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 (save 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 & Mac OS.

⇒ Git VCS tracks our Project folder and files.
⇒ The tracking of folder should be seperate.

"Every folder have unique GIT of their own."

⇒ Create a folder

• mkdir git-for-duops

Drawback of Linux File System → GIT - VCS

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

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

```
$ cd git-for-duops
$ git init
```
→ Git is initialized inside folder.
• .git

```
$ls -a
.git
```
→ .git (hidden folder), VCS is hidden there.

Eg Create a file

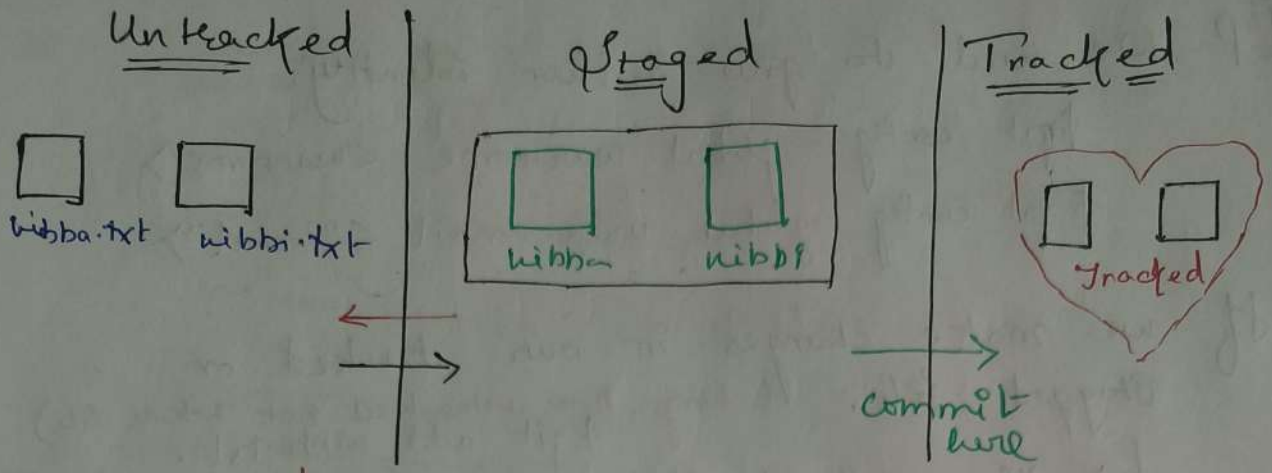
$ touch file.txt

$ git status ⟶ shows tracking status.

o/p "Untracked files"

Fun Eg: we have nibba nibbi,
- 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 nibba.txt nibbi.txt
$ git status
"Untracked"
$ git add nibba.txt nibbi.txt (Staging) → add file / stage to git
$ git commit -m "adding" (Tracking)
"Deleting"
$ rm nibbi.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 nibbi.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. (changes bcm untracked not whole file)
$git add nibhi.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 restore a file, no need to "$git add" again

$git add. → add all files to git.

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

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

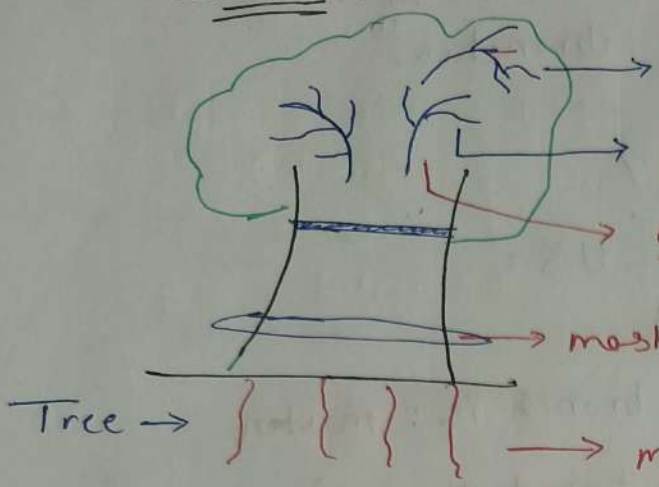
→ we get unique version ID.

