

Git & Github

- * Linus Torvalds made git.
- * He found out, the VCS (version control system) is very important for source code (Program), to maintain the history / checkpoints of our code.
- * One can go back and see the version (same point) of our code, files or shell scripts.
 - "Helps to rollback / revert to previous changes."

GIT: Global Information Tracker
→ is VCS


- Git is also called "source code management"

Extra:


To run linux on windows use

- WSL
- Gitbash: linux simulation (with limited commands support)

Note: No need to install git on
◦ Linux & MacOS.

- ⇒ Git VCS tracks our Project  folder and files.
- ⇒ The tracking of folder should be separate.
 - "Every folder have unique git of their own."

⇒ Create a folder

• mkdir git-for-drops 

Drawback of
Linux File System

- 1) Can't restore files
- 2) Can't manage versions
- 3) Can't understand the time, who edited file.

GIT - VCS

- 1) Can restore file
- 2) Can manage versions
- 3) Can understand author
- 4) Find time, when edited.

`$ cd git-for-drops`
`$ git init`

Git is initialized
inside folder.

• git

`$ ls -a`
`• git`

• git (hidden folder), VCS is hidden there.

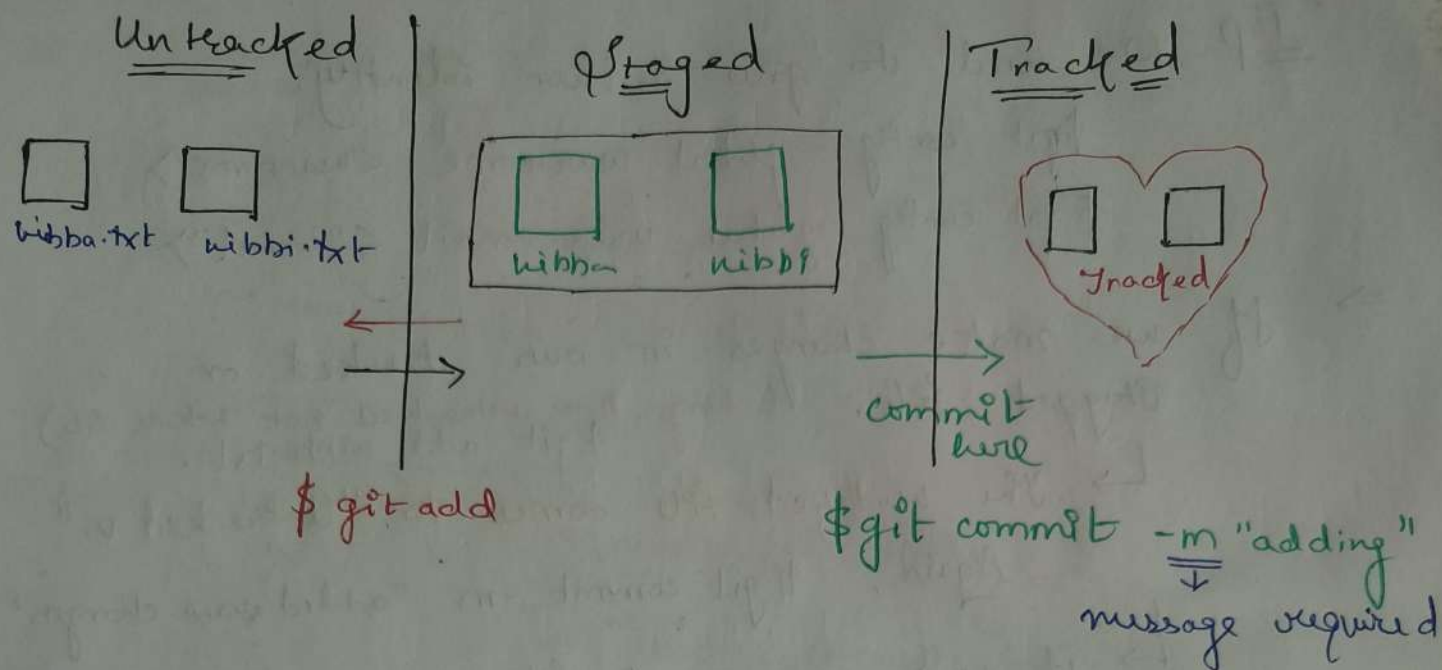
⇒ Create a file

`$ touch file.txt`

`$ git status` → shows tracking status.
o/p "Untracked files"

Fun Eg: we have nibba nibho, • they meet (untracked)
• They add as friend (staged)
• They like each other (tracked)

Three Stage of
Relation.



