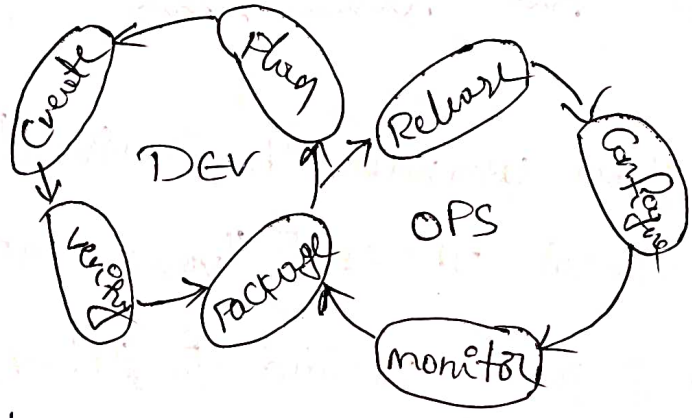
DevOPS

DevOPS:- The main Aim of DevOPS is Releasing the software to users by collaboration of developers & operational teams. it is combination of cultural philosophies, Practices & tools that increases an organizational ability to deliver application.



Devops tools:-

Git:- it's version controlling tool which is used to store code & manage the code. it can be managed by complex Projects. this tool mostly can used by developers. we need to do two things there.

i) Download Git (Gitbash) & you can do it on server (EC2 server)

ii) Code Hosting Platforms

*) Github.com, Gitlab, Bitbucket, Travis CI, Circle CI

whatever write codes by developers they are put into Github repository. So if we want make any changes then we cloned the code & do changes in Gitabash

after then Push into repository.

Generally repository lives in folder(local) & Github online Host.

Repository:- A Repository Contains all Projects files & each files revision history. it is Place nothing but where the code placed. Every Project has their own dedicated Repository.

Git (Gitbash) Config with Github Account:-

Syntax:-

git Config --global user.name "harisettivasu"

git Config --global user.email "vasu060895@gmail.com"

Here username & gmail Provided for github account Credentials.

for checking syntax:-

git Config --global -e [shows usr & email in ".git Config" file]

(or)

git Config --list (or) git Config --global --list

Help Command:- git help Config

notepad++ Config with git:-

1) Download notepad++ file download & Install in windows.
2) Add the "notepad++" Path in System variable.

Goto this Properties → Advanced system settings →
Environmental variables → select Path in system variable
→ Add Path here (application file location Path notepad++).

3) open the Gitbash & check here notepad++ in command line

Syntax:- notepad ++

Alies in Git:- Git alies are power full work-flow tool that create shortuts to frequently used Git Commands.

↳ notepad++ .bash_profile

1) Create ".bash_Profile" file throught notepad++ & vi editor & open the ".bash_Profile" file.
2) Enter alies npp="notepad++.exe -multiInst -nosession"
3) Here shortut for npp is added instead of notepad++.

Default editor notepad++ in Gitbash:-

↳ git config --global core.editor "notepad++.exe -multiInst -nosession"

↳ for creating Alies shortut for Command do

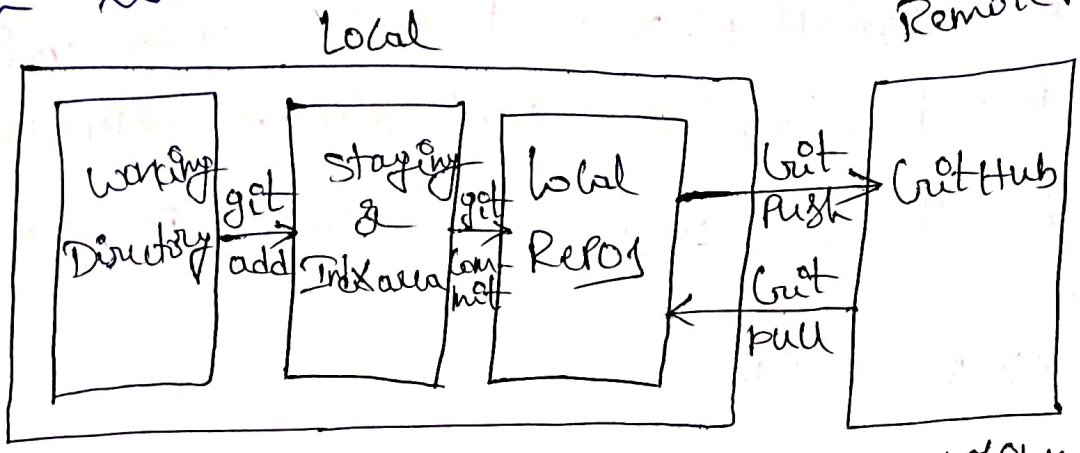
git log --online --graph --all

Syntax:-

git config --global alias.allCommits "log --online --graph --all"

here instead of "git log --online --global --all" Command & simple Command you can use "git allCommits".

