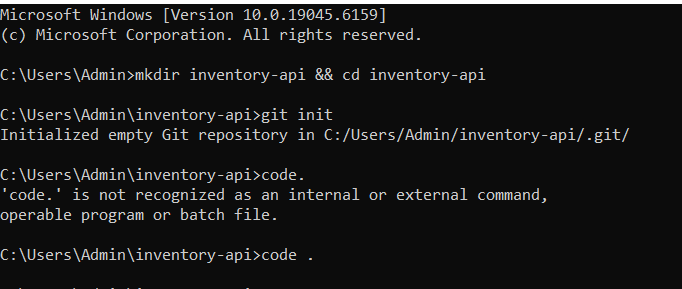
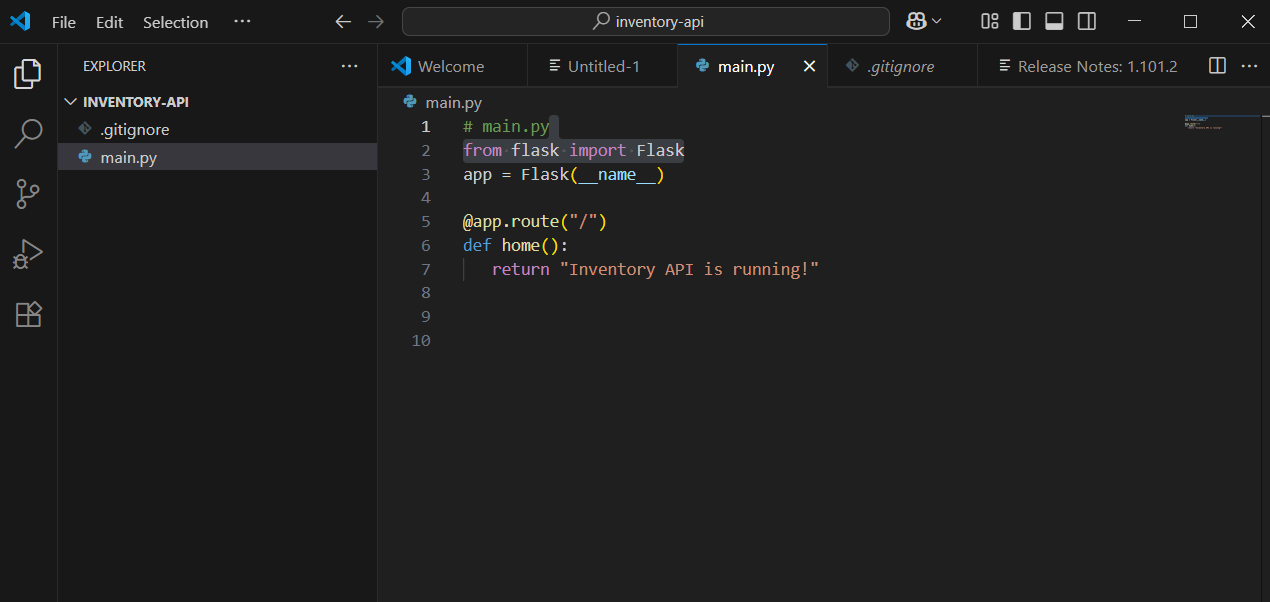
**Real-World Git & GitHub Labs (with VSCode)**  
**Lab 1: Version Control a Real-World Python Microservice  
Scenario:**You’re developing a small backend microservice for a DevOps automation project and want to version-control it properly using Git and GitHub. **Concepts Practiced:**git init**,** add**,** commit**,** push**,** pull**,** .gitignore**,** git stash  
**Tools Used:** Git CLI, GitHub, VSCode  
**Steps:**1. **Create your project folder and open it in VSCode:**  
  
mkdir inventory-api && cd inventory-api   
git init  
code .

* mkdir creates a new folder.
* cd moves into that folder.
* git init starts Git version tracking.
* code . opens the folder in VSCode.

  
  
2. **Create a file main.py and add sample code in VSCode:**  
*# main.py*  
from flask import Flask  
app = Flask(\_\_name\_\_)  
@app.route("/")  
def home():  
   return "Inventory API is running!"



