Git Initialization

Step 1: Install Git

Download from website

Step 2: Check Git Version After Installation

- git --version

Step 3: Configure Git

- git config --global user.name "Your Name"
- git config --global user.email "Your Email"

Step 4: Check Configuration Settings

- git config --list

Step 5: Create New Folder or Redirect to Your Working Directory

- Create New Folder: mkdir <folder-name>
- Go To Folder: cd <folder-name>

mkdir: Makes a new directory.

cd: Changes the current working directory.

List the files in Current Working Directory

- ls

Step 6: Initialize Git on Your Working Folder

- git init

  It creates a hidden folder to keep track of changes.

List the hidden files in Current Working Directory

- ls -la

Command to display the Current Working Directory

- pwd

Git Staging Area

Step 1: Add Files to Staging Area

Command To Add Single File to Staging Area

- git add <file-name>

Command To Add All Files to Staging Area

- git add .

OR

- git add --all

Command To Remove file from staging area

- git reset <file-name>

OR

- git restore --staged <file-name>

Step 2: Commit Your Progress and Changes

- git commit -m "Your Message About Changes"

Adding commits keep track of our progress and changes as we work.

When we commit, we should always include a message.

Step 3: Check The Status of Repository

- git status

OR

- git status --short

git status --short flags are:

?? - Untracked files

A - Files added to stage

M - Modified files

D - Deleted files

Git History and Revert

1. Get The History of Commits for Repository
   - git log
   - git log --oneline
   - git show <commit-hash>

2. View The Changes Between Commits and Working Tree
   - git diff

3. Revert The Changes
   - git reset --hard <commit-hash>
     - It moves the "HEAD" pointer to the specified commit.
     - Use with caution because it delete changes and commit history permanently.

   - git revert <commit-hash>
     - Useful when you want to undo changes in a specific commit but keep the rest of the history
     - Unlike "git reset", it doesn't remove the commit from history.
     - After the revert "add file to staging area and commit".

   Hash: The unique identifier of a commit to which you want to revert.

4. Modify the most recent commit.
   - git commit --amend -m "New Message"
     - It updates the latest commit with a new commit message.

5. Change The Default Code Editor in Your System to VS Code
   - git config --global core.editor "code --wait"

Git Branch

1. Create A New Branch
   - git branch <branch-name> OR
   - git checkout -b <branch-name>
     - Create a new branch, and move to it, if it does not exist

2. Lists All the Branches in The Current Repository.
   - git branch OR
   - git branch --list

3. Switch Between Branches
   - git switch <branch-name>
     - It is a newer, more intuitive command introduced in Git 2.23.

     OR
   - git checkout <branch-name>
     - It is an older, traditional command introduced in Git 1.0.

4. Delete Branch
   - git branch -d <branch-name>

5. Rename Branch
   - Rename The Current Branch
     - git branch -m <new-branch-name>
   - Rename A Branch You're Not On
     - git branch -m <old-branch-name> <new-branch-name>

6. Merge Branches

   Step 1. Switch To the Target Branch (main)
   - git switch <target-branch>

   Step 2. Merge The Source Branch (feature-branch)
   - git merge <source-branch>

Example: Merging the ”feature” branch into the ”main” branch.

1. git switch main
2. git merge feature

Git Rebase

Git Rebase

・Git rebase is used to move or integrate changes from one branch onto another by rewriting the commit history.

・git rebase allows you to change the base of your branch

- git rebase <base-branch>

<base-branch>: The branch you want to rebase your current branch onto (e.g., main or master).

Git Rebase Steps:

Step 1: Switch to the Branch You Want to Rebase (e.g., feature)

- git switch feature

Step 2: Rebase onto the Target Branch (e.g., main)

- git rebase main

Step 3: Resolve Conflicts (if any)

・Git will pause the rebase if there are conflicts.

・Resolve conflicts in the conflicted files.

・Stage the conflicts resolved files

- git add <file-name>

Step 4: Continue the rebase

- git rebase --continue

・If you want to abort the rebase

- git rebase –abort

Git Stash

Git Stash

・Stash is a way to save your changes in a temporary location.

・It is useful when you want to make changes to a file but don't want to commit them yet.

- git stash

・View the stashed list

- git stash list

・Save Stash with a Message

- git stash push -m "work in progress on X feature"

・Apply the stash

. Apply the most recent (latest) stash.

- git stash apply

・Apply the specific stash

- git stash apply stash@{n}

. Applying and dropping the stash

. Applies the most recent stash and deletes it from the list.

- git stash pop

. Applies the specific stash and deletes it from the stash list.

- git stash drop stash@{n}

. Drop the Stash

. Drop (Delete) the recent Stash

- git stash drop

. Drop a specific stash from the stash stack.

- git stash drop stash@{n}

. Clear all stashes from the stash stack.

. git stash clear

. Stash Specific Files

- git stash push -m "message" <file name>

. Stashes only the specified file(s) with an optional message.

Connect Git with GitHub

Step 1: Install Git

- Download Git
- Install Git
- Verify Installation
  - git --version

Step 2: Set Up Git Configuration

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

If Git is already installed and configured on your machine, skip steps 1 and 2.

Step 3: Create a GitHub Account

- Go to GitHub
- Log in to GitHub

Step 4: Create a New Repository on GitHub

- Click on the "+" icon in the upper-right corner and select **New repository.**
- Name your repository (e.g. my-project).

- Choose whether it should be public or private.
- Optionally, initialize with a README, you can skip this if you already have one locally.
- Once you've configured the repository settings, click the **Create repository** button.

Step 5: Link Your Local Repository to GitHub

- Navigate to your local project
  - cd /path/to/your/project

- Initialize a Git repository (if not already done)
  - git init

- Connect local repository to the GitHub repository.
  - git remote add origin <remote-url>

  <remote-url>: URL of the remote repository that you want to add.

  origin: Name of the remote repository.

- Verify the remote connection
  - git remote -v

Step 6: Commit Your Changes Locally

- Add files
  - git add <file-name>
- Commit changes
  - git commit -m "commit message"

Step 7: Push Changes to GitHub

- git push -u origin main

Step 8: Verify the Push

- Go to your GitHub repository page.
- Refresh the page, and you'll see the code you pushed listed in the repository.

Step 9: Future Changes (Workflow)

- For any new changes you make:
  1. Make changes to your project.
  2. Stage and commit the changes.
     - git add <file name>
     - git commit -m "commit message"
  3. Push the changes in GitHub
     - git push

Step 10: Pull Updates from GitHub

- git pull origin main