* What if nibbi go out (deleted) ?

```
$ git init
$ touch wibba.txt wibbi.txt
$ git status
"untracked"
$ git add wibba.txt wibbi.txt (Staging)
$ git commit -m "adding" (Tracking)
"Deleting"
$ rm wibbi.txt - Successfully got out from
                  FS of Linux
$ git status - But tracked in git vcs.
              "Here with help of Git
                we can restore file"
```

`$ git restore wibbi.txt`

Tip We need to give author identity.

\$ git config --global user.name <"username">

\$ git config --global user.email <"email">

⇒ If we make changes in our tracked or staged file. (change but untracked not whole file)
\$ git add wibhi.txt.

↳ The modified file comes to "Untracked v."

Again, \$ git commit -m "added new changes"

↳ A new version of file comes. (Untracked)

Note If we overwrite a file,
no need to "\$ git add" again

\$ git add. → add all files to git.

\$ git commit -m "commit" → commit all staged files to git.

\$ git log → can see unique versions of our changes (commit)

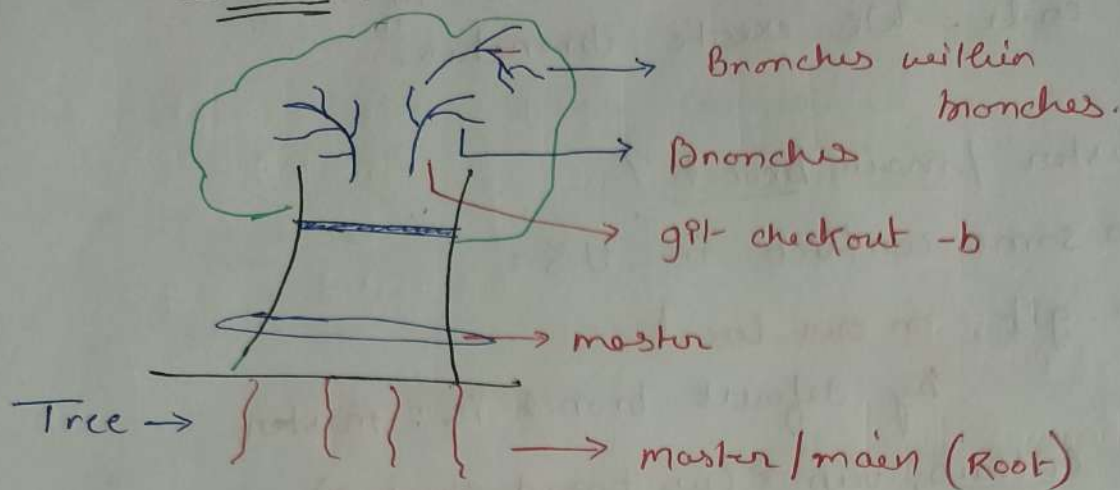
→ we get unique version ID.

→ Working in a team, in logs we get dedicated author name & email.

"Git is v. good for cross team - collab"

→ Shows "commit history"

* Branch *



↳ "Some concept in GIT"

⇒ Creating a new branch

`$ git checkout -b dev`

Creates new branch

dev branch working tree will be diff.
(Eg: All commits)

If we avoid creating new branch
+ stay in some branch (we avoid -b)

`$ git checkout master`
"Switched to master"

master branch working tree will be diff.

Eg `$ git checkout -b dev` → inside dev branch

`$ git status`
o/p "nibba", "nibbi"

`$ touch nibbu.txt`

→ created in dev

`$ git status`

o/p "dev branch" "nibbu"

`$ git checkout master` → Switched to master.
`$ git status`
"No nibbu found" → No file in master

⇒ "To avoid disturbance / conflict in main / master code. We create branches."