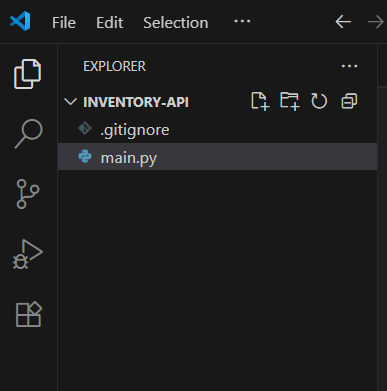
* This is a simple Python app using Flask.

3. **Create a .gitignore:**  
  
echo "\*.pyc" >> .gitignore  
echo "\_\_pycache\_\_/" >> .gitignore

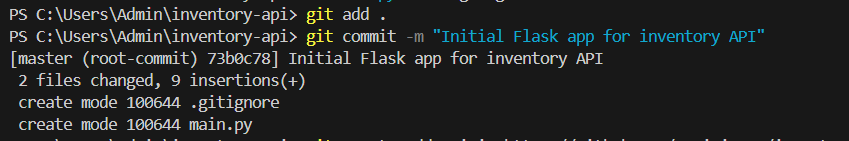
echo "venv/" >> .gitignore

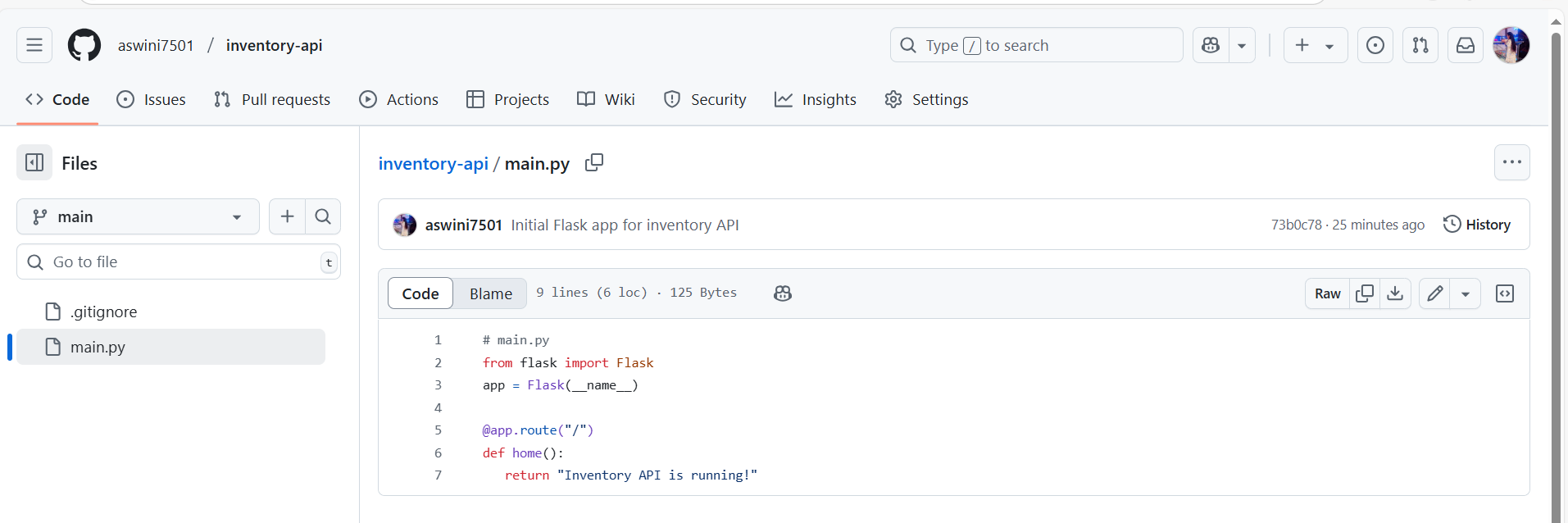
git3.PNG

* git ignore tells Git which files/folders to ignore (like compiled or virtual environment files).



