Hands on Journey from Git to GitHub

P.S. This document is part of the **FREE DevOps for Beginners Cohort** run by Pravin Mishra. You can start your DevOps journey for free from his YouTube Playlist.

Overview

In this exercise, we will be completing multiple assignments to accomplish the following setup.

- CodeTrack— Initial Git Setup (Local Only)
- Tracking and Staging Changes in a CodeTrack Project in Git
- Branching Workflow — Add & Verify a Contact Page in Git
- Setting Up GitHub for CodeTrack
- Collaborating on Mini-Finance with GitHub

Prerequisites

- Git installed (Git Bash on Windows, Terminal on macOS/Linux).
    - Install the latest Git from the url - https://git-scm.com/downloads
- Basic terminal navigation.
- Account in GitHub(Preferable)

Instructions

Assignment 5- CodeTrack— Initial Git Setup (Local Only)

Task 1 — Create a Local Project Directory

We will simulate the creation of a new **CodeTrack** project for new development.

- Navigate to your preferred working location (Documents/Desktop/projects).

Windows (Command Prompt or PowerShell or Git Bash):

1. mkdir CodeTrack
2. cd CodeTrack

macOS/Linux (Terminal):

1. mkdir CodeTrack && cd CodeTrack

Initialize Git:

1. git init

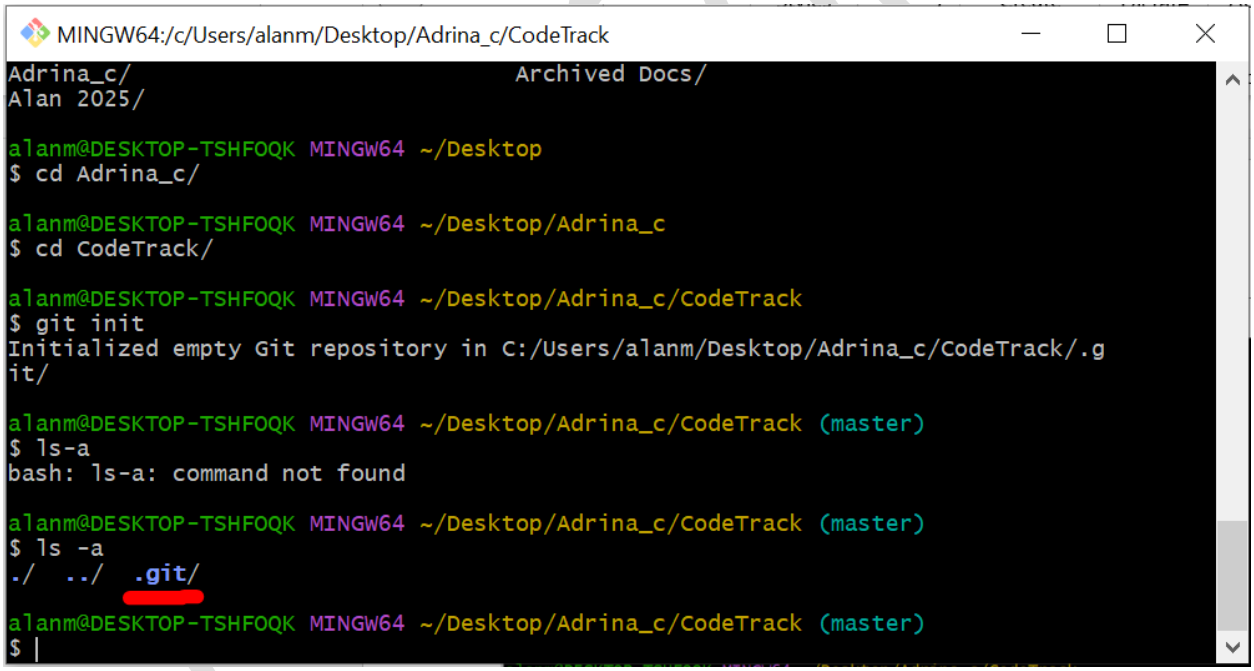
Expected:
Initialized empty Git repository in .../CodeTrack/.git/

(Optional check)

1. ls -a # or: dir /a (Windows)

You should see a hidden .git folder.

Screenshot A: Initialized empty Git repository(git init) || the .git folder listed with ls -a



Task 2 — Configure Git Locally for CodeTrack

Configure identity **only for this repo** (recommended for team/enterprise scenarios when identities differ by project).

1. git config --local user.name "Your Name"
2. git config --local user.email "your.email@example.com"

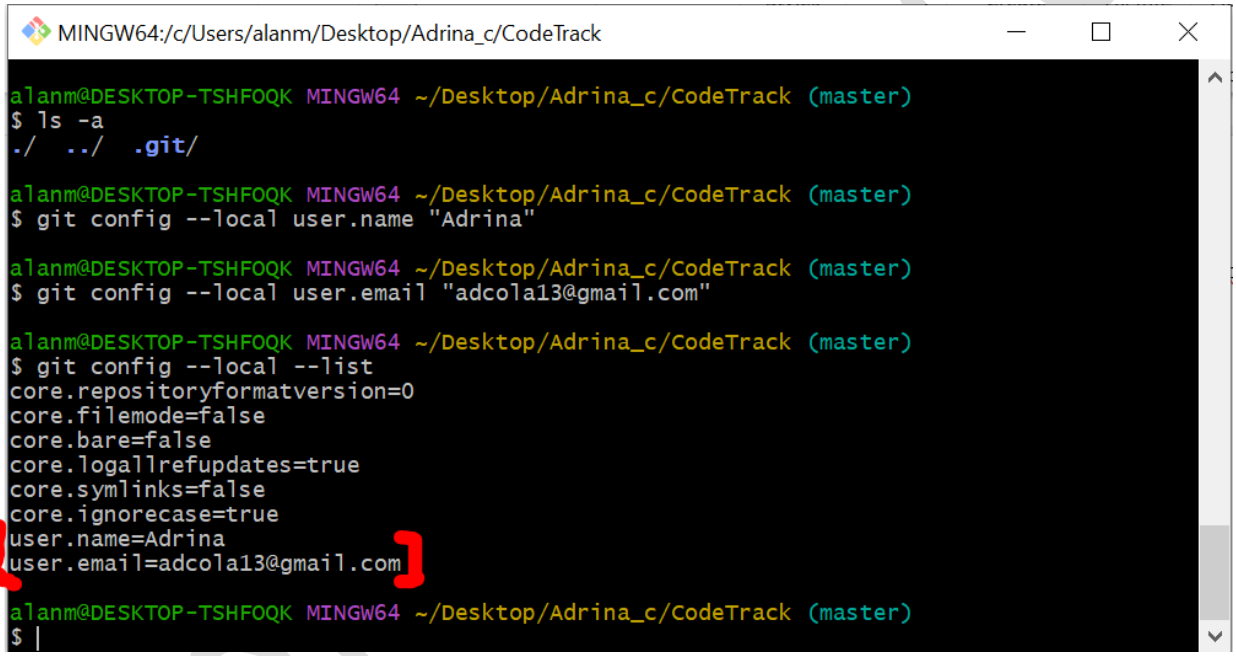
3. git config --local --list

Expected: output includes:

1. user.name=Your Name
2. user.email=your.email@example.com

Tip: If you'll push to GitHub and want to keep your email private, use GitHub's noreply email (e.g., 12345678+username@users.noreply.github.com).

Screenshot B: Set up your local Git identity(git config --local –list)



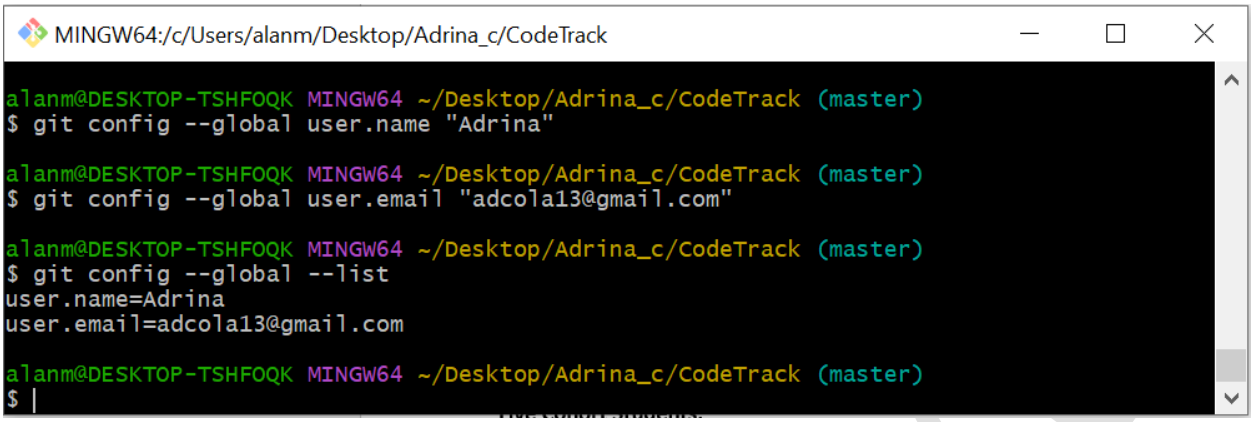
Task 3 — Configure Git Globally (Optional, Recommended)

Sets your default identity for **all repos** on this machine.

1. git config --global user.name "Your Name"
2. git config --global user.email "your.email@example.com"
3. git config --global --list

Expected: output lists your global user.name and user.email.

Screenshot C: Set up your global Git identity(git config --global –list)



When would you prefer to use **local configuration** instead of **global configuration**?

Global configuration is used to standardize your identity across all personal projects.

Local configuration is used to override your identity per project (e.g., work vs personal).

Assignment 6- Tracking and Staging Changes in a CodeTrack Project in Git

Task 0 – Navigate to Project Folder

Since you have already set up the Project Folder in the last assignment, navigate to it:

- **On Windows (Command Prompt):**
    1. cd path\to\CodeTrack
- **On macOS/Linux (Terminal):**
    1. cd ~/path/to/CodeTrack

If you're unsure of your current directory, use the below command to check:

1. pwd

Task 1: Create and Modify Files

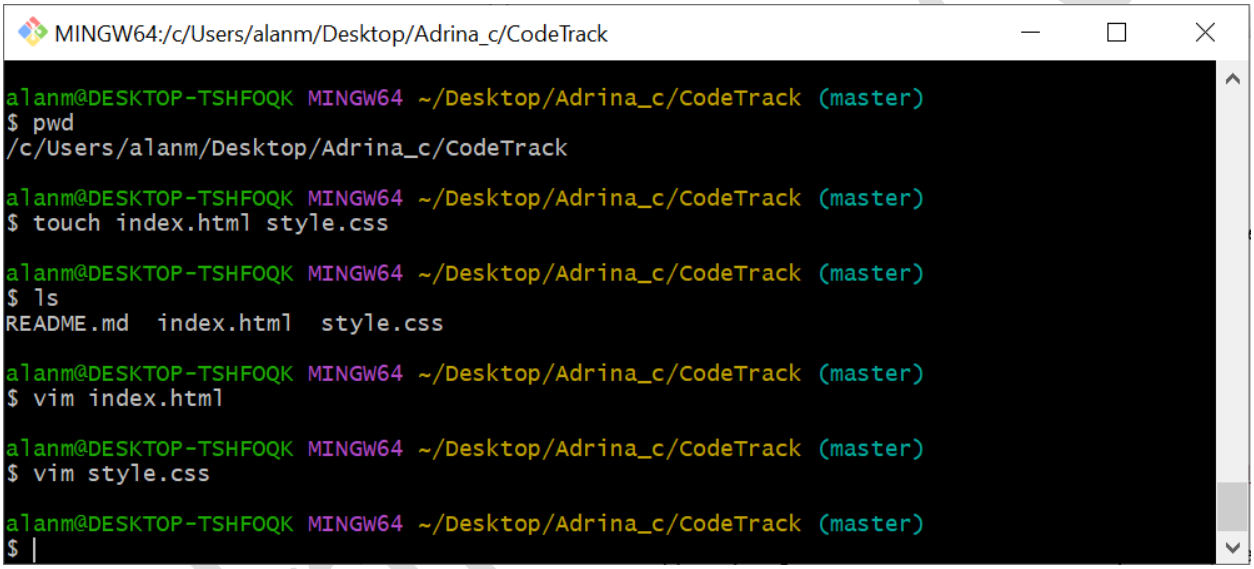
1. **Create two new files inside the CodeTrack directory:**
    1. touch index.html style.css

(For Windows users who don't have touch, use echo > index.html and echo > style.css.)

2. **Verify that the files have been created by listing the directory contents:**
    1. ls

Expected Output: You should see **index.html** and **style.css** in the file list.

3. **Modify index.html and style.css using a text editor.**
    - Go to GitHub and open the repository named "**Week-2---Git-GitHub-Assignment**".
    - Copy everything from the index.html file and the style.css file, then paste them into the same files in your own project.

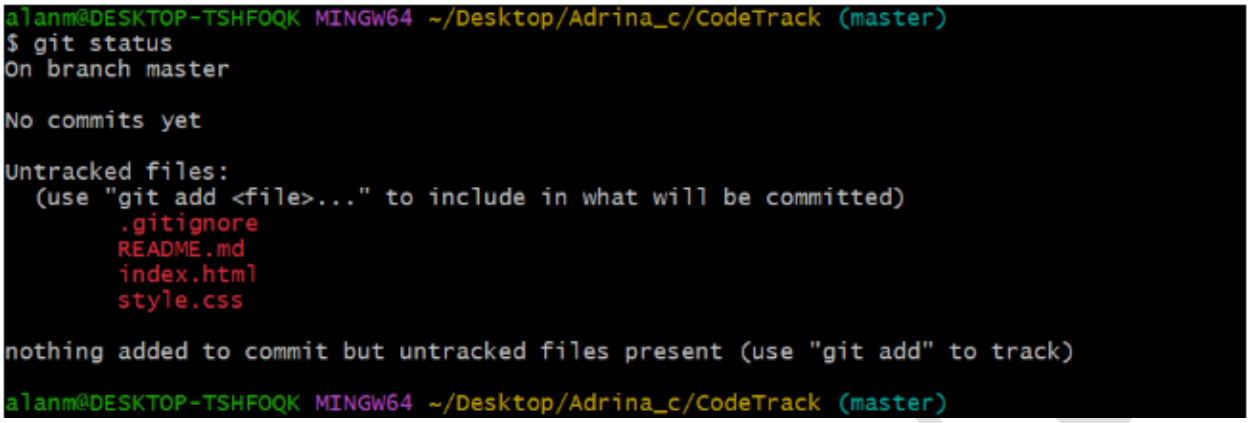


Task 2: Track the Files Using Git

1. **Check the repository status:**
    1. git status

Expected Output:
Git will list both index.html and style.css as **untracked** files.

Screenshot A: Output of git status before adding files (showing untracked).



2. **Stage the files for tracking:**
    1. git add .

