**Question 1. Set up a Git repository named WebProject. Configure Git with your details and ignore all**

**.DS\_Store and node\_modules/ files. Connect to a remote repository at**

**https://github.com/user/web-project.git.?**

**=>Git Commands for Repository Setup**

**1️. Initialize the Repository**

<<git init WebProject>>

🔹 *This initializes a new Git repository in a directory named WebProject.*

**2. Configure Git**

<<git config user.name "Your Name">>

<<git config user.email [your.email@example.com](mailto:your.email@example.com)>>

🔹 *Sets the username and email for Git commits in this repository.*

**3. Create a .gitignore File**

🔹 **Command to create the file (Linux/macOS):**

<<echo -e ".DS\_Store\nnode\_modules/" > WebProject/.gitignore>>

🔹 **Command to create the file (Windows - PowerShell):**

<<New-Item -Path WebProject\.gitignore -ItemType File>>

<<Add-Content WebProject\.gitignore ".DS\_Store`nnode\_modules/">>

🔹 *This prevents .DS\_Store (macOS metadata files) and node\_modules/ (npm dependencies) from being tracked in Git.*

**4. Add Remote Repository**

<<cd WebProject>>

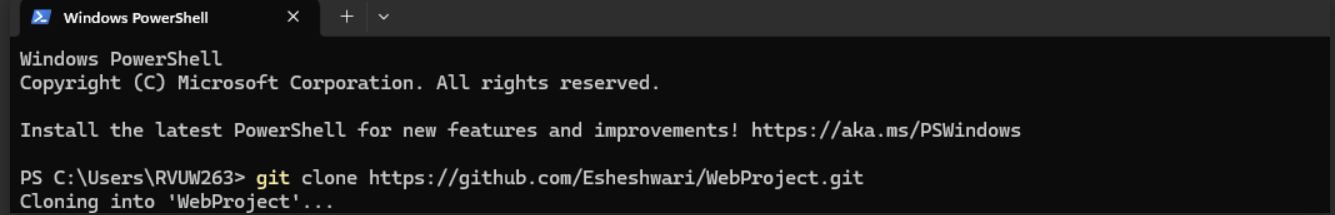
<<git remote add origin https://github.com/user/web-project.git>>

🔹 *Links the local repository to a remote repository named origin.*

**5. Verify Remote Repository**

<<git remote -v>>

🔹 *Checks if the remote repository was added correctly.*



**Questions 2. In the feature-login branch, create a file login.html. Add HTML code for a basic login**

**form and commit. Add CSS in a new file style.css to style the form and commit these**

**changes separately.?**

**=>1. Create and Switch to the Branch**

**<<git checkout -b feature-login>>**

**🔹 *Creates a new branch named feature-login and switches to it.***

**2️. Add HTML File (Login Form)**

**<<echo '<form>Login Form</form>' > login.html >> # Create login.html with basic form**

**<<git add login.html>> # Stage the file for commit**

**<<git commit -m "Add basic login form">> # Commit the HTML file**

**🔹 *Creates login.html with a simple form and commits it separately.***

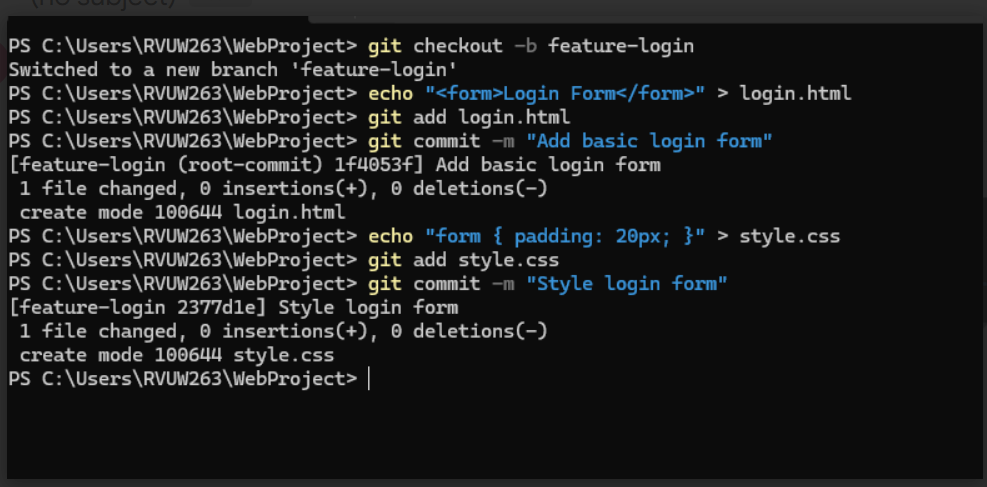
**3️. Add CSS File (Form Styling)**

**<<echo 'form { padding: 20px; }' > style.css>> # Create style.css with basic styling**

**<<git add style.css>> # Stage the file for commit**

**<<git commit -m "Style login form">> # Commit the CSS file**

**🔹 *Creates style.css for styling the form and commits it separately for better version control.***

****

**Question 3. Merge feature-login into the development branch. Ensure that the merge does not fast-**

**forward so that the merge is explicit in the project history.**

**=>1️. Switch to Development Branch**

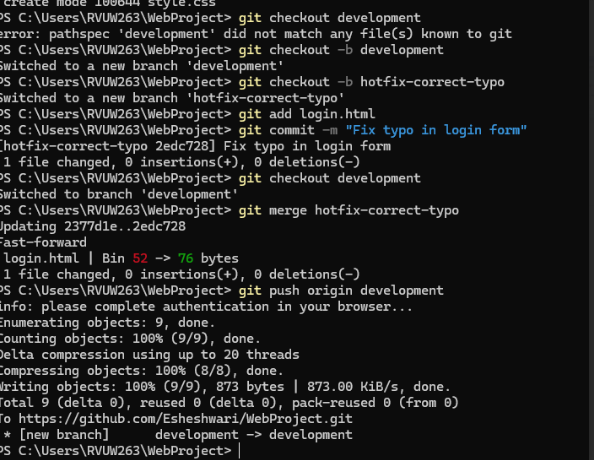
**<<git checkout development>>**

**🔹 *Switches the working directory to the development branch to integrate the new feature.***

**2️. Merge with No Fast-Forward**

**<<git merge --no-ff feature-login>>**

**🔹 *Merges changes from feature-login into development. The --no-ff option ensures a merge commit is created, preserving the feature branch's history.***