→ 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
```

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

creates new branch

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

```
$ git checkout master
"Switched to master"
```

→ master branch working 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:

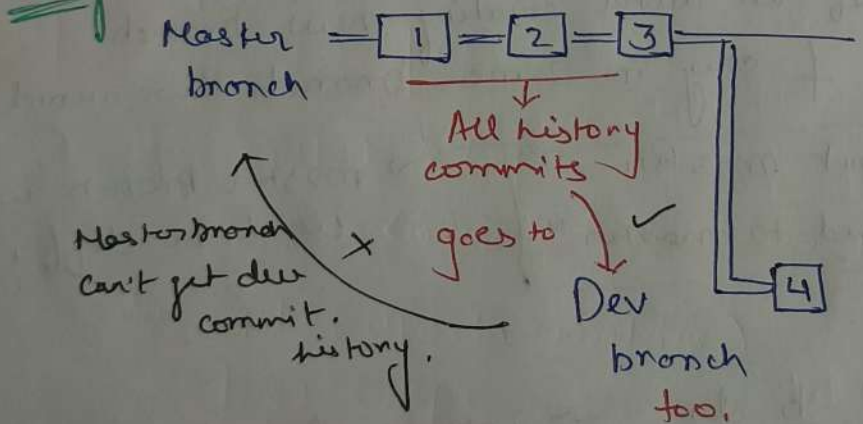
1) In git, on our local
   by default branch is: master

2) On GitHub, vsp → (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

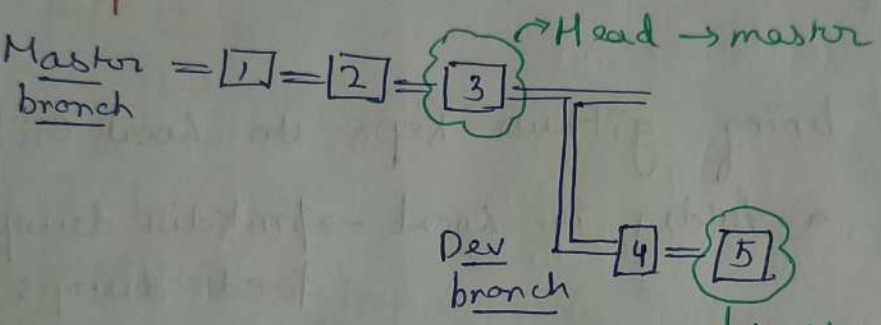
Both are same
"git switch dev"
↕
"git checkout dev"

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 lastest commit.

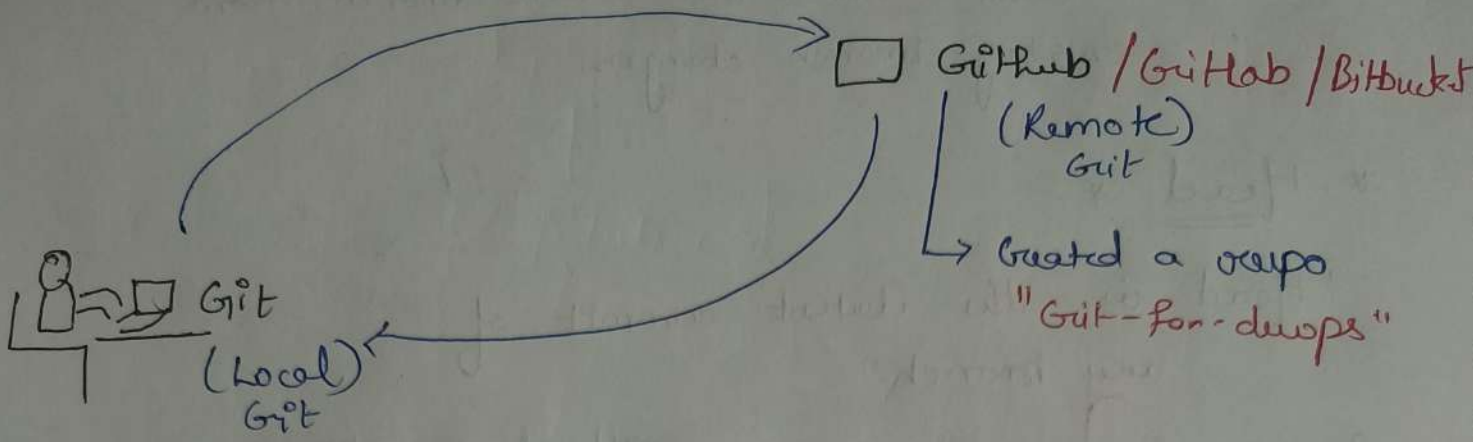
To check : $ git log --oneline (shows log in oneline)

O/P (HEAD → master)

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

Git	GitHub
• Is a VCS (version control sys), a tool	• GitHub is online platform allows to store code, collobrate with developers as VCS (provide UI)
• Create git init is same as	⇒ Create a new repo

