What is Git?

Git is a distributed version control system that enables multiple developers to work together efficiently on a project. It tracks changes in source code, supports collaboration, and maintains a complete history of code changes.

Git Repository:

A Git repository is a storage space for your project where Git tracks changes. There are two types:

Local Repository: Stored on your local system.

Remote Repository: Hosted on platforms like GitHub, GitLab, or Bitbucket.

Commands:

```
git init
```

```
git clone <repo-url>
```

What is `git remote`?

Git remote links your local Git repository to a remote one, allowing you to push and pull changes.

Commands:

- a remote repository
  `git remote add origin <url>`
- View configured remote repositories
  `git remote -v`

Git push:-

git push sends your committed changes to a remote repository.

Command:

```
git push origin main
```

Git pull:-

git pull fetches and integrates changes from a remote repository into your local branch.

Command:

```
git pull origin main
```

What is Git Merging?

Merging incorporates changes from one branch into another. This is commonly used when finishing a feature or bug fix and integrating it into the main branch.

Commands:

- `Switch to main branch`
  `git checkout main`
- `Merge 'feature' branch into 'main'`
  `git merge feature`

Common Git Operations:-

Operation	Command Example	Description
Initialize Repo	`git init`	Create a new Git repository
Check Status	`git status`	Show current state of the working directory
Add Files	`git add .`	Add all files to staging area
Commit Changes	`git commit -m "message"`	Save changes to local repository
View Commit History	`git log`	View the history of commits
Create Branch	`git branch feature`	Create a new branch
Switch Branch	`git checkout feature`	Switch to a different branch
Merge Branch	`git merge feature`	Merge a branch into current branch
Push Changes	`git push origin branch`	Push changes to remote repository
Pull Changes	`git pull origin branch`	Pull and merge changes from remote branch
Add Remote	`git remote add origin <url>`	Connect to a remote repository

Git Merging Strategies:-

Git offers multiple merge strategies depending on the project and workflow:

(1)Fast-Forward Merge

Applied when there is a linear path from current branch to the target branch.

Keeps history simple.

```
git merge --ff feature
```

(2)Three-Way Merge

- Occurs when branches diverged; Git creates a new merge commit.
- Preserves history of both branches.

```
git merge feature
```

(3)Squash Merge

- Combines all changes into a single commit.
- Useful for clean history.

```
git merge --squash feature
```

```
git commit -m "Squashed commit"
```

(4)Rebase (Alternative to Merge)

Rewrites history to make commits linear.

```
git rebase main
```

Git Branching Strategies:-

Branching strategies define how teams structure branches in a repository.

(1)Git Flow

- Feature, develop, release, and hotfix branches.
- Suitable for large projects.

(2)GitHub Flow

- Simple workflow: feature branches and pull requests to `main`.
- Ideal for continuous deployment.

(3)GitLab Flow

- Combines feature-driven development with environment-based deployment.
- Includes `main`, `pre-production`, and `production` branches.

(4)Trunk-Based Development

- Developers work in small, short-lived branches.
- Merge to `main` frequently.

Advantages of Git:

1. Distributed Version Control

Every developer has a full copy of the repository, including history.

Enables work offline and allows easy collaboration.

2. Fast Performance

Most operations are performed locally (e.g., commits, diffs, branches).

Much faster than centralized version control systems like SVN.

3. Branching and Merging

Lightweight and fast branch creation.

Easy to experiment, isolate features, and merge them when ready.

4. Data Integrity

Uses SHA-1 hashing to protect code and history against corruption and changes.

Every change is securely tracked.

5. Open Source and Free

Git is free to use and has a large community.

Constantly updated with new features and bug fixes.

6. Supports Non-Linear Development

Handles multiple parallel development streams efficiently.

Ideal for teams working on various features or fixes at the same time.

7. Collaboration Friendly

Tools like GitHub, GitLab, and Bitbucket make collaboration easy.

Pull requests, code reviews, and issue tracking are integrated.

8. Staging Area

Allows selective commit of changes.

Helps in organizing commits logically.

9. Efficient Handling of Large Projects

Scales well with large codebases and many developers.

10. Widespread Industry Use

Industry standard for version control.

High demand skill in software development roles