****

**Question 4. Create a hotfix branch to correct a typo in login.html. Make the change, commit it, and**

**then merge this branch back into development using a fast-forward merge.**

**=>Git Commands for Creating and Merging a Hotfix**

**1️. Create and Switch to Hotfix Branch**

**<<git checkout -b hotfix>>**

**🔹 *Creates a new branch named hotfix for making urgent corrections and switches to it.***

**2️. Make the Correction and Commit**

**<<sed -i 's/Login Form/Corrected Login Form/g' login.html>> # Correct the typo in login.html**

**<<git commit -am "Fix typo in login form">> # Add and commit the change in one step**

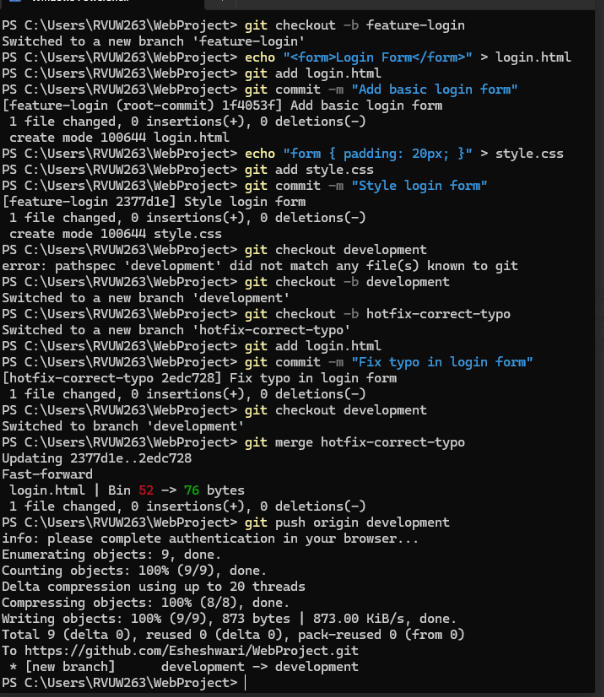
**🔹 *Uses the sed command to replace incorrect text in login.html and commits the fix.***

**3️. Merge Back to Development**

**<<git checkout development >># Switch to the development branch**

**<<git merge hotfix >> # Merge the hotfix changes into development**

**🔹 *A fast-forward merge is used since this is a simple correction and does not require preserving the hotfix branch history separately.***

****

**Question 5. Tag the current state of development as release-1.0 and push all branches and tags to the**

**remote repository. Demonstrate how to check out this tag into a separate directory for**

**deployment testing.?**

**=>1️. Tag the Current State of development as release-1.0**

**<<git checkout development # Ensure you are on the development branch>>**

**<<git tag -a release-1.0 -m "Release version 1.0" # Create an annotated tag>>**

**🔹 *Tags the current state of the development branch as release-1.0 with a message.***

**2️. Push All Branches and Tags to the Remote Repository**

**<<git push --all origin>> # Push all branches to the remote repository**

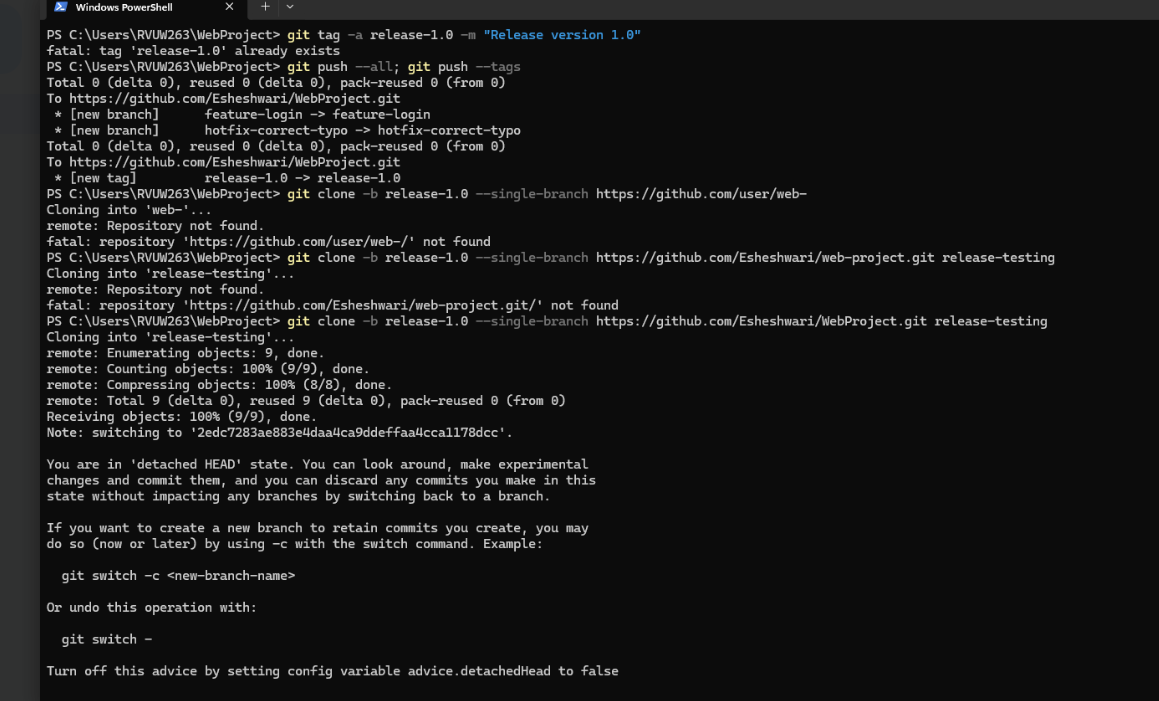
**<<git push –tags>> # Push all tags to the remote repository**