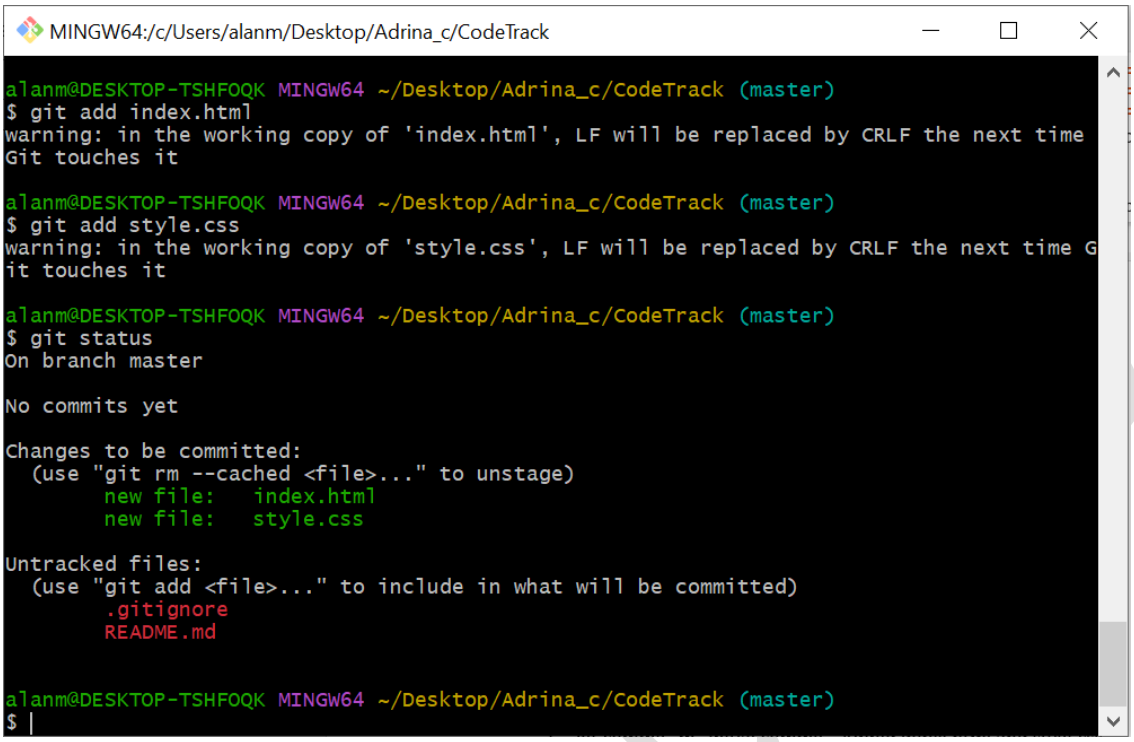
or stage them one by one:

    2. git add index.html
    3. git add style.css

3. **Verify that the files are now staged:**
    1. git status

Expected Output:
The files should now be **staged** and marked as **"Changes to be committed."**

Screenshot B: Output of git status after adding files (showing staged).



Task 3: Commit the Changes

1. **Commit the staged files with a meaningful message:**
    1. git commit -m "Initial commit - Added index.html and style.css"

Expected Output:

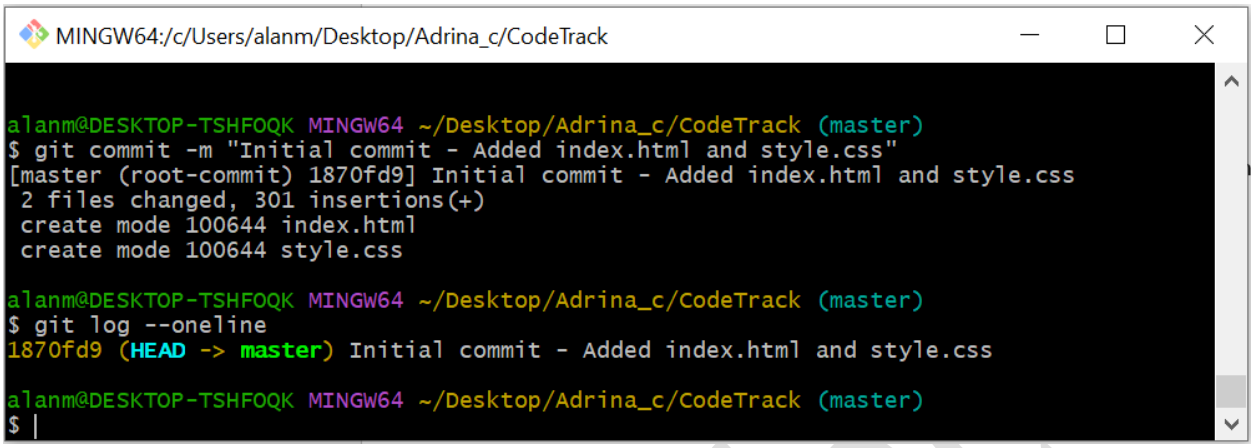
2. [master (root-commit) 1a2b3c4] Initial commit - Added index.html and style.css
3. 2 files changed, 10 insertions(+), 0 deletions(-)

2. **Verify the commit history:**
    1. git log --oneline

Expected Output:

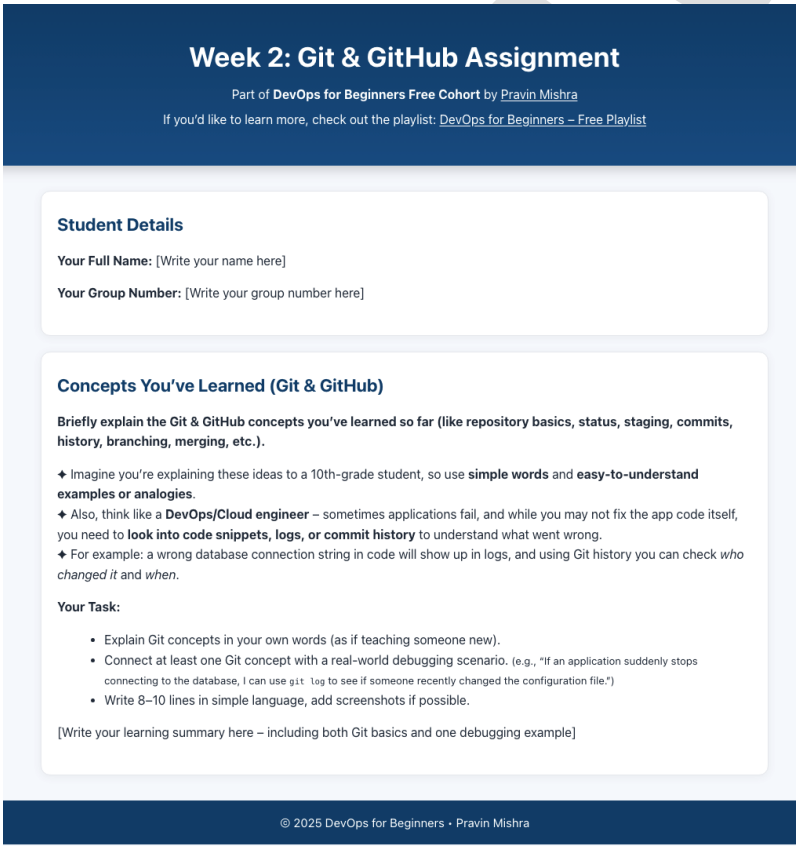
2. 1a2b3c4 Initial commit - Added index.html and style.css

Screenshot C: Output of git log --oneline after the first commit.

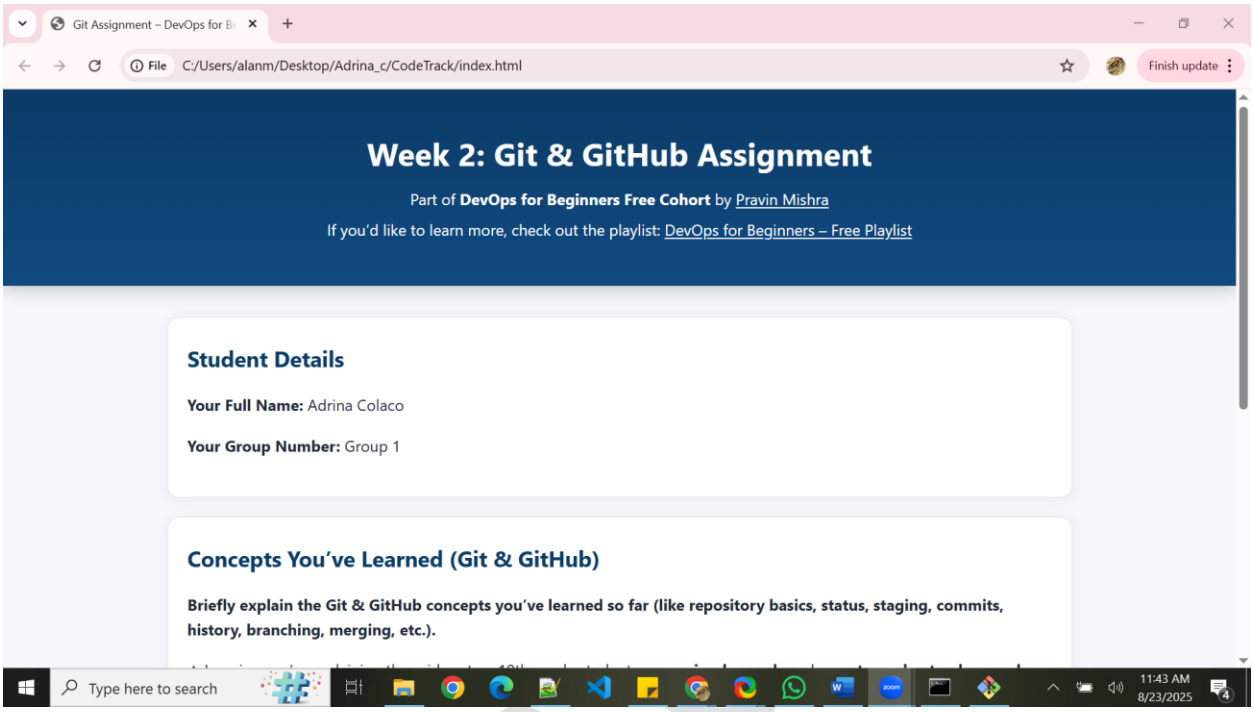


Task 4: Modify a File and Commit Again

1. Next, open the index.html file in your favorite browser. (Tip: right-click the file → choose **Open With Browser**).
2. Read the instructions written inside index.html and follow them step by step to complete the assignment. It must be like below:



3. After finishing, move on to the next steps.



4. **Check the repository status:**
    1. git status

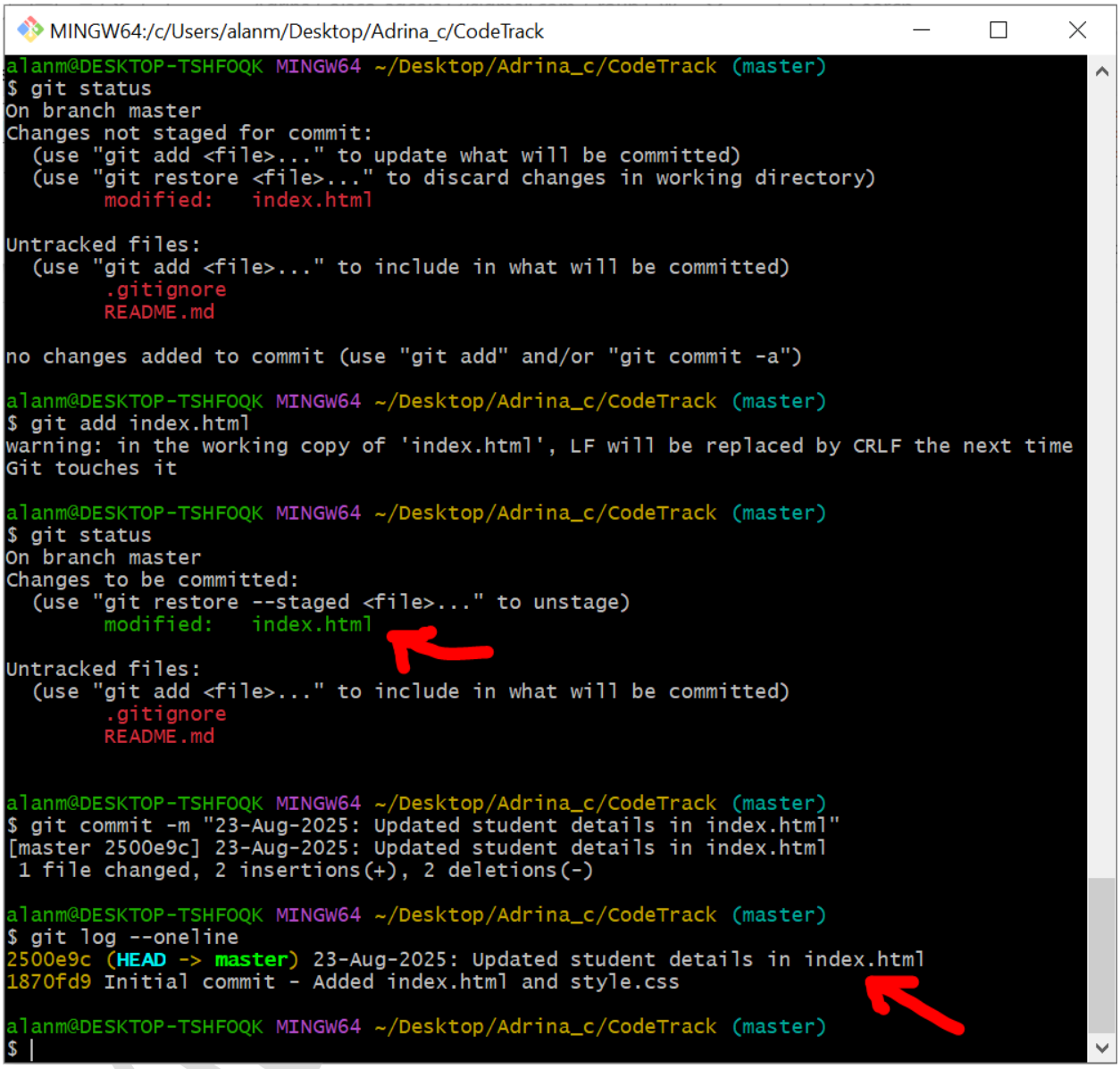
    Git will show index.html as **modified**.

5. **Stage the modified file:**
    1. git add index.html
6. **Commit the changes with a message:**
    1. git commit -m "Meaningful message"
7. **Verify commit history again:**
    1. git log --oneline

Expected Output:

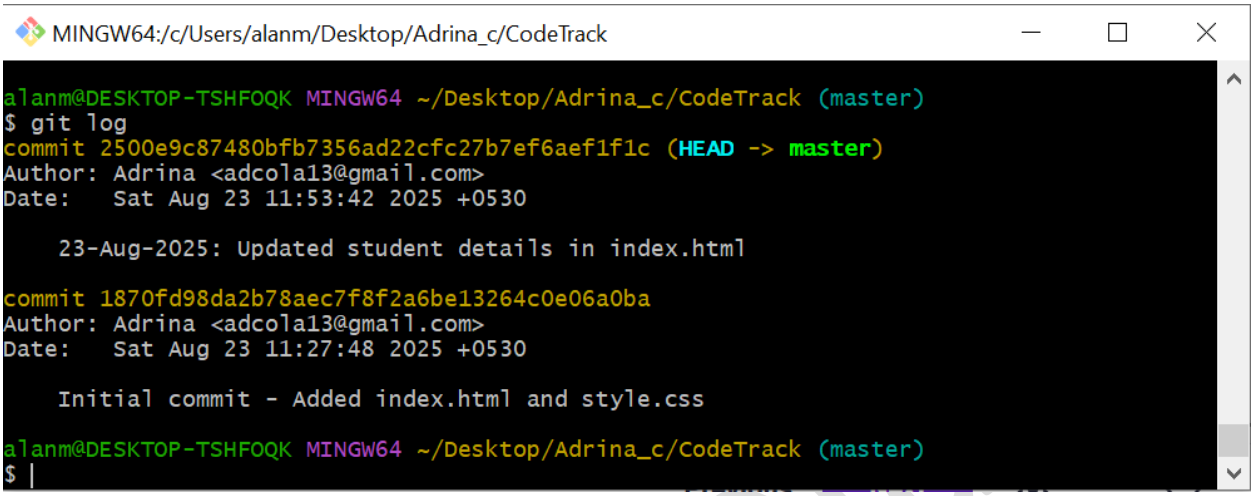
2. 3d4e5f6 Updated heading in index.html
3. 1a2b3c4 Initial commit - Added index.html and style.css

Screenshot D: Output of git log --oneline after the second commit.



