

## Git & GitHub

- Dr. Upadhyay

(Associate Software Engineer @ Spansion Tech.)

Git v/s GitHub :

Git → \* A distributed version control system that tracks the changes in a file / repository.

\* Version control system is nothing but save changes over time.

- track the changes to the files over time.
- it helps to developers to collaborate, maintain history, manage diff versions of a project.

\* It uses distributed model, which means each clone contains complete history.

GitHub → A cloud based hosting service for git repos

\* nothing but website that stores Git repos online

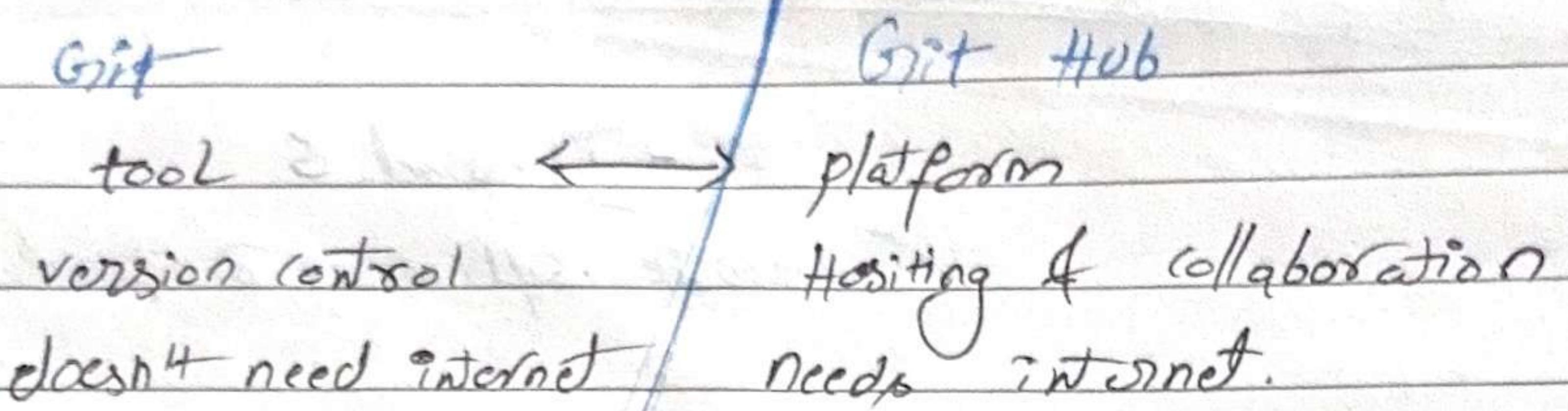
\* It provides collaboration features like pull request, issues, code review, CI/CD, etc.