.md → markdown file



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

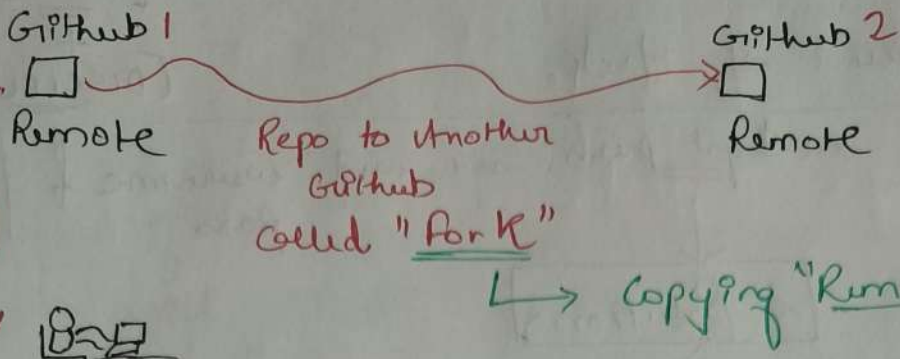
⇒ create a folder in local → $mkdir devops
$ cd devops

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

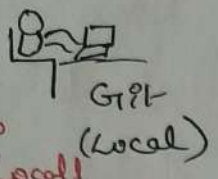
$Git init ✗

⇒ 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)



↳ Copying "Remote to Remote"



↳ Copying "Remote to Local"

⇒ 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 Github.

We use "Personal Access Tokens" (classic)

↳ "Generate New Token (classic)"

↳ Give Repo ☑ Access.

☑ Generate Token.

[FOCUS] + [DISCIPLINE]

PAT
↳ is kind of Hack.

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_ed5912.., id_ed25...pub
(public)

SSH Setup for Push

(local server to server remote) Push → server 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)

A Private Key
local
- $ cat id_ed25.pub (copy)
- Local already have private key.
- Go to Rep "git-for-dwops"
- 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

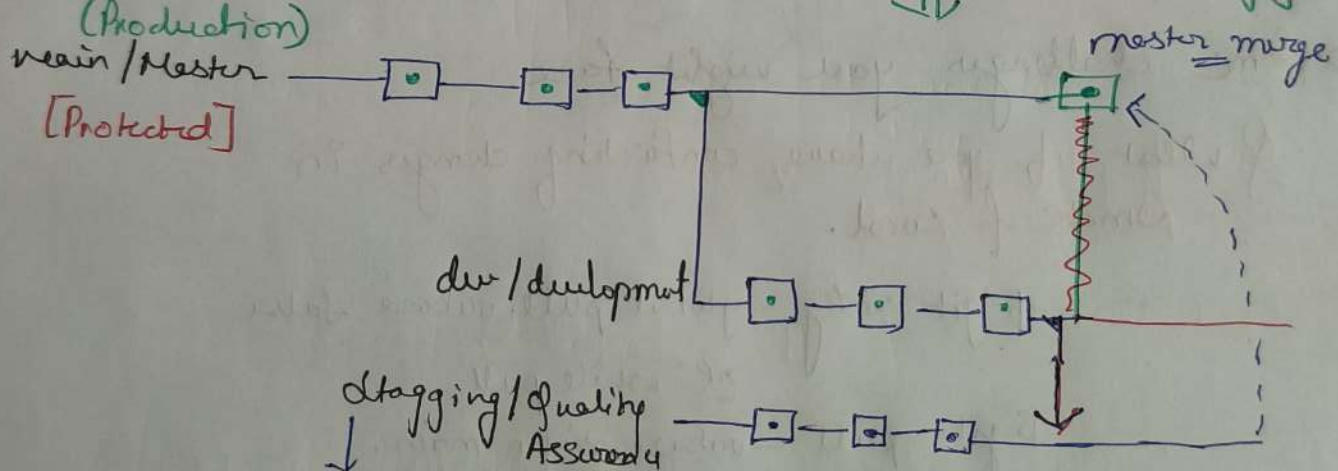
