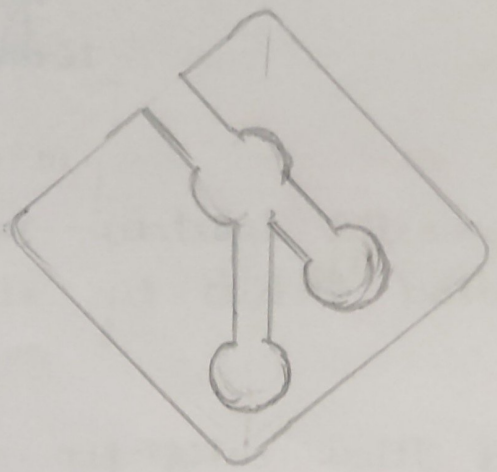
Git.....



* Git :→

(Global Information Tracker)

Open source distributed version control system

- Discovered by = linus
  windows = Gitbash

- Connect EC2 Instance
  terminal = git --version
  not found
  Install = Sudo yum install -y git
  mkdir demogit
  cd "

* Git workflow :→

3-phases/stage
1) Working directory
2) Stage
3) Commit

1) Working directory =
Vi demo.txt - untrack files
(If u delete git didn't bother)

2) Stage
demo.txt - all files will be track

3) Commit
demo3.txt, hi hello how r u?
username, mail, timestamp

terminal: ls -al => displays the regular directory

ls -a => shows the hidden file (. ..)

git init => convert regular folder into git folder

ls -a => . . git - now its git folder

* Configure the Username & Email by git
  - Outside the directory

git config --global user.name "Vaishnavi"

" " " .email "vaishmayo5@gmail.com"

git config --list

↳ displays username = vaishnavi
email = vaishmayo5@gmail.com

* The very first file what we create the git will not get to know that it is in working directory

vi demo.txt

[Hi]

cat demo.txt

git Status

↳ no commits yet
untracked files

If u delete the file = demo.txt

↓

rm "

git doesn't bother about it

Sudo useradd @meshussain

Sudo passwd "

Su - @meshussain

whoami)

↳ "

* Converting untracked file into tracked file.

git add .

↳ all untracked files will get tracked

git add demo.txt

↳ Specific file will be tracked.

git status

↳ new file: demo1.txt

If I delete this demo1.txt

rm demo1.txt

git status

↳ it shows

changes to be committed.

newfile demo1.txt

changes not staged for commit

deleted demo1.txt

- If I need to restore these files

git restore demo1.txt

* Commit the file

git commit -m "updated file demo1.txt"

↳ u can do it to multiple files

git log = shows the number of commit

git --help = provides all the commands of git

"git commit" = is used to save changes to your local git repository
* Its like Snapshot

- I made changes in vi demo.txt
... im vaishnavi.
↳ this change I made

git status

Note: If you do small changes in the code it should be committed if not it will through the error.

git commit -m " "
git log
↳ it displays the commit 5bf90e1

↓
git show paste the id.

↓ it shows all the commits

it displays ↓ the modified things

git log --oneline => it displays single line command output

git diff => displays the changes

git add . ——→ Staging area.

git tag ——→ Allocate versioning

↳ after commit one code generates

5abc...

↓

- It unable to track this so versioning is important
- It helps to remember all your commits

1) lightweight tags = Semantic (0.0.0)

2) Annotated "

↳ "this is the minor vision annotated"

0.1.0

If u run git tag

↓

nothing can be seen

git tag V0.1.0

↓

it will automatically get attached to latest commit

- If you want add ~~commit~~ tag to another commit

git tag V0.1.0 "5abc..."

↓ ↓

Semantic paste id

git log

git tag -d V0.1.0

↳ delete the tag

* plugin = increase my functionality (version, helper)
module = function

* Rest api = Communicate with 3rd party
Api (Application programming Interface) = interact with two different applications

git tag -a V0.1.0 -m "updated the minor version"

- -a → annotated (a message, tagger's name, email, date)
- V0.1.0 → name u want to assign to the tag
- -m "updated the minor version" → message describing the tag.