Q Why master / main branch?

⇒ Due to some conflict in US:

1) In git, on our local

by default branch is: master

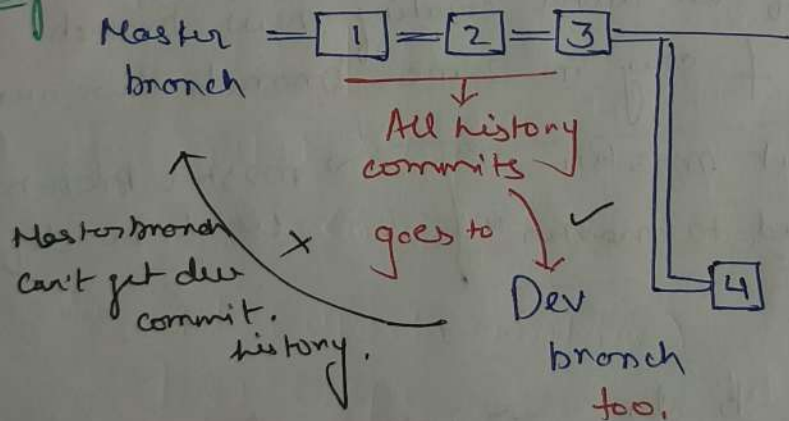
2) On GitHub, vup → (us based startup)

by default branch is: main

⇒ "Every branch maintains own logs"

Note: The sub branch of dev branch will also include commits of master.

Diagram



Q To see current branch?

\$ git status

Q To see all branches?

\$ git branch

O/P * dev
master

Both are same
"git switch dev"

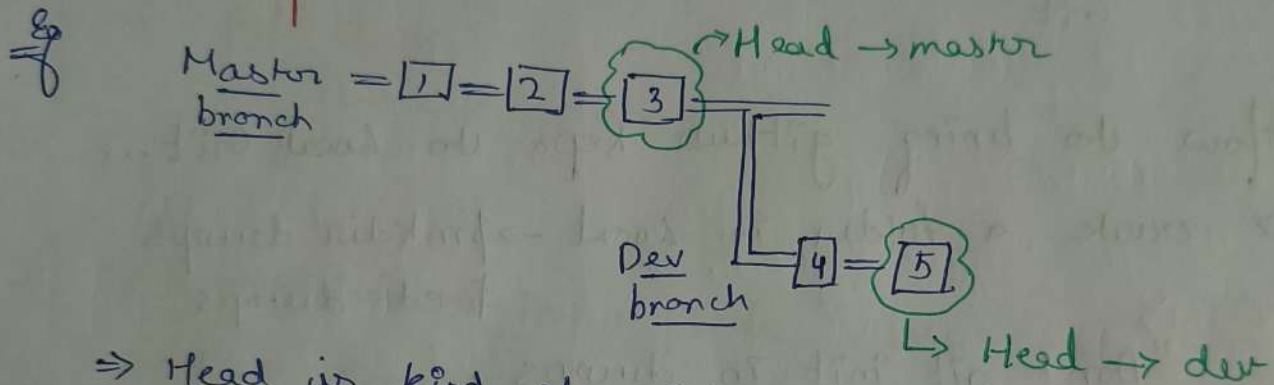
↓
"git checkout dev"

Note "Git branch" : creates empty branch

"Git checkout -b" : will create branch from existing branch changes.

* Head *

"Head is the latest commit of any branch"



⇒ Head is kind of pointer, points the latest commit.

To check : `$ git log --oneline` shows log in oneline
o/p (HEAD → master)

Note : In git status, we will be able to get parent changes too, in logs.

