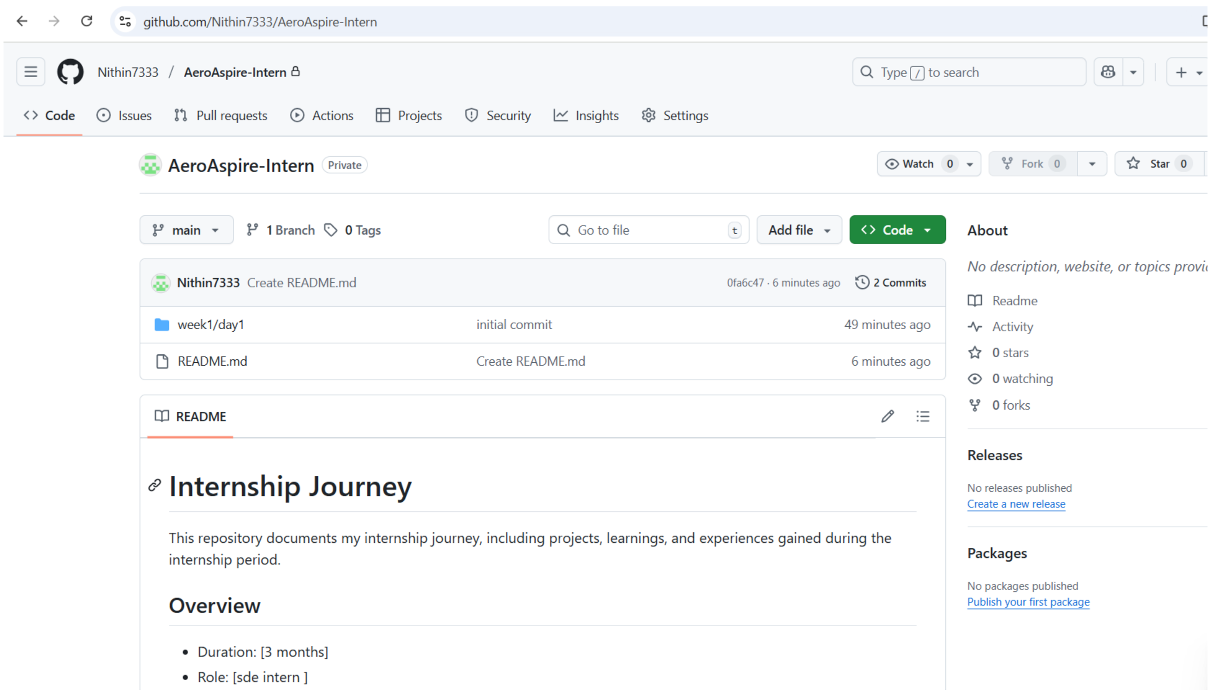
AeroAspire Intern

Nithin K Y

Day3:- Git basics & branches



Nithin7333 Create README.md		0fa6c47 · 6 minutes ago 2 Commits
week1/day1	initial commit	49 minutes ago
README.md	Create README.md	6 minutes ago

Internship Journey

This repository documents my internship journey, including projects, learnings, and experiences gained during the internship period.

Overview

- Duration: [3 months]
- Role: [sde intern]

1. First, create a new folder for your project or go to the one you want to use.
2. Open your terminal (command prompt) and type git init to turn that folder into a Git repository.
3. Add the files you want to save into that folder (your code, documents, etc.).
4. Tell Git to pay attention to all those files by typing git add . (this stages all the files).
5. Now, create a snapshot of those files with a message describing what you did by typing:
   git commit -m "Initial commit"

6. If you want to save this project to a remote server like GitHub, link your local repo to the remote using:
   git remote add origin [your-repo-URL]
7. Finally, send your snapshot to the remote server with:
   git push -u origin main.

For creating branch:

1. Open your terminal and go to your project folder.
2. To create a new branch and jump onto it right away, type:
   git checkout -b your-branch-name
   This makes a new workspace where you can try new things without messing up your main work.
3. If you just want to create the branch but stay where you are, use:
   git branch your-branch-name
4. Later, if you want to switch to that branch, type:
   git checkout your-branch-name
5. Once your new branch is ready and you want to share it online, push it with:
   git push -u origin your-branch-name

Question:-

What is the workflow from making changes → staging → commit → push?

The workflow in Git is simple: you first make changes to your files in the project folder on your computer. Once you're happy with those changes, you "stage" them which means telling Git exactly which files you want to prepare for saving. After staging, you "commit" your changes, which means taking a snapshot of those staged files with a message explaining what you changed. Finally, if you want to share your latest work with others or back it up online, you "push" the committed changes to a remote repository like GitHub. This process helps keep your project organized, tracked, and easily shared.

What is a merge conflict: what causes it, and how do you resolve?

A merge conflict in Git happens when two branches have changes to the same part of a file and Git doesn't know which version to keep during a merge. This usually occurs when:

- Two people edit the same line in a file differently,
- One branch deletes a file while the other branch modifies it,
- Or when a file is renamed in one branch but changed in another.

Git cannot automatically decide how to combine these conflicting changes, so it stops the merge and marks the files with conflict markers.

To resolve a merge conflict, you:

1. Open the file with conflicts, look for markers showing conflicting sections.
2. Manually choose which changes to keep (yours, theirs, or a merged version).
3. Remove the conflict markers and save the file.
4. Stage the resolved file with git add <file>.
5. Commit the resolution with git commit to complete the merge.

Describe what happens under the hood with git commit: what objects are stored? (briefly)

When you make a commit in Git, it creates several objects behind the scenes. First, Git stores the content of each file as a "blob" object. Then, it creates a "tree" object, which organizes these blobs by recording file names and the directory structure. Finally, Git creates a "commit" object that points to this tree and includes important information like who made the change, when, a message describing the change, and a link to the previous commit(s). Think of it as saving a full snapshot of your project with details about the change and a connection to its history, so you can track everything efficiently over time..