**🔹 *Ensures that both branches and tags are available in the remote repository.***

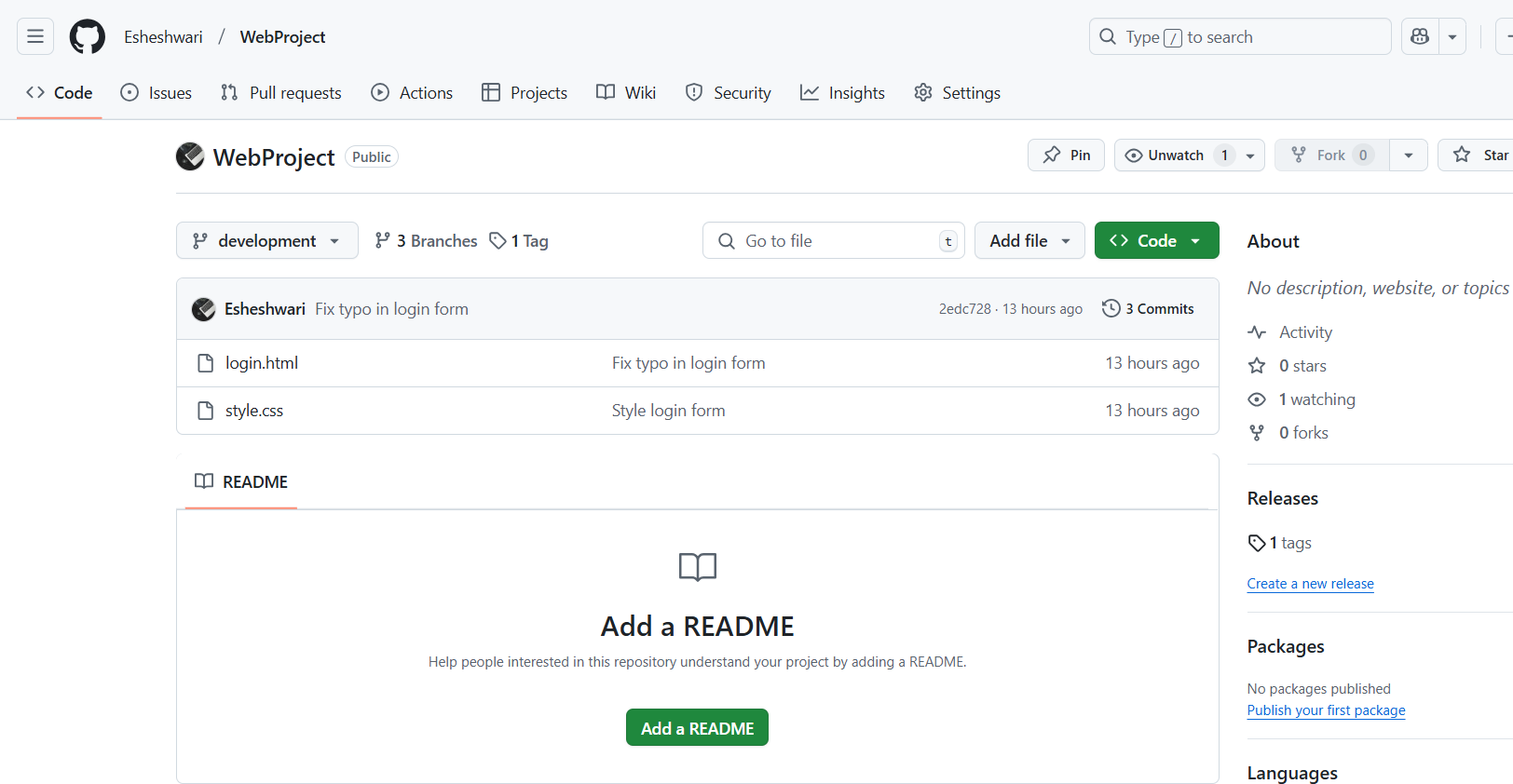
**3️. Clone the Repository into a Separate Directory for Deployment Testing**

**<<git clone --branch release-1.0 https://github.com/user/repository.git deployment-test >>**

**🔹 *Clones only the release-1.0 tag into a new directory named deployment-test for testing.***

****

**Final repository:**

****

**#Set-2 questions:**

**Question 1. Initialize a new Git repository named ApiDevelopment. Configure Git with your username**

**and email. Set up .gitignore to ignore bin/ and obj/ directories. Add a remote repository**

[**https://github.com/yourusername/api-dev.git**](https://github.com/yourusername/api-dev.git)**.?**

**=>1️. Initialize the Repository**

**<<git init ApiDevelopment>>**

**🔹 *Creates a new Git repository in the ApiDevelopment directory.***

**2️. Configure Git**

**<<git config user.name "Your Username">>**

**<<git config user.email** [**your.email@example.com**](mailto:your.email@example.com)**>>**

**🔹 *Sets up Git with your username and email for commit attribution.***

**3️. Set Up .gitignore**

**🔹 Command to create the file (Linux/macOS):**

**<<echo -e "bin/\nobj/" > ApiDevelopment/.gitignore >>**

**🔹 Command to create the file (Windows - PowerShell):**

**<<New-Item -Path ApiDevelopment\.gitignore -ItemType File >>**

**<<Add-Content ApiDevelopment\.gitignore "bin/`nobj/" >>**

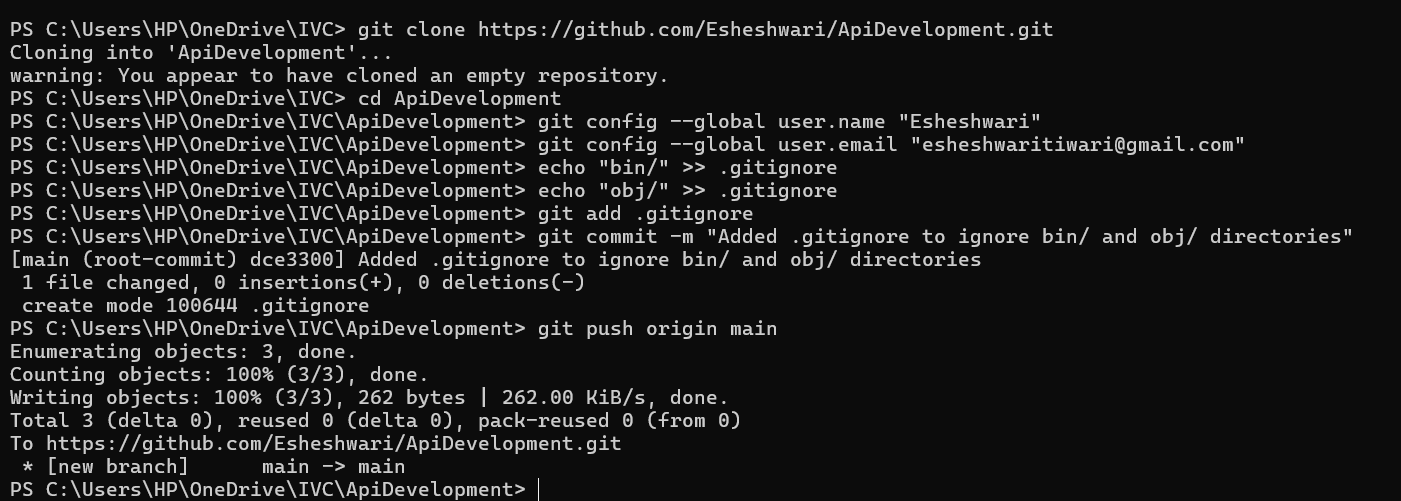
**🔹 *This prevents bin/ and obj/ directories (used for compiled files) from being tracked by Git.***