Git

- Is a VCS (version control sys), a tool

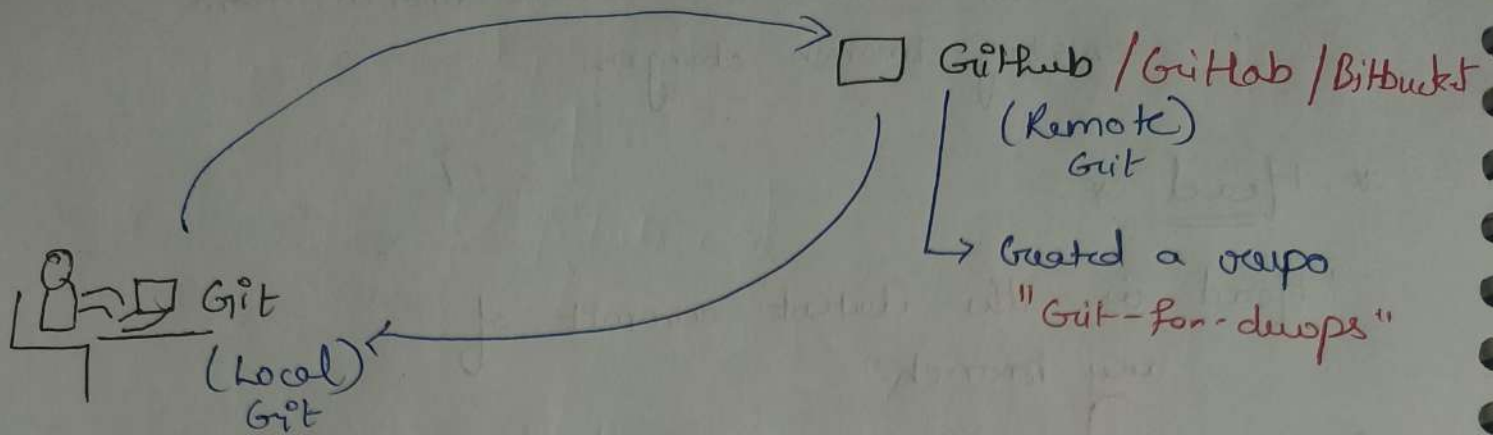
- Create git init is same as

GitHub

- GitHub is online platform allows to store code, collaborate with developers as VCS (provide UI)

- Create a new repo

.md → markdown file



⇒ How to bring github Repo to local Git...

⇒ create a folder in local → `$ mkdir devops`
`$ cd devops`

"Don't git init in devops, we are already importing a repo from Github"

`$ git init` X

⇒ Go to • Github > Repo

• Go to <code> → local → HTTPS → copy

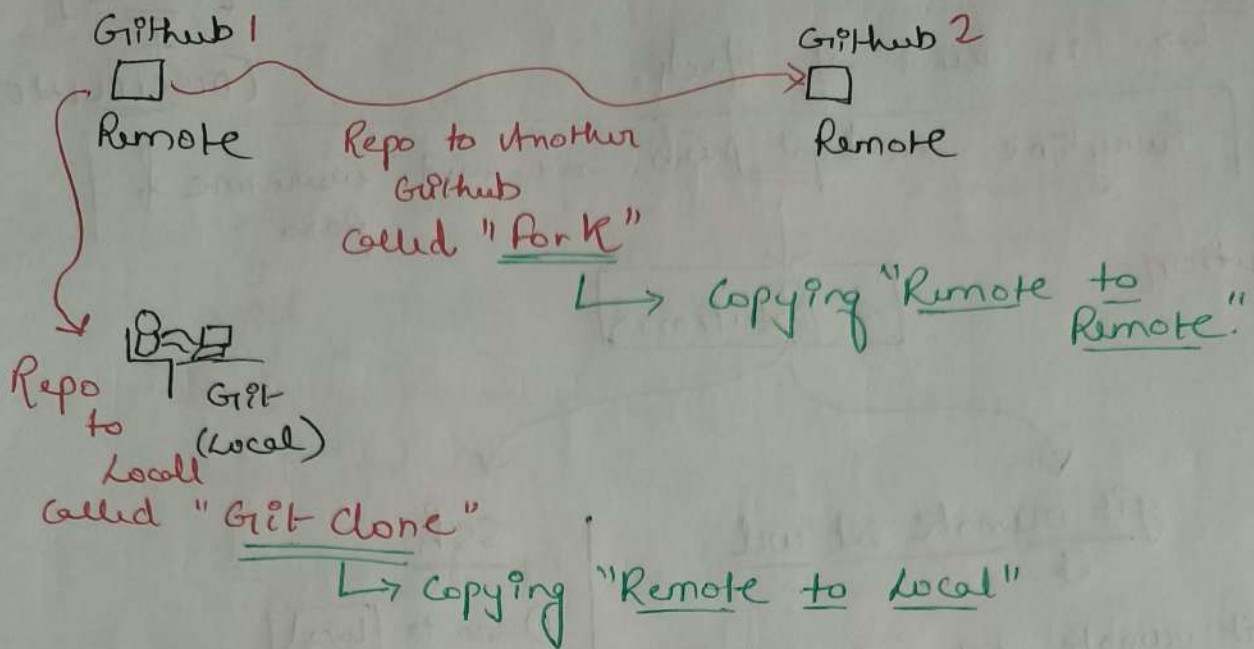
⇒ Go to • Git (Ubuntu machine)

