Git and Github.

1. What is Git actually ?

⇒ The first thing we should know is that →

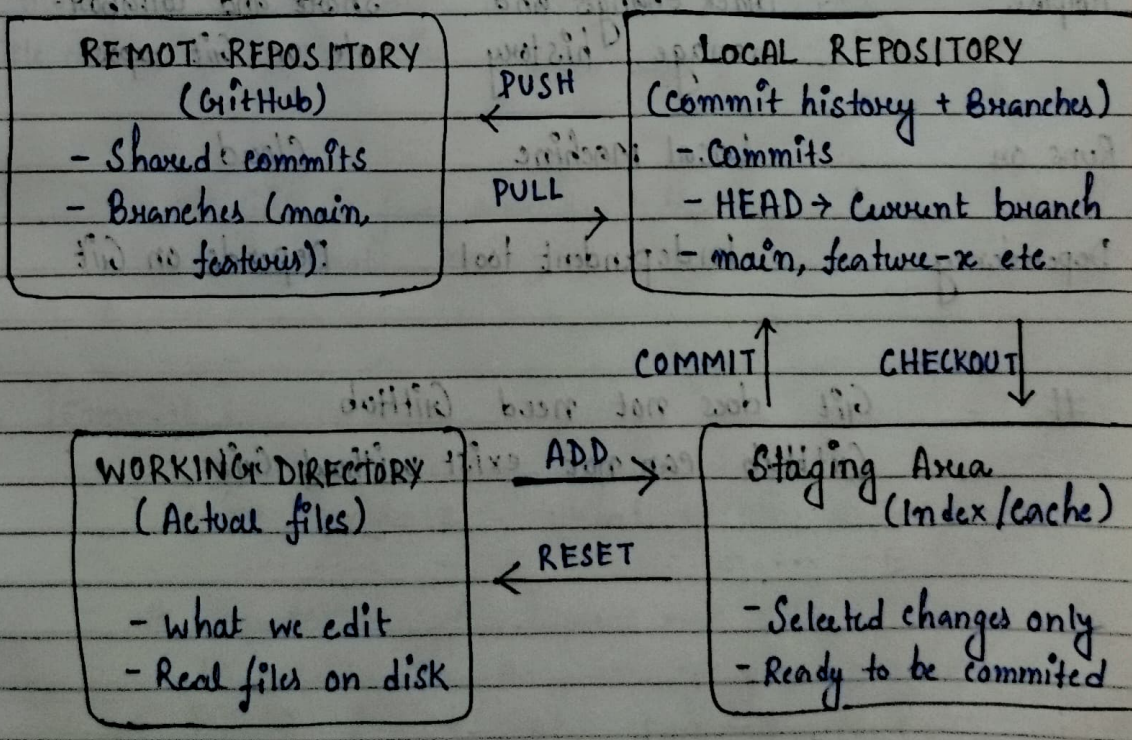
Git ≠ Github.

Git is a distributed version control System (DVCS)

↳ What does it mean ?

↳ i) Every developer has a complete copy of the repository.
ii) That copy includes full commit history
iii) No permanent dependency on a central Server.
iv) We can commit, branch and inspect history offline.

2. Git as a System



3. Why need Git ?

Without Git	With Git
° No history of changes	° Every changes are recorded
° No accountability	° Can move backward
° Collaboration will be difficult	and forward in time
	° Multiple developers can work parallelly

4. Git Vs Github

Points	Git	GitHub
Defination	Distributed version Control System	Remote hosting & Collaboration Platfrom
Purpose	Track changes and manage history	Share and collaborate on Git repos
Runs on	Local Machine	Cloud
Dependency	Independent tool	Depends on Git

Git does not need GitHub
GitHub can not exist without Git.

5. REPOSITORY

A Repository is → A project directory
is → A hidden .git folder

.git folder contains
- Entire commit history
- All branches
- All metadata
- Git's internal database

If .git is deleted Project is no longer a Git repo.

{* Commands}

Create local repo → git init

6. Git's Internal Data Model

Git stores data as objects

Blob	Tree	Commit	Head
File content	Directory structure	Snapshot + metadata	Pointer to current branch

Git is fast. because it stores Snapshots not file changes.

7. Commits

A Commit = Snapshot of the whole project.

Commits are for humans not git.
It tracks the progress of your project and Provide detailed story regarding your project to viewer.

Each commit contains →
- Snapshot of tracked files
- Commit message
- Author name and email
- Timestamp
- Parent commit hash

{* commands *}

- git status // shows which files are untracked, modified, staged
- git add file.txt // Stages only file.txt
- git add . // stages all changes in the current directory.
- git commit -m "clear explanation of what changes"

// create a new git from the stage files
Saves a snapshot of the project into the local repository.

8. COMMIT HISTORY & INSPECTION

Git keeps a directed history graph not a straight line.

{* COMMANDS *}

View history

- git log
- git log --oneline

Inspect changes

git show <commit-hash>

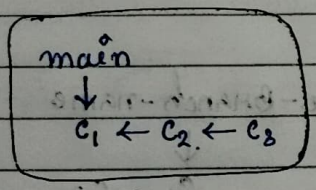
it allows :→
- Auditing
- Debugging
- Rollbacks
- Accountability

9. Branches

Branch ≠ copies / duplicate folder.
Branch = Lightweight pointer to a commit.