GitHub is an example, the alternative to GitHub → GitLab

→ Bitbucket

→ Azure DevOps

Basically



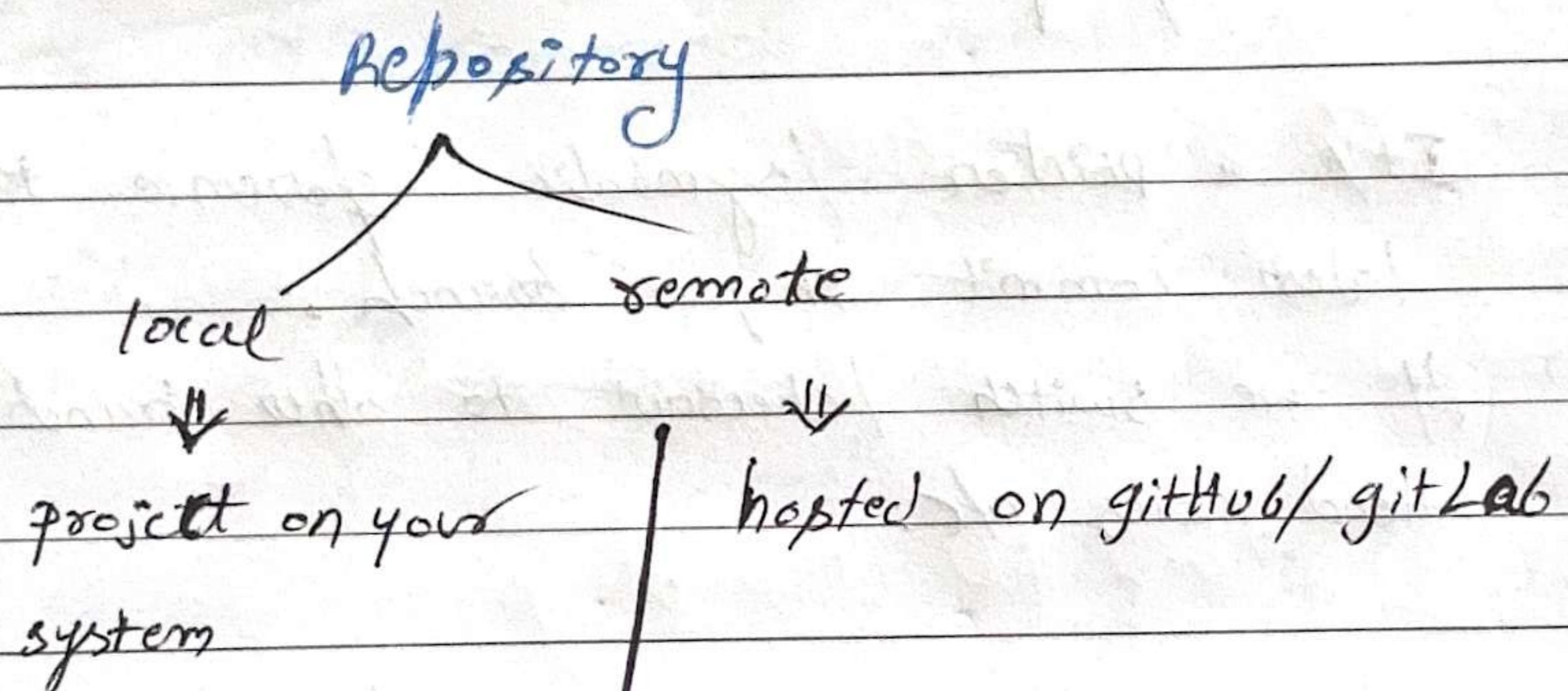
If we have to explain this in simple way

- In your laptop, write "Hello world" program & save. This is only present in your local storage & none of friends can access.
- If you push / add this program to GitHub / gitLab etc & provide access as public everyone can able to see the your program & can download. → This is GitHub.
- If you want to change your code, like adding another function then Program in local & program in GitHub is diff. This diff is recognized by Git & tells you that there is a diff.
- You have do some steps like committing, pushing & then Merge into the original code which is hosted on GitHub for all this Git provides some commands called Git commands.

## Git Concepts

Repository  $\Rightarrow$  its nothing but project folder managed by Git

Eg: on your laptop you just create a folder called Projects. Inside this you just add files of your projects. The folder project is called Repo



~~all changes local~~  
all changes local  
until you make it [ if branch has to remote then from local  
it has to pushed & then merge into the  
as remote by pushing ] main.

How repository is used by git to track changes?

$\Rightarrow$  git stores all metadata for a set of files & directories including history of changes, branches & configuration

Commit ; A snapshot of code/project at a point of time  
\* It's nothing but an object of a code that contains tree (directory structure), parent commit author info, time & mainly unique SHA-1 hash

An unique identifier for each commit.

→ very helpful while merging or reverting one particular commit.

Branch ; An copy of Project used for development

HEAD ; It's a pointer /symbolic reference to the latest commit in your branch.

If we switch / checkout to other branch in repo Head moves etc.

Working directory, Staging & Repository :