**4️. Add Remote Repository**

**cd ApiDevelopment**

**<<git remote add origin https://github.com/yourusername/api-dev.git >>**

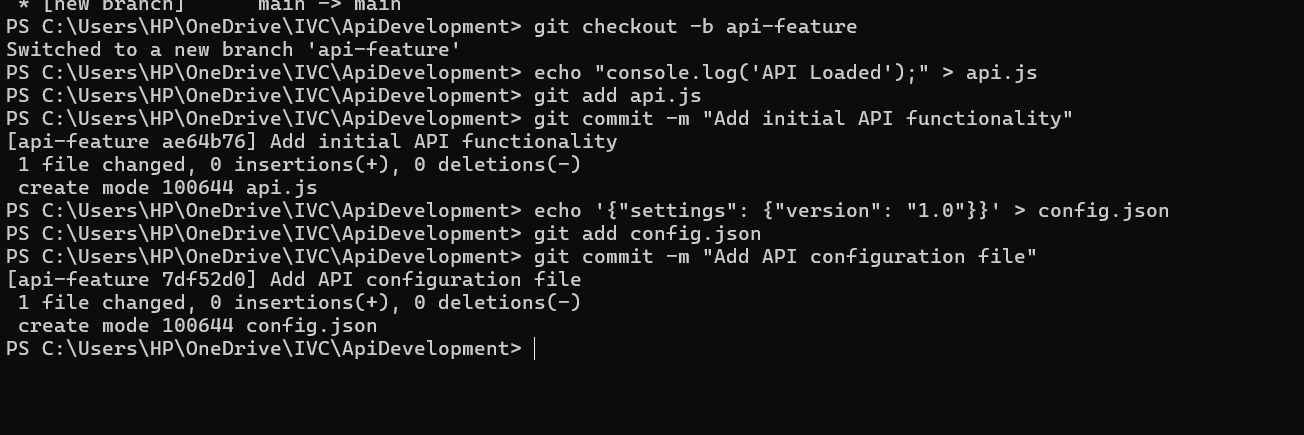
**🔹 *Links the local repository to the remote GitHub repository named origin.***

****

**Question 3. In a branch called api-feature, develop two files: api.js and config.json. Write basic**

**functionality in api.js and configuration settings in config.json. Commit each file**

**separately?**

* **1️. Create and Switch to Branch**
* **<<git checkout -b api-feature>>**
* **🔹 *Creates and switches to a new branch named api-feature, allowing independent development.***
* **2️. Add and Commit api.js**
* **<<echo "console.log('API Loaded');" > api.js >>**
* **<<git add api.js >>**
* **<<git commit -m "Add initial API functionality">>**
* **🔹 *Creates api.js with a basic API loading message, stages it, and commits it separately for clear version history.***
* **3️. Add and Commit config.json**
* **<<echo '{"settings": {"version": "1.0"}}' > config.json >>**
* **<<git add config.json >>**
* **<<git commit -m "Add API configuration file">>**
* **🔹 *Creates config.json with basic configuration settings, stages it, and commits it separately to keep changes modular and manageable.***
* 

**Question 3. Create a pull request from api-feature to main on GitHub. Simulate a review process by**

**commenting on the pull request and making additional changes as requested by a reviewer.**

**=>1️. Push to Remote**

**<<git push origin api-feature>>**

**🔹 *Pushes the api-feature branch to the remote repository, making it available for review.***

**2️. Create Pull Request**

**🔹 Action: Go to GitHub → Navigate to your repository → Click Pull Requests → Click New Pull Request  
🔹 Select: api-feature as the source branch and main as the target branch  
🔹 Submit: Add a title and description, then click Create Pull Request**

**🔹 *Pull requests facilitate code reviews, ensuring quality and collaboration before merging changes into main.***

**3️. Simulate Review and Apply Changes**

**🔹 Example Reviewer Comment: "Please add error handling to api.js."**

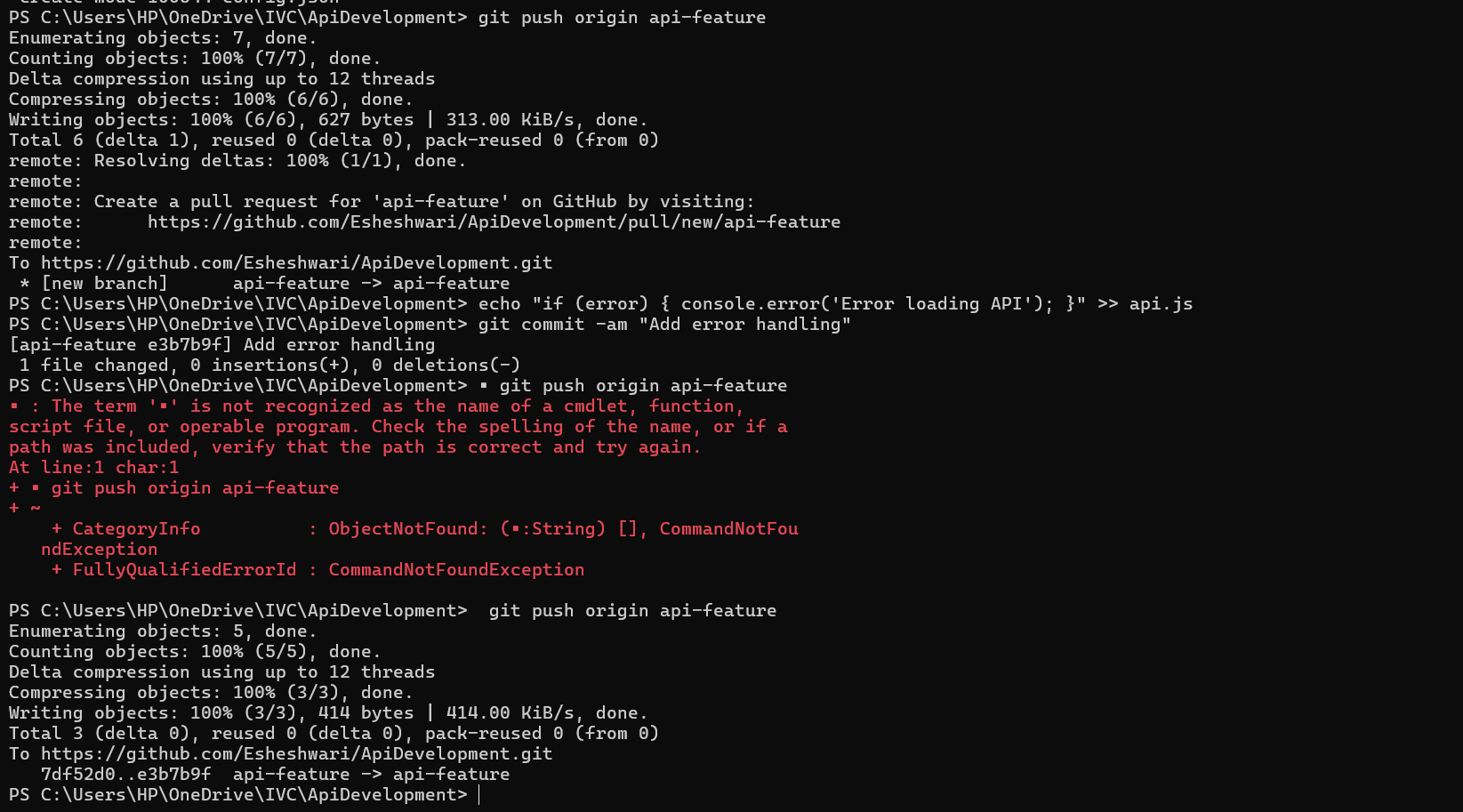
**🔹 Modify api.js to include error handling:**