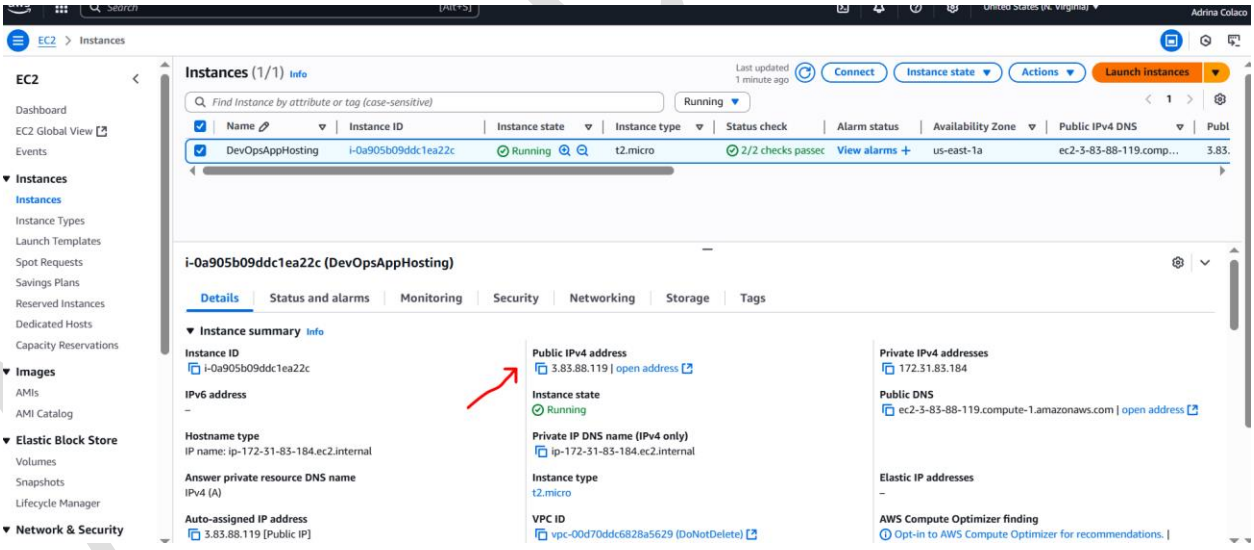
To see all details of the commits use below command

git log

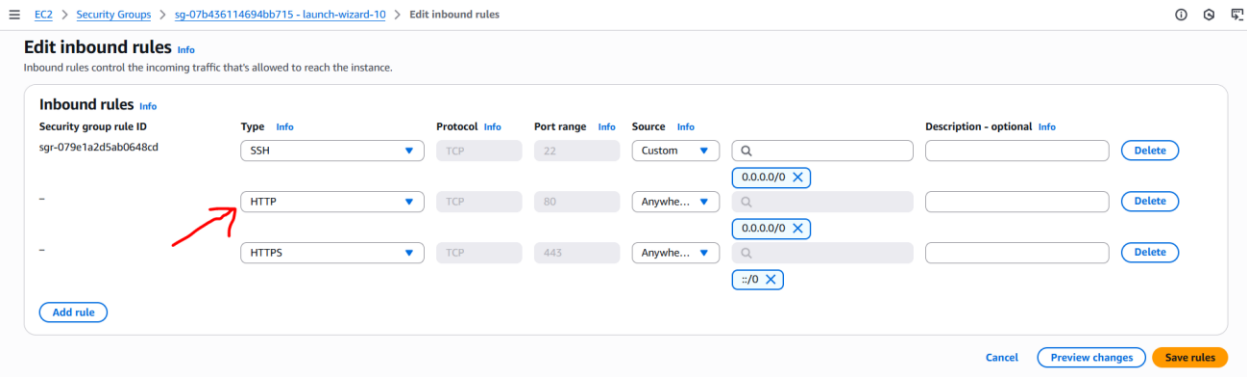


Task 5: Deploy this application on EC2 Instance (as done in the Linux section)

1. Launch an EC2 Instance



Make sure to add HTTP to the inbound rules of the respective security group while creating the instance.



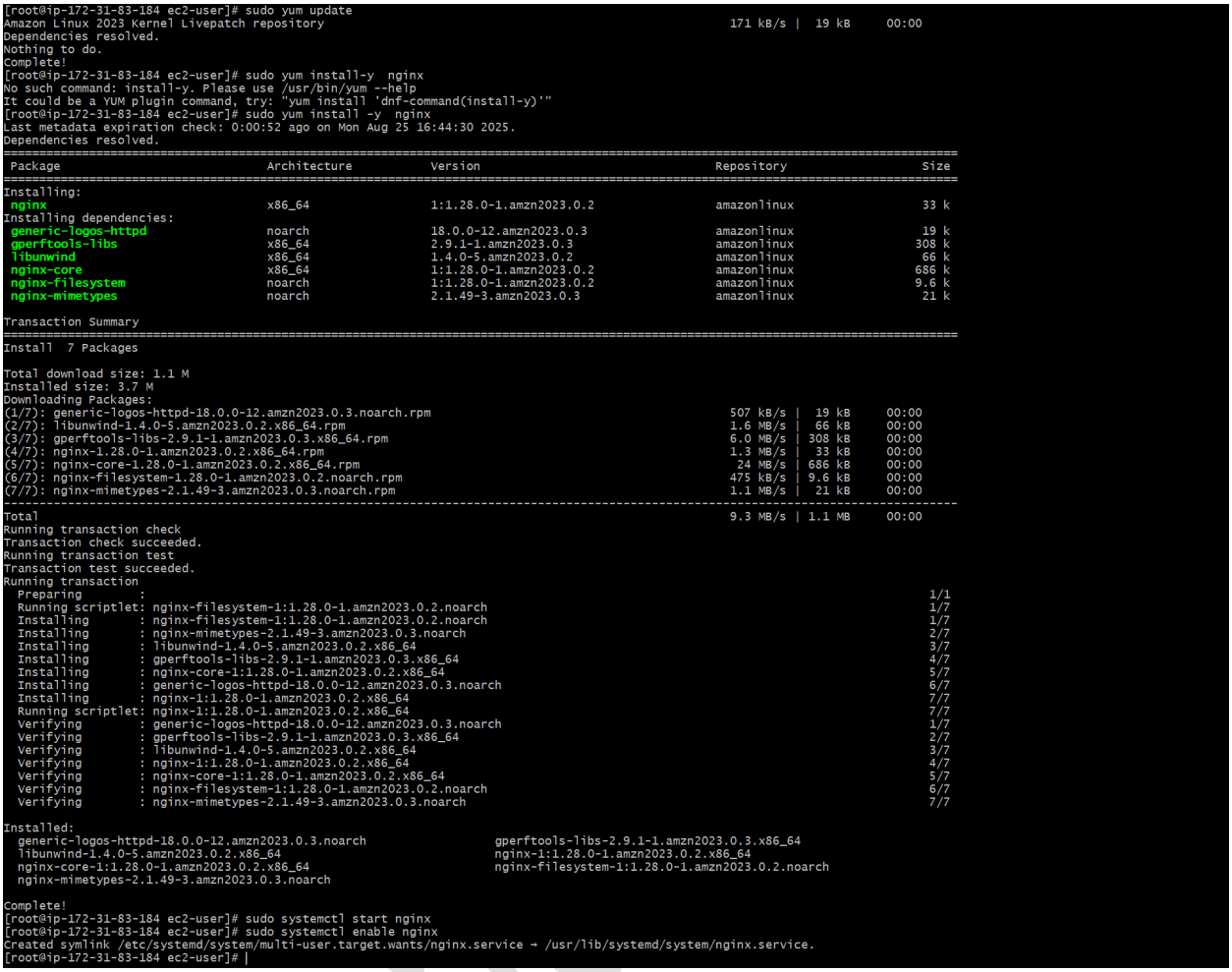
2. Connect to Your EC2 Instance
    - Open a terminal on your local machine.
    - Use SSH to connect:
        1. ssh -i your-key.pem ec2-user@<EC2-Public-IP>

```
alanm@DESKTOP-TSHFOQK MINGW64 ~/Desktop/Adrina_c/CodeTrack (master)
$ ls
23_8       DevopsAppHosting.pem  contact.html  mini_finance/  pubkey.pub
23_8.pub   README.md             index.html    pubkey         style.css

alanm@DESKTOP-TSHFOQK MINGW64 ~/Desktop/Adrina_c/CodeTrack (master)
$ chmod 400 "DevopsAppHosting.pem"

alanm@DESKTOP-TSHFOQK MINGW64 ~/Desktop/Adrina_c/CodeTrack (master)
$ ssh -i "DevopsAppHosting.pem" ec2-user@ec2-3-83-88-119.compute-1.amazonaws.com
The authenticity of host 'ec2-3-83-88-119.compute-1.amazonaws.com (3.83.88.119)' can't b
e established.
ED25519 key fingerprint is SHA256:g7UoQAK62Grd9bLc7knKAPekUotjFJXJJYh4Mlb02Bc.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-83-88-119.compute-1.amazonaws.com' (ED25519) to the li
st of known hosts.
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-83-184 ~]$
```

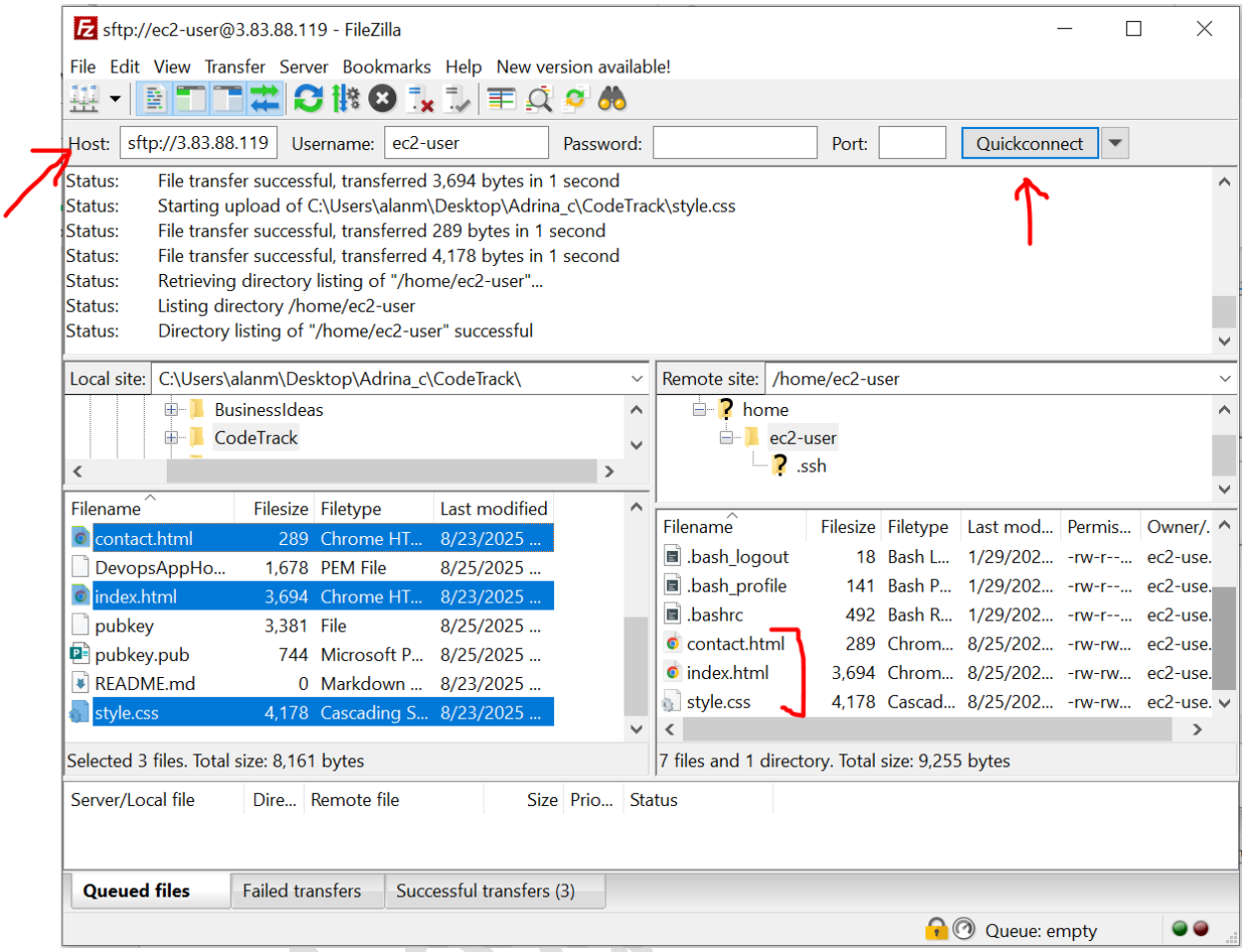
3. Update Packages *(as done in Linux section)*
4. Install Nginx *(as done in Linux section)*
5. Start Nginx Service



6. Deploy Your Application *(same as Linux section)*

    - Copy your project files (index.html, style.css, etc.) into Nginx's web directory:

        1. Use FileZilla to move the project located on the local machine to EC2(MobaXterm can also be used)
        2. Go to Edit → Settings → Under Connection → SFTP → add the PEM key that downloaded while creating your EC2 instance.
        3. Quick Connect to the server using sftp and move the files from local machine to EC2 server.

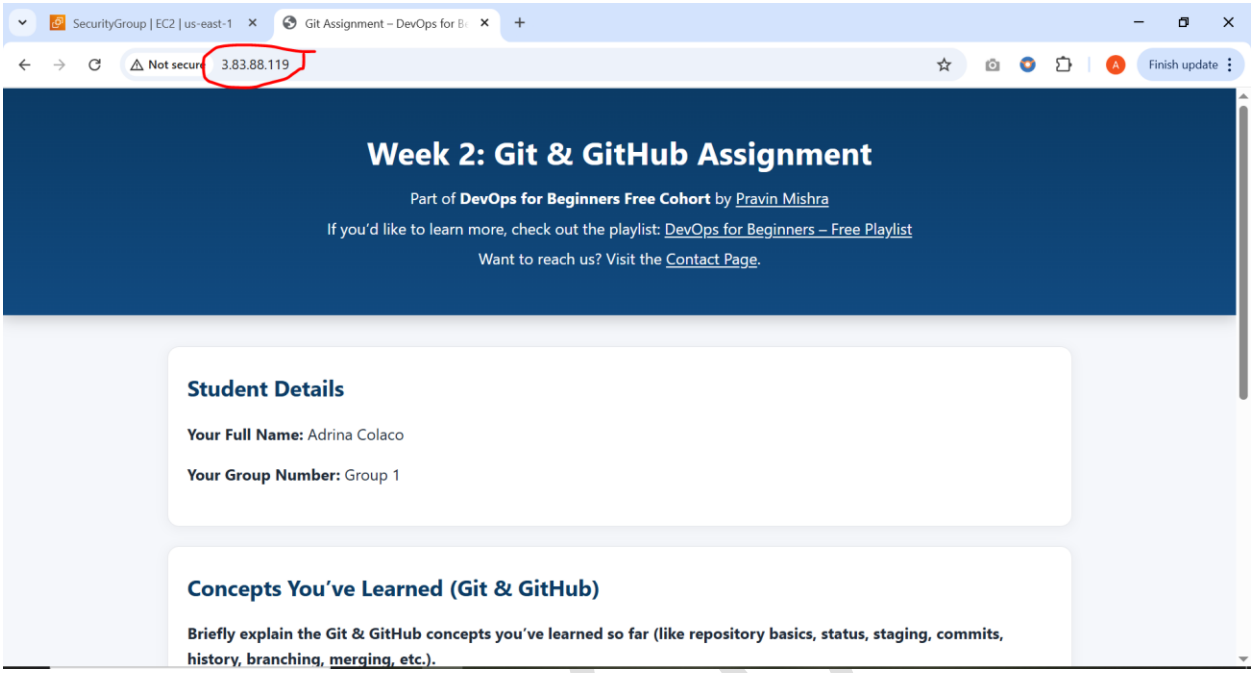


4. Copy the required web files to the NGINC web directory.
    1. sudo cp /home/ec2-user/index.html /usr/share/nginx/html/index.html
    2. sudo cp /home/ec2-user/contact.html /usr/share/nginx/html/contact.html
    3. sudo cp /home/ec2-user/style.css /usr/share/nginx/html/style.css

```
[ec2-user@ip-172-31-83-184 html]$ sudo cp /home/ec2-user/index.html /usr/share/nginx/html/index.html
[ec2-user@ip-172-31-83-184 html]$ sudo cp /home/ec2-user/contact.html /usr/share/nginx/html/contact.html
[ec2-user@ip-172-31-83-184 html]$ sudo cp /home/ec2-user/style.css /usr/share/nginx/html/style.css
[ec2-user@ip-172-31-83-184 html]$ ls
404.html  50x.html  contact.html  icons  index.html  nginx-logo.png  poweredby.png  style.css
[ec2-user@ip-172-31-83-184 html]$
```

7. Access the Application
    - Open a browser and go to:
        1. http://<EC2-Public-IP>

You should see your application running successfully.



