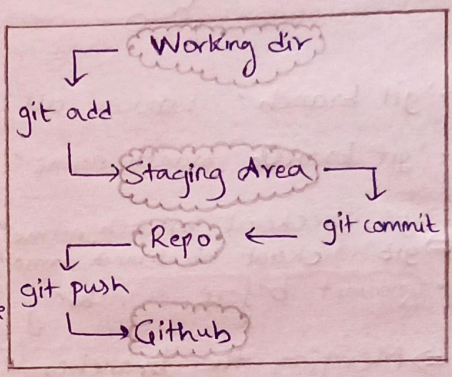
GIT

- 'git --version' : shows version of git
- 'ls' : shows all directories / files
- 'git init' : → initializing git, so now it tracks our files
  ↳ one time per project
- '.git' : → a hidden folder to keep history of all files and sub-folders.
  ↳ consits : "HEAD, hooks, refs, config, info, description, objects" files initially, & it grows in size as we make progress.
- 'commit' ~ check point (like game)
- 'pwd' : shows present working directory
- 'git status' : → shows whether our folder has already initialized git ?
  → shows any 'commits' to be done on branch, name will be shown.
- 'cd' : changes directory



- 'git add file_1 || git add .' :
  ↳ adds file_1 to tracking zone. we can also write multiple files here
  ↳ adds all 'uncommitted' / changed files under tracking of git.
- 'git commit -m "message" : while committing, it's imp. to add a message inside - " ".

⇒ Stage : → git init
→ create file / files
→ git add file1 file2 || git add .
→ git status

⇒ Commit : → git commit -m "a good descriptive message"
→ git status
→ 'git log' : → used to view history of committed changes within a Git repo.
↳ log of commits with commit hash

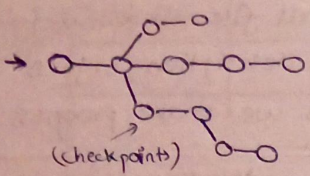
- 'git config --global user.name "Ank Ver" ' : set a git username
- 'git config --global user.name' : confirm, you have set correct git username
- 'git config --global user.email "av@g.ai" : set a git email

- 'git log --oneline' : shows git logs in 1 line

⇒ '.gitignore' : → don't want to track some files
→ ex: node_modules, API key, etc
→ get template online, patterns can be tricky.
↳ (type 'gitignore generator' online)

⇒ Branches :

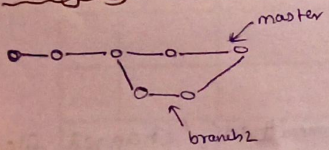
→ Like an alternate timeline
→ You are always on some branch



→ 'Head → master' : Head points to where a branch is currently at.

- 'git branch' : shows all possible branches available
- 'git branch branch_name' : creates a new branch named "branch_name".
- 'git checkout branch_name' : switches to 'branch_name' branch.
- 'git checkout -b branch_name' : Creates & switches to 'branch_name' branch.
- Commit before switching to another branch

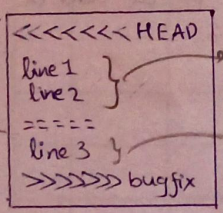
⇒ Merging the branches :



→ 'git merge branch2' : merges branch2 with 'master' branch, provided we are currently on 'master' branch.

→ 'git branch -d branch1' : deletes 'branch1' branch

→ git tries its best to resolve conflicts :



branch that you're on.

conflict that came from another branch. (merge)

Keep whatever you want, remove markers & save.

Suppose we want changes of other branch to be saved, then whatever is in "()" → remove those & save. That's it. Conflict resolved.

⇒ 'git diff' : → shows the changes btw working directory (current file) & (its just convention) the staged changes (file which is in staging area).

→ How to read diff :

'represents lines of file 1' → • a → file 1 & b → file 2 (is nty but file 1, but changes made over time, so saving as file 2)

doesn't mean removed → • (- - -) file 1

• (+ + +) file 2

{indicates changes in file} → can be shown for multiple files

'represents lines of file 2' → doesn't mean added → • changes in line & little preview of it.

→ So you made changes, → did 'git add .' → Now you can use this command:

→ 'git diff --staged' : shows differences btw index i.e (staged changes) & last commit
↳ Shows in 'vim mode' → to exit → enter 'q'.

→ If we want to see differences between 2 specific commits:
↳ 'git log --oneline' : shows commit id with commit messages, basically git log.
↳ 'git diff <commitId1>..<commitId2>' (or) 'git diff <commitId1> <commitId2>' : shows differences btw both commits.

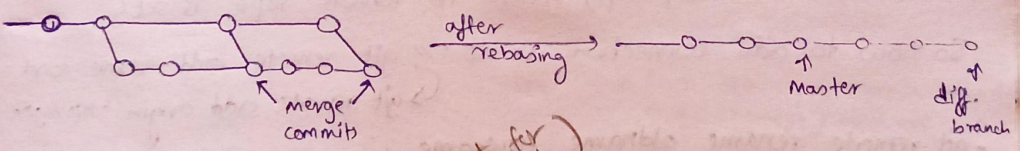
⇒ 'git stash' : → Create a repo, work & commit on main
→ switch to another branch & work
→ conflicting / unsaved changes don't allow to switch branch, without 'commits'

→ 'git stash' : you can switch branch
→ 'git stash pop' : bring back those changes
→ 'git stash apply' : apply changes & keep them in stash
→ Stash can be moved to diff. branches as well.