**<<echo "if (error) { console.error('Error loading API'); }" >> api.js >>**

**<<git commit -am "Add error handling" >>**

**<<git push origin api-feature >>**

**🔹 *Updates api.js with error handling, commits the change, and pushes it to the api-feature branch, automatically updating the pull request.***

****

**Question 4. Implement a continuous integration pipeline using GitHub Actions in the api-feature**

**branch. Write a basic YAML file that runs a script saying echo "CI Build Passed".**

**Commit and push these changes.**

**=>1️. Create GitHub Actions Workflow Directory**

**<<mkdir -p .github/workflows>>**

**🔹 *Ensures the correct directory structure for GitHub Actions workflows.***

**2️. Create CI YAML File (ci.yml)**

**<<echo "name: CI Build >>**

**on: [push]**

**jobs:**

**build:**

**runs-on: ubuntu-latest**

**steps:**

**- uses: actions/checkout@v2**

**- name: Run a one-line script**

**run: echo 'CI Build Passed'" > .github/workflows/ci.yml**

**🔹 *Defines a GitHub Actions pipeline that runs on every push, executing a simple build step.***

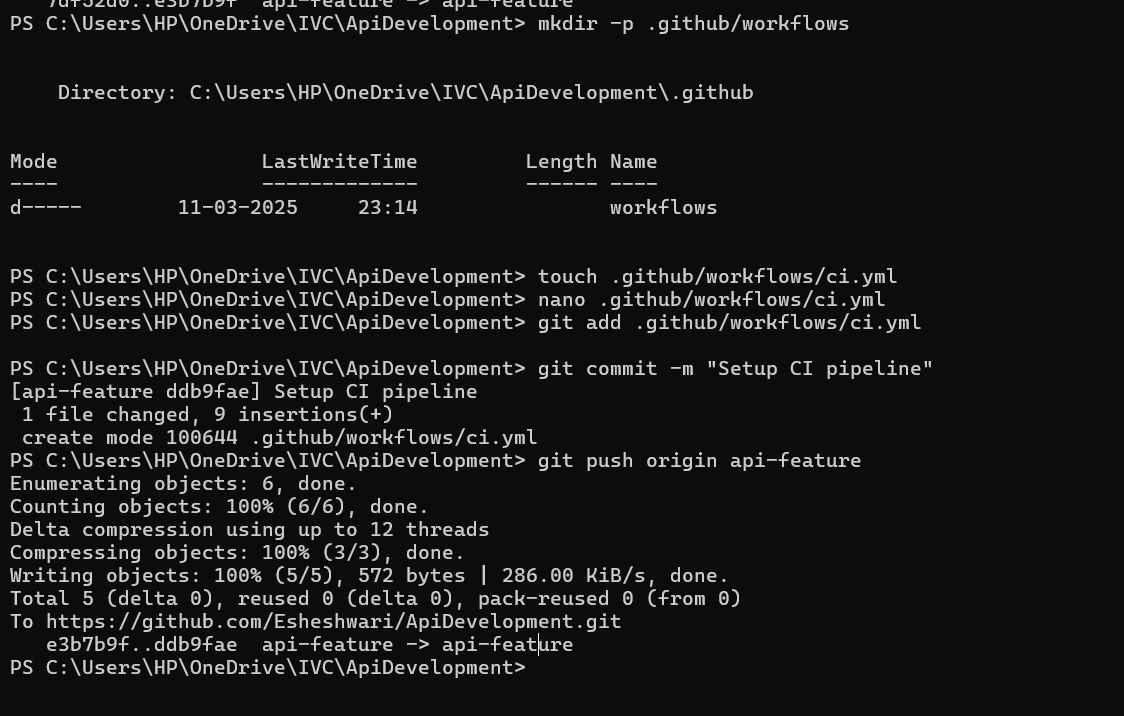
**3️. Commit and Push the Workflow File**

**<<git add .github/workflows/ci.yml >>**

**<<git commit -m "Setup CI pipeline" >>**

**<<git push origin api-feature >>**

**🔹 *Stages, commits, and pushes the CI workflow to the api-feature branch, enabling automated CI checks on GitHub.***

****

**Question 5. After merging api-feature into main, tag the final merge commit as v2.0 and demonstrate**

**pushing the tag to the remote repository. Also, provide instructions on how to revert to the**

**previous version v1.5 if necessary.?**

**=>1️. Merge api-feature into main and Tag as v2.0**

* **Switch to main branch**

**<<git checkout main>>**

* **Merge api-feature into main**

**<<git merge api-feature>>**

* **Tag the final merge commit as v2.0**

**<<git tag -a v2.0 -m "Release version 2.0">>**

* **Push the updated main branch along with the new tag**

**<<git push origin main –tags>>**

**📝 This merges api-feature into main, tags the merge commit as v2.0, and ensures that the tag is available in the remote repository for tracking versioned releases.**

