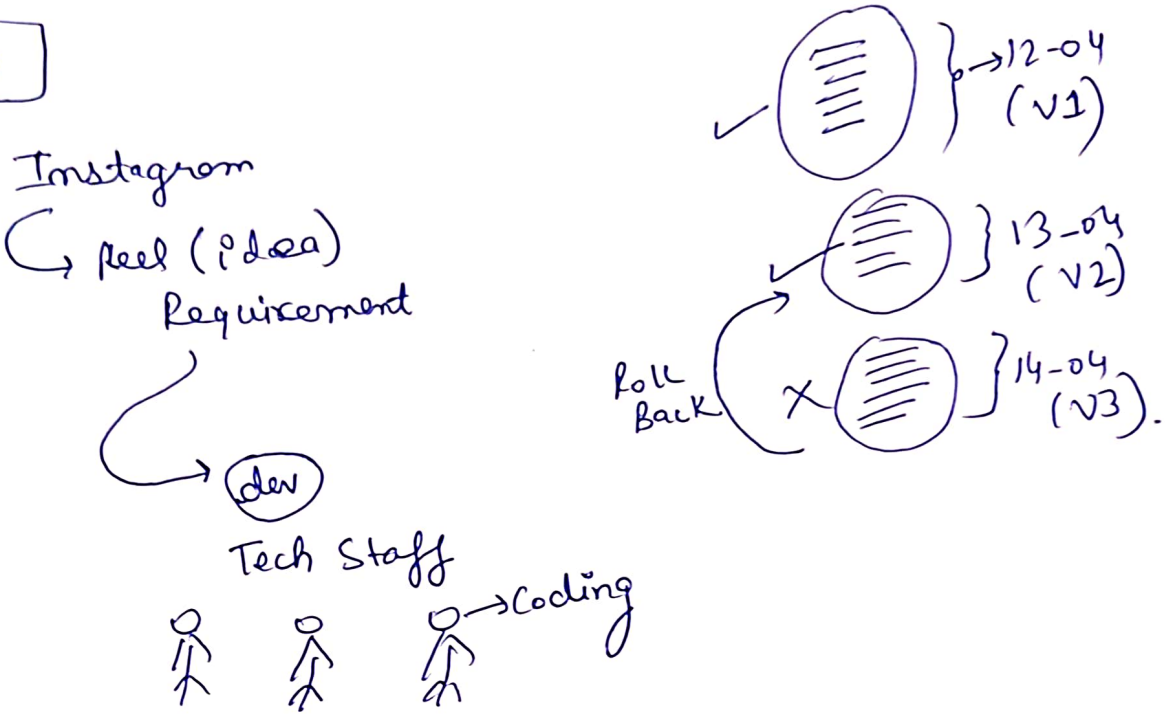
Git & GitHub

Day 1

Flow Diagram ⇒



VCS ⇒ Version Control System.

Git ⇒ DVCS (Distributed Version Control System).

↓

Tool using (VCS)

→ Git help to maintain Diff versions, track changes line by line.

Practical Implementation Git bash

Folder ⇒ Respective files.

⇒ **Create Different Version**

⇒ git init] Git creates a Hidden folder Behind the scenes. (Git initialization).

(2) ⇒ Create a new file.