Note You should always write origin while push / pull

$ 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 seperate branch called dev/development.

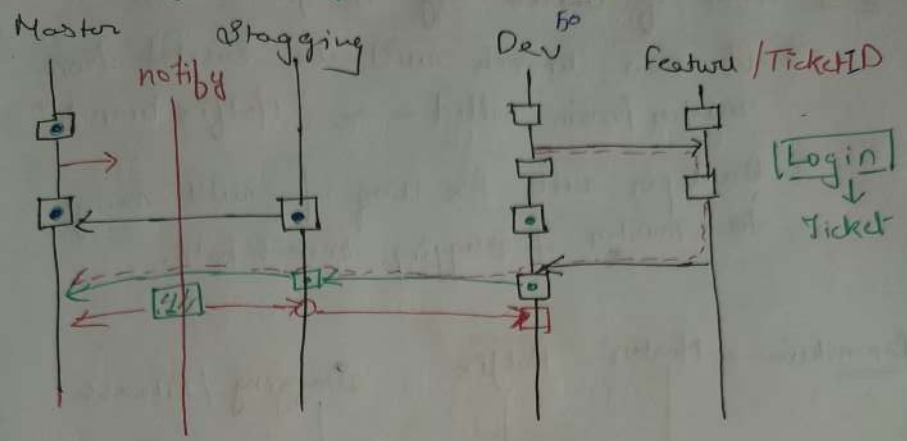
① Branch Strategy



Here dev branch changes accepted or discard.

↳ Acpt → Sent to Master(merge)

(2) Branching Strategy (Production level) **



* Suppose we have 50 developer working on "dev branch"
  - ↳ The branch will get populated, so → Feature branch (Invented)

Q What is feature Branch?
⇒ Suppose working on "Login func" of webpage.
↓
Ticket will be created (with ID)

Eg 50 dev working on 50 diff features
All developers will be assigned 50 feature branch will come from "dev branch"

"Dev branch to Feature" → "Feature branch will release update / done" work → "To dev branch back." → "Test to Stage"

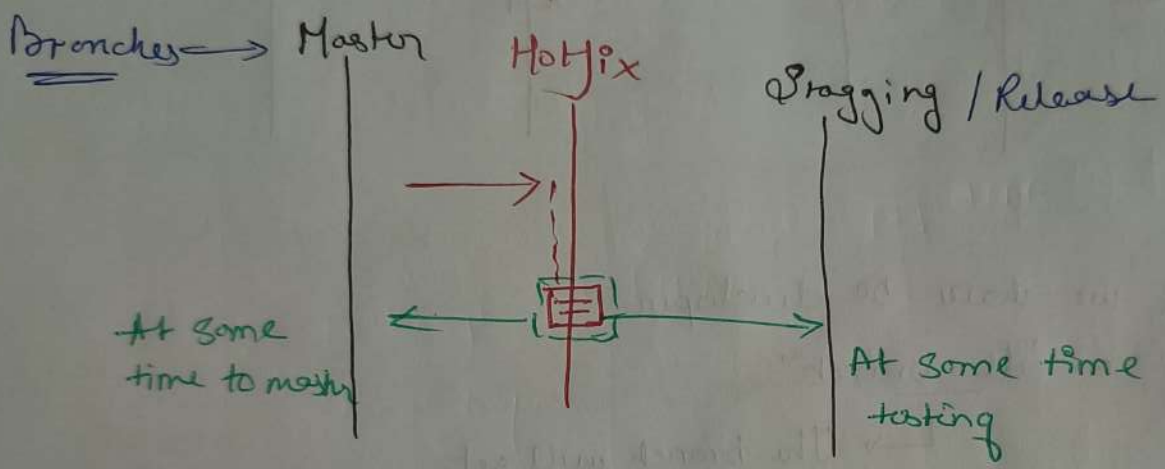
↓

Production ← Master

Q 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 & Stagging branch both.



Lets Hands on Practical

Go to Local → Remote Folder Git Repo

/github/git-for-devops

