### **What is Git?**

Git is a version control system that tracks changes in your code. It helps teams collaborate, manage versions, and avoid overwriting each other's work.

### **Setup and Configuration**

You need to identify yourself to Git so that it knows who made the changes.

git config --global user.name "Your Name" # Set your name

git config --global user.email "your.email@example.com" #Set your email

**Check Configuration:** git config --list # See all current configurations

### **Clone a Repository**

To copy a project repository from a remote server to your computer.

git clone <repository\_url>

### **Understanding Stages in Git**

1. **Working Directory**: Your local files and changes.
2. **Staging Area**: Changes you’ve marked to be committed.
3. **Repository**: The saved history of changes

### **Tracking and Committing Changes**

**git status #** To see the state of your files (e.g., modified, new, or untracked).

**git add -** To tell Git which changes to save in the next commit.

git add <file\_name> # Add a specific file to the staging area

git add . # Add all changes in the current directory

**git commit -** To save a snapshot of the changes in the repository.

git commit -m "Clear and descriptive message about the changes"

### 

### **Working with Branches**

* + A branch is like a separate workspace where you can make changes without affecting the main code.
  + Example: Use a branch to add a new feature while others work on the main branch.

git branch # List all branches

git branch <branch\_name> # Create a new branch

git branch -d <branch\_name> # Delete a branch

git branch feature/login # Create a branch for login feature

git checkout <branch\_name> # Switch to a branch

git switch <branch\_name> #Alternative modern way to switch

git switch feature/login

**git merge**

To combine changes from one branch into another.

git merge <branch\_name>

Ex: Step 1. git checkout main

Step 2. git merge feature/login # Merge login feature into main

### 

### **Fetch Latest Changes**

git fetch

Downloads commits, files, and refs from a remote repository into your local repository. It does not merge them into your current working branch automatically.

### **Collaborating with Remote Repositories**

**git remote -** To manage connections to remote repositories.

git remote add origin <repository\_url>

# Links your local Git repository to a remote repository

git remote -v # View connected remotes

git remote add origin https://github.com/user/project.git

**git pull -** To fetch the latest changes from a remote repository and merge them.

git pull origin main

**git push -** To send your local commits to the remote repository.

git push origin <branch\_name>

git push origin feature/login

### **Viewing Changes**

**git log -** To view the history of commits.

git log

**git diff -** To see what has changed between versions or branches.

git diff # View unstaged changes

git diff <branch1> <branch2> # Compare branches

### **Undoing Changes**

**git reset -** To unstage or remove changes.

git reset <file\_name> # Unstage a file

You might use git reset --hard <commit\_hash> if:

* You want to completely discard changes made after a certain commit.
* You want to revert your repository to a specific stable state.

**Example**

Current State:

You have the following commit history: A -> B -> C -> D (HEAD)

And your working directory contains changes related to commit D.

Run Command:

git reset --hard B

Result:

The commit history is reset to: A -> B (HEAD)

* Commits C and D are no longer part of the branch (but may still exist in Git's reflog temporarily).
* The working directory and staging area now reflect the state of commit B.

**git revert -** To undo changes from a specific commit, without affecting the rest of the history.

git revert <commit\_hash>

**git checkout -- <file\_name>-** To discard changes in a file.

git checkout -- src/Main.java