Text File (git_tracking_summary.txt):

- Listing all Git commands used so far with a short explanation for each.

Git_Tracking_Summary.txt

Assignment 7- Branching Workflow — Add & Verify a Contact Page in Git

Task 0 — Start from your existing repo

Navigate your **existing CodeTrack** project from the last assignment:

1. cd path/to/CodeTrack
2. git status
3. git branch

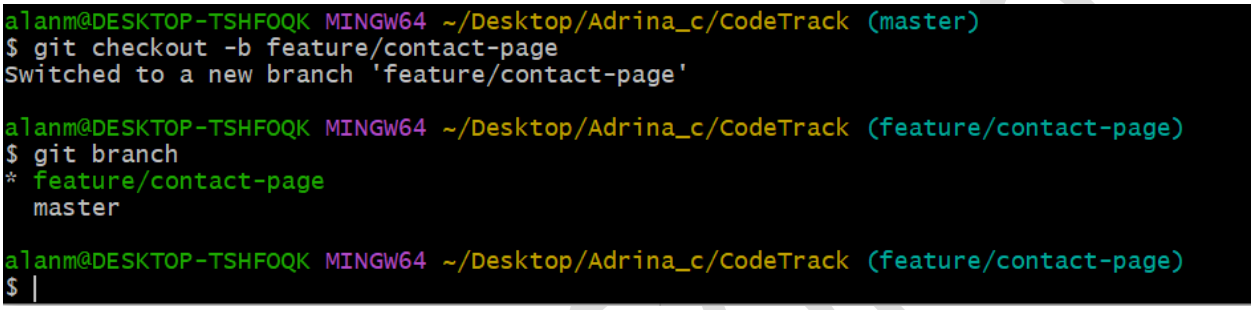
Ensure you're on main (or master).

Task 1 — Create and switch to a feature branch

1. git checkout -b feature/contact-page
2. git branch

Expected: * feature/contact-page.

Screenshot A: git branch output right after creating feature/contact-page (shows * feature/contact-page).



Task 2 — Add contact.html (in the branch)

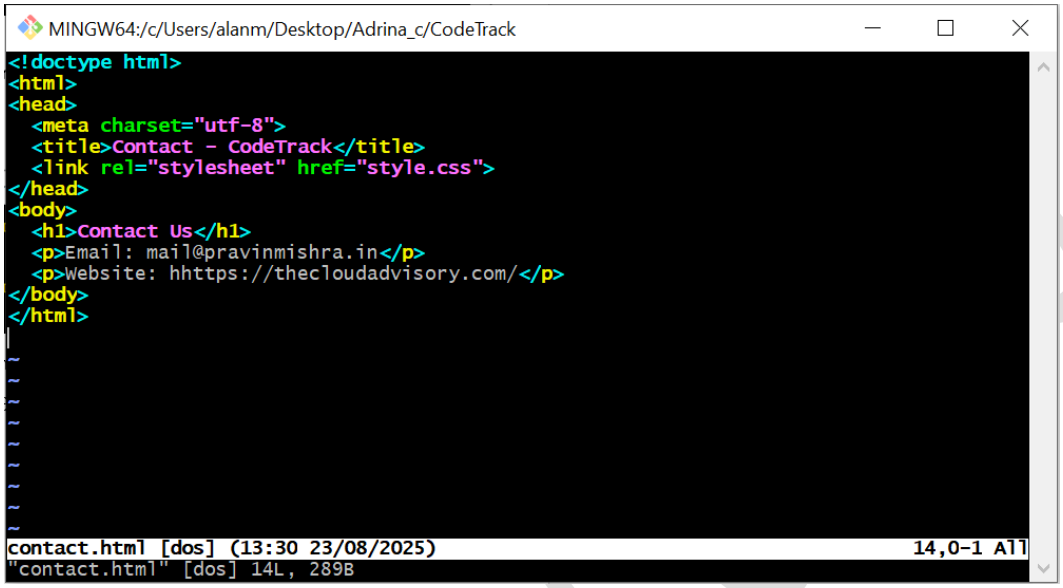
Create the file and add content:

1. # macOS/Linux
2. touch contact.html
3. # Windows PowerShell alternative:
4. # ni contact.html

contact.html

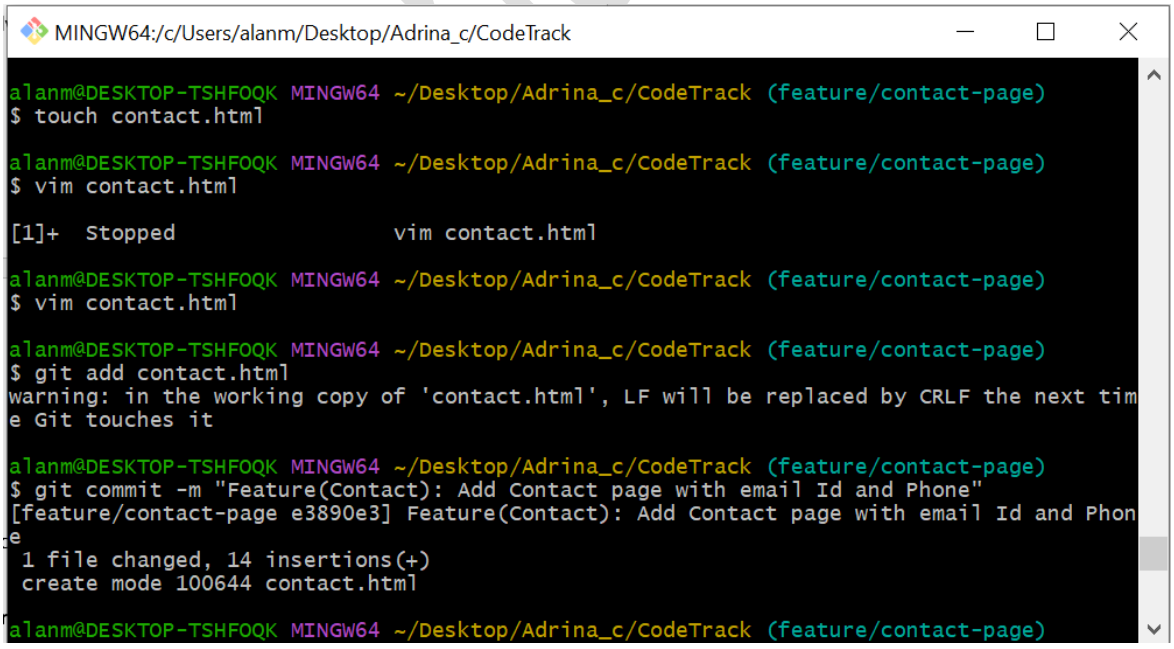
```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Contact - CodeTrack</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Contact Us</h1>
```

```
  <p>Email: mail@pravinmishra.in</p>
  <p>Website: hhttps://thecloudadvisory.com/</p>
</body>
</html>
```



Stage & commit:

1. git add contact.html
2. git commit -m "feat(contact): add contact page with email and phone"

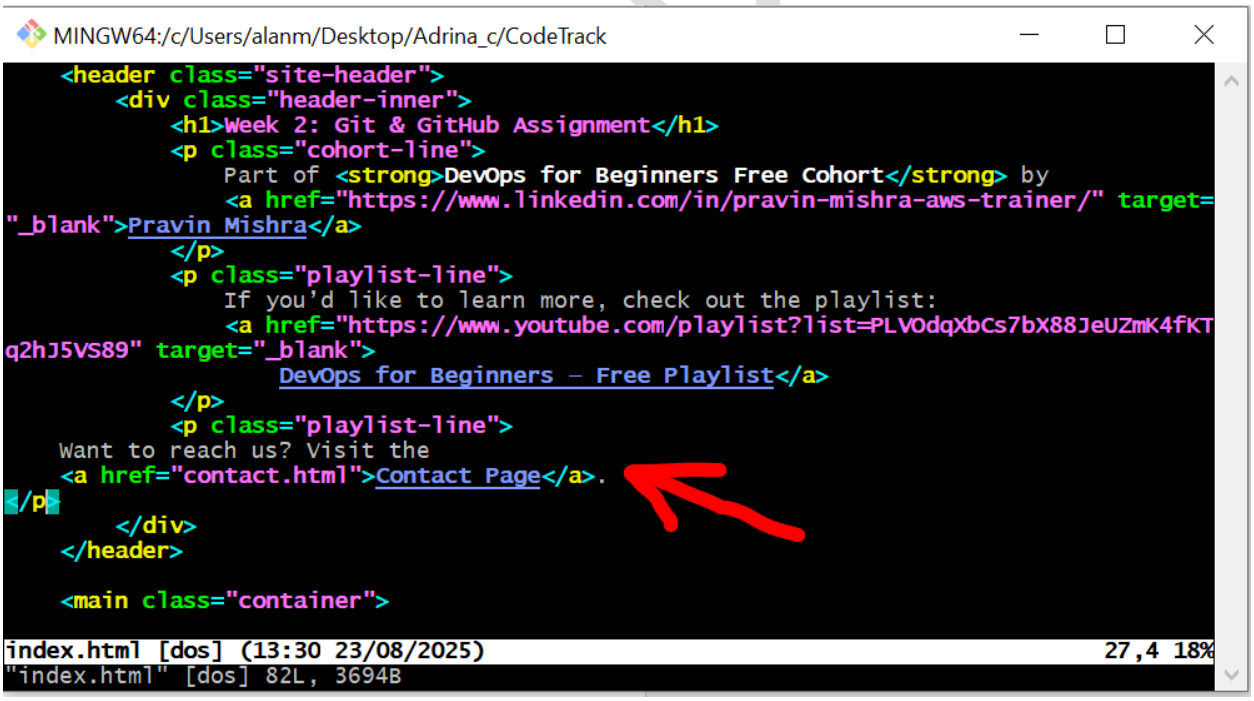


Task 3 — Add a link to the contact page in index.html (still on the branch)

Insert this **new paragraph** directly **below** the existing playlist paragraph (the one with class playlist-line):

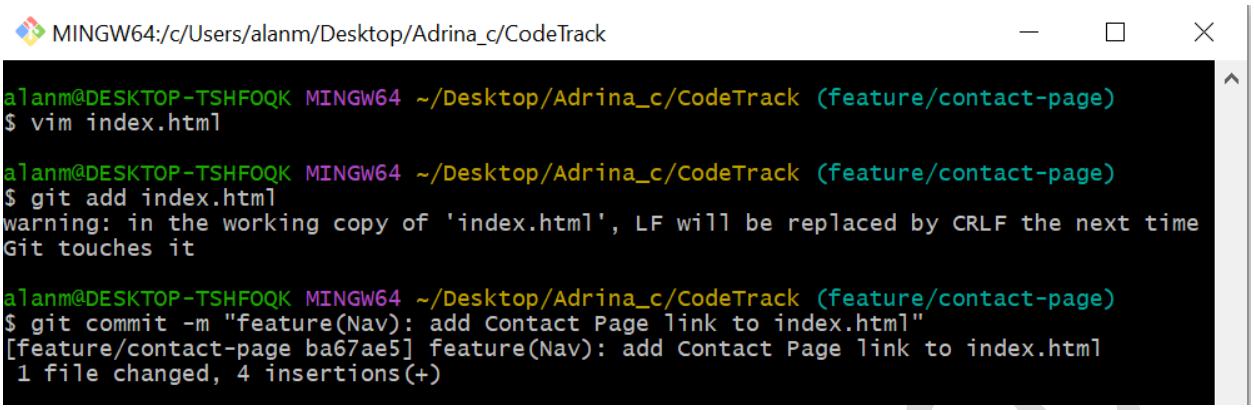
```
<p class="playlist-line">
  If you'd like to learn more, check out the playlist:
  <a
href="https://www.youtube.com/playlist?list=PLVOdqXbCs7bX88JeUZmK4fKTq2h
J5VS89" target="_blank">
    DevOps for Beginners – Free Playlist
  </a>
</p>

<!-- ADD THIS NEW PARAGRAPH RIGHT BELOW THE ONE ABOVE -->
<p class="playlist-line">
  Want to reach us? Visit the
  <a href="contact.html">Contact Page</a>.
</p>
```



Save, then:

1. git add index.html
2. git commit -m "feat(nav): add Contact Page link to index.html"



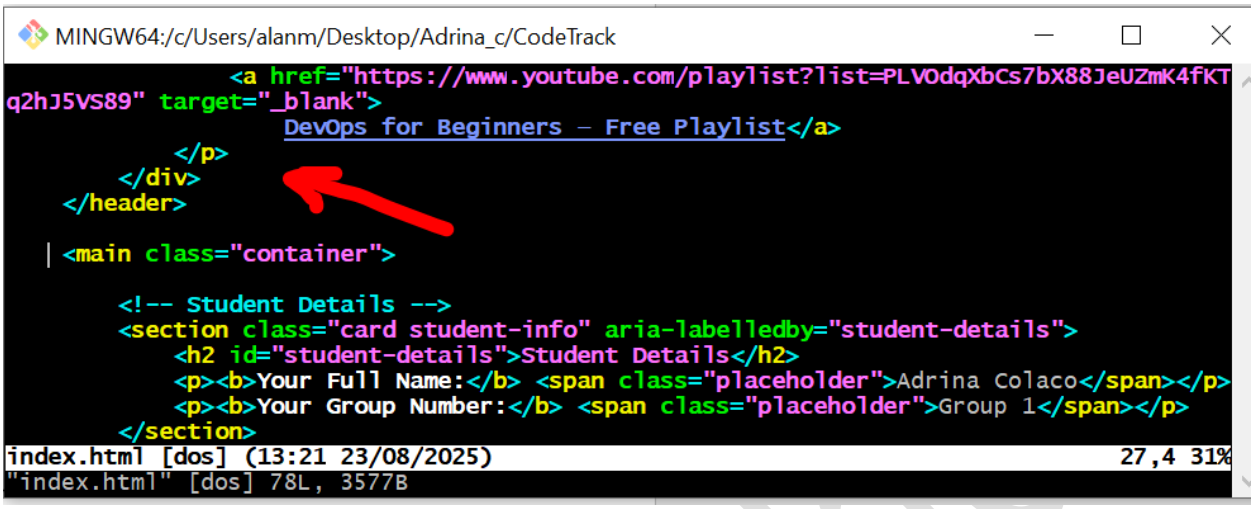
Task 4 — Verify isolation (switch back to main)

1. git checkout main
2. ls

Screenshot B. git branch output after switching back to main.

- contact.html should **not** be in main yet.
- Open index.html in your browser → the **Contact Page** link should **not** exist on main yet.

(This proves your changes live only on the branch.)

