Git Training

Git is a **distributed version control system** designed to track changes in files (most commonly source code) during software development. It allows multiple developers to work on the same project simultaneously, each with their own local copy of the repository, and then merge their changes together.

- **Distributed**: Unlike centralized systems where a single server holds the repository, every developer using Git has a full copy (including the complete history) of the project on their machine.
- **Version Control**: Git records each set of changes as a *commit*, effectively creating snapshots of the project over time. This makes it easy to revert to earlier versions, review histories of modifications, and collaborate without overwriting each other's work.
- **Collaboration**: Git supports branching, merging, pull requests, and other features that facilitate teamwork on projects large or small.

Because of its efficiency, reliability, and robust branching and merging capabilities, Git has become the de facto standard for source code management.

At a low level, Git works with "objects" (Commit, Tree, Blob, and Tag). Each commit can have one or more parents, representing the history of changes. Understanding this internal structure makes troubleshooting easier and allows for more effective use of advanced Git features.

Branching and Merging

Branch: A separate line of development.

Merge: Combining one branch into another, potentially requiring conflict resolution.

Fast-Forward Merge: If no parallel commits have been made, Git simply advances the branch pointer (fast-forward).

Three-Way Merge: When parallel changes exist in different branches, Git uses three reference points (the two sets of changes and the common ancestor) to merge.

Stashing

Git Stash temporarily stores your current, uncommitted changes. This is extremely handy if you need to switch tasks or branches without losing work in progress.

Common commands:

- **git stash save "message":** Save current changes to stash with an optional message.
- **git stash list:** List all existing stashes.
- **git stash apply**: Apply the latest (or a specified) stash.
- **git stash pop:** Apply the latest stash and remove it from the list.

Git Flow and Other Branching Models

Git Flow: A structured workflow for managing branches. It typically involves **main** branches **master** and **develop**, along with temporary feature/, bugfix/, release/, and hotfix/ branches.

Rebase

Rebase is a method of rewriting history, allowing you to take commits from one branch and reapply them on top of another branch. The main benefit is producing a cleaner, linear history. However, caution is needed when rebasing public/shared branches because rewriting published history can cause issues in team collaboration. Always coordinate with team members before rebasing on a shared repository.

Cherry-Pick

Cherry-Pick allows you to copy a specific commit from one branch to another without merging the entire source branch history. This is useful when you want to transfer a few vital changes to your main branch but don't want to include all other changes in the source branch.

Example:

```
git cherry-pick <commit-hash>
```

Advanced Conflict Resolution

When multiple people edit different parts of the code in parallel, **conflicts** are likely to occur. To resolve them:

1. Open the conflicting files.
2. Examine both your changes and the other branch's changes and keep the correct combination.

3. Save the file and create a new commit.
4. In more complex scenarios, use specialized merge tools (e.g., KDiff3 or Meld) for three-way comparison.

Best Practices

1. **Descriptive Commit Messages**: Keep commit messages concise and informative.
2. **Use Separate Branches**: Create a new branch for each feature or bugfix to maintain clarity and traceability.
3. **Pull Recent Changes**: Always ensure your local repository is up to date before merging or rebasing.
4. **Avoid Rebasing Public History**: If commits have been pushed to a shared repository, avoid rebasing unless you have team consensus.

Git Quick Start Guide

Step 1: Install Git

Download Git from the official website: https://git-scm.com/downloads

Step 2: Initialize a Git Repository

```
git init                # Initialize Git in the folder
git add .               # Add all files to staging
git commit -m "Initial commit: Add greet function"  # Save changes locally
```

Step 3: Connect Local Repository to GitHub

```
git remote add origin https://github.com/yourusername/repository.git  # Set the GitHub remote
git branch -M main        # Rename branch to 'main' (if needed)
git push -u origin main    # Push initial commit to GitHub
```

Step 4: Clone an Existing GitHub Repository

git clone https://github.com/yourusername/repository.git # Clone repository
cd repository # Change into the repository folder

Step 5: Daily Workflow

1. Pull Latest Changes
git pull origin main # Fetch and merge changes from GitHub
If up-to-date, Git will say "Already up to date."
If there are conflicts, resolve them manually.

2. Make Local Changes

Modify your files or add new code.

3. Add and Commit Changes

git status # Check current state of your working directory

git add . # Stage all changes
Or
git add -p # Stage changes interactively (partial commits)

git commit -m "Your commit message" # Commit your changes

4. Push Changes to GitHub
git push origin main # Upload your changes to GitHub

Git Configuration
git config --global user.email "you@example.com"
git config --global user.name "Your Name"