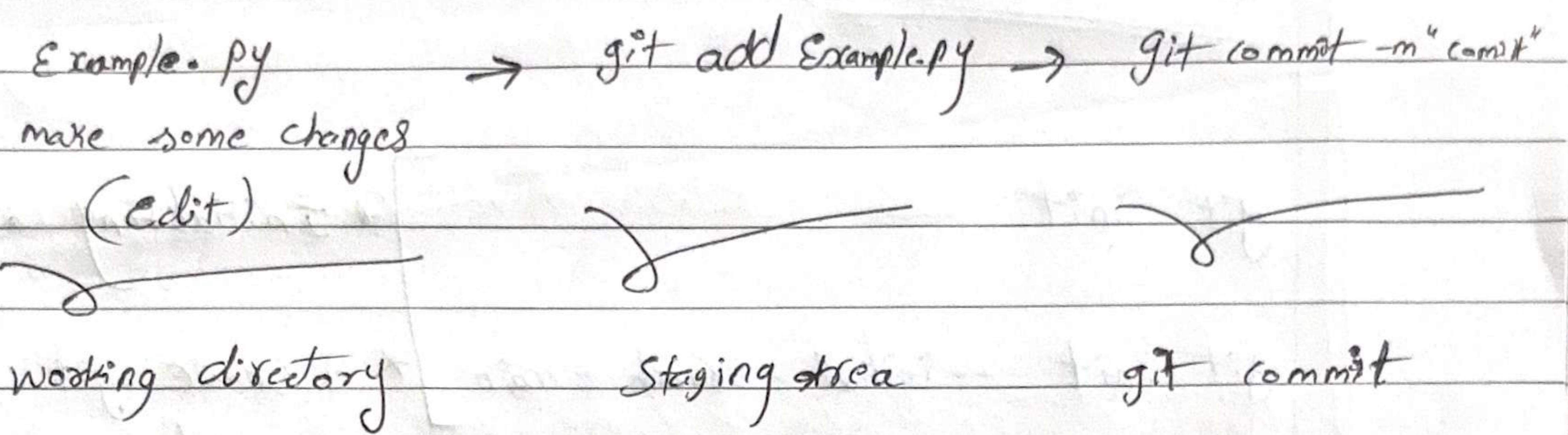
Current state of files in a file system

A binary file that stores info about what will go into the next commit

git directory containing all git data structures.

files which are added using "git add" are moved to Staging area

Eg:



## Git configuration & setup

all git commands starts with git --->

git config [ --global | user.name "username" ] set your  
 git config [ --global | user.email "EmailId" ] identity.

git has 3 levels of settings

- system-wide : stored in /etc/gitconfig, applies to all users
- User-wide : ~/.gitconfig applies to current user
- project-specific : .git/config, applies to current repo

## Git Commands :

git init

\* Initializing a repository

git init --initial-branch=main

\* It creates .git directory with subdirectory for objects & configuration files.

git add file-name → add specific file to staging area

git add . → add all files

git add \*.js → add specific file types

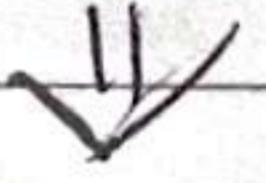
git add -i → add files interactively.

git add -u → add tracked files (skip new files).

### Checking Status

git status → Show repo status, gives which file are tracked, not added to staging Area.

git status -s → Show in short format



Show state of working directory & staging area by comparing file bases & timestamps.

making  
commit

`git commit -m "message"` → commit with message

`git commit -m "short message" -m "Detailed message"`  
commit with <sup>↓</sup> detailed message

Add +  
commit

`git commit -am "message"` → Add & commit in one step  
only tracked files only

Edit  
last  
commit

`git commit -- amend` → Amend / edit the last commit  
instead of creating new commit, it  
modifies the most recent commit in the  
history.

`git commit -- amend -m "Edit message"` → edit last commit  
message

`git add missed_file.txt` ] → add missed file into the previous  
`git commit -- amend` commit

Show  
History

`git log` → view commit history

`git log --oneline` → view commit history one line per commit

`git log --graph --oneline` → show graph of branches

`git log -p` → Show changes <sup>in</sup> each commit

`git log --author = "name"` filter by author name

clone

git clone [url : http://github.com/username/repo.git]

repo

→ clone a repo into system/directory

git clone url my-project → done into specific dir

git clone -b branch-name url → clone a specific branch

cloning a repo → creates a complete copy of repo  
including all branches tags & history.

The remote repo is automatically added  
as "origin" >

git diff → Shows unstaged changes.

## Branch & Merging ;

A branch is just lightweight, movable pointer to a specific commit.

It's nothing but a version of the main project, where developers can make the changes without affecting original version (before merging).

