Git & GitHub: Complete Guide

Git:

- Git is a **distributed version control system (VCS)** that helps you track changes in your code over time.

- Record every modification made to your code
- Revert to previous versions if something breaks
- Work on multiple features simultaneously using **branches.**
- It runs locally on your computer.

GitHub:

- GitHub is a cloud-based platform built on top of Git. It allows developers to store code online, collaborate with others, and manage projects more effectively.
- Pull requests for reviewing code.
- Issues for tracking tasks and bugs.
- CI/CD integrations for automation.
- Public or private repositories for sharing or securing code.

Difference Between Git & GitHub:

Feature	Git	GitHub
Type	Version control system	Cloud-based hosting platform
Function	Tracks changes locally	Stores code online and enables collaboration
Usage	Works offline	Requires internet for cloud features
Collaboration	Limited to local team	Supports global collaboration, pull requests, and forks

Why Version Control is Important:

- Maintain a history of changes.
- Collaborate without overwriting each other's work.
- Quickly revert to previous versions.
- Manage multiple features via branches.

Git & GitHub Work Together:

- Local Changes with Git: You make changes to your project locally using Git commands (git add, git commit).
- Push to GitHub: Upload your changes to a remote repository on GitHub (git push).
- Collaboration: Team members can clone, pull, and contribute to the same repository.
- Sync & Merge: GitHub facilitates merging changes, resolving conflicts, and reviewing code via pull requests.

Git Basics (Beginner):

Installing Git:

Git needs to be installed on your machine to track code versions.

Windows: Download from git-scm.com and follow the installer.

Linux:

- **sudo apt install git # Debian/Ubuntu**
- **sudo yum install git # CentOS/RHEL**

Mac:

brew install git

Git Configuration:

Before starting, set your username and email for commits.

- git config --global user.name "Your Name"
- git config --global user.email "your.email@example.com"

--global applies settings for all repositories.

git config –list

Shows all your Git configuration settings (username, email, editor, colors, credentials, etc.) in one place.

Creating a Local Repository:

To track a project with Git, initialize a local repository.

git init

- Creates a .git folder to store version history.
- After this, Git starts tracking changes in this project.

Cloning a Repository (git clone):

To copy an existing repository from GitHub or another remote:

git clone https://github.com/user/repo.git

- Creates a local copy of the project with full history.
- Useful for collaborating on existing projects.

Git Workflow:

- **Working Directory:** Where you edit files.
- **Staging Area:** Where changes are prepared for commit (git add).
- **Repository:** Permanent history stored in commits (git commit).

  Edit files → git add → git commit → Repository

Checking Status:

git status

- Shows files ready to commit, new files, and changes not staged.

Adding Files:

```
git add file.txt      # Stage a specific file
git add .             # Stage all changes in the current directory
```

- Only staged files are included in the commit.

Committing Changes:

```
git commit -m "Add feature X"
```

- Each commit is a snapshot of the project at a point in time.
- Use meaningful messages to describe changes.

Viewing Commit History:

```
git log
```

- Shows commit ID, author, date, and message.

Branching Basics:

Branches let you work on features or fixes without affecting the main code.

- **List branches:**

  ```
  git branch
  ```

- **Create a new branch:**

  ```
  git branch feature-branch
  ```

- **Switch branches:**

  ```
  git checkout feature-branch
  ```

- **Merge branches:**

  git merge feature-branch

Deleting Branches:

- Remove branches that are no longer needed.

  git branch -d feature-branch # Deletes merged branch safely
  git branch -D feature-branch # Force delete even if not merged

Git Intermediate Concepts:

Staging & Unstaging Changes:

- **Stage changes:**

  git add file.txt # Adds file to staging area

- **Unstage changes:**

  git reset file.txt # Removes file from staging area

Undoing Changes:

- **Discard local changes (unstaged):**

  git checkout -- file.txt # Reverts file to last committed version

- **Revert a commit:**

  git revert <commit-id> # Creates a new commit that undoes changes

Viewing Differences:

- **Show unstaged changes:**

  git diff # Compare working directory with staging area

- **Show staged changes:**

  git diff --staged # Compare staging area with last commit

Stashing Changes:

- **Save changes temporarily:**

  git stash # Stores changes in a stash

- **Apply stashed changes back:**

  git stash pop # Restores the latest stash

Tagging Commits:

- **Create a tag for a commit:**

  git tag v1.0 # Tags current commit as v1.0

- **List all tags:**

  git tag # Shows all tags

Working with Remote Repositories:

- **Add remote repository:**

  git remote add origin <url> # Link local repo to remote

- **List remotes:**

  git remote -v # Shows configured remote URLs

Pulling & Pushing Changes:

- **Fetch & merge changes from remote:**

git pull origin main # Updates local branch with remote changes

- **Push local commits to remote:**

git push origin main # Uploads commits to remote repository

Git Advanced Concepts:

- **Rebasing**

  git rebase branch-name # Reapply commits on top of another branch to create a linear history

- **Cherry-picking Commits**

  git cherry-pick <commit-id> # Apply a specific commit from another branch to the current branch

- **Reset vs Revert**
- **Hard Reset:**

  git reset --hard <commit-id> # Move branch to a specific commit and discard all changes in working directory and staging

- **Soft Reset:**

  git reset --soft <commit-id> # Move branch to a specific commit but keep changes staged for next commit.

