Git

If you're completely new to Git and version control, this guide will walk you through everything from installation to making your first commit.

1. What is Git and Why Use It?

What is Git?

Git is a tool that helps you keep track of changes in your files. Think of it as an "undo" button for code.

Why Use Git?

- Allows you to save different versions of your work
- Helps in collaboration when working with a team
- Makes it easy to track who made changes and when
- Prevents loss of important files

2. Getting Started: Installing Git

Step 1: Download and Install Git

- **Windows:** Download Git and follow the installation steps.
- **Mac:** Install using Homebrew:
- `brew install git`
- **Linux (Ubuntu/Debian):**
- `sudo apt update`
- `sudo apt install git`

Step 2: Verify Installation

After installing, check if Git is installed correctly:

```
git --version
```

This should return a version number (e.g., `git version 2.34.1`).

3. Configuring Git (One-Time Setup)

Before using Git, set up your name and email. This information is recorded with each change you make.

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

To verify the settings:

```
git config --list
```

4. Creating Your First Git Repository

A **repository (repo)** is a folder where Git keeps track of changes.

Step 1: Initialize Git in a Folder

1. Open the terminal (or Git Bash on Windows).
2. Navigate to your project folder:
3. `cd path/to/your/folder`
4. Run the following command:
5. `git init`

This creates a hidden `.git/` folder, where Git stores all version history.

5. Making Your First Commit

A commit is like a "save point" in a video game—it records your changes so you can go back if needed.

Step 1: Check Status

Before saving changes, check the status:

```
git status
```

This will show which files are new or modified.

Step 2: Add Files to Staging Area

Git doesn’t track new files by default. You must **add** them first.

To add a specific file:

```
git add filename.txt
```

To add all files:

```
git add .
```

Step 3: Commit Changes

Now, save your changes using:

```
git commit -m "First commit - added new files"
```

6. Connecting Git to GitHub (Optional, for Online Backup & Collaboration)

Git works locally, but you can use **GitHub** to store your code online and collaborate with others.

Step 1: Create a GitHub Account

- Go to GitHub and sign up.

Step 2: Create a Repository on GitHub

1. Click the **"+"** button in the top-right corner.
2. Select **"New repository"**.
3. Name it (e.g., "my-first-repo") and click **Create repository**.

Step 3: Link Your Local Repository to GitHub

1. Copy the GitHub repository link (it looks like `https://github.com/your-username/repository.git`).
2. Run the following command in your terminal:
3. `git remote add origin https://github.com/your-username/repository.git`
4. Push your local commits to GitHub:
5. `git push -u origin main`

   If you see an error about "main" not existing, rename your branch first:

   ```
   git branch -M main
   git push -u origin main
   ```

7. Working with Branches (For Organized Development)

Branches allow you to work on new features without affecting the main project.

Create a New Branch

```
git branch new-feature
```

Switch to the New Branch

```
git checkout new-feature
```

Or, using a single command:

```
git checkout -b new-feature
```

Merging a Branch to Main

Once you've finished working on a feature, merge it back to the main branch:

```
git checkout main
git merge new-feature
```

8. Collaborating with Others on GitHub

When working in a team, you must pull the latest changes before pushing.

Pulling Updates from GitHub

```
git pull origin main
```

This ensures you have the latest code.

Pushing Your Changes

After committing your changes:

```
git push origin main
```

9. Undoing Mistakes in Git

Made a mistake? No problem!

Undo the Last Commit (Without Losing Changes)

```
git reset --soft HEAD~1
```

Remove the Last Commit Completely

```
git reset --hard HEAD~1
```

⚠ This deletes the last commit permanently.

10. Git Cheat Sheet (Quick Commands)

Command	Description
`git init`	Initialize a new Git repository
`git status`	Show the current state of the repository
`git add .`	Add all files to the staging area
`git commit -m "message"`	Save changes with a message
`git log`	View commit history
`git clone <repo>`	Copy a remote repository
`git branch new-feature`	Create a new branch
`git checkout new-feature`	Switch to a branch
`git merge new-feature`	Merge a branch into `main`

Command	Description
`git push origin main`	Upload changes to GitHub
`git pull origin main`	Download the latest changes

11. Best Practices for Using Git

✅ **Commit often** – Save changes in small, meaningful commits.
✅ **Write clear commit messages** – Explain what was changed and why.
✅ **Use branches** – Work on new features separately from the main code.
✅ **Pull before pushing** – Always update your local repository before uploading changes.
✅ **Use `.gitignore`** – Exclude unnecessary files (e.g., log files, temporary files).

---

12. Finally thought

1. Install Git and set up your username/email.
2. Initialize a Git repository (`git init`).
3. Add files to tracking (`git add .`).
4. Commit changes (`git commit -m "message"`).
5. Connect to GitHub (optional).
6. Push and pull changes (`git push` and `git pull`).
7. Work with branches for feature development.
8. Undo mistakes safely if needed.