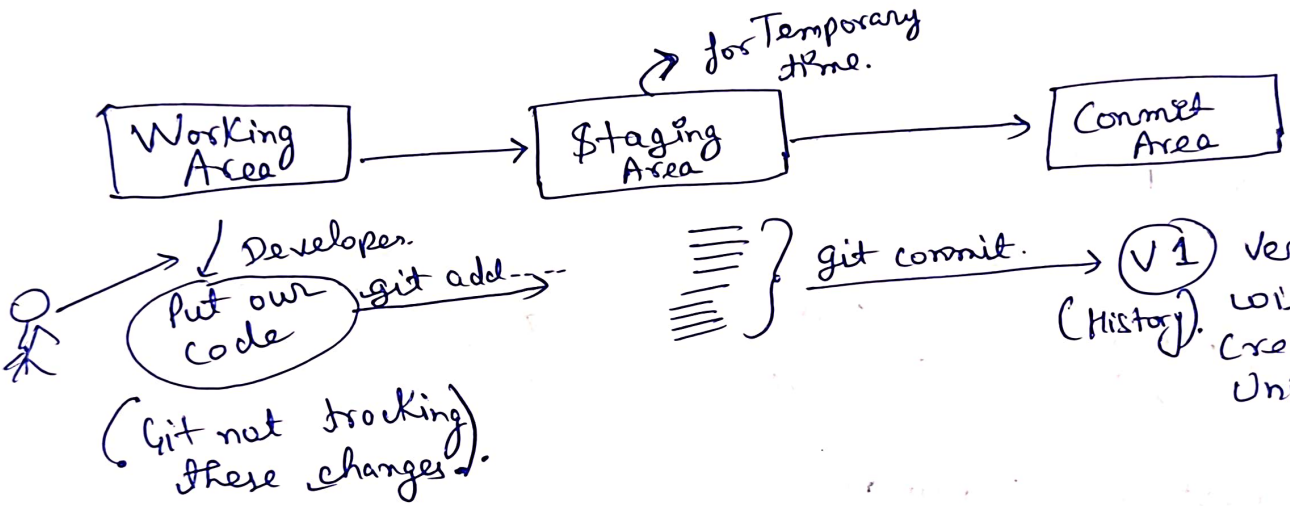
(3) git status (Not added in Staging part).

(4) git add a.txt (file will get added to Staging Area).

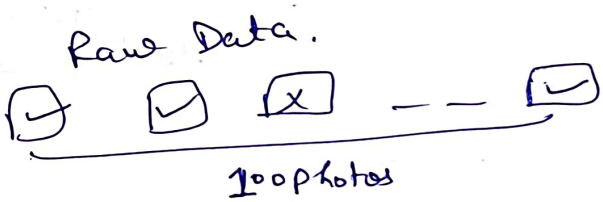
(5) As soon as we do `git log` ⇒ Show all History

(6) git commit -m "Demo".

(7) If do not want to create a version then will not `Commit` the changes.



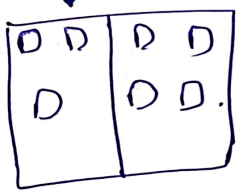
(Ex) `Photographer` Photos



↓ Select Good photos.

Commit (Put them on Album).

↓



⇒ git add.] Commit all the files from the Working to Staging Area.

⇒ .get to older versions as well.

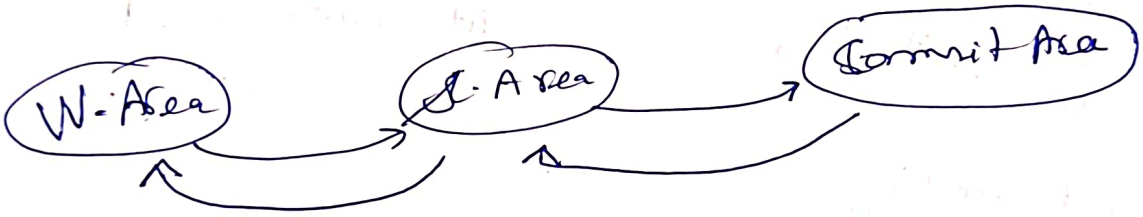
⇒ git log.

⇓ git reset _commit_Pd

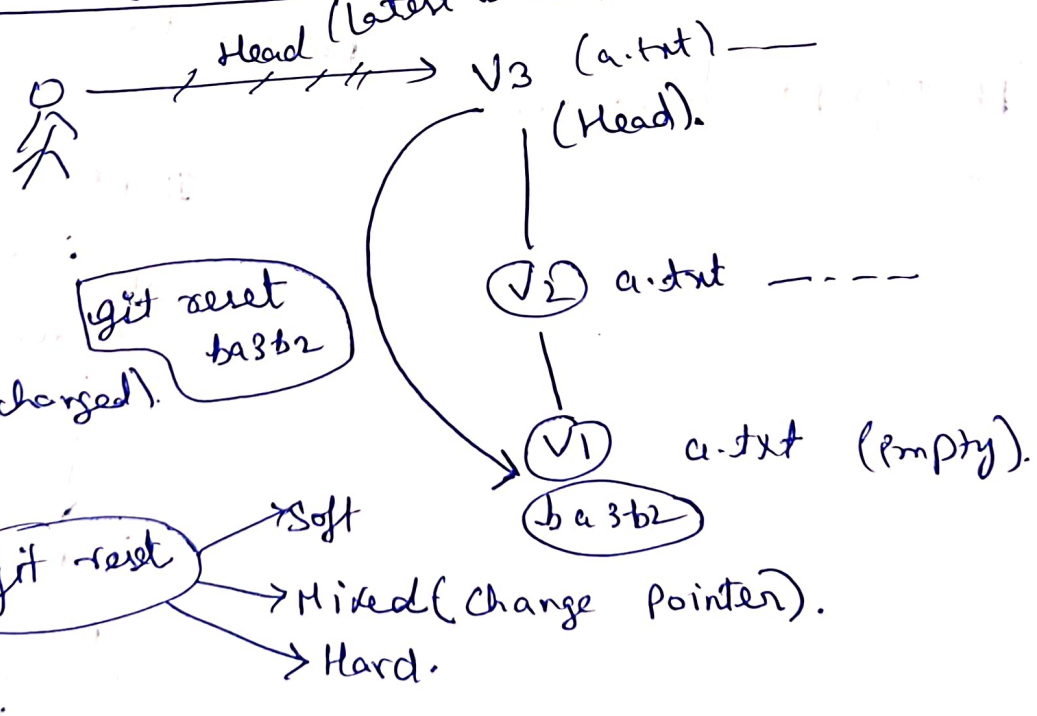
few characters as well.

