Git and Github Course

This is my personal summary note for what I actually learn from the Git and Github version control course from Udemy.In this document i will mention all what i learn from the course including my personal experience of learning so let start.

Let's start by defining Git and Github today as a developer. Git and Github are the most powerful software for project versions, controls,updates and collaborations.
For the building of applications teams are needed and Git and Github permit to facilitate the collaboration between workers by providing control and previous versions of our programs and work.

Today Git and Github has become a game changer for developers and Data analyse.Since Git is a cross platform app it works on all our systems.

To install git and github on Mac and linux you can find the documentation on that on the internet.
Installing Git on macOS and Linux

Welcome! In this article, we'll guide you through:

- Installing Git on **macOS** and **Linux (Ubuntu/Debian)**
- Setting up your **GitHub account**
- Basic Git configuration after installation

Let's get started! 🚀

1. Installing Git on macOS

There are two common ways to install Git on macOS:

Option 1: Install Git Using Homebrew (Recommended)

If you have **Homebrew** installed (a package manager for macOS), you can install Git easily.

Open your Terminal and run:

1. brew install git

Once installed, verify by checking the version:

1. git --version

If you don't have Homebrew installed, you can install it by running:

1. /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

Note: Homebrew is a free and popular package manager for macOS.

Option 2: Install Git via Xcode Command Line Tools

Another option is to install Git through Apple's Xcode Command Line Tools.

In your Terminal, run:

1. xcode-select --install

A pop-up will appear asking you to install the developer tools — accept it.

After installation, verify Git:

1. git --version

2. Installing Git on Linux (Ubuntu/Debian)

Installing Git on Linux systems like Ubuntu is straightforward.

Step 1: Update Your Package Index

Open your Terminal and run:

1. sudo apt update

Step 2: Install Git

Then install Git by running:

1. sudo apt install git

Step 3: Verify the Installation

Check the Git version:

1. git --version

3. Setting Up a GitHub Account

GitHub is a platform for hosting and collaborating on Git repositories.

To create a GitHub account:

- Open your web browser and go to **github.com**.
- Click on **Sign up** in the top right corner.
- Follow the prompts to create your free account.

Note: GitHub is free for public and private repositories.

4. Configuring Git with Your GitHub Account

Once you have a GitHub account, set up Git locally with your name and email.

In your Terminal, run the following commands:

1. git config --global user.name "Your Full Name"
1. git config --global user.email "your_email@example.com"

To verify your Git configuration:

1. git config --global --list

You should see your username and email listed.

1. Creating a Github repository
In other to create a github repository we need to type a set of commands that is
1. git init: This command permit in install git in a given folder on file and so with a small .git file is been created inside our folder
2. git add : This command has the role to permit us to add our files into our repository the command git add is been followed by the name of the file we want the add for example if the file we want to add is output.txt we gonna type the command **‘git add output.txt’.**But in the case we want to add a full contain of a folder or all the files into a folder we gonna used the command **‘git add .’**Where the dot represent all files.
3. git commit : The git commit is the command been used to show when the project has been modified or change accounting to time generally for our version 1 of a project we used the statement **‘initial commit'.The** set of commands used for github to use the git commit are **‘git commit -m initial commit’.**

How let try to understand how git tracks changes in other projects
To understand how all this works we need to know the differcent between **staging Area,Working Area** when adding a new file into our work we are actually using the command **git add ‘’**which actually acts in the staging zone
Now how doesn’t git track files ?? alright git doesn(save files but git tracks the snapshot of the prevent event which permits it to make fast changes at any time without charging the process.
So git can be considered as a smart library which operates on its own for doing analysis and other stuff.

How Let learn How to master Branching with Git and Github in git we can create different branches for the same project so that a team can easily sub-divide the work into smaller pieces and at the end mesh all the breaches into one for the finale product.So can also consider that an individual corking on a project was to save Version 1 and keep it functional while developing version 2.
To create a new branch using git we use the command **git branch features-xyz** and this will duplicate the actual branch you are working with.Or we can use the more modern command **‘git switch features-xyz’** and work with it.

Once we have created a new branch we can do all modifications to our files in peace and this will not affect our project.To mesh the branches we used the command **git mesh feature-xyz** and this will mesh our main branch with our master branch.

Note:If you face a merge conflict do not panic git will ask you to solve the problem manually.

Now let's move on to the concept of connecting our projects to our github account in order to create a new repository directly from our accounts we need to to click on the new icon and then give a name to our new repo.
After that we will clearly find the commands to provide into the page .

There command ***'git main -M (branch)'*** is a command used to specify the branch we want to use for the storing of our project but by default this is set up to the main branch in the case we have another branch name as master we gonna have the following for branching***'git main -M master'ℏ.***

In order to upload the project we are going to use the command ***'git pull -u origin main'*** . The main command is valid for the main branch. For other branches we have a different branch name.

Now let learn how to folk a repository on github which permit to copy someone works or repo from github and then with that we can actually work from the repository and do what you want.They folking can be found on the github repo icons and this permit dev to always go forward in their task and with this we can now clone the github repo on our pc for working with.But when folking a repo we usually folk repositories on new branches when working so that we should not mix our work.