Git workflow!-



Git clone!- It is a Git command line utility which is used to target an existing repository & create a clone or copy or download of the target Repository.

Syntax!-

↳ git clone "HTTPURL" "name"

Git add!- It is used to Add those changes to the staging Area.

Syntax!- Git add ⊙

Git Statu!- It is used to display the state of the working directory & the staging area (tracking & untracking files).

Syntax!- git status (or) git statu --short

Git Commit!- It is used for send the changes from staging Area to local Repos.

syntax!-

↳ git Commit -m "message"

↳ git Commit (or) git Commit -v

for the first initial commit we don't have a Parent but second Commit Parent is the before Commit. that means new Commit is linkup with the Previous Commit.

Git Push!- It is used to upload local Repository Content to remote Repository.

Syntax!- ↳ git Push
↳ git Push origin master

.Git!- It is used to tracking the files whatever you changed. this is a Hidden file. if you want to see in Git bash terminal then enter,

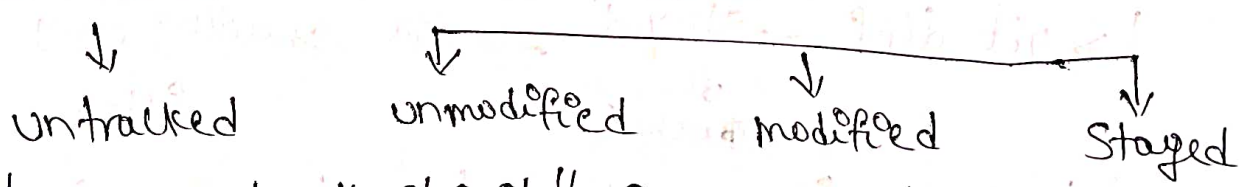
↳ ls -al → for shows all files & hidden also

when we are cloning the Project from remote repos, Automatically ".Git" file hidden file is Canning

Status life cycle!- there are two Status files,

i) tracked files!- whatever files Git knows those files are tracked files

ii) untracked files!- whatever files Git unknows those files are untracked files.



when we do "git init" Command then all files Comes in untracked stage.

when we do "git clone" Command then all files Comes in tracked Stage.

NOTE!- we can't see unmodified files in git Status after Completing staging Area. if you want edit then goto vi editor & notepad++.

.gitignore :-

create ".gitignore" file through vi editor. Goto that files activate the insert mode.

Rule :- Comments & Blocklines

Enter,

comments can be placed..

test.txt → test.txt file will not show in git status

*.txt → .txt files will not shown in entire Project in git status

!test.txt → only i need test.txt file in git status

Remaining all .txt files i don't need.

/test.txt → only current directory test.txt file will not shows.

Git diff :- it shows content differently in staging Area & file & working directory file.

Syntax :-

↳ git diff → shows content difference in staged & working directory

↳ git diff --staged → for showing only staged files.
or
--cached

Git log :- It is a utility tool to review & read a history of everything that happens to a repository.

Syntax :-

↳ "git log" → for all commits info.

↳ "git log -P" → for result 2 commits.
-2

↳ git log --online --all

↳ git log --online --graph --all

Branch!- It is a create, list, rename & delete branches its pointer.

↳ when we getinilizing (git init), default branch is comming (i.e master branch).

↳ Head will shows currently which branch you staying or currently where you are.

Syntax!-

↳ git branch -a → shows all branches here.

↳ git branch "Branchname" here → for creating branch

↳ git Checkout "Branchname" here → for switching to Particular branch.

↳ git checkout -b "branchname" → for creating branch & checkout to the that branch.

NOTE!-

→ If you want make any changes, first you can create branch then switched to that branch & make here changes.

→ Again switched Previous Branch (master) & here Command " git merge "Branchname" "

→ So we getting changes in master branch also.

or

→ Direct you can push (changes branch) to remote Repository here you can "Pull request" Conform for which branch you need to merging.

→ git branch -d "Branchnamehere" → for deleting Particular branch.

fast forward merge:- It means whatever you merging two commits from two different branches is consisting of same ancestor.

↓

means before commit is same for two branches.

Recursive Strategy:- It means whatever you merging two commits from two diff. branches is consisting of two diff ancestor.

↓

means before commit for two different branches is different.

Syntax:-

↳ git branch --merge → it shows what branches are merged.

↳ git branch --no-merge → it shows what branches are unmerged.

↳ git branch -a → shows all branches.

Commands:-

→ git restore --staged <filename> → for undo option staging Area to unstaging (working Area).

→ git reset --soft HEAD~1 → for undo option (Recharge) Commited Area to staging Area.