⇓ git log (came one step back).

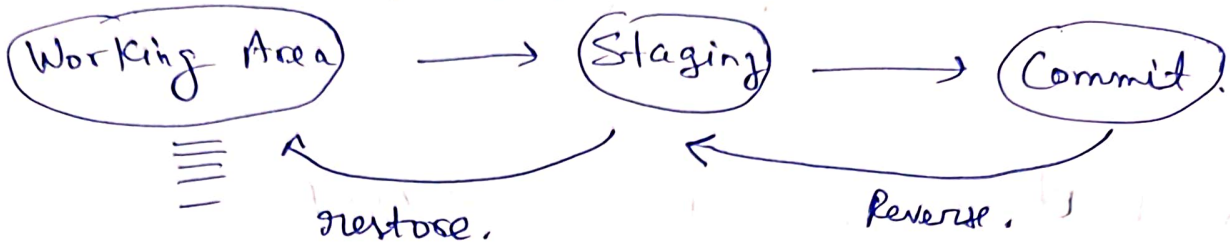
⇒ Reverse Part



→ git restore filename



⇒ Mixed reset will take Data Back from Commit Area, Remove from Staging Area as well.

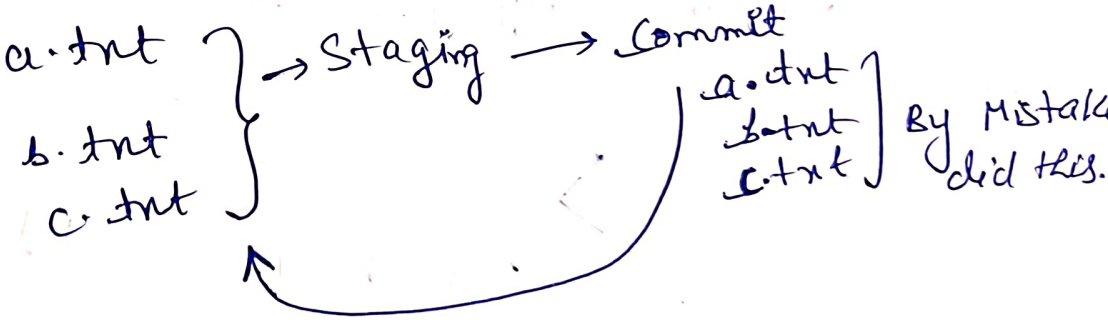


⇒ Soft Reset Remove from Commit Area but keep in Staging Area (forgot some file to be committed).

a.txt b.txt c.txt should also be added in abcd Commit.

Staging → Commit (abcd)

Diagram



git reset --soft commit id or pointer can use as well.

git rest --soft HEAD~1

Pointer

one step back.

latest Commit reference.

⇒ By default Mixed Reset will be applied.

git reset --mixed HEAD~1.

↳ unstaged the changes.

⇒ Hard Reset — git reset --hard HEAD~1.

(Remove all the Data from the 3 stages).

Basic Concepts

Why Git Called DVCS → each Developer will Have their own copy of repository.

Centralized Repository

Push Command → pushes the code to Centralized Repository.