- **Revert:**

  git revert <commit-id> # Undo a commit by creating a new commit without altering history

- **Working with Submodules**

  git submodule add <repo-url> path # Add an external repository as a submodule in your project

  git submodule update --init # Initialize and update submodules

- **Resolving Merge Conflicts**

  Manual: edit conflicting files, then

  git add <file> # Mark conflicts as resolved

  git commit # Complete the merge

  Using tools: git mergetool # Open merge tool to resolve conflicts visually

- **Git Internals Overview**

  HEAD → points to the current commit

  Index → staging area for next commit

  Working Tree → your current files in the project directory

- **Advanced Logs & History**

git log --oneline --graph --all # Visualize branch history as a graph

git reflog # View all changes to HEAD, including resets and rebase

- **Git Aliases for Efficiency**

  git config --global alias.st status # Create shortcut 'git st' for 'git status'

  git config --global alias.co checkout # Create shortcut 'git co' for 'git checkout'

GitHub Basics:

- **Creating a Repository on GitHub**

  Go to GitHub → Click **New Repository** → Add name, description, choose public/private → Click **Create**

  Creates a remote repository to store your project online.

- **Cloning GitHub Repositories**

  ```
  git clone https://github.com/user/repo.git   # Copy a GitHub repository to your local machine
  ```

- **Connecting Local Repository**

  ```
  git remote add origin https://github.com/user/repo.git  # Link local repo to GitHub remote
  ```

- **Pushing Local Commits to GitHub**

  ```
  git push -u origin main   # Upload local commits to the GitHub repository
  ```

- **Pulling Updates from GitHub**

  ```
  git pull origin main      # Fetch and merge changes from GitHub into local branch
  ```

Collaboration & Workflow on GitHub:

- **Forking a Repository**

  Description: Create your own copy of a repo to work independently.

  Example: On GitHub, click **Fork** on https://github.com/opensource/project → It creates https://github.com/yourusername/project.

- **Creating Pull Requests**

  Description: Propose merging your branch changes into another branch (usually main).

  Example: After committing on feature-login branch:

- Push branch:

  ```
  git push origin feature-login
  ```

  On GitHub, click **Compare & Pull Request**, add description, then **Create Pull Request**.

- **Branching Strategies**

  Git Flow: Structured branching for releases and hotfixes.

  Example: develop → feature-login → merge → release → main

  Feature Branches: Isolate features.

  ```
  git checkout -b feature-dashboard
  ```

  Hotfixes: Quick fixes to production.

  ```
  git checkout main
  git checkout -b hotfix-typo
  ```

- **Reviewing Pull Requests**

  Description: Team reviews code before merging.

  Example: On a PR, click **Files Changed** → add comments → **Approve** or **Request Changes**.

- **Resolving Conflicts on GitHub**

  Description: Fix merge conflicts via web editor or locally.

  Example (locally):

  ```
  git pull origin main     # Resolve conflicts in files
  git add resolved_file.txt
  git commit -m "Resolve merge conflict"
  git push origin feature-login
  ```

- **Using Issues & Labels**

  Description: Track tasks, bugs, or enhancements.

  Example: Create an issue titled Login page bug and add labels: bug, high-priority.

- **Assigning Tasks & Mentions (@username)**

  Description: Assign tasks and notify team members.

  Example: In issue or PR comment:

  @alice Can you review this PR?

  Assign the issue to a teammate using **Assignees** in GitHub.

Advanced GitHub Features:

- **GitHub Actions (CI/CD Pipelines)**
- **Explanation:** Automates tasks like building, testing, and deploying code whenever changes are pushed or a PR is created.
- **Example:** Automatically run tests on every push.

yaml

```
name: CI Pipeline

on: [push]

jobs:

  test:

    runs-on: ubuntu-latest

    steps:

      - uses: actions/checkout@v2      # Check out code

      - run: npm install               # Install dependencies

      - run: npm test                  # Run tests
```

- **GitHub Pages (Static Site Hosting)**

  Explanation: Host static websites (HTML/CSS/JS) directly from a GitHub repo.

  Example:

  Push website files to main or gh-pages branch.

  Enable GitHub Pages in **Settings → Pages**.

Access the site at:

https://username.github.io/repo

- **Projects & Kanban Boards**

  Explanation: Organize tasks, track progress, and manage project workflows visually.

  Example:

- Create a project board Sprint 1
- Columns: To Do, In Progress, Done
- Add issues or PRs as cards → move them across columns as work progresses.

- **Security & Access Control**

  Branch Protection Rules: Prevent accidental changes to critical branches.

  Example: Protect main branch → Require PR approval → Prevent direct pushes.

  Two-Factor Authentication (2FA): Adds an extra layer of security for GitHub login.

- **Releases & Versioning**

  Explanation: Package stable versions of your software with version numbers and release notes.

  Example: Tag a release:

  ```
  git tag -a v1.0.0 -m "Initial release"
  git push origin v1.0.0
  ```

- **Managing Large Files (Git LFS)**

  Explanation: Git LFS stores large files outside normal Git history to keep repo size small.

  Example:

  ```
  git lfs track "*.psd"          # Track Photoshop files
  git add .gitattributes
  git commit -m "Track large PSD files using Git LFS"
  git push origin main
  ```

- **Webhooks & API Integration**

  Explanation: Notify external services when specific GitHub events occur (e.g., PR merge, push).

  Example:

  Configure a webhook to notify Slack when a PR is merged.

  Or trigger CI/CD pipelines in Jenkins or CircleCI automatically.

•

Practical Git & GitHub Tutorial (Beginner → Advanced)

Requirements

- Linux VM (Ubuntu/CentOS) or any Linux environment (e.g., VirtualBox, AWS EC2, WSL)
- Internet connection
- GitHub account: https://github.com