- \* `git branch` → list branches
- \* `git branch -a` → list branches including remotes.
- \* `git branch feature-name` → create a new branch.
- \* `git checkout -b feature-name` → create & switch new branch
- \* `git checkout branch-name` → switch to branch.  
↳ checkout is also works, switch is also work exactly same as checkout.
- \* `git branch -m old-name new-name` → Rename branch.
- \* `git branch [-d] branch-name` → delete branch.  
↳ instead of -d use -D for force delete.
- \* `git merge branch-name` → merges the changes from branch-name into the current branch.
- \* `git merge --squash branch-name`  
→ Merge with combining all commits into one.

Eg; Imagine the project repo is named as 'A'  
'A' contains original codebase for the project-  
which is the main branch.

Now you are the developer who needs to develop  
a new feature (say writing new endpoint).

You have to create new branch called 'B'  
As of now whatever present in 'A' is also  
present in "B"

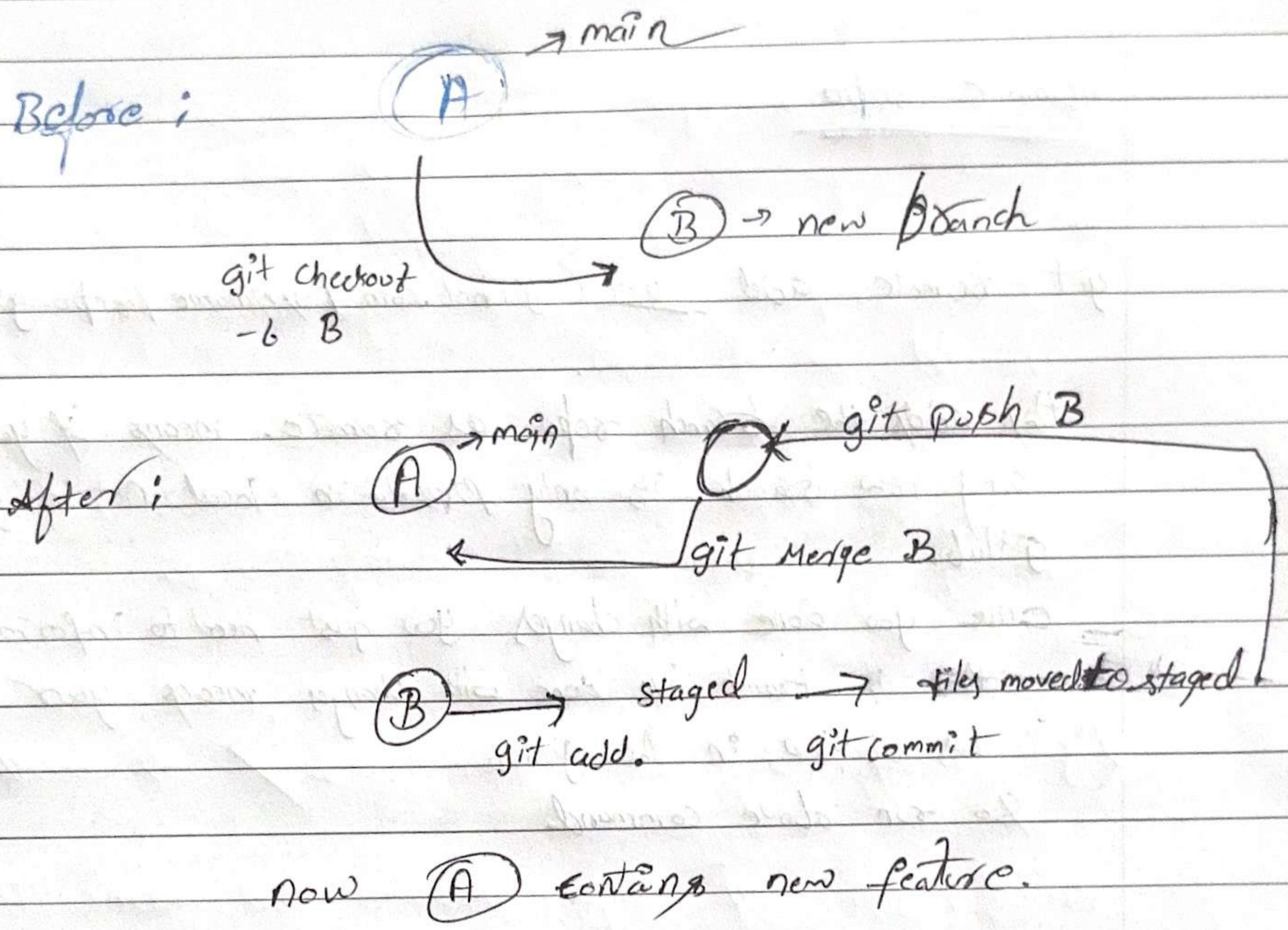
You starts working / writing an code logic in branch  
'B', tested in local works, then you have to  
merge into 'A' so that the new feature need to  
be deployed in production (lower env.).