• `$ cd devops`

• `$ git clone <paste repo link>` • git → Repo

O/P Git-for-devops (folder)

↓
Inside this folder we will have • git (hidden)



⇒ How to find origin of code? which is local or remote?

\$ git remote -v

README.md → in capital (syntax)

⇒ Will push the file from local to remote..

\$ git push

Note Password based authentication, stopped by X Github.

We use "Personal Access Tokens" (classic)

↳ "Generate New Token (classic)"

↳ Give Repo ☒ Access.

☒ Generate Token.

PAT

↳ is kind of Hack.

[FOCUS]

+
[DISCIPLINE]

Note | Everytime we \$ git push, we need username & pass

Solution:

2 Solutions

git remote set-url

↓

Hack

① \$ git remote set-url origin
https://<token>@github.com
<url>

ssh

- 1) Go to local
(/home/ubuntu)
- 2) cd .ssh
- 3) \$ ssh-keygen
Enter, Enter

O/P id_rsa12..., id_rsa25...pub
↓
public

SSH Setup for Push

(local server to server remote)
Push

A Private Key
local

B Public Key

GitHub

- Go to GitHub Settings
- Go to SSH & GPG Keys
- Add new SSH Key

→ Paste (Public Key on Remote)

• Go to Repo → <code> → SSH (copy URL)

• \$ cat id_rsa25.pub
(copy)

• Local already have private Key.

• Go to Rep "git-for-dummies"

• Change URL

→ \$ git remote set-url origin <paste URL>

→ Old origin changed to SSH

Note Once SSH URL setup done,
no need to do authentication again
while push.

⇒ what if there is any change on ^{remote} local,
we will "Pull"

- Local to Remote : Push
- Remote to Local : Pull

"Go to local" (eg EC2 Ubuntu)

cmd: `$ git pull origin main` → branch
name

⇒ Some challenges you might face

⇒ what if you have conflicting changes in
Remote & local.

`$ git config --global pull.rebase false`
or while pull
`$ git pull --rebase origin main.`

⇒ Diff b/w Pull / Clone

* only changes of remote are pulled.