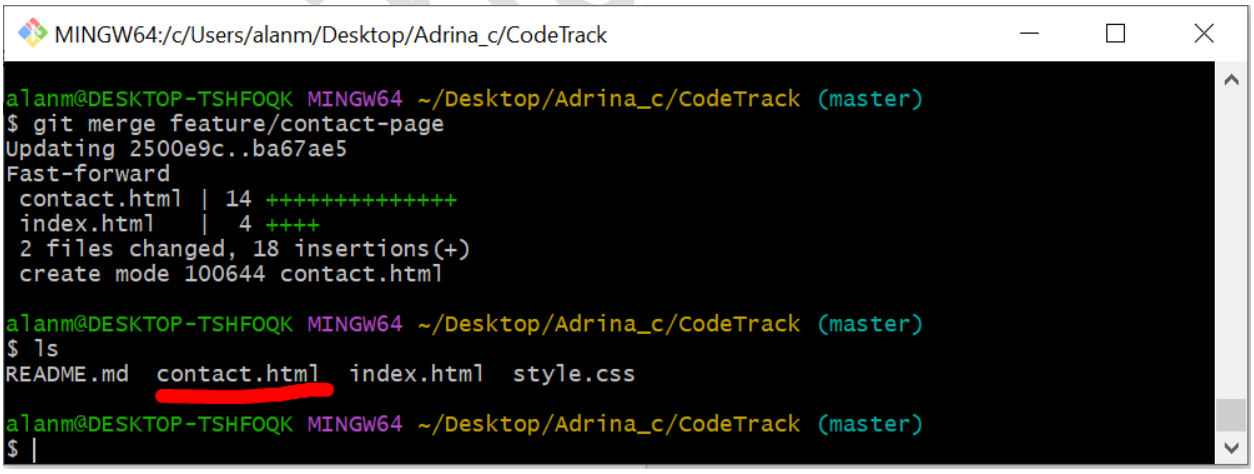


Task 5 — Merge the feature branch into main

1. git merge feature/contact-page

Now verify:

- ls → contact.html is present.
- Open index.html in your browser → you should see the **Contact Page** link.
- Click the link → it should open **contact.html**. ✅



Task 6 — Inspect history (nice graph)

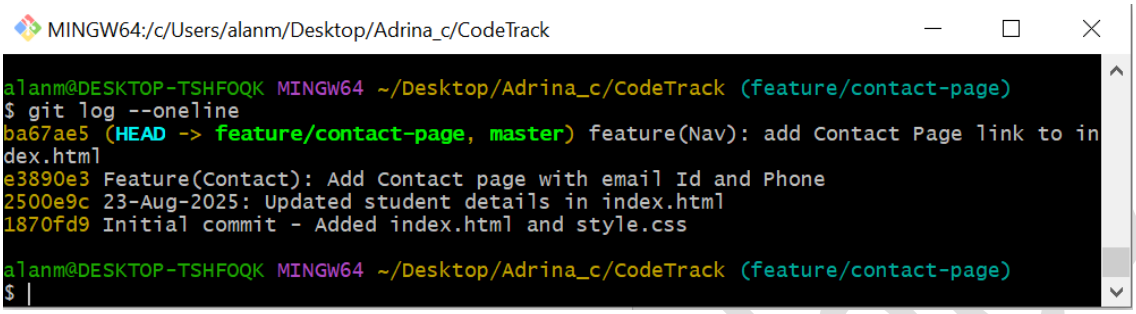
1. git log --oneline --graph --decorate --all

(Optional cleanup)

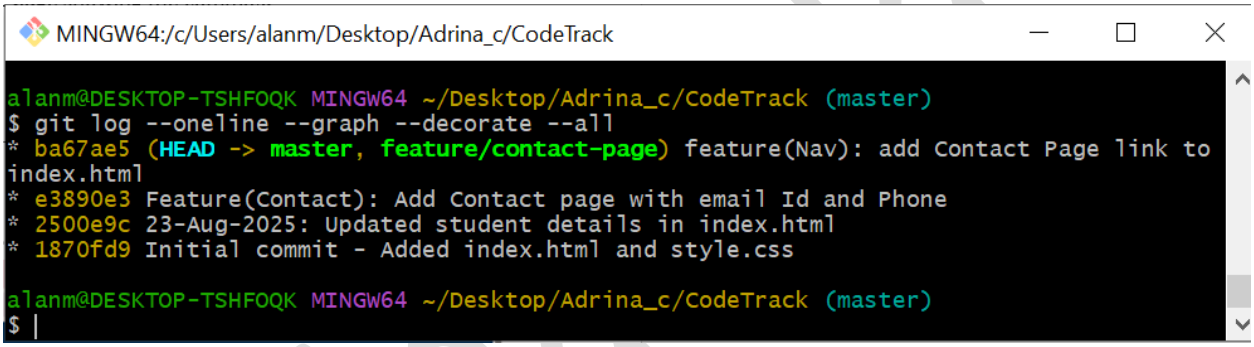
1. git branch -d feature/contact-page

Screenshot C. git log --oneline while on feature/contact-page showing the commits:

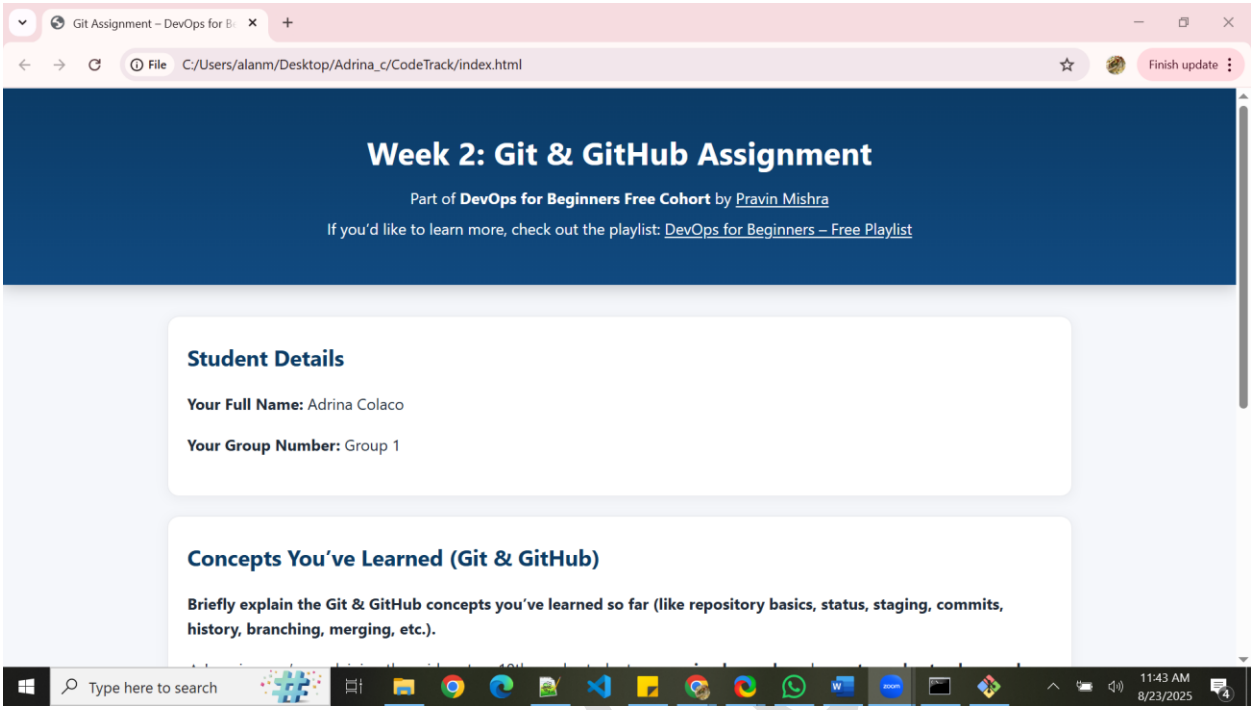
feat(contact): add contact page...
feat(nav): add Contact Page link...



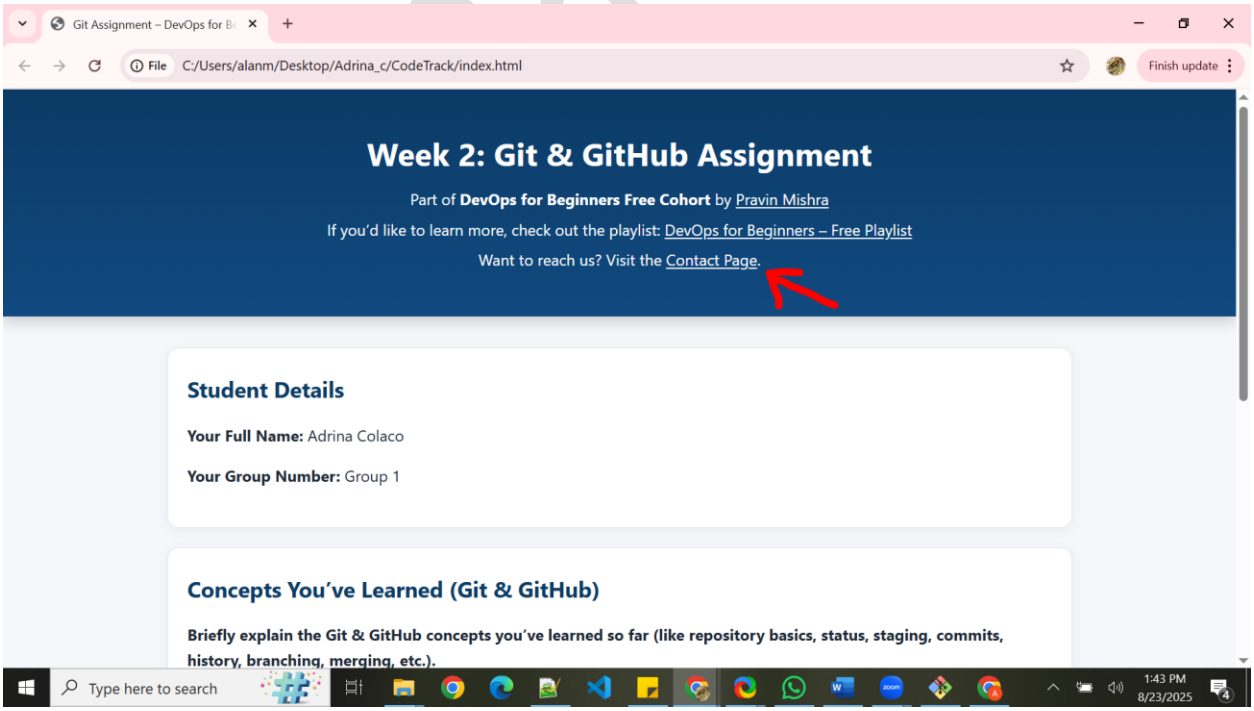
Screenshot D. git log --oneline --graph --decorate --all after merging into main.



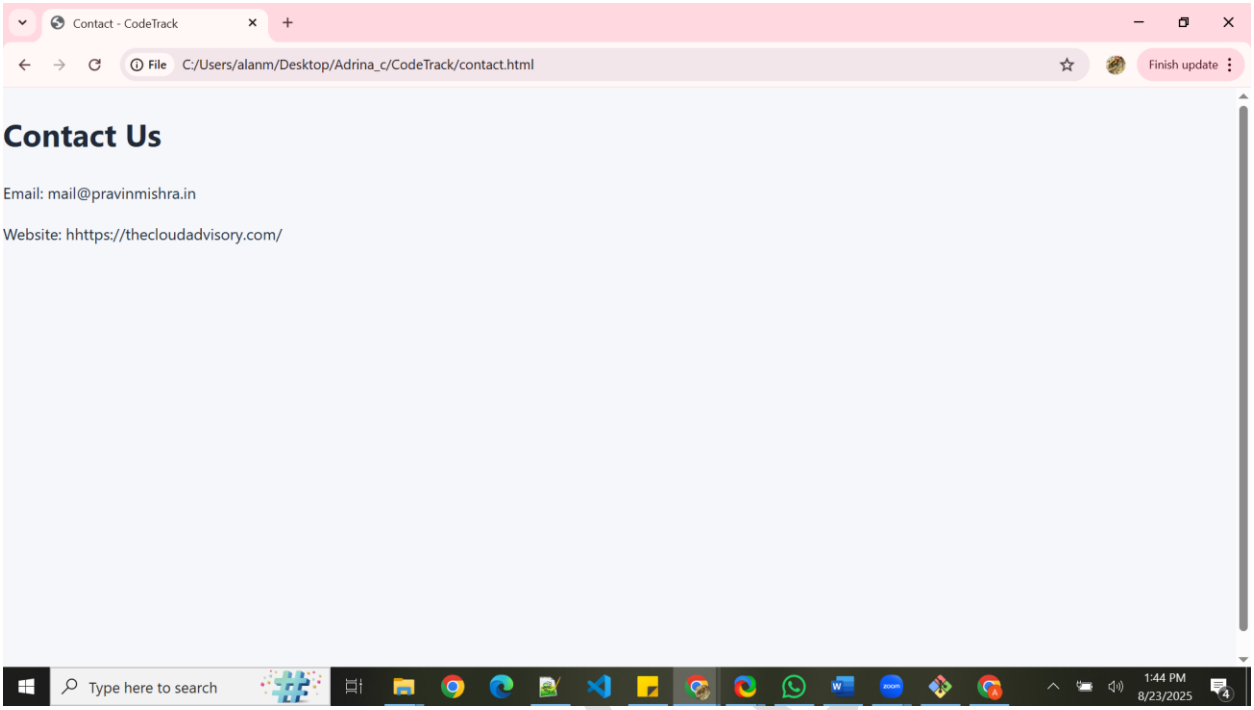
Screenshot E: Browser screenshot of index.html **before merge** (no Contact Page link visible).



Screenshot F: Browser screenshot of index.html **after merge** showing the new **Contact Page** link.



Screenshot G: Browser screenshot of the loaded **contact.html** (after clicking the link).



Explain why the link wasn't visible before the merge and why it appears after.

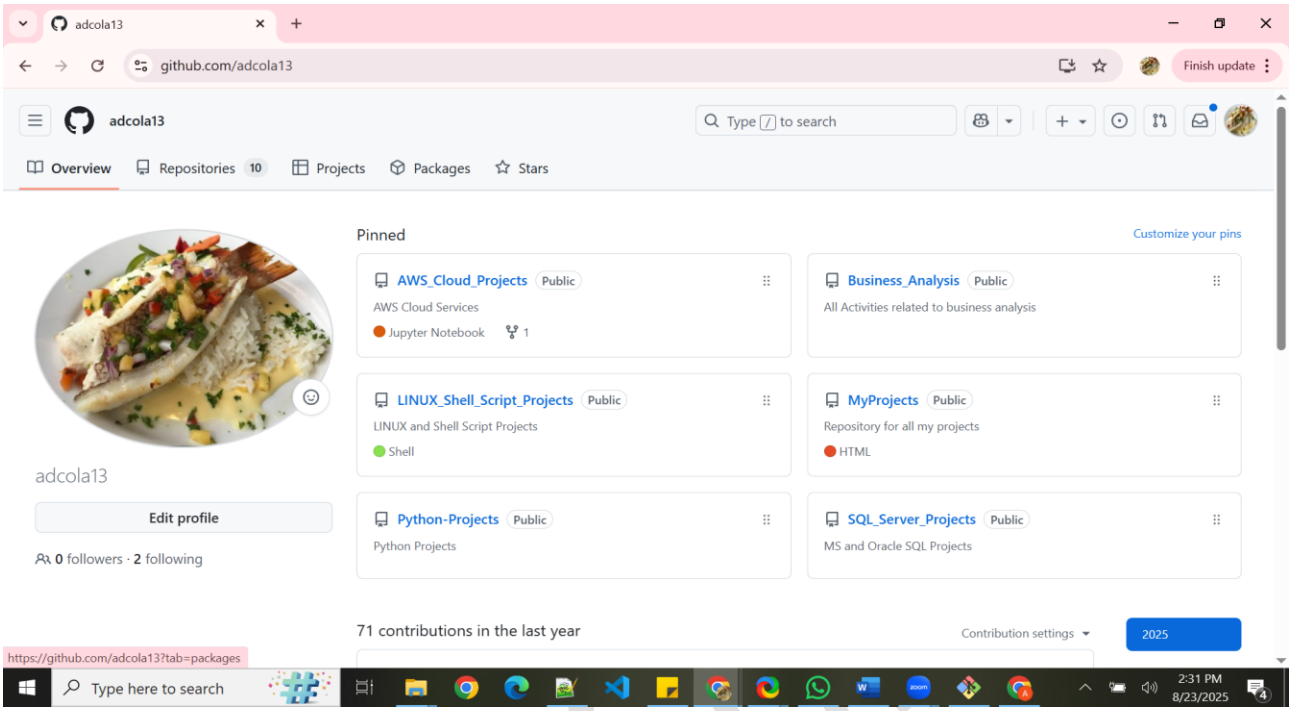
The changed code from the Branch was not yet present in the Main(master) branch before Git Merge.
Once Git merge command was run, the code linking to Contact.html file moved to the index.html file present in main(master) branch as well.