* Git Stash :-

git stash = provides a backup / storage

- If I have created demo1.txt
demo2.txt

now, demo1.txt have untracked it is working directory
git stash can be only applied for tracked / which is is staggiog.

git stash -u
↓
it consists both untracked (working directory files) + tracked files.

git stash list

- If I have worked on demo1.txt & completed with commit
Now, I want to work on demo2.txt

↓

I want to bring back the file

git stash list

git stash@{0}:

(delete manually)

better to use this ← git stash apply ⇒ it will keep the history

git stash pop ⇒ it will not

git stash apply stash@{0}

↓

C/P

ls -al

git add .

git commit -m " "

git stash drop stash@{0}

↓

delete the file

* Undo the Commit :→

git revert →

git reset

- If I want to undo the commit & keep the history of the file we use

"git revert"

↓

Commit id → history

git rebase - reapply commits on top of

- **reset** = undo the commit
  - ↳ it consists 3-degrees
    - --soft
    - --mixed
    - --hard
  - It will always point to the latest commit.
  - It will also show that which commit we are in

* **reset --hard** = cannot bring back the file again

reset
↳ commit 2 head
commit 1 head - master

* **reset --soft** = staging area
  git status - it will show something is there.

* **reset --mixed** = working directory.

revert = commit id
reset = move the head
↳ reset current head to the specified state.

If you want git do don't consider the file

vi demo.txt | .gitignore

*.log
↳ wildcart character (give some extension)

vi demo.log

file ignored

* HTTP - Webpage protocol.
* SSH = protocol of remote access

Github
↓
repository
↓
.gitignore
↓
License
↓
http/ssh
↓
we need token
↓
Settings of Github
↓
Developer Setting
↓
OAuth / personal tokens
↳ to access the Github API
↓
Token (classic
↓
password
↓
Note, Expiration, Select any option

* git remote -v => It will says whether there is any repository attached to my local system.

git branch => default branch.

* `git branch -m main`

↳ it will convert from master to main

git branch

↳ * main

* Add remote to the local repository :→

`git remote add origin` https://github.com/abbadiomer/intellipaat.git

↓ https

copy from github paste here.

`git remote -v`

`git push -u origin main`

username : email

password =

} files created can be seen in Github

* If I have made some changes in Github file demo2.txt

if I command git cat demo2.txt

↓

it won't show the changes here.

So we use

`git pull`

↳ check the remote repository it contains changes

`git fetch origin main` ⇒ only see the changes from remote to local repository

`git diff main origin/main`

↓

Compare the remote & local repository

```
git pull
cat demo2.txt
```

`git clone` ← paste url from github → copy & paste from remote repository to local repository.

`git rm demo2.txt` → deletes the file

↓ git status

if you want to restore it

`git restore --staged demo2.txt`

```
git restore demo2.txt
git status
git add .
git commit -m "updated"
git push -u origin main
username:
password:
```

Branching Strategy

Branch = Separate line of development you can work on different features in different branches without affecting the main code.

* release management = procedure to deploy the application.

`git branch`

↳ *main

`git branch feature`

`git branch`

↳ feature

*main ⟶ its in main branch

↓

to switch from main to feature branch

`git checkout feature`

↳

*feature

main

* Git merge: ⟶

* Vi demofeature.txt

↳ Hello, I'm vaishnavi

git add.

git commit -m "committed in feature branch"

git log --oneline

git branch

↳ feature

*main

`git merge feature` ⟶ combine the changes from one branch into another.

git log --oneline

↝ never do in repository do in local history

Rebase :→

* vi demorebase.txt
  git add..
  git commit -m "rebase example"
  git log -oneline
  git branch
  git rebase feature
  git log --oneline
  ↳ (HEAD → main, feature

* Merge & Rebase :→

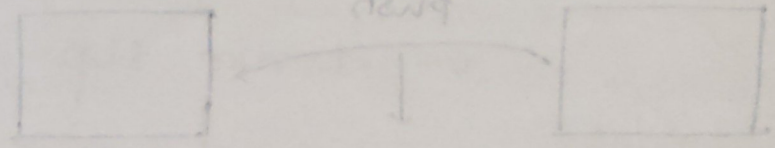
(Same meaning)

are used to bring the changes from one branch to another

- merge will create a commit id
- git will not show commit in rebase on your github repo because it will not show in events in order

