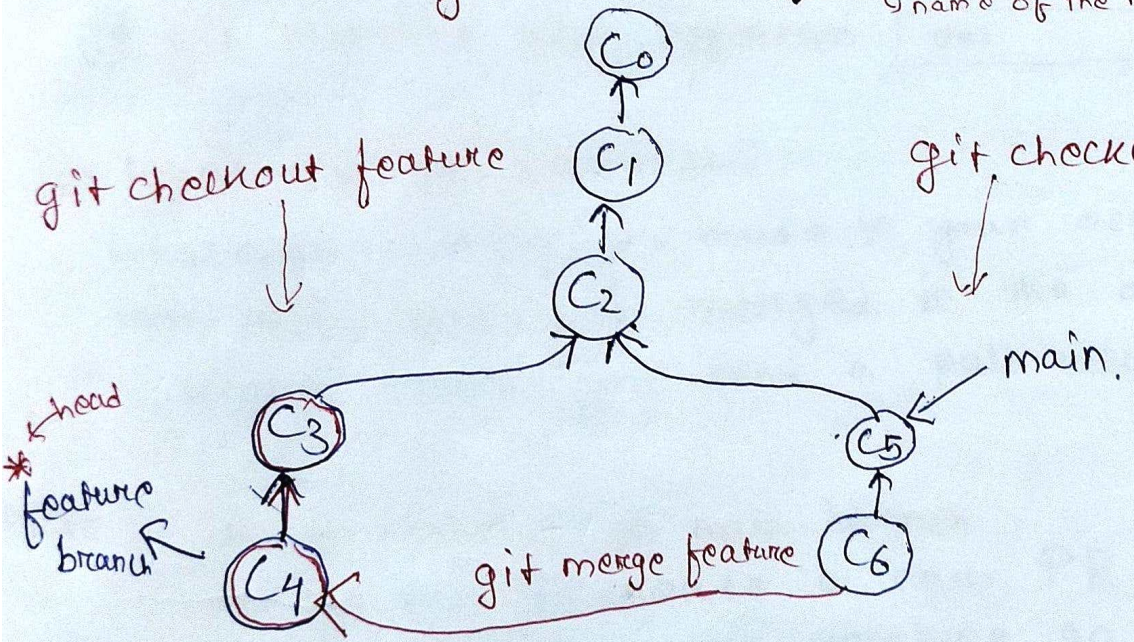


Use of branch

whenever we are working working on a project and you are resolving a bug or creating a new feature always create a separate branch.

(13) To create new branch

```
git branch feature
          └ name of the branch.
```



(14) git checkout feature

→ brings the head to the branch feature

(15) git merge feature

↓

Merges feature branch with main branch.

Working on existing Projects on GitHub

If we want to make any changes on existing GitHub Projects we cannot directly make any changes

1st we have to create a copy of the project in our own account

or

it is called Forking (cloning)

Why should we FORK?

Forking creates a clone in our account and now we can do anything we want with it.

(16) git clone [ ] ↓ url.

What is upstream and adding it to local?

From where we have forked a project or the original project is called an upstream url.

(17) git remote add upstream [url]

What is Pull request?

whatever changes you made in your account will only get merged in the original account when you send a pull request

Note :- Never commit on main branch

Always try to create a new PR for every new feature you are working on or every bug you are removing.

Making Forked projects even with main project

(18) git fetch --all --prune

all branches — the one's related also

(19) git reset --hard upstream/main

*Now the forked project is same as main project.

SUMMARY

CHEAT SHEET

1. git init → Initialize the git in folder.
2. git status → Gives the status
3. git add file-name → Add file to tracked file.
4. gid add. → Add all file
5. git log → keep history of all commits
6. git add remote origin <url> → Add url as a name of origin to git.
7. git branch -m main → Set the parent branch as main
8. git push -u origin main → push code to origin(url)
9. git remote -V → list of all the url attached to a folder.
10. git clone <url> → fork and get url from someon's other project.
11. .gitignore → To ignore selected file. Trick to untrack file
12. git rm text.txt → rm + git add.
13. git restore --staged <filename>
    git restore <filename>
14. git commit --amend → To make minor changes to committed file.
15. git reset HEAD <filename> → Switch file from stagging area to modified.
16. git checkout -- <filename>
17. git stash → Undo changes and save to local memory
18. git restore <filename> → ① staged → modified.
    ② modified → unmodified

→ git stash pop → Undos changes
→ git stash clear → Clears up all the stash.