**2️. Revert to a Previous Version (v1.5) if Needed**

* **Checkout the previous version (v1.5)**

**git checkout v1.5**

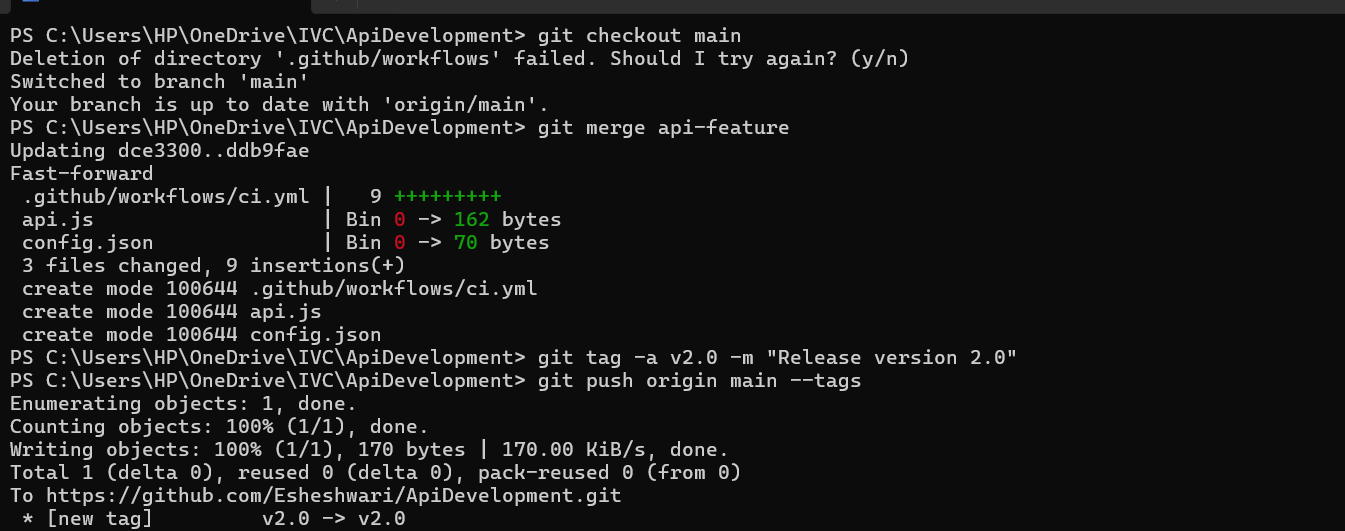
**📝 This allows reverting the working directory to version v1.5, which is useful for debugging or rollback scenarios.**

**3️. Roll Back main to v1.5 (If Required)**

* **If you need to reset main to v1.5, force reset the branch**

**<<git reset --hard v1.5>>**

**<<git push --force origin main>>**

****

**#Set 3:**

**Question 1. Create a new repository named 'SampleProject', configure your user details, create a**

**README file, and push the setup to a remote repository.?**

**1️. Initialize the Repository**

* **Command:**

**<<git init SampleProject>>  
Initializes a new Git repository locally in a folder named SampleProject.**

**2️. Configure User Details**

* **Commands:**

**<<git config user.name "Your Name">>**

**<<git config user.email** [**youremail@example.com**](mailto:youremail@example.com)**>>**

**These commands set up your identity for Git, ensuring commits are attributed correctly.**

**3️. Create and Commit README.md**

* **Commands:**

**<<echo "# SampleProject" > README.md>>**

**<<git add README.md>>**

**<<git commit -m "Initial commit with README">>**

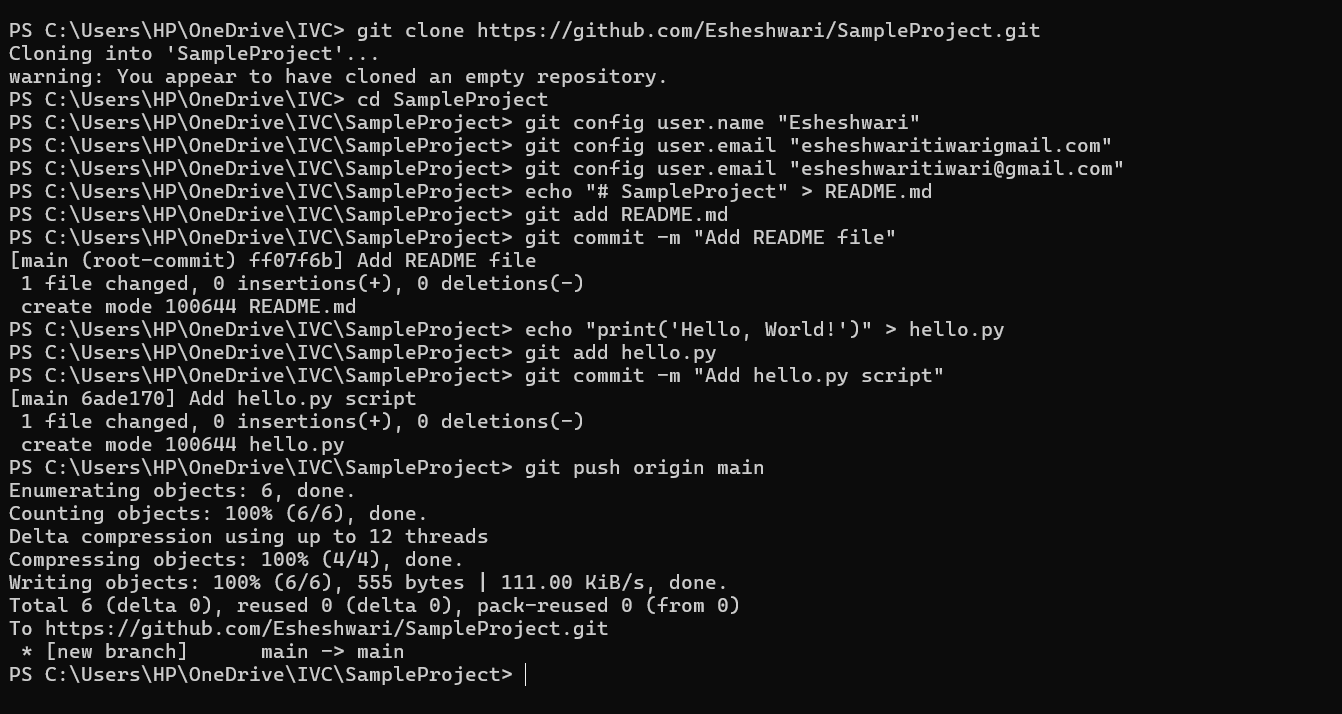
* **Justification:** 
  + **Creates a basic README.md file to document the project.**
  + **Uses git add to stage the file.**
  + **Commits the file to the repository with an initial message.**