Initial State

#At initial state only one branch main exists.

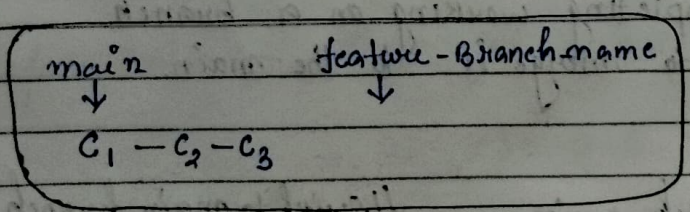


{ main = lable pointing to C3
Head = points to main
C1, C2, C3 = commits

After creating new branch.

{* commands}

git branch feature-Branch-name.



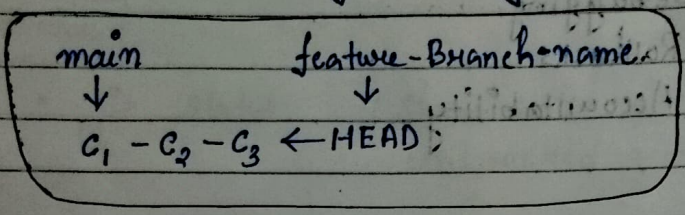
← points to current commit. we have to switch branch to use.

Both branches, point to the same commit NO new commit is created. NO files are copied.

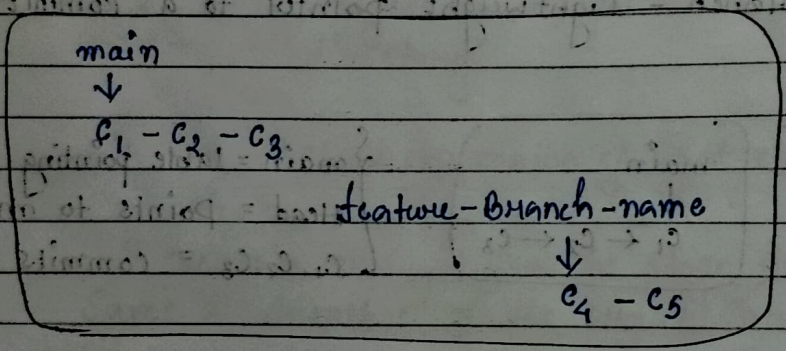
{* Commands}

git switch feature-Branch-name

Moves HEAD to the (feature-Branch-name) branch.
Updates working directory to match that branch.



NOW we are using new created branch and leaves the main branch untouched.



HEAD → • A pointer to the current branch
• Indicates where new commits will go

10. MERGING

After completing working on a branch we have to merge it to the main.

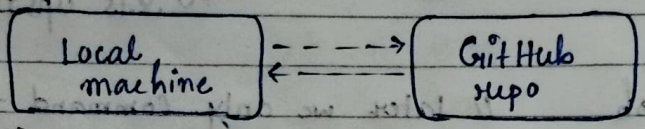
{* commands}

git switch main //Switch to main branch.
git merge feature-Branch-name.

Git creates a merge commit combines history.

11. REMOTE REPOSITORIES

A remote is a reference to another Git repo (usually GitHub)



{** commands}

Add remote

git remote add origin [Link of the repo from GitHub]

Verify

git remote -v

12. Push, Pull, Fetch

We have three places.

(Working Directory) → (Local Repository) → (Remote Repo)
[files] [commits] [GitHub]

PUSH

git push sends local commits to the remote repository.

Before Push :→ Local → A-B-C (HEAD)
Remote → A-B

After push → Local → A-B-C
Remote → A-B-C // Repo gets updated.

{* command}

```
git push origin main  // First t
git push -u origin main  // First time only
                         -u Sets upstream.
git push  // later we only command this
```

git fetch

git fetch downloads commits from GitHub, but:

- does not merge
- does not change our files
- does not touch our current branch.

it updates remote-tracking branches like → origin/main

Before fetch :→

Local main → A-B

Remote main → A-B-C-D

After fetch :→

Local main → A-B

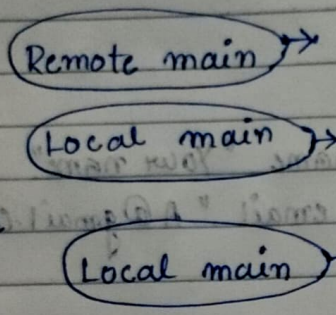
origin/main → A-B-C-D

{* commands}

```
git fetch origin.
```

git pull

git pull = git fetch + git merge.



13. Clone

{*commands}

git clone (Link)

This → copies code
copies full history
Sets remote automatically.

14. Pull request

- A pull request, requests merging one branch into another.
- It Enables →
  - code review
  - Discussion
  - Quality control

Created on GitHub not locally.

15. .gitignore

used to exclude files from tracking.

Git commands as phases.

Phase : 0

Install Git
↓
git --version
↓
git config --global user.name "Your name"
git config --global user.email "A@gmail.com"

phase : 1

Create project folder
↓
Cd Project folder
↓
git init

Phase : 2

Edit / create files.
↓
git status
↓
git add .
↓
git commit -m "meaningful message"