* whole repo get clone

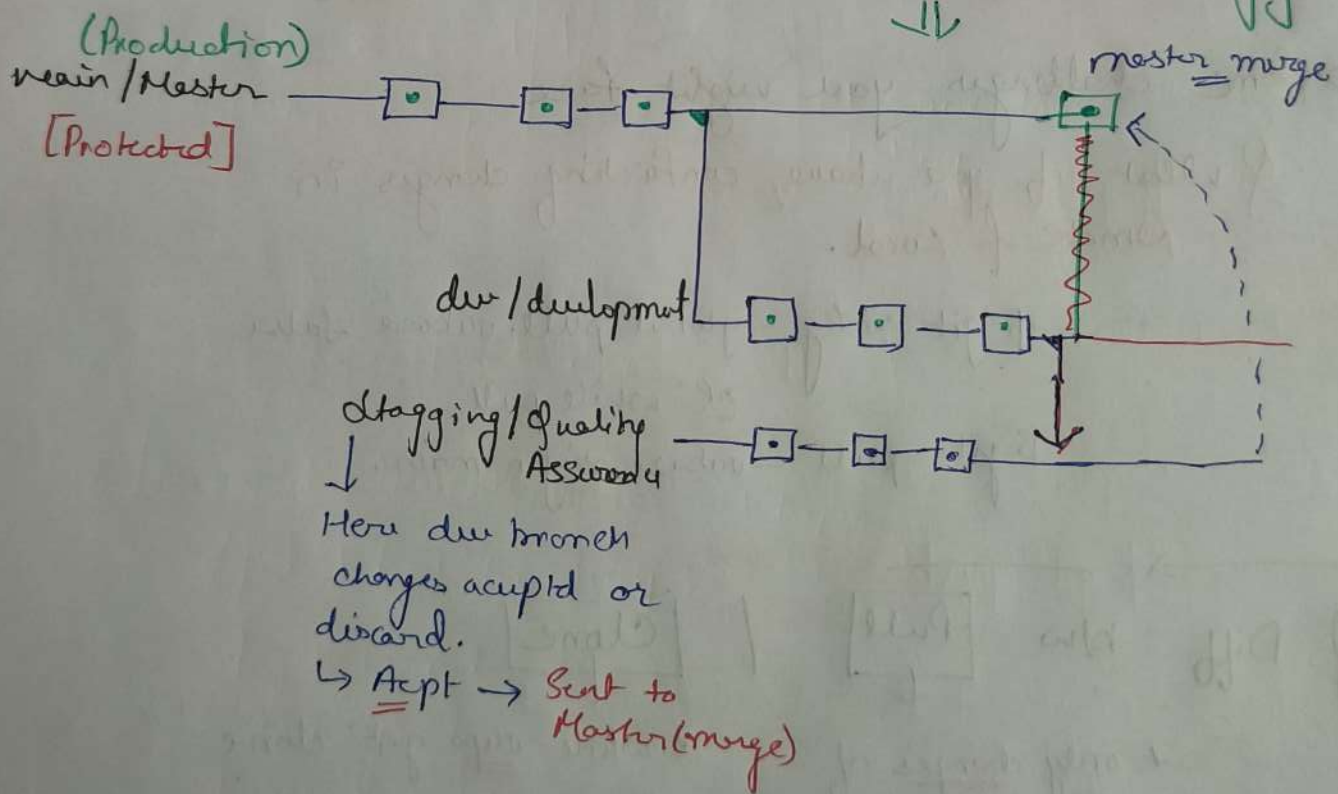
Note You should always write origin while
push / pull

`$ git pull origin main` (preferred) ✓
or
`$ git pull` (this not preferred) ✗

* Branching Strategies *

- In our xyz project, we have master/main branch.
- "Master code generally used in production".
- Master branch is protected. Anyone can't commit to master/main branch directly.
- "We don't use "master b." for testing."
- For Testing we create separate branch called dev/development.

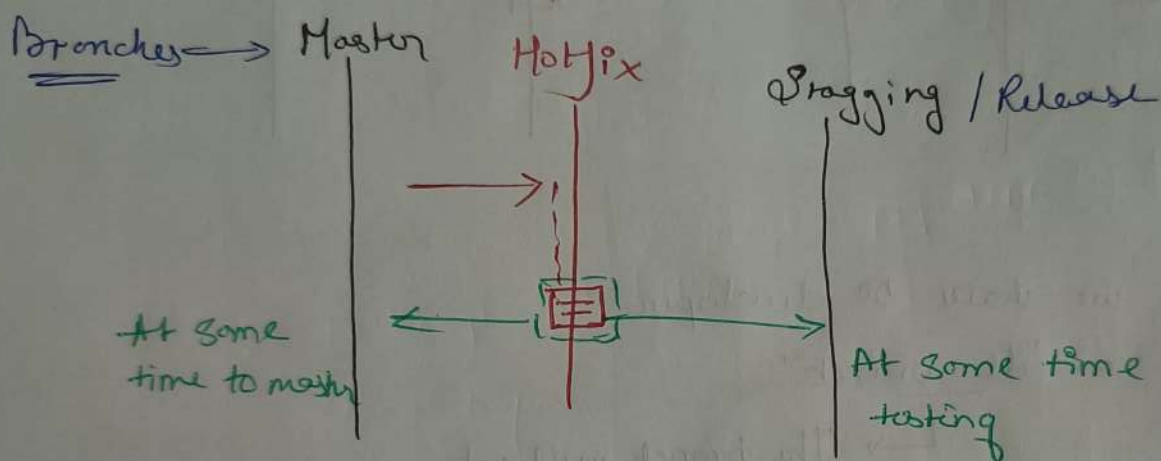
① Branch Strategy



What if there's bug in production?