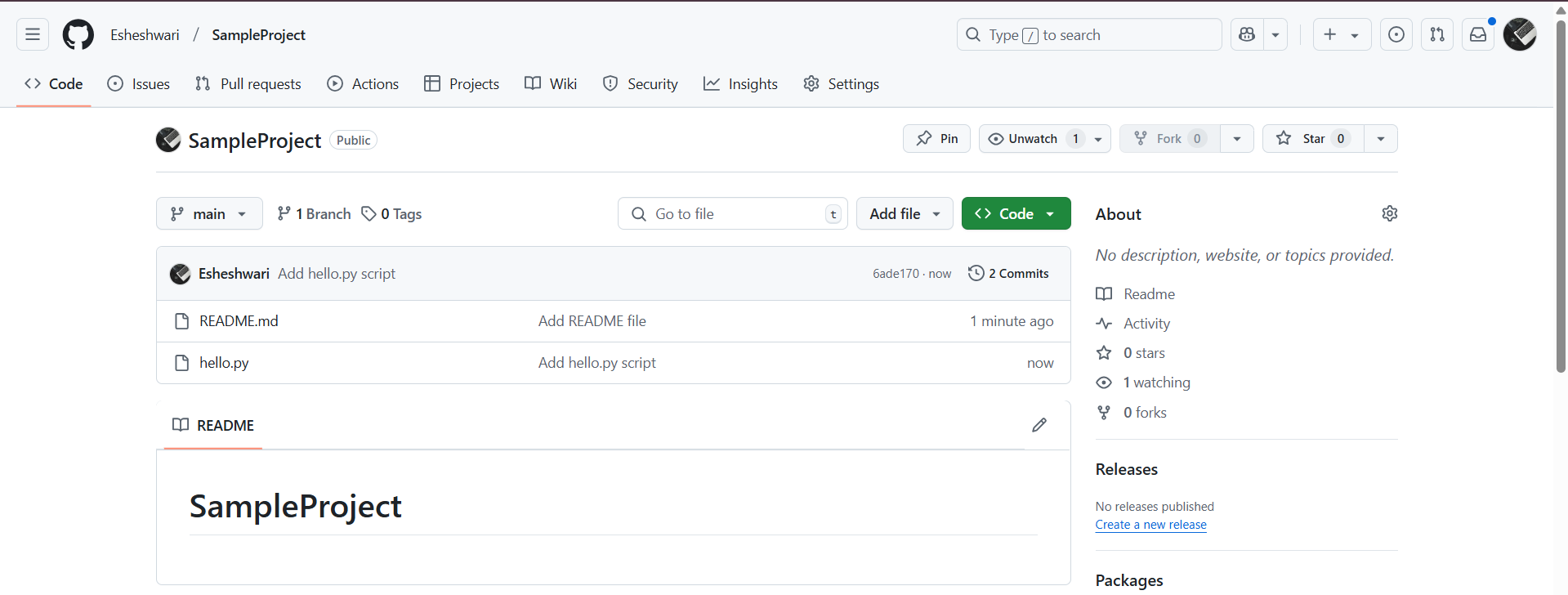
**4️. Add Remote and Push to GitHub**

* **Commands:**

**<<git remote add origin https://github.com/yourusername/SampleProject.git>>**

**<<git push -u origin main>>**

****

****

**Question 2. Create a branch named 'feature-enhancement', make changes to a file named 'app.js',**

**commit these changes, and merge this branch back to the 'main' branch.?**

**=>1️. Create and Switch to a New Branch**

* **Command:**

**<<git checkout -b feature-enhancement>>**

* + **Creates and switches to a new branch named feature-enhancement.**
  + **Helps in isolating feature development from the main branch.**

**2️. Modify app.js and Commit the Changes**

* **Commands:**

**<<echo "console.log('Feature added');" > app.js>>**

**<<git add app.js>>**

**<<git commit -m "Add new feature to app.js">>**

* + **Updates app.js with a new feature.**
  + **Uses git add to stage the file.**
  + **Uses git commit to save the changes with a descriptive message.**

**3️. Merge feature-enhancement into main**

* **Commands:**

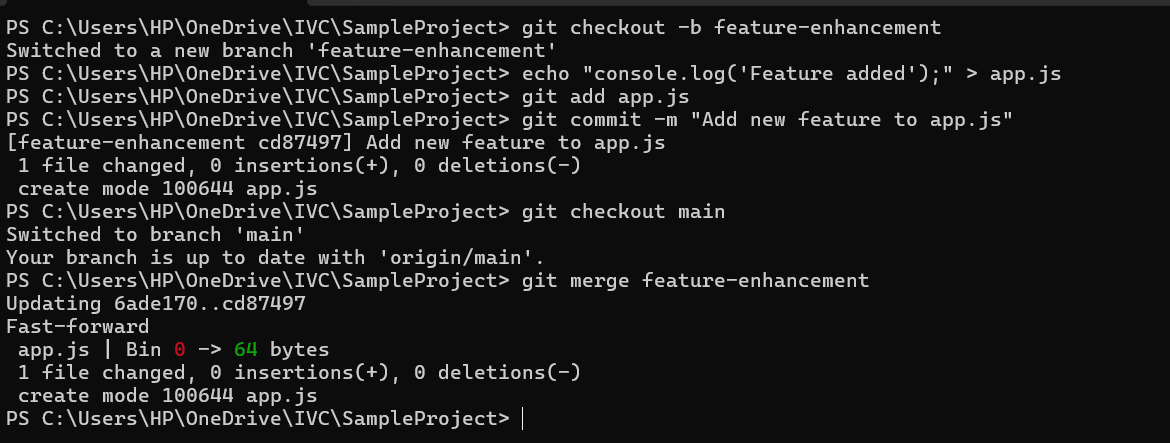
**<<git checkout main>>**

**<<git merge feature-enhancement>>**

* + **Switches back to the main branch.**
  + **Merges the feature-enhancement branch, integrating the new feature into the main project.**

