What is Git?
Git is a **version control system** — like a time machine for your code.

- It allows us to save checkpoints (commits).
- We can go back to previous versions.
- We can track changes made to the code.
- Multiple developers can work on the same project without affecting each other because each environment is isolated.

Git Configuration
Whenever I make a commit, I want my name and email to appear.

```
git config --global user.email "raghuvalluru18@gmail.com"
git config --global user.name "raghu"
```

Initialize a Repository

```
git init
```

Used to initialize a directory as a Git repository.
After this, a hidden .git folder is created.
.git acts like a database — it stores:

- All commits
- Snapshots
- History
- Branch information

Working Directory
The place where we write and modify code.
Git is not aware of new files here until we tell it to track them.

Staging Area
We bundle files from the working directory and prepare them for commit.

- Adds all files in the current folder to staging.
- . means everything in this directory.

```
git add .
```

Repository
After committing, files are permanently saved as a snapshot.

```
git commit -m "initial commit"
```

File States
A file can be:

- **Untracked** – New file, not tracked by Git (U)
- **Modified** – Existing file changed
- **Staged** – Added using git add
- **Committed** – Saved permanently in repository

HEAD

HEAD
HEAD means the latest commit.
Whenever we create a new branch, it is created from the current HEAD.

Git Logs
To see commit history:

```
git log
```

```
git log --oneline
```

Shows commit metadata and information.

Git Ignore
We don't usually add every file to the repository.
.gitignore is used to exclude files or folders.

.gitkeep
Used to keep an empty folder inside Git (because Git doesn't track empty folders).

Branching
To see branches:

```
git branch
```

Creating and Switching Branches
Switch to main:

```
git switch main
```

Create a new branch:

```
git switch -c feature_1
```

Merge
If another developer made changes and we want to include them into main, we merge.

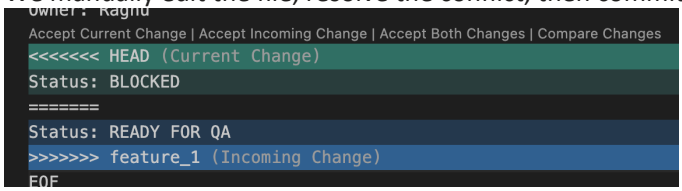
```
git merge feature_1 -m "merging 1"
```

A merge commit is created when combining histories.

Merge Conflict
A merge conflict happens when:

- Two branches modify the same file or same lines.
- Git cannot automatically decide which version to keep.

We manually edit the file, resolve the conflict, then commit.



Bugfix Branch
We create a bugfix branch when:

- There is a production bug.
- We want to isolate the fix.
- We don't want unfinished feature code in main.

```
git switch -c bugfix
```

Rebase
Instead of merge, we can use rebase.

```
git rebase main
```

Rebase:

Rebase:

- Moves or reapplies commits onto a new base commit.
- Creates a cleaner, linear history.
- Alternative to git merge.

Rebase is sometimes called fast-forward when no extra merge commit is created.

Used to **cancel a merge** when conflicts happen.

```
git merge --abort
```