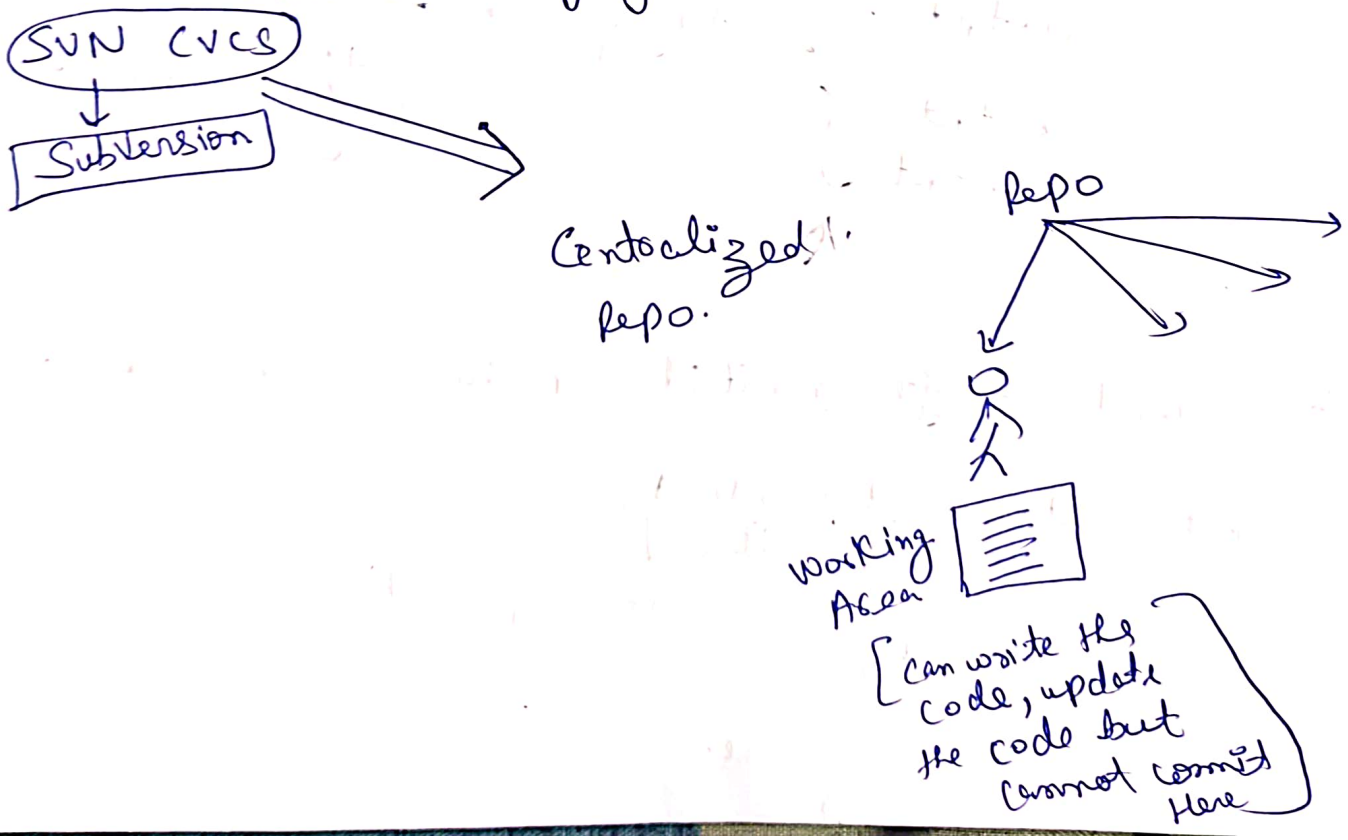
CVCS → Centralized Virtual Control System.

↳ suitable only for limited Team Members.

SVN CVCS

↓

Subversion

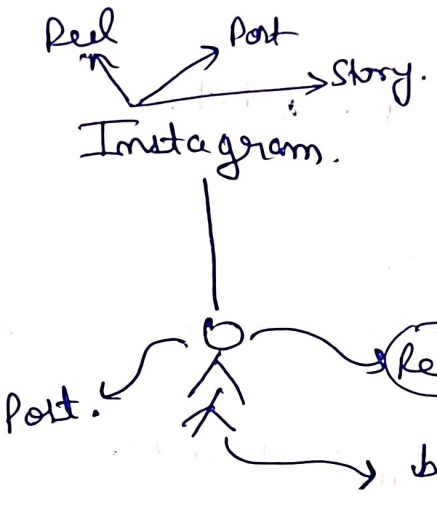


DVCS

git → local git repo { Can do everything Here and then Push the code to Centralized Repo }.

