__/__/____

Git & GitHub

- Drupad. S
(Associate Soft Engineer @ Speridian tech.)

Git v/s GitHub :

Git → ** Distributed version control system that tracks the changes in a files / repository.

* version control system ; is nothing but save changes over time.
  - track the changes to the files over time.
  - it helps to developers to collabrate 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 repo's
* nothing but website that stores Git repo's 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

Git	Git Hub
tool ⟷	platform
version control	Hosting & collaboration
doesn't need internet	needs internet.

If we have to explain this in simple way

→ on your laptop, write "Hello world" program & save This is only present in your local storage & none of friends can not access.

→ if you push / add this programm to github / gitLab etc & provide access as public Everyone can able to see the your program & can download. → This is github.

→ if your want to change your code, like adding another function then Program in local & program in github is diff this diff is recongized 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 → 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**

Repository
- local ⇓ project on your system
- remote ⇓ hosted on github/gitlab

~~[illegible]~~
all repo are local until you make it as remote by pushing

if repo/branch has to remote then from local it has to pushed & then merge into the main.

How repository is used by git to track changes?

⇒ 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 helpfull 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** :

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 files which are added using "git add" are moved to staging area	.git directory containing all git data structures.

__/__/____

Eg;

Example.py → make some changes (Edit) → git add Example.py → git commit -m "commit"

working directory — Staging area — git commit

Git configuration & setup

{All git commands starts with git ---}

git config --global user.name "username"
git config --global user.Email "Email/Id"

} set your identity.

⇓

git has 3 levels of settings

→ system-wide : stored in /etc/gitconfig, applig to all user
→ 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 create .git directory with subdirectory for objects refs. & configuration files.

Adding to staging area

`git add file_name` → add specific file to staging area

`git add .` → add all files

`git add *.jp` → 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

⇓

Shows 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 detailed message

Add + commit

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

Edit last commit

`git commit --amed` → Amed / Edit the last commit instead of creating new commit, it modifies the most recent commit in the history.

`git commit --amed -m "edit message"` → edit last commit message

`git add missed_file.txt`
`git commit --amed`] → add missed file into the previous 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 in each commit

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

__/__/____

clone repo

git clone https://github.com/username/repo.git url
→ 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 a 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 light weight, movable pointer to a specific commit.

its nothing but a version of the main project, where developer's can make the changes without effecting orgical 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 B 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 deploye in production (lowerenv).

→ You add files which your modified
→ adds a commit message
→ Push / Push upstream to make the branch remote
(we will know this concepts in upcoming page)

Now Branch 'B' has new feature upon the
old project

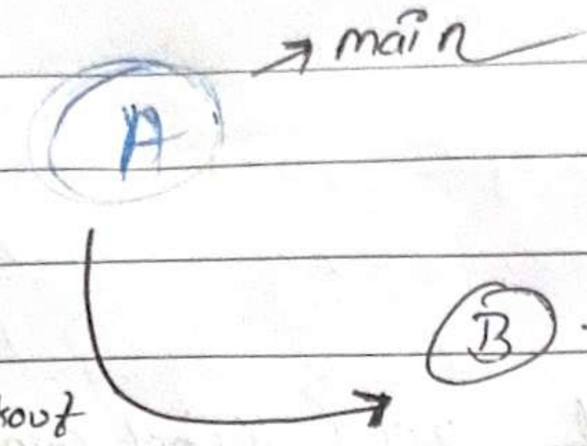
B = A + new feature

So Merge B into A

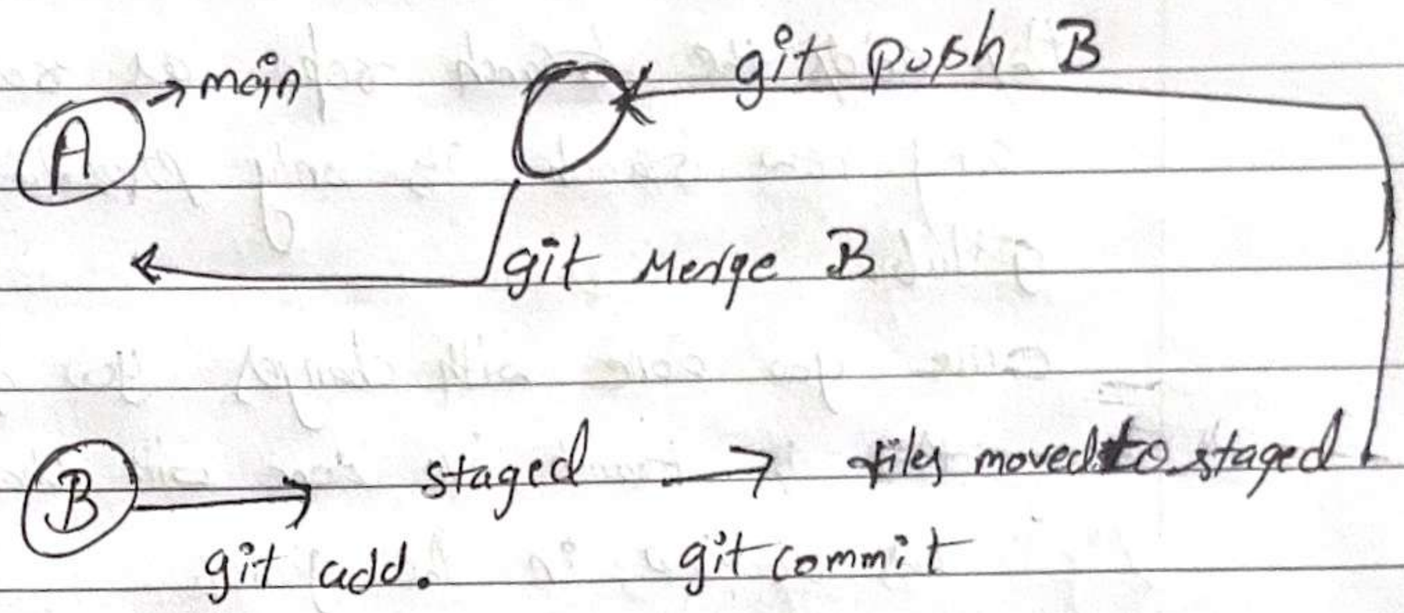
This how git branch, repo works.

__/__/____

Before :



After :



now (A) contains new feature.

Merge conflicts :

- when git can't automatically combine changes from diff branches.
- conflicts occur when the same line in a file have been modified in the diff branch being merged.

you have resolve 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 ive 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 are.
(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, it's [redacted] 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 loss 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 apply~~ 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@2y → 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 → can get commit_hash from github/gitLab
(or)
you can get it by git log
You will commit_hash with name.

__/__/____

* As a begginer its enough to know about git and github

* Most of the BE students developed / prepared projects for their colleges. add them into your git repo and provide git repo link in your resume.

* How to add projects into your git repo?

→ open the Project main dir
→ open cmd in the dir
→ git init
→ git add.
→ git commit -m " message "
→ In your github profile, create new repository & copy the repo 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

- Dnopod.S
Associate Software Engineer ⊙
Speridian technologies, Balore