GIT (version control), Git commands

1. Introduction

Git is a distributed version control system used to track changes in code, collaborate with teams, and manage project history.

2. Setup & configuration

a. Sets your identity for commits.
   ➔ git config --global user.name "Your Name"
   ➔ git config --global user.email "you@example.com"

b. View your current configuration.

   ➔ git config –list

3. Starting a Repository

a. Initializes a new Git repository in the current folder.
   ➔ git init .
b. Creates a copy of an existing remote repository.
   ➔ git clone <repository-url>

4. Tracking Changes

a. Shows modified, staged, and untracked files
   ➔ git status
b. Stages changes (single file / all files) for the next commit.
   ➔ git add <file>
   ➔ git add .
c. Saves staged changes with a message.
   A "commit" in Git represents a snapshot of the project's tracked files at a specific point in time.
   ➔ git commit -m "Your commit message"

5. Branching

In Git, branching refers to the ability to diverge from the main line of development and work on new features, bug fixes, or experimental changes in an isolated environment without affecting the main codebase.

Commands used in git branching

a. Lists all branches in the repo or remote repo.
   - ➔ git branch
   - ➔ git branch -a

b. create a new branch
   - ➔ git branch <branch-name>

c. Switches to another branch.
   - ➔ git checkout <branch-name>
   - ➔ git switch <branch-name>

d. Creates and switches to a new branch.
   - ➔ git checkout -b <branch-name>

e. Merges changes from another branch into the current branch.
   - ➔ git merge <branch-name>

6. Remote Repositories

a. Shows connected remote repositories
   - ➔ git remote -v

b. Push commits from local branch to remote repositories's branch
   - ➔ git push <remote-origin-name> <branch-name>
   - ➔ eg. git push origin main

c. Downloads commits, branches, and files from remote (without merging).
   - ➔ git fetch

d. Fetches and merges changes from remote to local branch.
   - ➔ git pull <remote-origin-name> <branch-name>
   - ➔ eg. git pull origin master

7. Undoing changes

a. Unstages a file (keeps changes).
   ➔ git reset <file-name>
b. Discards local changes (restores file to last commit).
   ➔ git checkout -- <file>
c. Creates a new commit that undoes changes from a specific commit.
   ➔ git revert <commit-id>

8. Viewing History

a. Shows commit history
   ➔ git log
b. Compact commit history with branch graph.
   ➔ git log --oneline --graph –all
c. Shows differences between modified files and last commit.
   ➔ git diff

9. Summary

- Start project → git init / git clone
- Save work → git add → git commit
- Branching → git branch, git checkout -b
- Remote → git push, git pull, git fetch
- Undo → git reset, git revert
- History → git log, git diff

10. Best practices writing commit message

Commit messages are the story of your project — they help your future self (and teammates) understand what changed and why. Writing them well is a superpower.

- feat: → new feature
- fix: → bug fix
- docs: → documentation changes
- style: → formatting (no logic change)
- refactor: → code restructure without feature change
- test: → adding/updating tests
- chore: → build tasks, tooling, deps

Examples:

- fix(ui): correct navbar alignment on mobile
- docs(readme): update installation steps for Windows
- style(ui): format button component with Prettier
- refactor(api): extract user validation into separate function
- test(user-service): add unit tests for password hashing
- chore(deps): update axios to v1.7.0
- build(webpack): enable code splitting for better caching
- fix(auth): prevent login crash on empty password
- chore(cicd): update GitHub Actions workflow
- feat(auth): add Google login support
- feat(dashboard): add export to CSV option