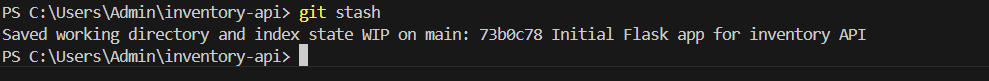
4. **Stage and commit your work:**  
  
git add .  
git commit -m "Initial Flask app for inventory API"

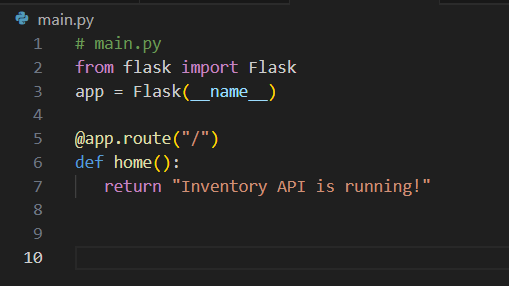


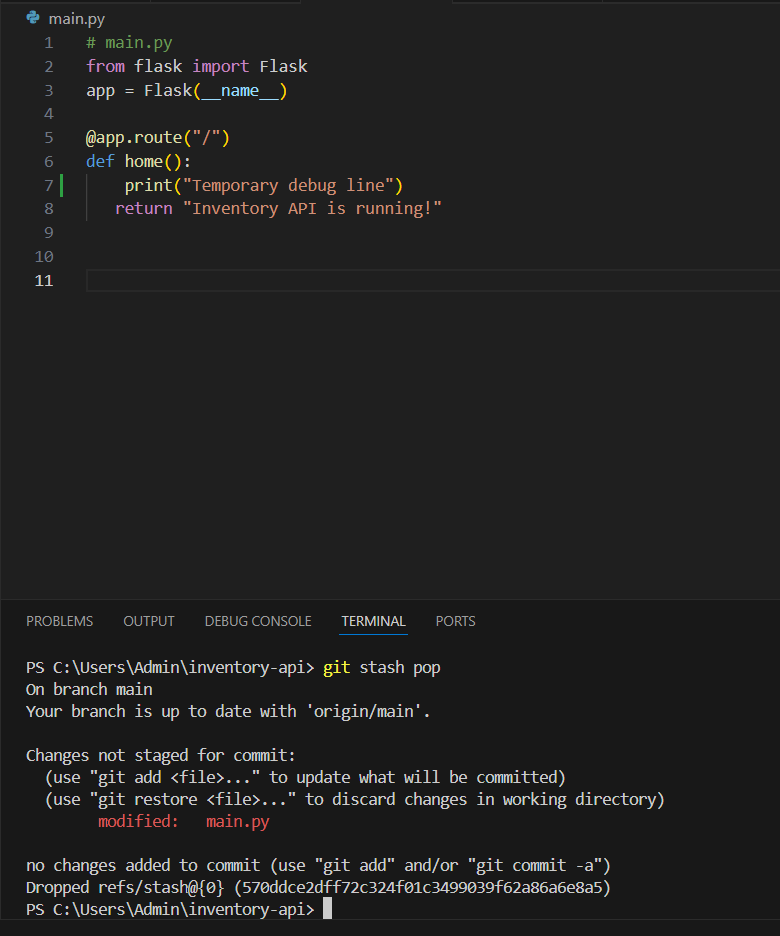
* git add . stages all files for commit.
* git commit -m saves your work with a message.  
    
  5. **Create a GitHub repository (e.g.,** inventory-api**)  
  Then connect it:**  
  git remote add origin <https://github.com/aswini_7501/inventory-api.git>  
  git branch -M main  
  git push -u origin main  
    
  
* Connects your local project to GitHub.
* Pushes your first commit to GitHub.

6. **Use git stash before switching tasks:**  
  
*# Make some uncommitted changes to main.py*  
git stash      *# Save changes*  
git stash pop  *# Restore later*

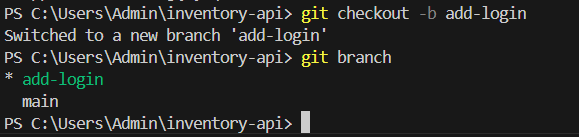
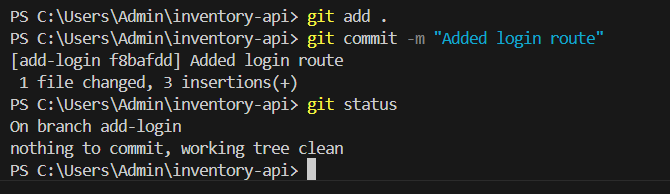
* Use stash when you need to switch tasks but aren’t ready to commit.
* pop restores your saved changes.