```
$ git branch
  * main
$ git checkout -b stagging
$ git checkout -b dev
$ git branch
    * dev
```

⟶ Inside Dev (Gvate) ⟶ $ vim Branching.md (Same file)

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

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

↓

To merge in Stagging

↓

$ 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

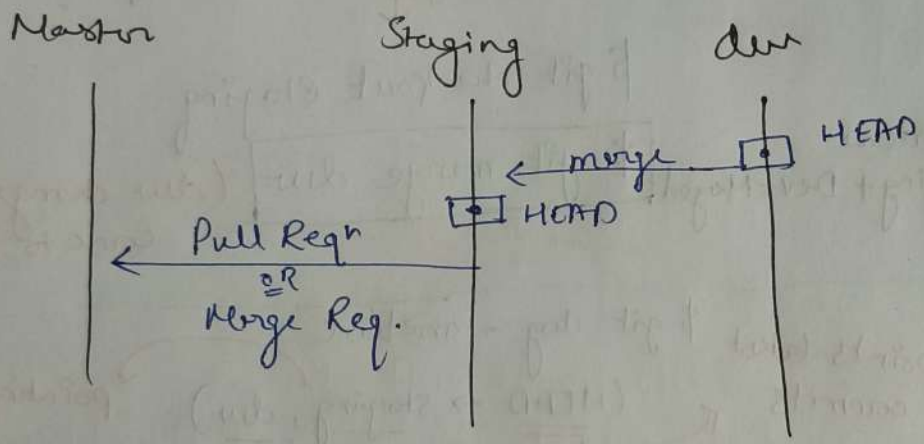
↓

Manage Access

↓

Add People

Now, Add Reviewer ← Invite



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

§ We pushed commands.md file from local to remote in dev branch

- Go to Pull Req →
- Stag ← dev (Select)
- Now Go to "Code Rabbit" (AI tool) → gives suggestion (can apply)

↓

Help to review to pull request

Signup & signin with GitHub

Go to ⇒ Pull Request
⇒ Comment Section &
"@coderabbitai full review"
↳ You will get suggestion
↳ Act As AI Reviewer.

Note: If you want manual human Reviewer
Eg user1

You can setup "Branch Protection Rules"
↳ Avoid "Merging part"
in "main branch"
from any other user.

Setup Rule: ↳ Go to Settings → (of git-for-devops)
↳ Go to Branches
↳ "Add classic branch Prot Rule"
↳ Branch name patter "main"
↳ Create (Protection done)

Imp Qns

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.