* **Branching Strategy** git branch.
  ↳ Master.

Flow Diagram



Branch will help to Manage work in more efficient Manners.

Master branch ⇒ Ready for Deployment only keep those changes.

⇒ git branch feature/reels.

⇒ git switch feature/reels.

⇒ git log (Same as we Have for Master).

⇒ Git switch master.

ll. ⇒ unaware about the change in feature/reels.

⇒ git branch → for another bug.

bug/post

⇒ git switch bug/post.

⇓

ls. (Real branch changes. not aware by this bug branch)

⇒ git log.

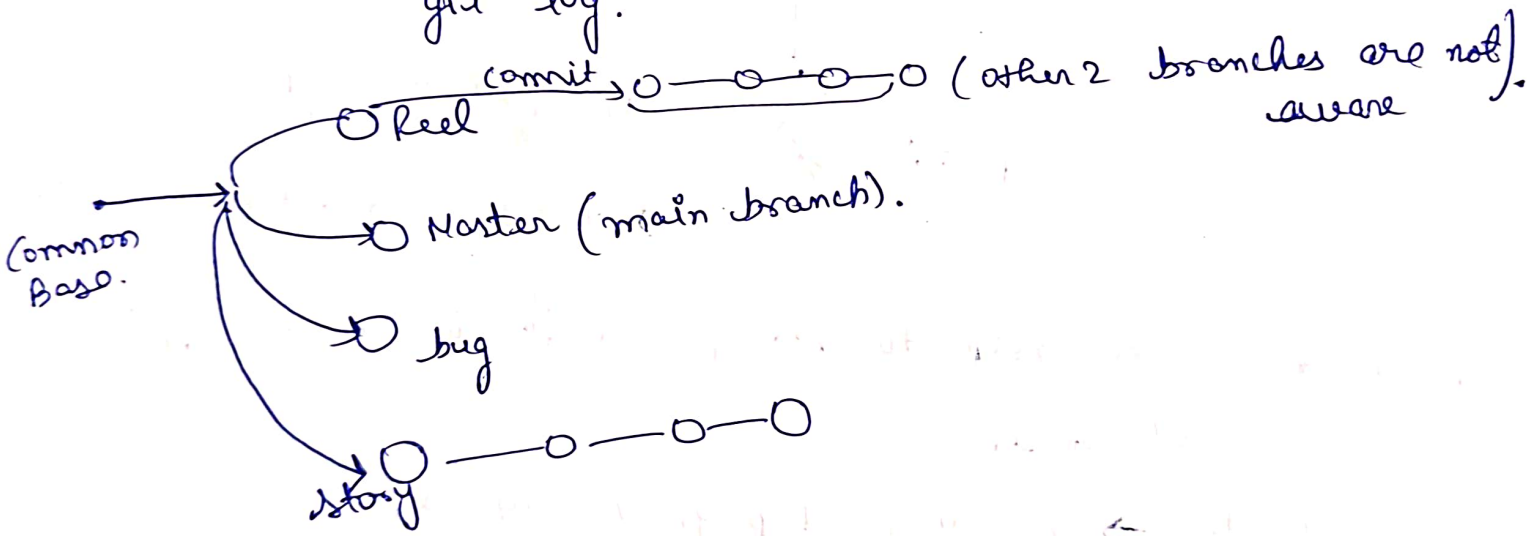
⇒ git switch feature/reels.

⇓

ls ⇒ open a.txt.

⇓

git log.