⇒ A new branch will be created from master/main called "Hotfix branch"

⇒ Developer will fix bug & will merge to master & Staging branch both.



Lets Hands on Practical

Go to Local → Remote Folder Git Repo

/github/git-for-devs

\$ git branch

* main

\$ git checkout -b staging

\$ git checkout -b dev

\$ git branch

* dev

→ Inside Dev

(Create) →

\$ vim Branching.md
(Save file)

Note If all files working fine in "dev"
we still need to merge in "Staging" then
(Production) "Master".

* Branching.md
(file in dev) → \$ git add branching.md
\$ git commit -m "Added Branching Strategies".

↓
To merge in Staging
↓

\$ git checkout staging
⇒ Branch Staging + Dev = Merged
\$ git merge dev (dev changes will come to staging)

\$ git log --oneline
points latest commits
(HEAD → staging, dev) pointing both

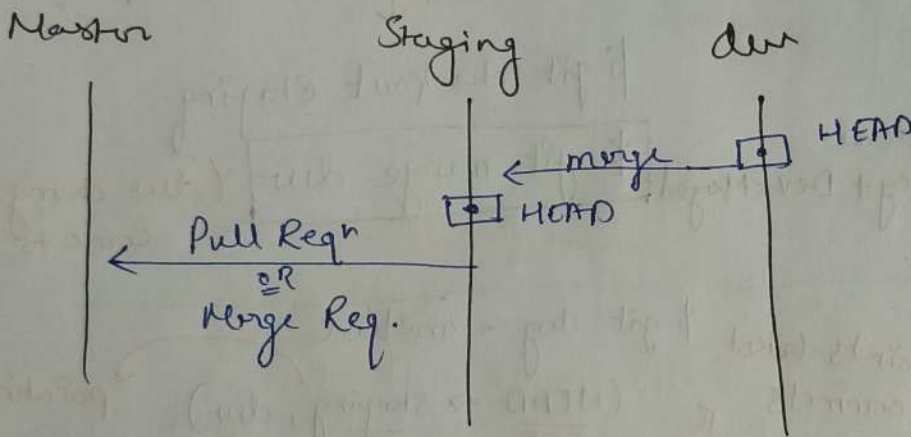
* Note To merge staging branch → master.
You should always do with "Github"
Not over local m/c, do it remotely.
Why?
↓

- # We can create a pull request on github & we can review that pull request with 10 other people.
- # But on local no one can review, but on github we can.

\$ git push origin staging

⇒ we pushed files of staging on remote (github)
⇒ on github stag we can add > Reviewers

* Go to Setting → Collaborators
↓
Manage Access
↓
Add People
↓
Invite ← Add Reviewer



Note When reviewer will ☒ Approve, then we can merge.

§ we pushed commends.md file from local to remote in dev branch

• Go to Pull Req →

• Stag ← dev ^{Select}

• Now Go to "Code Labbit" ^{AI tool} → gives suggestion (can apply)

Help to review to pull request
Signup + signin with Github

Go to \Rightarrow Pull Request

\Rightarrow Comment Section \downarrow

"@codewithritu full review"

\swarrow You will get suggestion

\rightarrow Act As AI Reviewer.

Note: If you want manual human Reviewer
 \rightarrow eg user!

You can setup "Branch Protection Rules"

\rightarrow Avoid "Merging part"

in "main branch"

from any other user.

Setup Rule: \rightarrow Go to Settings \rightarrow (of git-for-dwops)
 \rightarrow Go to Branches

\rightarrow "Add classic branch Prot Rule"

\rightarrow Branch name pattern "main"

\rightarrow Create (Protection done)

Imp fns

Note * A pull request proposes code to review before merging to main branch.

* Git stash: temp saves uncommitted changes.

* Fork in github is repo copy.