(get from 'git log --oneline')

- git checkout <Hash> → (detach Head) : new branch → Now if we want to come to present → "git checkout master"
git checkout HEAD~2 → look at 2 commit prior
git restore filename → get back to last commit version.

⇒ Rebase : → alternative to merging
→ clean up tool (clean up commits)



Steps:
- git init
// work & commit on main branch
- git switch -c footer
// work & add a footer in feature branch
- git commit -am "add footer text" (shortcut for add + commit)
- git switch main
// work & add hero section on main branch
- git switch footer
- git merge main
// work & add more info. in footers
// if there is more work on main, do more merge
// get on footer branch & rebase
- git rebase main

// work & add more to footer branch
// move to footer branch
- git rebase main

// NEVER REBASE WHEN YOU ARE ON 'MAIN' BRANCH, NEVER REBASE COMMITS THAT YOU HAVE SHARED PUSHED TO GITHUB

→ Rebase command is done from side branch & not 'main' branch.
→ Now rebase is done from other branches only, then in git graph you can see that all the branches are merged into 1 line.

* Then suppose you go to main branch & do changes, again you can commit it & see that another branch is being shown.

Basically, (2 initially) we have another commit while 'merging' branches & now we don't have that after rebasing.

→ If conflicts occur while rebasing → read errors;

e.g. git rebase main

// conflict occurs → (rebase → in halt state)
→ to abort & get back to state before 'git rebase', ↳ 'git rebase --abort'
→ Once conflict solved → 'git add/rm <conflicted_files>'
then → 'git rebase --continue'

⇒ Github Discussion:

→ Git is a software & GitHub is a service to host git online
{GitHub = collaboration + Backup + Open Source}

Steps:

- Create a new repo. (parallelly do 'git init' in whatever file you're working)
- No need to add readme file (if we already have file in our vscode & it is git initialized) → (readme.md) → markdown
- git remote -v : checks if there are any remote repo's set up or not. (o/p empty means no remote repo. is set)

So now to add connection we do : // git remote add name url
↳ git remote add origin https://github.com/...git

- git remote rename oldname newname : if we want to rename our old name (origin) to new name.
- git remote remove name : If we want to remove the name.
- git push <remote> <branch> : <remote> → where I have to push; <branch> → branchname which you want to push
- git push origin main : It will push your code to github but when next time you do 'git push', when you updated files & want to push, then that command 'git push' won't work, as there is connection established yet.

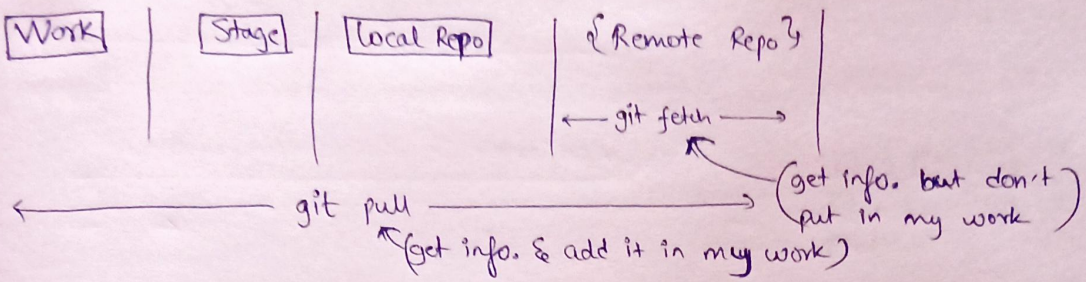
(we can push any branch)

- git push -u origin main : '-u' setup an upstream that allow you to run future command 'git push' & it push the code directly to github.

• git clone <URL> → of any repo., & now we can bring that repo.'s code onto our system.

When you clone a repo, you get just main branch connected, rest of remote branches are not configured: git switch branch_name

• git branch -r : connects remote branch to local.



→ git pull = git fetch + git merge

→ git pull origin main (changes will be merged to main)

⇒ Github features on website like: • Adding collaborators • Readme file
• Markdown format • Adding Gists
• Codespaces • Dev Container

{ Kind of like virtual machines, but here it's configured based on what kind of code I am having. Ex: I have Golang repo, then while opening this, it will have configuration & installation done already. } ← Codespaces

⇒ Open Source:
- Talk
- Open an issue
- Get the issue assigned
- Work & add value
- Make PR & iterate over it
- Have Patience

→ Fork the repo; create branch, make changes; now we can do 'git push origin branch_name'; then in Github you can compare & pull request

⇒ • git revert: → purpose: reverts a specific commit by creating a new commit that undoes the changes made by the specified commit. (commit history not modified)

• git reset: → purpose: resets the current branch to a specific commit, discarding any commits after that point. (commit history modified)