- You add files which you modified
  - add a commit message
  - Push / push upstream to make the branch remote
    - ~~done~~ (we will know this concepts in upcoming page)
- now Branch 'B' has new feature upon the old Project

$$\text{B} = A + \text{new feature}$$

so Merge B into A

This how git branch, repo works.



# Merge Conflicts

~~when git can't automatically combine changes from diff branches.~~

\* conflicts occur when the same line in a file have been modified  
diff in the .diff branch being merged.

You have to solve conflicts manually then

git add -file list

git commit

~~git merge --abort~~

## git merge tool

Remote repo :

git remote add url (github.com /username /repo.git)

this adds the branch repo as remote. means if you create any new branch its only present in local not in github or gitlab.

Once you done with changes you just need to inform github that this ip branch i've done with changes means your branch has to view in the github.

So run above command.

To get all remote branches → git remote -v

Fetching & Pulling :

git fetch → get the changes / commits from remote / repo  
but it doesn't merge into local branch.

git fetch origin → fetches from specific remote

git pull → it will pull the changes (fetch + merge).

it always pulls from remote (main) branch into local branch

git pull origin branch-name → pull changes from branch-name

pushing changes

git push → push the changes to remote

it's nothing but updates the changes on top of the current remote branch.

git push origin branch-name → Push to Specific Branch.

git push -u origin branch-name → This will set branch as remote  
then try to push

nothing but (git remote add branch)

+

git push origin (branch)

Eg: Take previous example, (A) → contains main codebase (main branch)  
(B) → copy of A + your changes (locally)

So when you try to merge B into A, the steps need to follow →

(B) → make changes → git add → git commit → make B as  
remote → push changes to B.

now (A) & (B) are 2 diff branches in remote, I need  
only one branch as main so B has to merge / combine into (A)  
so git merge B

Take same Example , B has 3 commits means while you are doing changes, you did it in 3 levels

- You pushed all 3 changes/commits but Manager needs only one particular change and rest 2 is not needed as of now.

Either you have remove 2 commits (revert) OR select only one and merge (cherry pick).

{git cherry-pick commit-Hash (Id).} → cherry pick

Selects / gets only changes associated that Commit Id.

we can cherry pick from multiple commits like

git cherry-pick commit\_1 commit\_2

→ You working on a project in B branch, suddenly someone called needs another requirement which has highest priority. You need to change/ create new branch and start work on it, but you done so many changes in B branch if you switch to another branch it will lose

In other hand you need to commit before testing.

In this time you have to use { git stash }

This will save the changes locally & temporarily.

OK you saved it locally when you switch back branch (B)  
how to get your changes back?

~~git stash~~ git stash apply → get back the changes  
from latest stashed.

~~git stash~~ git stash pop → Apply + remove stash.

What if you stashed multiple times?

git stash apply stash@{2} → get back the changes  
from stash. 2nd.

### Revert

git revert → revert means get back to old changes by  
removing the changes i did.  
which is nothing but undo

git revert commit-hash → You can get commit hash  
from github/gitLab  
②

You can get it by git log  
You will commit hash with  
name.

- \* As a beginner it's enough to know about git and github
- \* Most of the BE students developed / prepared projects for their colleges - add them onto your git echo and provide git echo link in your resume.
- \* How to add ~~git~~ project into your git echo?
  - open the project main dir
  - open cmd in this dir
  - git init
  - git add .
  - git commit -m " message "
  - In your github profile, create new repository & copy the echo url (https)
  - git branch -M main
  - git remote add origin copied url >
  - git push -u origin main

That's it your project is safe within github

- Dnyaneshwar S

Associate Software Engineer @  
Opinion Technologies, Bangalore

You can use the below link for practice how git command works visually  
<https://learngitbranching.js.org/>