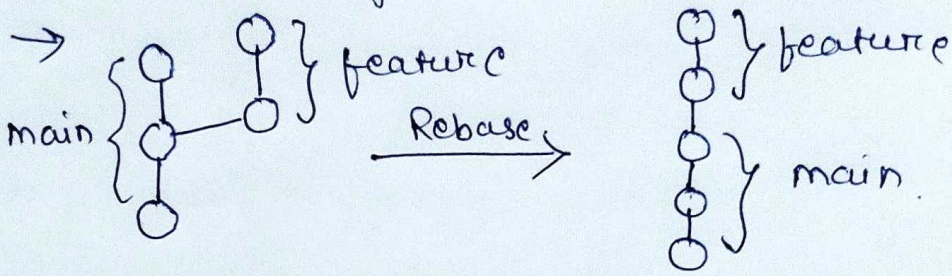
(19) git branch<branch name> → To create branch

(20) git checkout <branch name> → Takes you to the branch.

(21) git merge <branch name> → Merge all files to the checkout branch.

(22) git pull upstream main → Commit all the upstream changes to your local system.

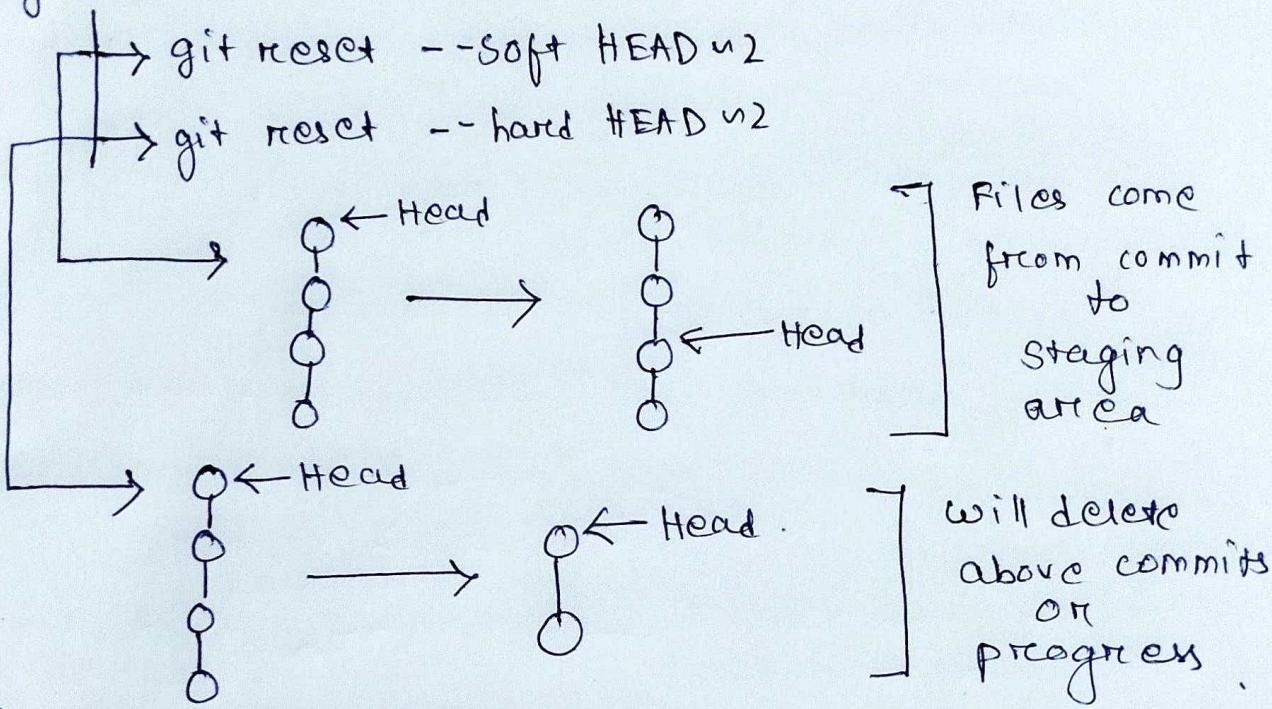
(23) git rebase →



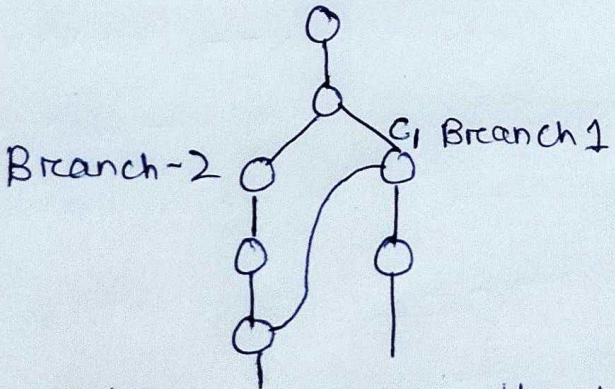
(24) git reset :-

→ git reset --soft HEAD~2

→ git reset --hard HEAD~2



(25) git cherry-pick



Can make copy of C_1 commit of branch 1 to branch 2.

(26) git reflog → shows the history of action in the repo

(27) git rebase -i log_code → Used to merge all the above commits to one commit.