⇒ git log --oneline (cleaner logs).

on master Branch

⇒ git merge feature/reels

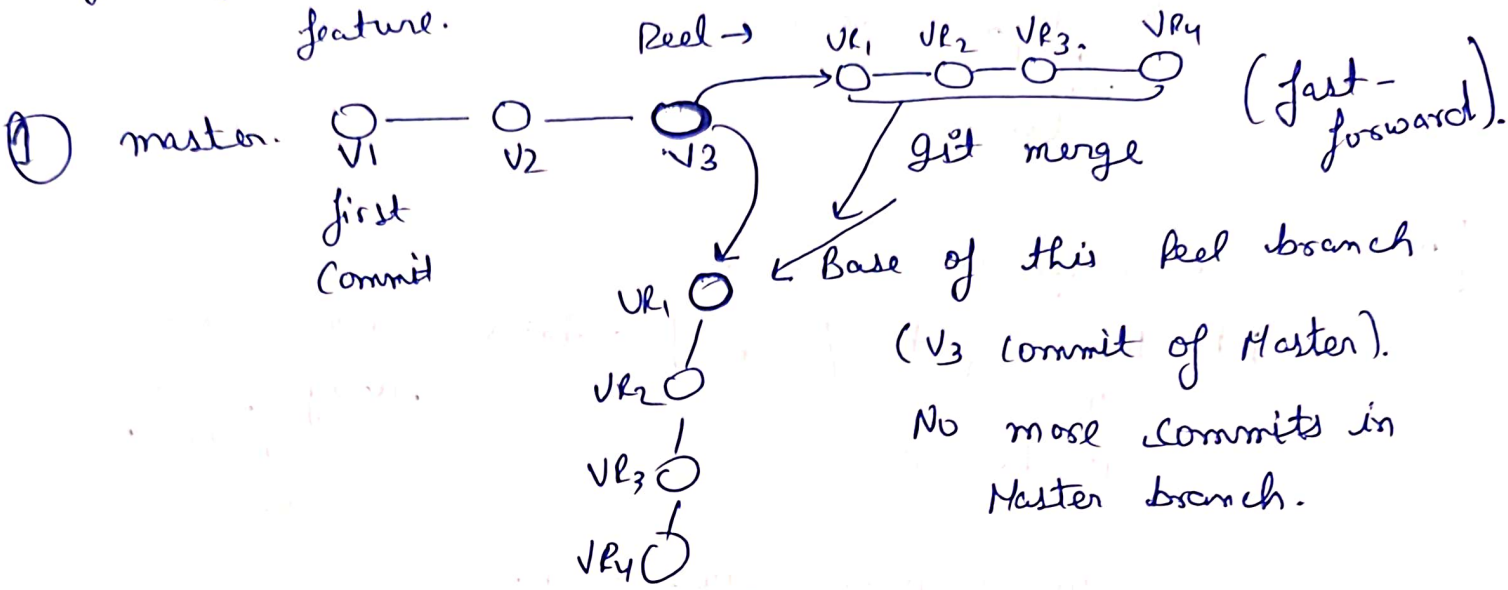
⇒ Make Proper use of VS code for Git.

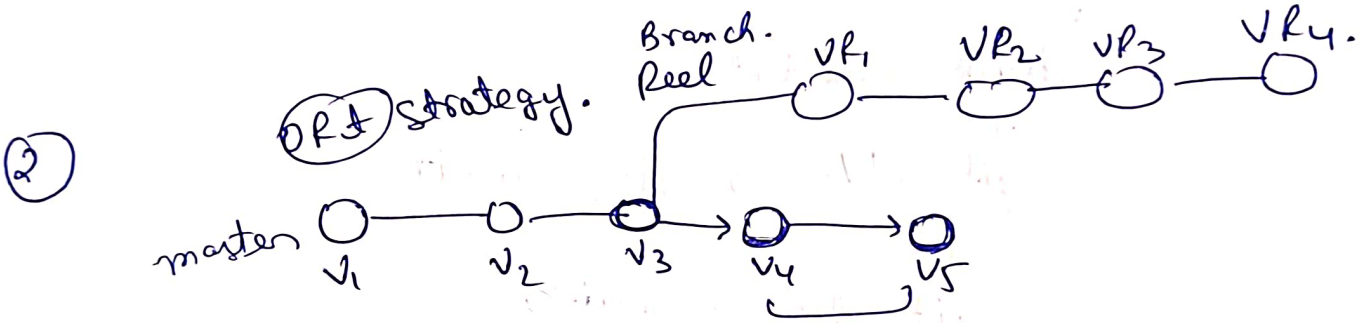
⇒ Each Branch will Have their own Commit History.

⇒ Merge Strategy

↙

fast-forward.





Practical Example

mkdir gitday1-secondpractical

⇓

cd --

⇓

ls.

⇓

git init

⇓

touch a.txt ⇒ git add . ⇒ git commit

(By default Master will be default Branch).