* If you have multiple files committed.

vi demo34.txt
vi demo35.txt
git cherry-pick



File System
cannot recover deleted files

Version Control System
recovered deleted files or rollback to a versioning

* Filesystem = The file you create will be untracked.
Git VCS = it gets tracked.

untracked (red) → Staging (green) → tracked

provides a option to track or untrack

git add . → all
or filename

git commit -m " "
↓
tracked.

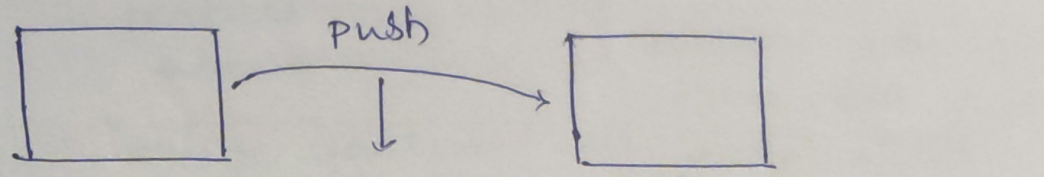
testing.py
↓
if I delete from here

to again make it unstage or untracked
git rm --cached files

git restore testing.py

* git add = untracked to staging
* git commit -m " " = Staging to tracked.

* push this files from local to github.
You should have repository in Github



Copy from github the url (push) & paste in terminal.

* To see if it is added / not
  = "git remote -v"

* PAT =>
  Github Settings
  ↓
  developer settings
  ↓
  PAT (personal access token)
  Token (classic)
  Generate new token
  use passkey
  Note = name something
  Expire = 7/30/90
  Select = repo
  ↓
  Generate token
  ↓
  c/p somewhere

Command: git remote set-url origin
https:// paste @github.com/ c/p the line generated after git remote -v (push)

↓

git remote -v

↓

git push origin master

↓

done [local to github]

Github
↓
repo

Clone
Github → local
↓
Go to Github
↓
code
↓
HTTP= copy & paste on terminal.

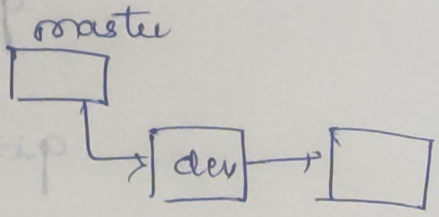
Fork
Github → Github
from my github to another github
(C/P) → Fork

* Without token how you can push code

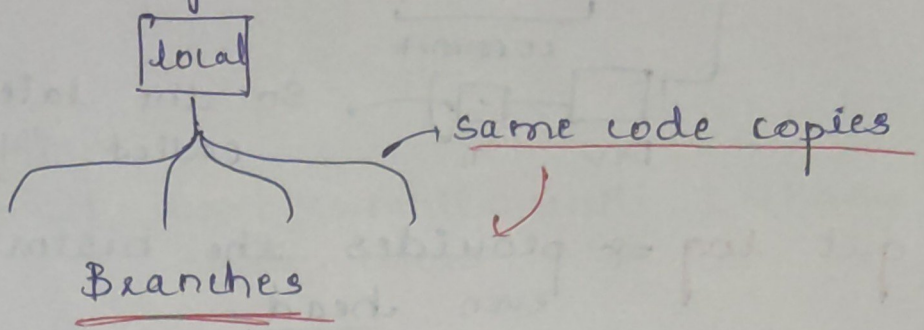
Github
↓
Settings
↓
SSH
↓
C/P

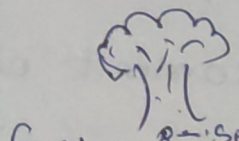
* "git pull" => to bring code from github to local
* "git push" => local to github.
* Branches :→