Assignment 8 - Setting Up GitHub for CodeTrack

Task 1 — Create a GitHub Account

1. Go to GitHub and click **Sign Up**.
2. Enter your **email, password, and username**.
3. Complete verification and click **Create Account**.
4. Once logged in, navigate to your **GitHub Dashboard**.

Expected Outcome: You now have a GitHub account and access to the dashboard.



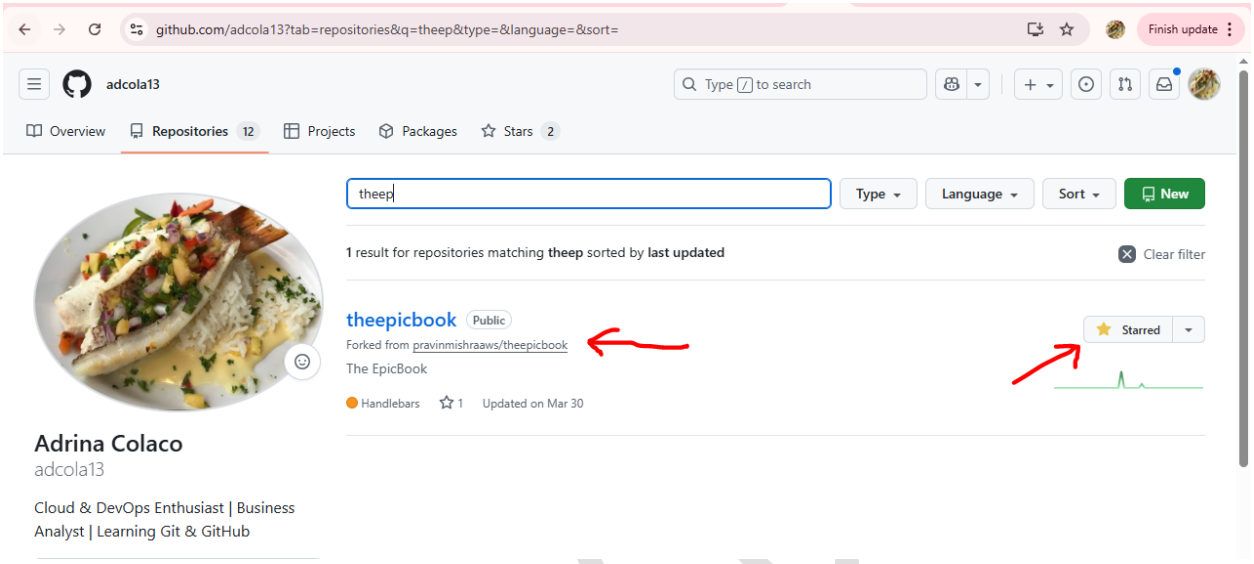
Task 2 — Explore GitHub Features

1. From the top menu, click **Explore**.
2. Browse **Trending Repositories** to see what's popular.
3. Use the search bar to find an open-source project (e.g., type theepicbook).
4. Click the ⭐ **Star** button on a repository that interests you.
5. Click **Fork** on any public repository to create your own copy.

Expected Outcome: You have explored GitHub, starred at least one project, and forked one repository.

Screenshot A: The **repository you starred.**

Screenshot B: The **repository you forked**.

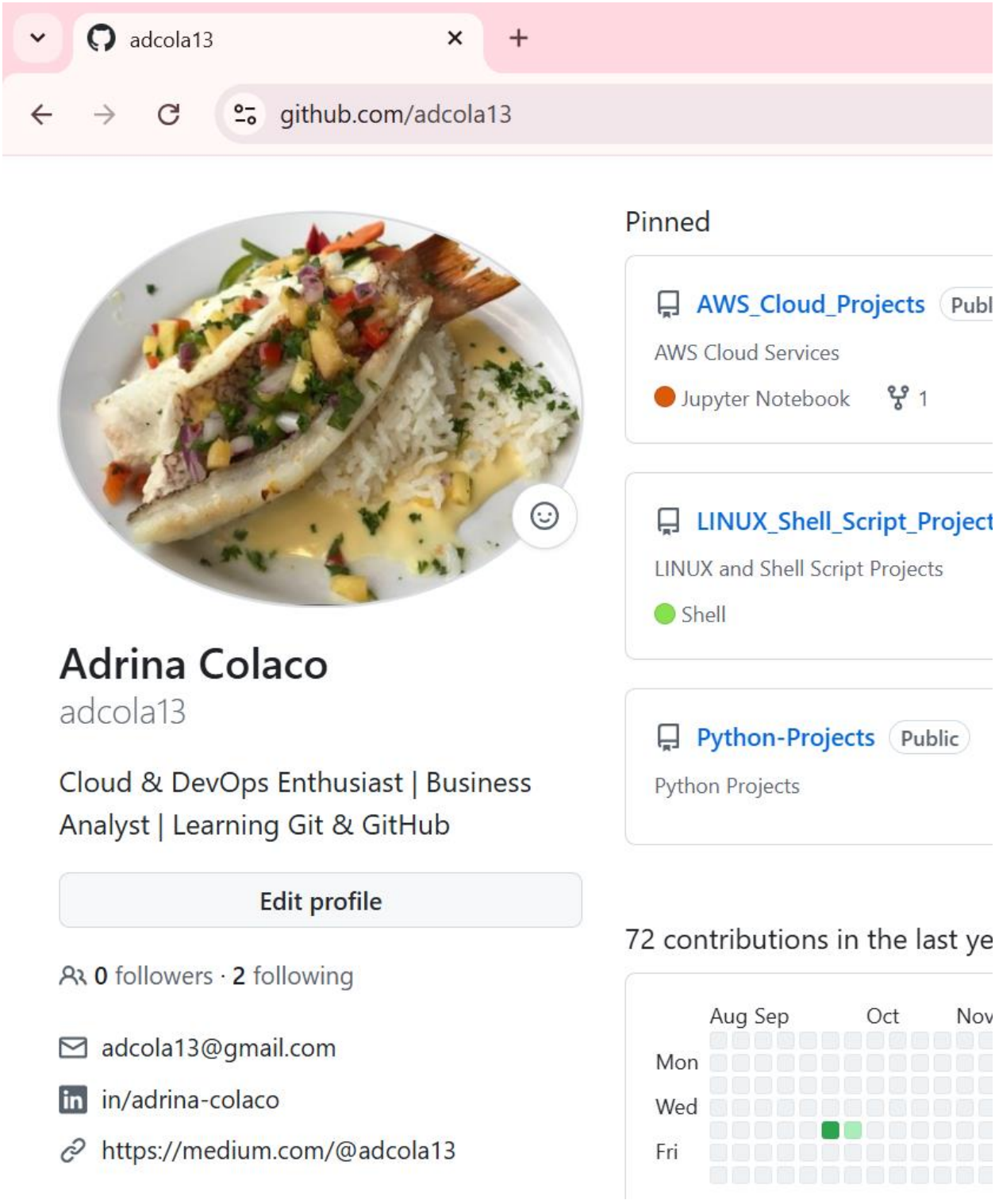


Step 3 — Update Your GitHub Profile

1. Click your profile picture (top-right) → **Your Profile**.
2. Click **Edit Profile** and:
    - Add a short **bio** (e.g., “Cloud & DevOps Enthusiast | Learning Git & GitHub”).
    - Optionally add **location**, **company/school**, and **social links**.
    - Upload a **profile picture** (optional but recommended).
3. Save changes.

Expected Outcome: Your GitHub profile looks personalized and professional.

Screenshot C: Your updated **GitHub profile page** showing your new bio.



Why is it important to have a professional GitHub profile as a developer?

A polished GitHub profile serves as a **live portfolio**, showcasing your real code, collaboration history, and growth trajectory—making it a powerful tool for credibility, visibility, and hiring potential.

Key Takeaways:

- **Real code > resume claims**: Viewers can assess your actual work and problem-solving style.
- **Reflects growth & commitment**: A history of contributions signals continuous learning.
- **Trusted evidence**: For those with limited formal experience, your GitHub can be the only proof you can code.

Assignment 9- Collaborating on Mini-Finance with GitHub

Task 0 — Access Existing Mini-Finance Code

The upstream repository exists at:

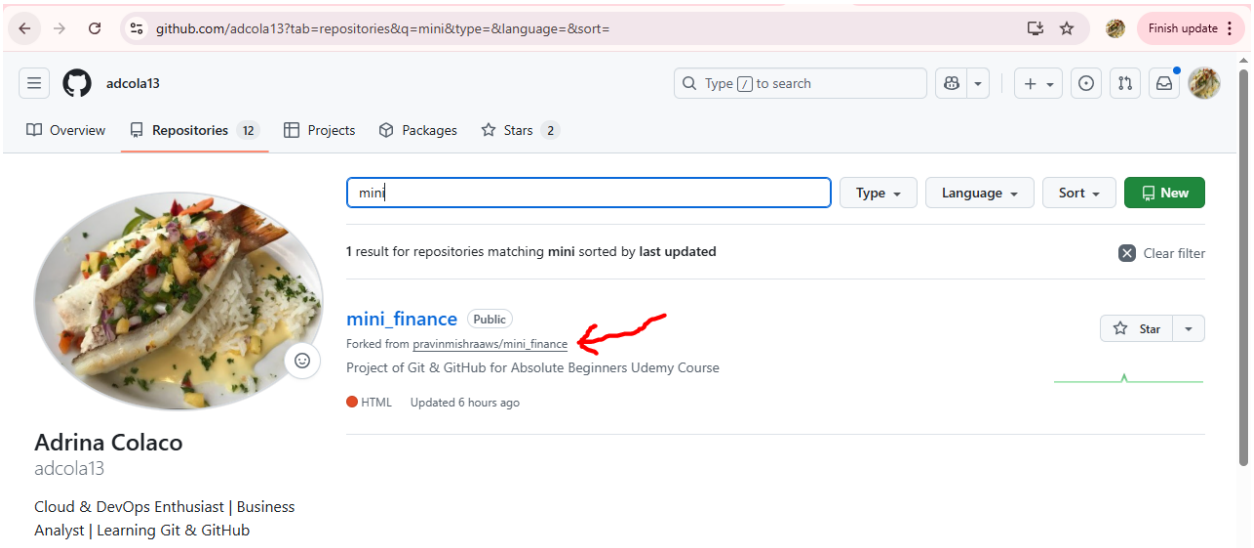
1. https://github.com/pravinmishraaws/mini_finance

We'll fork this repository to your GitHub account, clone, and work with it.

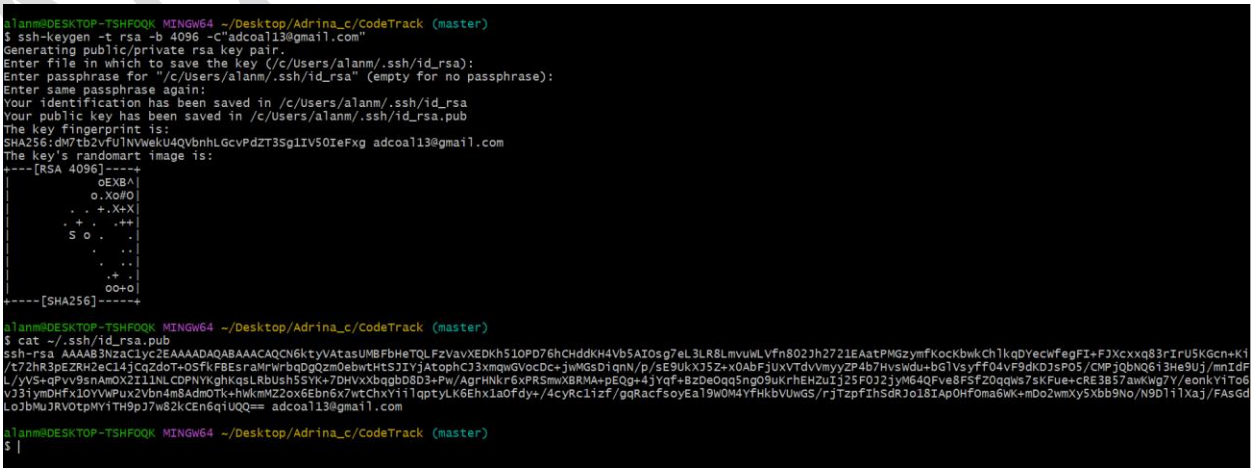
Task 1 — Fork & Authenticate

1. Login to GitHub and **Fork** the mini_finance repository into your account.

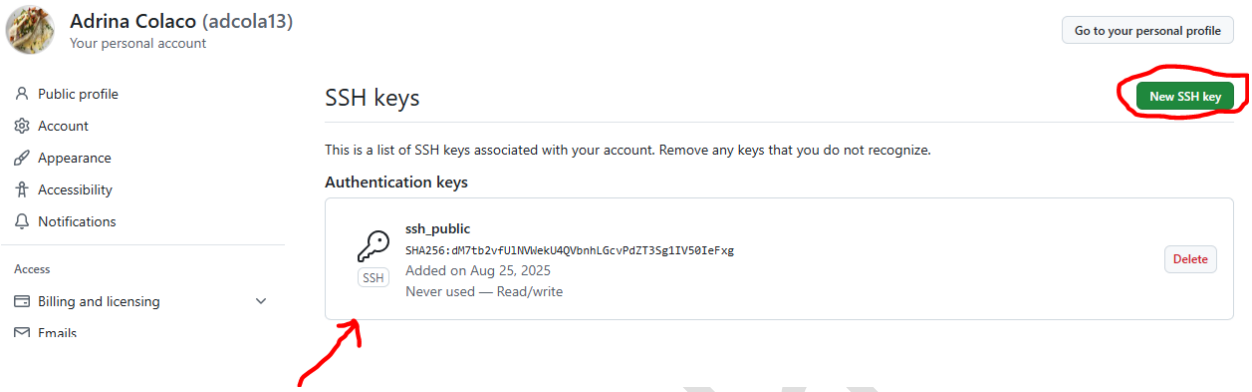
Screenshot A: The GitHub page showing your **forked mini_finance repo** under your account (URL visible).



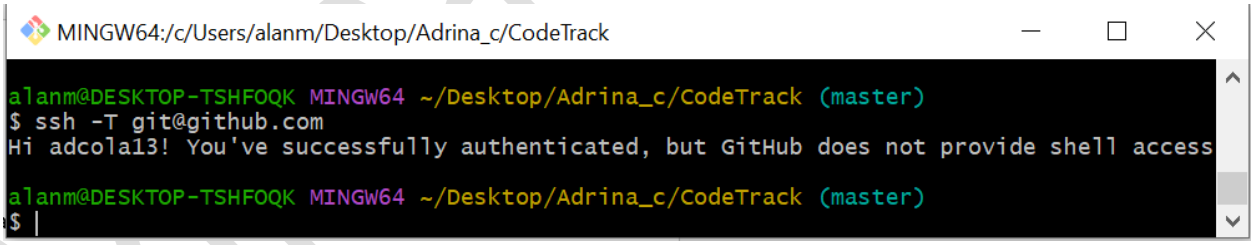
2. In your terminal, configure authentication if not already set:
    - **SSH** (recommended):
        1. **Generate an SSH Key (if not already created):**
            1. ssh-keygen -t rsa -b 4096 -C "your-email@example.com"
            2. Press **Enter** to save the key in the default location (~/.ssh/id_rsa).
        2. **Add the SSH Key to GitHub:**
            1. Copy the SSH key:
            2. cat ~/.ssh/id_rsa.pub



3. Go to **GitHub → Settings → SSH and GPG keys** → Click **New SSH Key**
4. Paste the key and save



3. **Test the SSH Connection:**
    1. ssh -T git@github.com
    2. If successful, you should see a message like:
    3. Hi yourusername! You've successfully authenticated, but GitHub does not provide shell access.