After we created the forked on github let go back to git and use the following set of commands

'git clone <repository-link>'
then move on to the clone repo with ***cd clone-repo-name*** and then just modified the readme file features by updating it and also used other features for handling the repo for us now usage.

Now let attack handling mesh errors in github when collaborating with our team members. We may get problems during our branch meshes and this can be very common so we need to handle that and correct this processing through the git terminal.

To solve the problem of merge conflicts we need to know first of all when to mesh conflict occurs to do that we need to type the command ***'git checkout -b feature-branch'***

In order to work like a pro when using github we need to set up an issue for automatic problem solving when working with respository.After that decide what type of error you would like to correct.
If we are fixing our readme file we gonna used the command '***git checkbox -b fix-readme'***

Common Git Errors & Fixes

Git is powerful, but like any tool, you might run into some confusing errors—especially when working in teams or switching branches. Here are the most common issues and how to handle them like a pro:

1️⃣ Merge Conflict

Error:

```
1. CONFLICT (content): Merge conflict in file.txt
2. Automatic merge failed; fix conflicts and then commit the result.
```

What it means:
Two branches changed the same part of a file.

Fix it:

.

Open the file, look for <<<<<<<, =======, and >>>>>>>

.

.

Decide what to keep

.

.

Delete conflict markers

.

.

Then run:

.

```
1. git add file.txt
2. git commit -m "Resolve merge conflict"
```

2️⃣ Detached HEAD State

Error:
You're not on a branch, you’re on a commit directly.

Fix it:
If you want to get back:

```
1. git checkout main
```

If you want to save what you're working on:

```
1. git switch -c new-branch-name
```

3️⃣ Push Rejected (Non-Fast-Forward)

Error:

1. Updates were rejected because the remote contains work that you do not have locally.

Fix it:

.

Pull before pushing:

.

1. git pull origin main

.

If still stuck, try:

.

1. git pull --rebase origin main

4️⃣ Accidentally Committed to the Wrong Branch

Fix it:

1. git switch correct-branch
2. git cherry-pick <commit-id>
3. git switch wrong-branch
4. git reset --hard HEAD~1

5️⃣ Forgot to Add a File Before Commit

Fix it:

1. git add missing-file.txt
2. git commit --amend

This adds the file to the last commit.

Pro Tip:

Use `git status` frequently—it tells you exactly what's going on. And don't panic—Git has ways to recover almost everything!

Commit Message Writing Tips

Writing clear, helpful commit messages is one of the easiest ways to improve collaboration and keep your project history readable. Here's how to write messages that your future self (and your teammates) will thank you for!

Why Good Messages Matter?

- They explain *why* you made changes (not just *what*)
- They help with debugging, code reviews, and tracking features
- They make open-source contributions easier to understand

Structure of a Great Commit Message

Basic Format:

1. Short, meaningful summary (max 50 characters)
2.
3. Optional detailed explanation in a second paragraph.
4. Mention related issues or PRs if needed.

Examples of Good Commit Messages

✅ `Fix typo in README`
✅ `Add login validation to sign-up form`
✅ `Refactor user profile component for clarity`
✅ `docs: update instructions for SSH key setup`

Avoid These

❌ `Update`
❌ `Fix stuff`
❌ `More changes`
❌ `Final commit for real this time`

> These messages don't say what or why — they aren't helpful later!

Pro Tips

- Use **present tense**:
  ✅ `Add feature`, ❌ `Added feature`
- Keep the first line short & focused
- Use **tags** for consistency:
  - `feat:` for new features
  - `fix:` for bug fixes
  - `docs:` for documentation
  - `refactor:` for code improvements
  - `style:` for formatting changes
  - `test:` for adding tests

Final Thoughts

A good commit message is like a journal entry for your codebase — make it count! When in doubt, ask: "Will someone understand this message in 6 months?"

Quick Tip: How to Use the Git Commit Editor (Vim)

Quick Tip: How to Use the Git Commit Editor (Vim)

In the mini project, when you run `git commit` without the `-m` option, Git may open a text editor (often **Vim**) to write your commit message.

If you're new to Vim, here's a quick guide to avoid getting stuck:

How to Write a Commit Message and Save in Vim:

1. **When the editor opens**, press `i` to enter **Insert mode**
   You'll see `-- INSERT --` at the bottom.
2. Now **type your commit message**, like:
   - Add header and footer sections to index.html
3. When you're done:
   - Press `Esc` to exit Insert mode
   - Then type `:wq` and press `Enter`
     This means "write and quit"

✅ Done! Your commit will now be saved.

Pro Tip:

You can skip the editor by writing your commit message directly in the terminal:

1. git commit -m "Your message here"

Let me know if you'd like a **screenshot**, **GIF**, or **short video demo** to go along with this tip—it can really help first-timers feel more confident!

Now let discuss about github action and task workflow automation
Github actions are mainly defined as automated scripts that run on github servers automation can be made for testing code deploying websites and more and this can be considered as having a bot that is going to automate tasks for you.
For the creation of github actions we gonna create a folder and name it **'github/workflows'** and inside that workflows folder create a file and name it as main.yml which gonna be the **main.yml** file where we gonna write all our automations. An inside out main.yml file we can write our automated scripts.