**4️. Push Changes to Remote Repository**

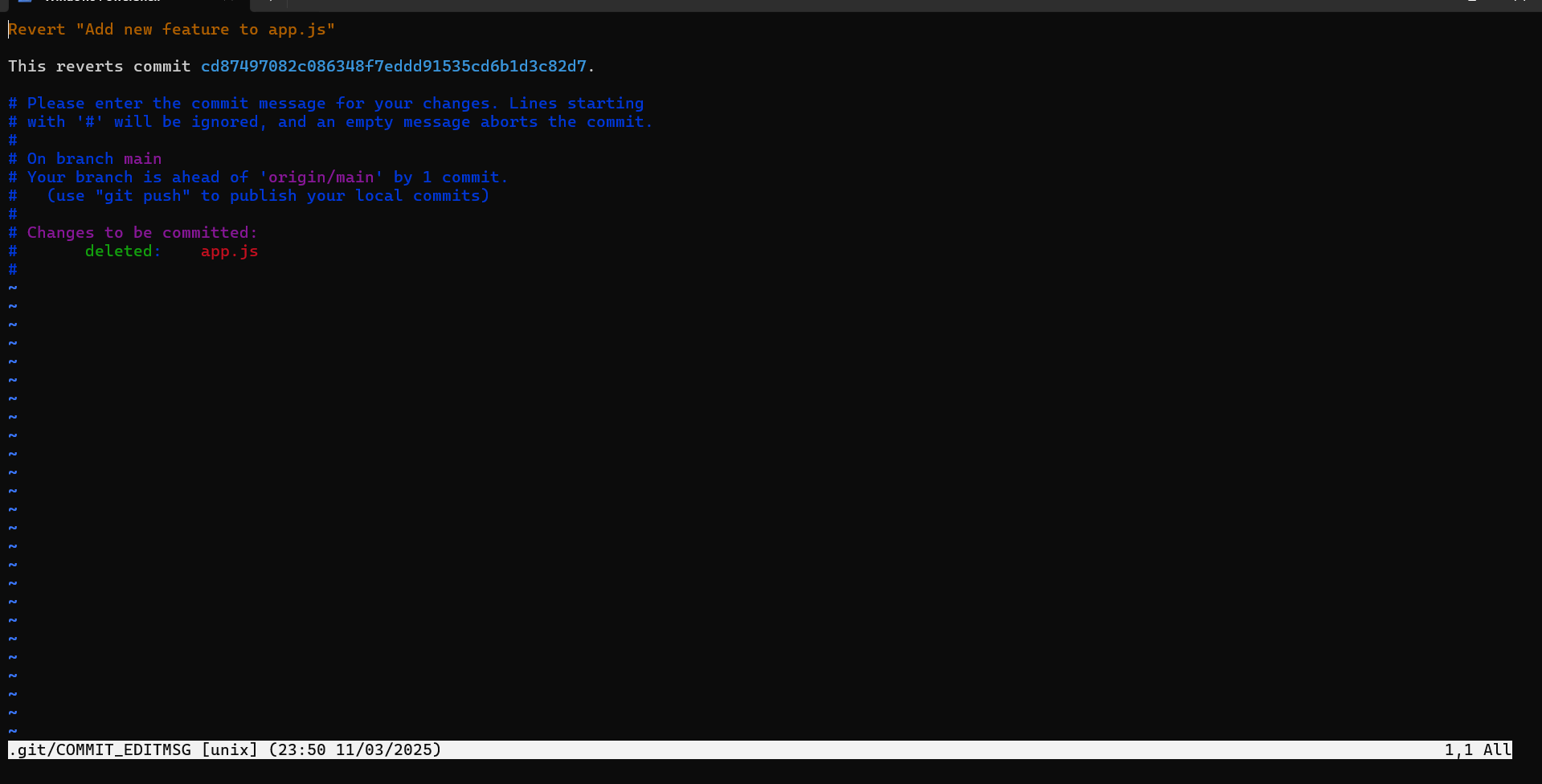
**<<git push origin main>>**

****

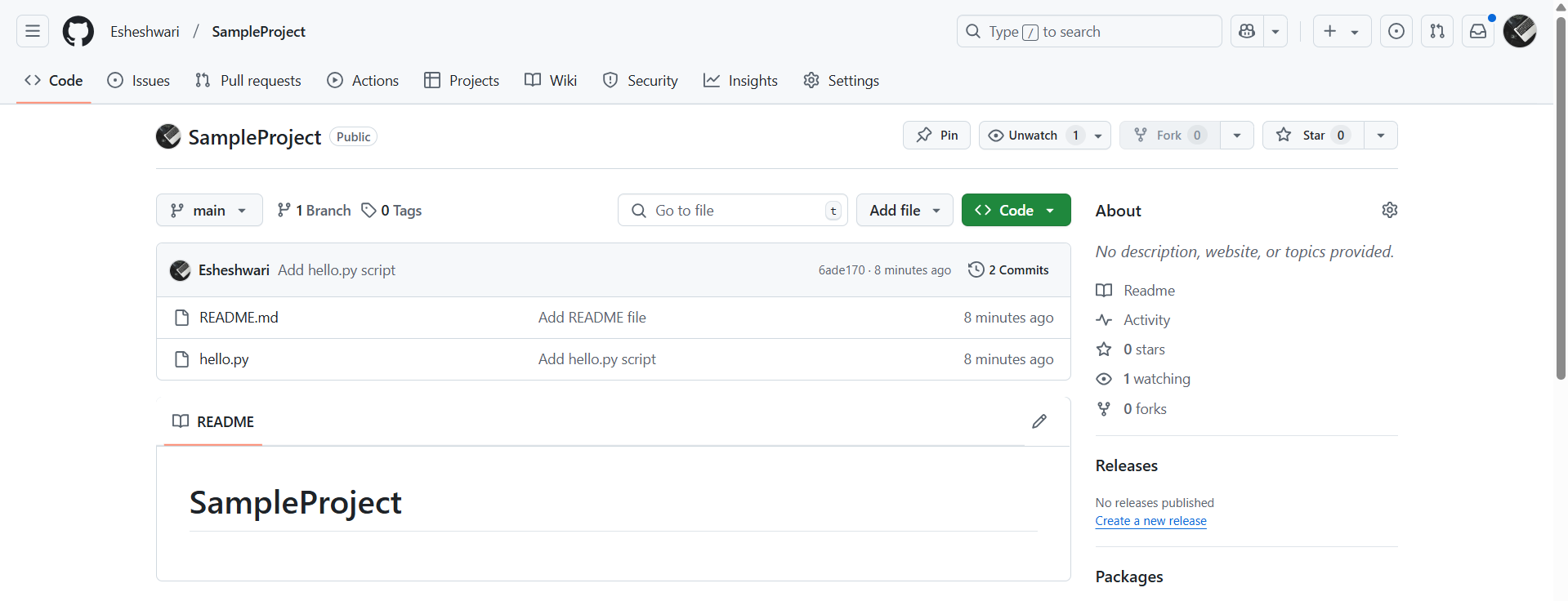
**Question 3. Simulate a scenario where you need to revert the last commit made to the 'main' branch**

**due to an error.**

* **Command: <<git revert HEAD>>**

****

**Final repository:**

****