4. **Use SSH for Git Commands:**
   When cloning a repository, use the SSH URL instead of HTTPS:
5. git clone git@github.com:yourusername/repository.git

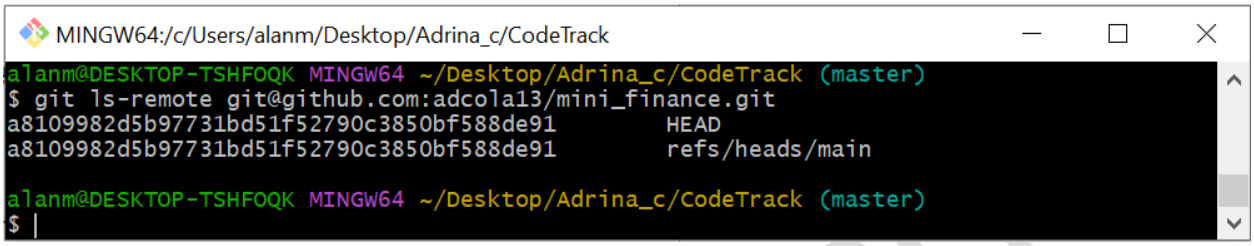
6. git config --global url."git@github.com:".insteadOf "https://github.com/"

- **OR HTTPS**:
    1. git config --global credential.helper cache

Test with:

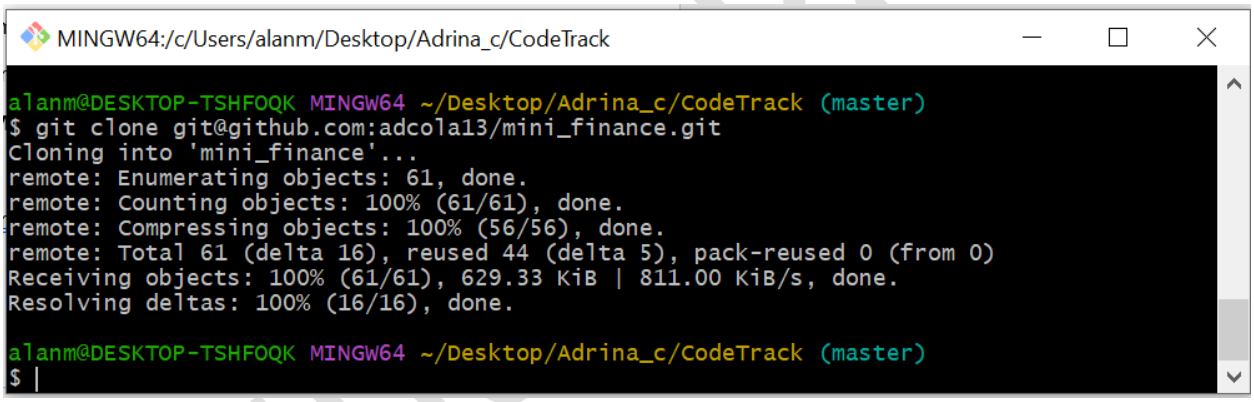
2. git ls-remote git@github.com:yourusername/mini_finance.git

3. **Expected outcome:** You have forked the repo and your terminal is authenticated for Git ops.



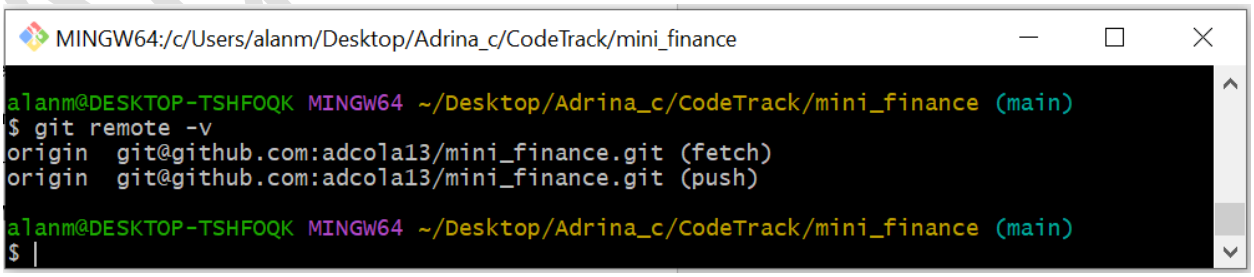
Task 2 — Clone Your Fork Locally

1. git clone git@github.com:yourusername/mini_finance.git



2. cd mini_finance
3. git remote -v

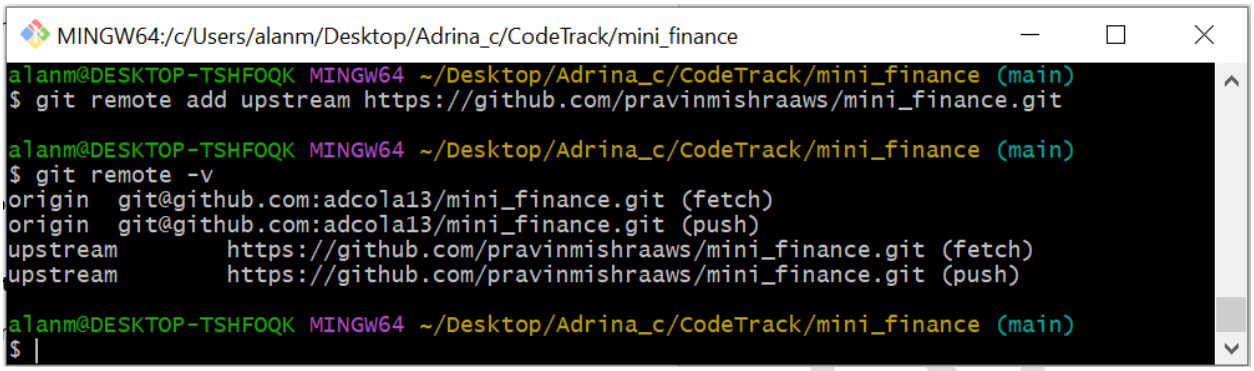
- origin should point to your fork.



1. Add an upstream remote to the original repo:
    1. git remote add upstream https://github.com/pravinmishraaws/mini_finance.git

Expected outcome: Local clone with origin (your fork) and upstream properly set.

Screenshot B: Terminal output of git remote -v after cloning, showing both origin and upstream.

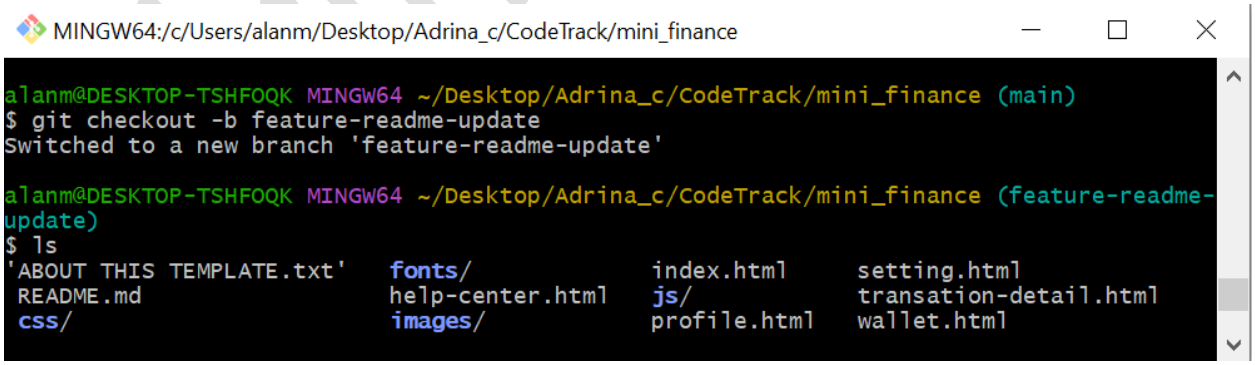


What is the purpose of the upstream remote in your workflow?

The **upstream remote** serves as an anchor to the original repository, enabling you to **fetch** the latest changes, keep your fork in sync, and make your contributions clean and up-to-date. It's essential for effective collaboration, especially in open-source or team-based workflows.

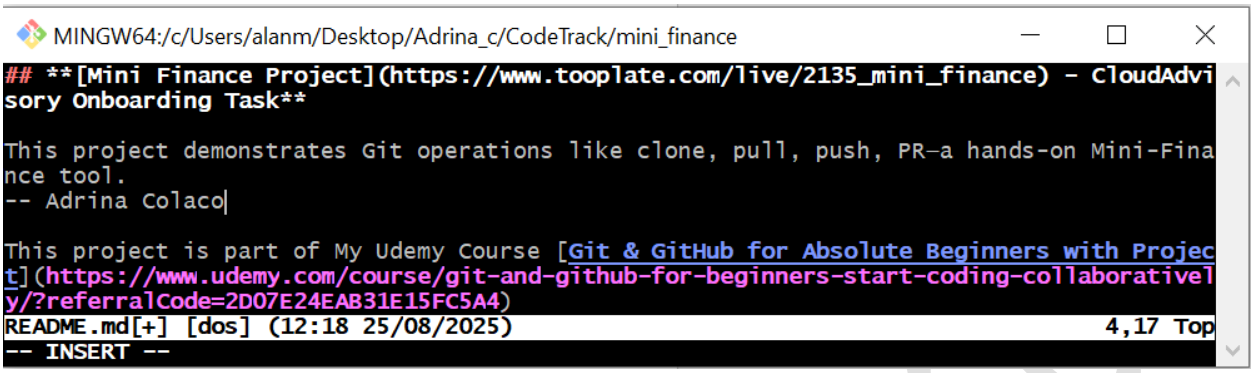
Task 3 — Create a Feature Branch & Make a Change

2. Create a new branch:
    1. git checkout -b feature-readme-update



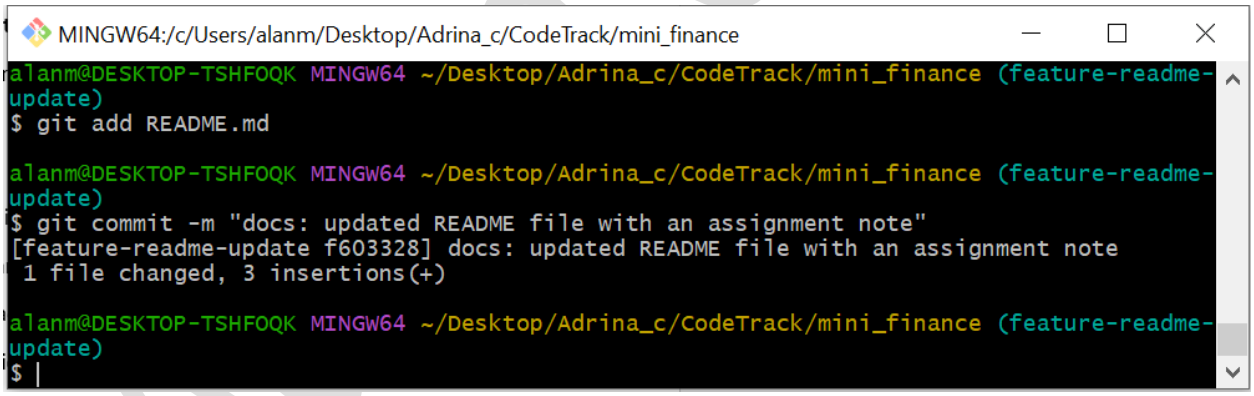
3. Open README.md and add a new section:

You may write: “This project demonstrates Git operations like clone, pull, push, PR—a hands-on Mini-Finance tool.”



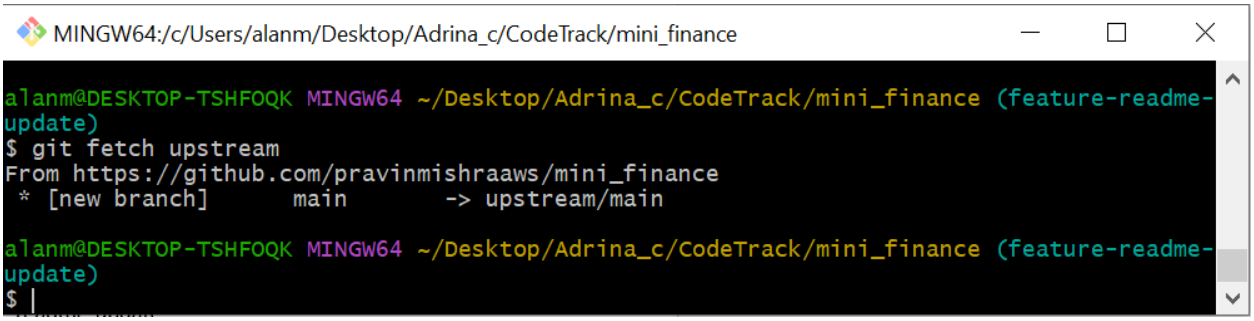
4. Save, then stage and commit:
    1. git add README.md
    2. git commit -m "docs: update README with assignment note"

Screenshot C: Terminal showing your commit on feature-readme-update. (e.g., git log --oneline -n 3)

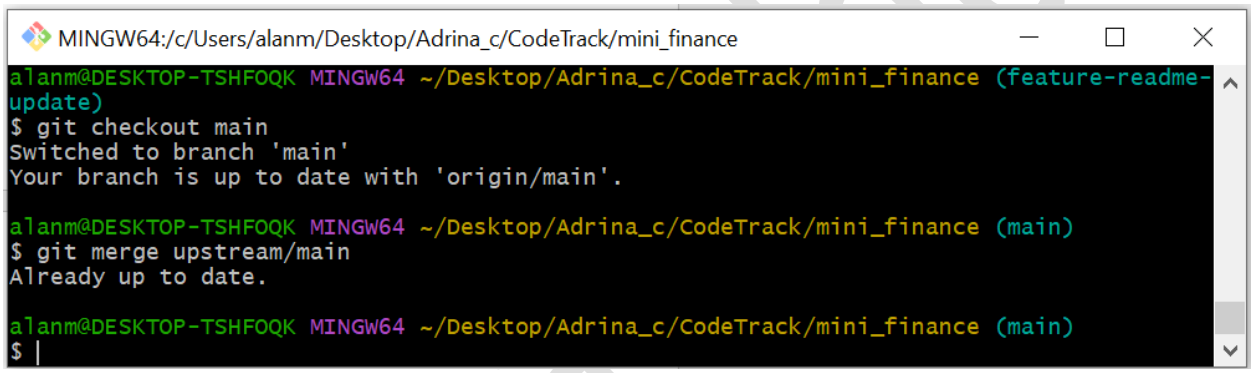


Task 4 — Pull From Upstream & Push to Origin

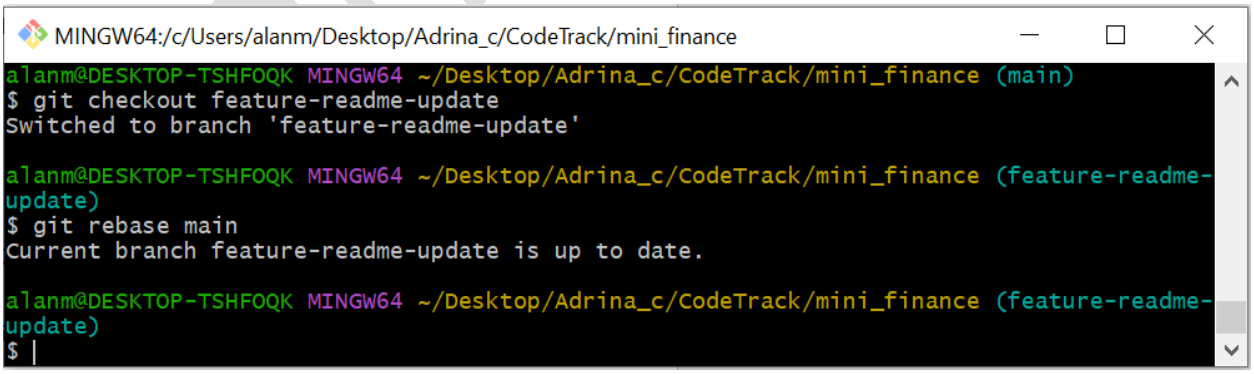
1. Sync changes from upstream’s main:
    1. git fetch upstream