→ git reset --hard HEAD~1 → for undo option from Commited Area to working Area (here deleted whatever you added recently).

↓

it pass to git Commit, all (next later) Commits deleted & Content also.

merge Conflict!- It is that occures when Git is unable to automatically resolve differences in code b/w two Commits. when we changing same file & same line on diff. branches, after we doing merge Command not Comming result merge Conflict error is Coming, that merge Conflict means occures because Git doesn't know which code to keep & which code is discard.

So you can go on vi editor & remove and take any of one change greaterthen lines & one code.

↓

Save & Git add .

↓

git Commit

Git fetch!- this Command that tells your local git to retrive the latest meta-data info. from the original.

↳ if some changes done on repository (Github) at that time those change doesn't come to your local Repos. So at the time "fetch" Command is used for. changes [like added files on remote Repos] Comes to origin/main (master).

↳ git fetch origin

Syntax!- git branch -a

*main
remote/origin/Head
remote/origin/main ✓

then Again you need merge from "origin/main"

branch to "main" branch.

Syntax:- git merge origin/main [switched in main branch]

↓

shows here is -a [new file also be added].

Git Pull:- This command used to update the local version of repository from a remote repository

↳ if some changes done on repos (Github) at that time those changes doesn't come to your local repos. So at the time "Pull" command is used for update the local repos as well as remote repos.

Syntax:- ↳ git (Pull) remote origin

↳ git (Pull) remote origin master/main.

when we are doing "git Push" before that you need done first "git Pull".

Git rebase:- It is Process to reapply commits on top of another base tip. the function of rebase tool is same as of merge tool.

↳ git log --oneline --graph --all

* - Commit sweetbox branch
* - Commit master branch
* - Commit main branch

} for eg when we do rebase

↳ git log --oneline --graph --all

* — Commit main branch
|\ * Commit master branch
* — Commit main branch

} for eg when we do merge.

P4 merge tool:- It is visual diff tool that displays the differences blw file versions & helps you to resolve conflicts & merge competing version into one.

~~Git stat~~

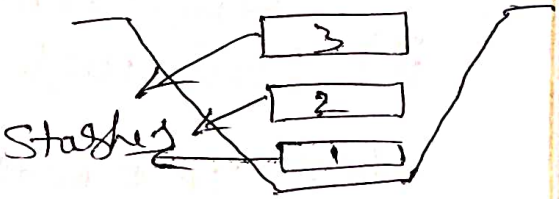
Git stash:- This Command takes your uncomited changes (both staged & unstaged). only
↳ Commits (add, ~~Commit~~) only modified & ready to add files.

Syntax:-

↳ git stash or git stash push → for going to delete in Dustbin

↳ git stash list → for showing all stashes here.

Dustbin (stack)



↳ git stash apply → for recent stash change we want to backup but this stash not deleted in dustbin

↳ git stash drop → for deleting recent stash.

↳ git stash POP → for recent stash apply & deleted.

↳ git stash clear → for stashes are cleared & droped.

Git clean:- It is used for deleting the only untracked files.

Syntax:-

↳ git clean -f → forcefully deleted untracked files [only files].

↳ git clean -f -d → forcefully deleted untracked files & folder.

Task(1) clone from Remote Repos!-

↳ first clone the content from Remote Repository.

↳ Here devit you can change or update the code in (master/main) branch & add, commit & push to remote Repos.

&

Here you can change in another branch also.

After changing you have two ways for going code to Remote Repos.

i) merge this branch to (main/master) & add to remote Repos.

ii) devit push this code to remote Repos.

here In Remote Repos, you are getting pull request & merge here to main/master branch. here you can delete branch also.

Task(2) from local to remote Repos sending code!-

step(1)!-

↳ first Create Empty Repos in Github.

↳ Here Project files first you need to do git init & adding & add & Commits changes here.

↳ Here you need to add Empty remote URL repository & then git push to empty repos.

Suppose Project folder (Contains files) are there in local System.

↓

git init

↓

git add

↓

git commit -m "message here"

↓

git remote add origin URL (Empty Repos)

↓

git Push --set-upstream origin master

or

git Push -u

↓

then you got Project folder on Remote Repos.

6. Git cloning through SSH:-

If you want cloning through ssh you must have to genrate Public & Private keys in terminal. So GOTO Gitbash & Putty,

↓

ssh-keygen

↓

After ".ssh" Directory created & in that directory we having a files.

→ id_rsa.Pub (Public key)
→ id_rsa (Pt key)
→ known_hosts
→ known_hosts.old

↓

Here see Public key by using Cat Command & Copy that entire key.

↓

GOTO remote Repos → Setting → "SSH & GPG keys" option → Here add the ssh key (Public key)

↓

the use git clone through ssh option.

↓

then files downloaded to local Repos.