⇒ add something in file ⇒ git commit

⇒ touch b.txt ⇒ same as we do for a.txt

⇓

git log --oneline

⇓

ls.

Now create a feature branch

git branch feature/reel] contains exact commit history from master branch. as (Base)

⇓

git branch.

⇓

git switch feature/reel.

⇓

touch reel.txt

⇓

git add. ⇒ git commit.

⇓

git log --oneline.

⇓

git switch master.

⇓ ls

⇓

git log --oneline.

⇓

vi a.txt ⇒ add ____ and commit.

⇓ git log --oneline.

⇓

git switch feature/reel (Notice arg.

Tools to Visualize Commit History

GitKraken

⇓

Update data in any file

⇓

git add . ⟹ git commit "second by reel"

⇓

git switch master

⇓

Make some changes Here then add & commit.

⇓

git switch feature/reel

⇓

one more commit (save and commit).

⇓

git log --oneline.

⇓

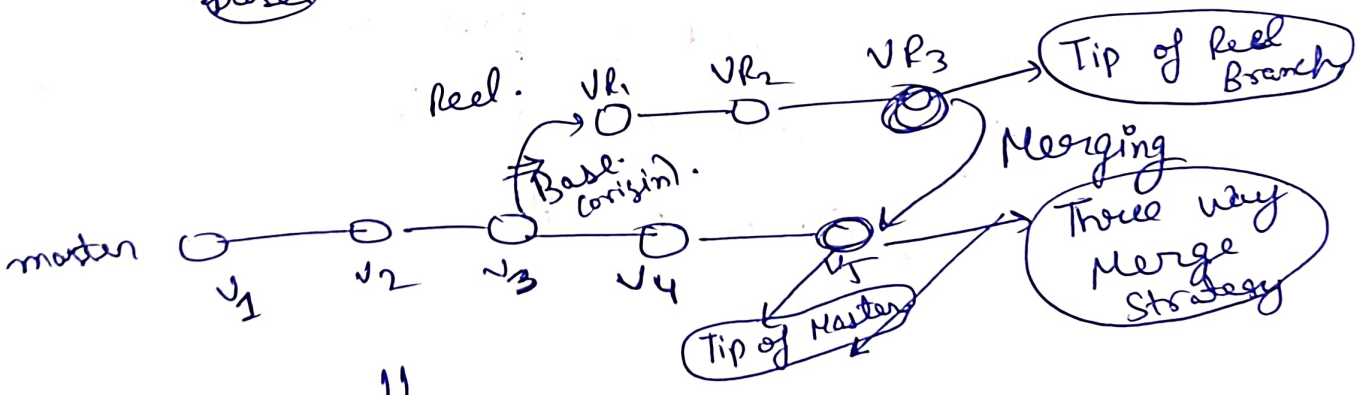
git switch master

⇓

git log --oneline.

V.Imp

* Initially both master & feature(reel) Have the Common base.



⇓

git merge feature/reel) Right Now on master Branch.

⇓ add message

⇓ ORT Strategy.

⇒ one extra commit is there which is Merge commit. (To say only some merge has been Happen).

⇒ git diff (compare the changes b/w versions).

⇒ git commit takes snapshot of the complete project.

⇒ a.txt → 10Kb
file created
13th April (11:13 AM)
Commit (Snapshot 1)

20Kb
b.txt created → a.txt, b.txt
Commit (Snapshot 2)

(1Kb)
c.txt file created (Upate B.txt) (21Kb)
Commit (Snapshot 3) → a.txt, b.txt, c.txt

⇒ git creates snapshot and store them as blob. (Binary Data).

⇒ for each blob creates one Hash value as well.
→ with the help of Hash value git will identify whether a value is being changed in a particular commit or not.

⇒ git abord (to abord the conflicts).

More Merging Strategies

ORT replaced with recursive Strategy

1.) Multiple branches ⇒ wants all the merge in Main branch

→ Create 2 branches (bug/123)

→ git switch ~~branch~~ bug/123

⇓

1 Commit done.

(rm -rf filename).

→ git switch ~~branch~~ bug/124

⇓

1 commit done.

→ git log --oneline

→ git switch bug/123

→ git log --oneline

→ git switch master.

→ git merge -s octopus bug/123 bug/124

Octopus Strategy

two branches will be merged to master branch.

→ ls] Both branches changes are there.

→ git log --oneline