2. git checkout main
3. git merge upstream/main



2. Switch back to your feature branch:
    1. git checkout feature-readme-update
    2. git rebase main # optional but recommended

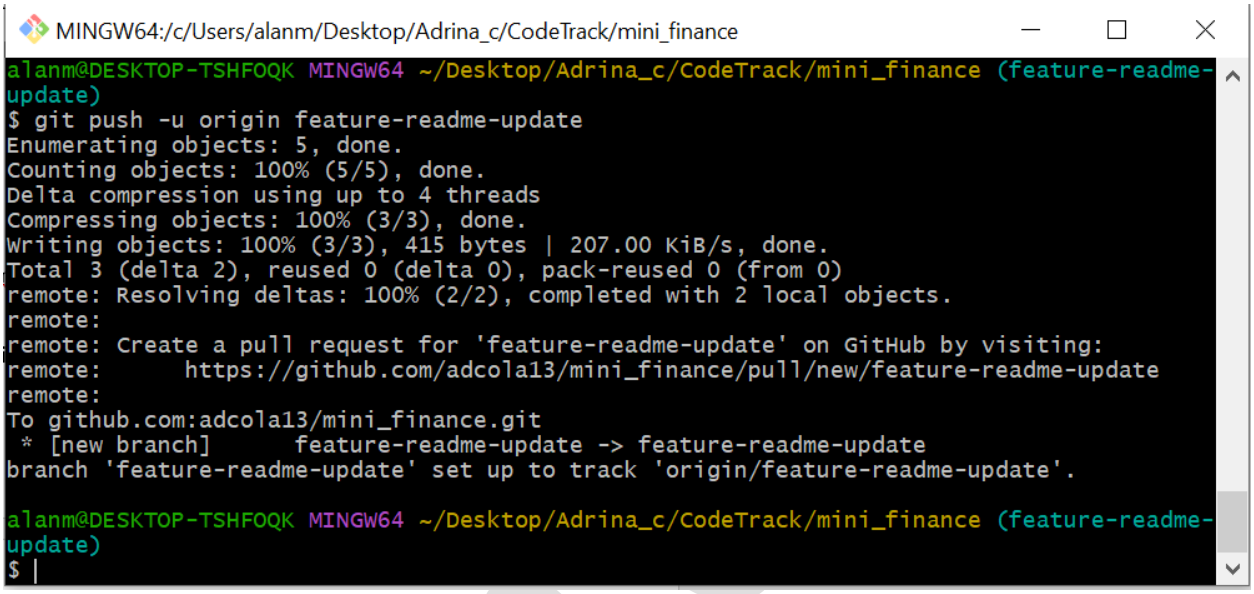


Why did you rebase or merge from upstream/main before pushing?

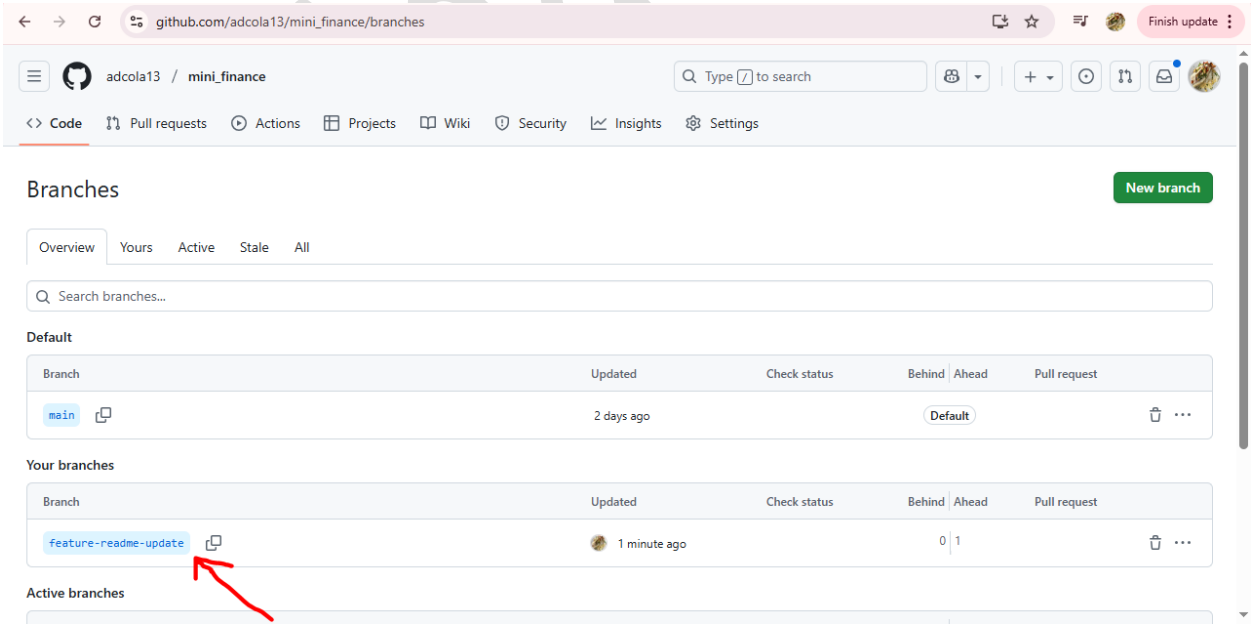
Rebasing or merging from upstream/main (or upstream/master) before pushing is primarily about keeping your work in sync with the latest changes in the main project and ensuring your commits integrate cleanly.

3. Push your branch to your fork:
    1. git push -u origin feature-readme-update

Screenshot D: Confirmation of git push -u origin feature-readme-update.

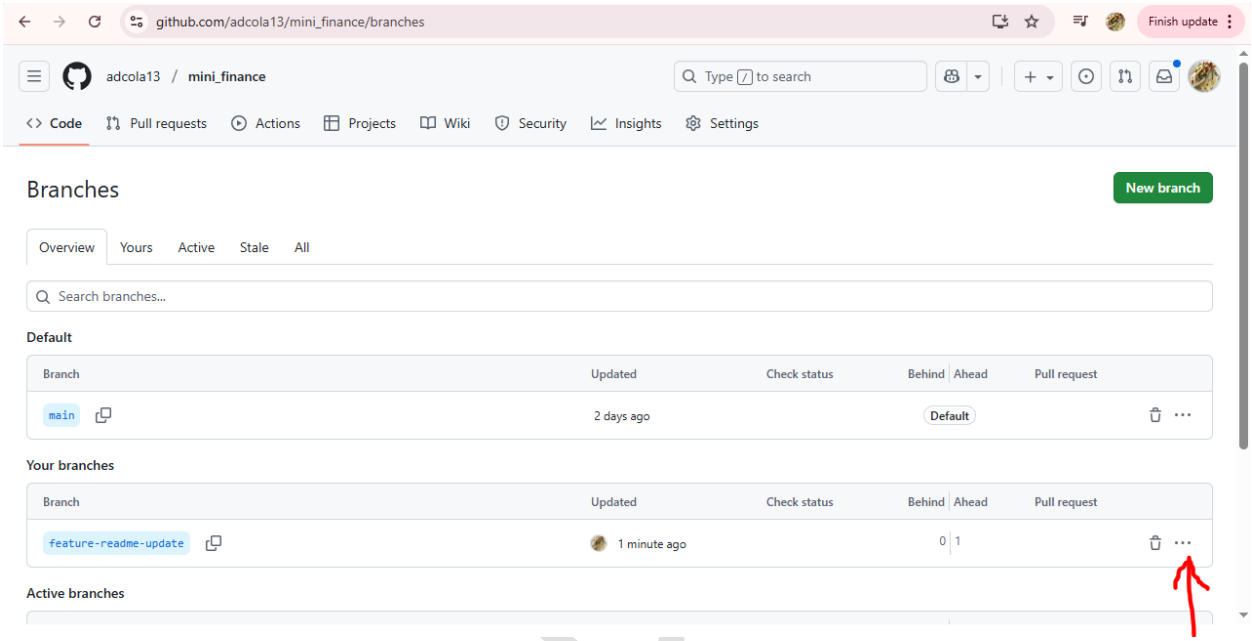


Expected outcome: Your feature branch is available on GitHub under your fork.



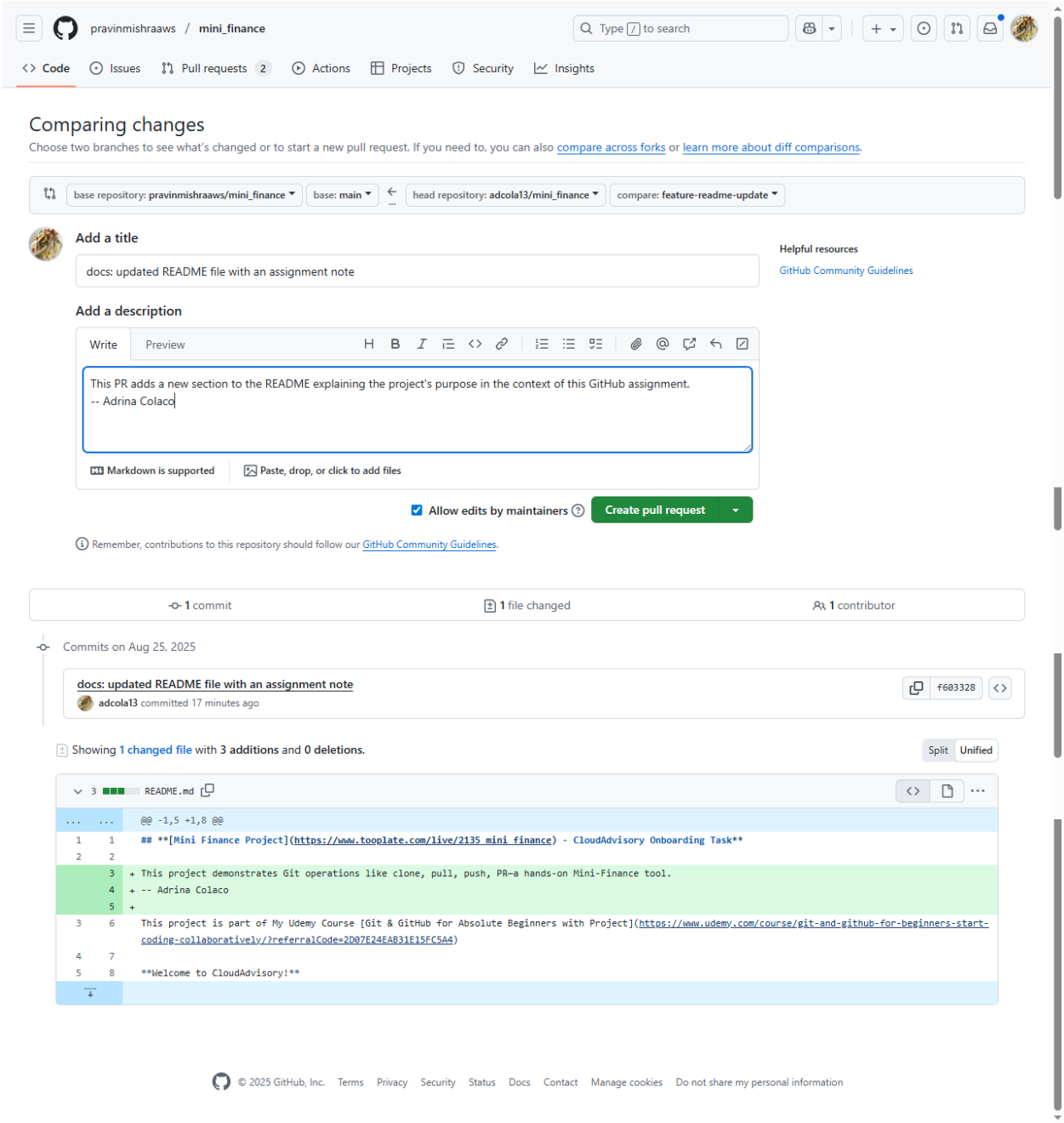
Task 5 — Create a Pull Request

1. Go to your fork on GitHub.
2. Click on 3 dots and then on **New Pull Request**.



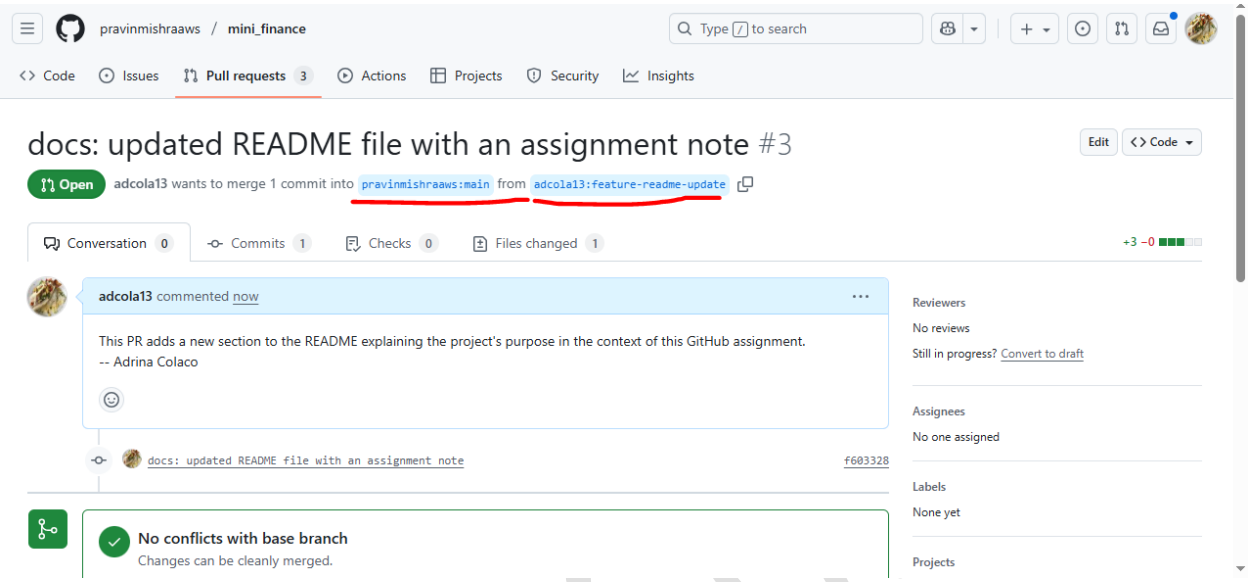
3. Make sure it's targeting pravinmishraaws/mini_finance:main from your feature-readme-update branch.
    1. Title: "docs: update README with assignment note"
    2. In the body, add a short description:

    "This PR adds a new section to the README explaining the project's purpose in the context of this GitHub assignment."



4. Submit the **Pull Request**.

Screenshot E: The GitHub UI showing **open Pull Request** (title and description visible).



Why is creating a Pull Request an important step in team collaboration?

Pull Requests are essential in modern Git workflows because they:

- Ensure high code quality via structured review
- Enable clear communication and knowledge sharing
- Provide transparent documentation
- Support automated checks and mitigate risks
- Streamline collaboration at scale—especially in distributed environments

The End