git branch
git branch dev.
git switch dev



- Every branch in git maintains its own copy



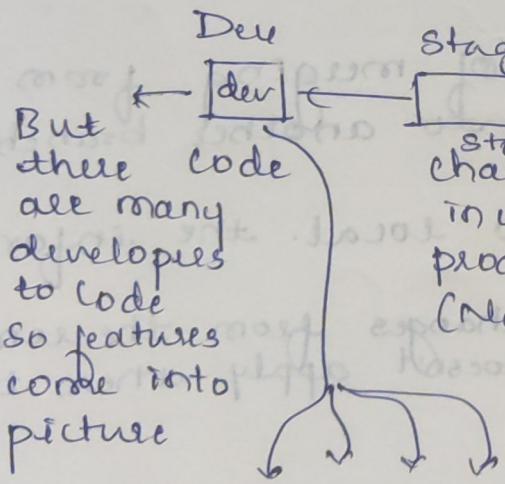


Github
main ← seed
- Master (local)
- from same seed I can create another branch (dev) [git branch dev]

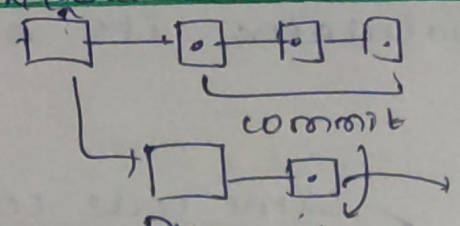
I can switch the branch => git switch dev

Note :→ Every branch in git maintains its own copy

- git status
  - └ on branch dev
- git switch master
  - is → doesn't show the file created in dev



Master - Main branch



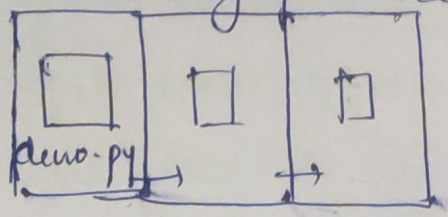
So the latest commit is called "HEAD"

* git log => provides the history of git & even head.
  ↳ represent latest commit on your repository.
* git log --oneline => displays single line
  9e148f
  380cf4d.
* git switch dev
  " checkout " } both are same - to switch

push dev branch to github
"git push origin dev"

* Merge git dev to master on github
  Go to pull requests
  ↓ ↓ create
  Base = master compare = dev.
* pull request: The process of merging from one branch to another branch.
* git fetch = from remote to local. the information
  ↳ It gets the latest changes from the remote repository, but doesn't apply them to your code yet
* Hooks :→

untracked staging tracked.



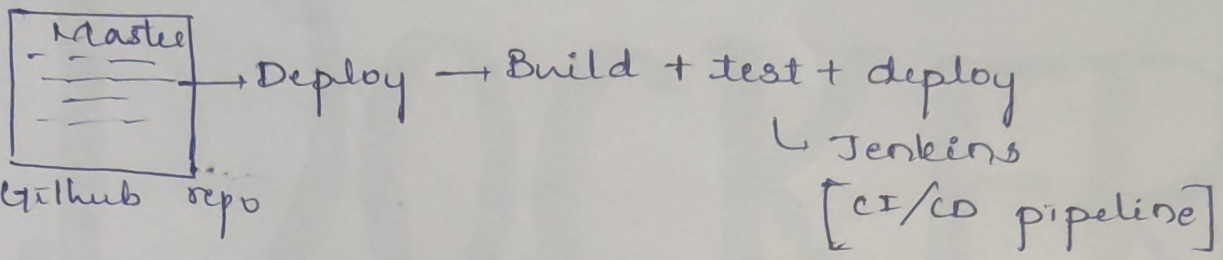
I wrote code & did the commit - I get errors, no syntax, so hooks helps to know which kind of changes I want to get commit before com

ls -a
cd .git
cd hooks
ls
vim pre-commit

* pre-commit hook ⇒ it won't commit until I change the code errors

* Github Actions :→

It is a inbuild tool of Github where we can build, test & deploy.



Select text file ⇒ Actions
Github actions scans the code
pylint is used here

git branch : list, create, or delete branches