**Lab 2: Feature Branching and Merge Conflict Simulation  
  
 Scenario:**You’re working on a login feature while your teammate updates the same file with an environment config. You’ll simulate and resolve a merge conflict using Git + VSCode.  
 **Concepts Practiced:**git branch**,** git checkout**,** git merge**, resolving conflicts  
Tools Used:** Git CLI, VSCode  
**Steps:**1. **Create a new branch for a feature:**  
  
git checkout -b add-login

  
  
  
2. **Edit main.py in VSCode:**  
  
@app.route("/login")  
def login():  
   return "Login Page"  
  
Save and commit:  
  
git add main.py  
git commit -m "Added login route"  
  


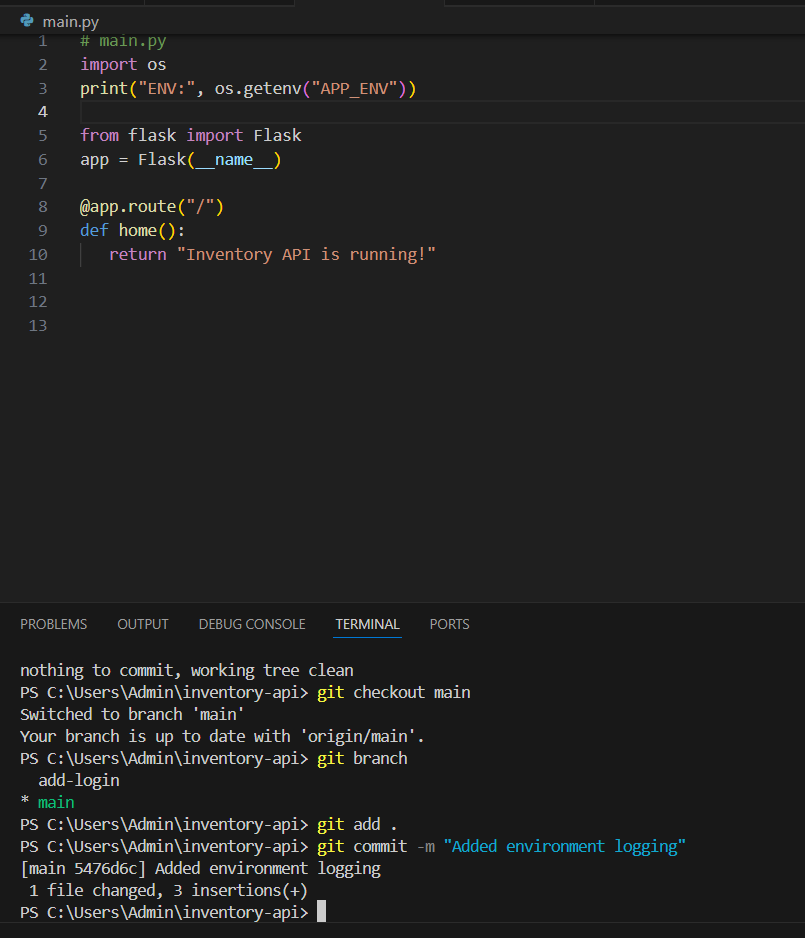
3. **Switch to main and make a conflicting change:**  
  
git checkout main

Add this in the same area ofmain.py**:**

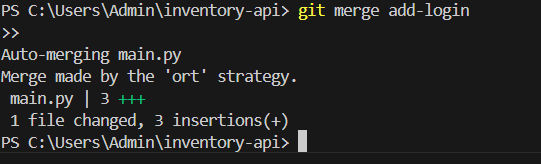
import os  
print("ENV:", os.getenv("APP\_ENV"))

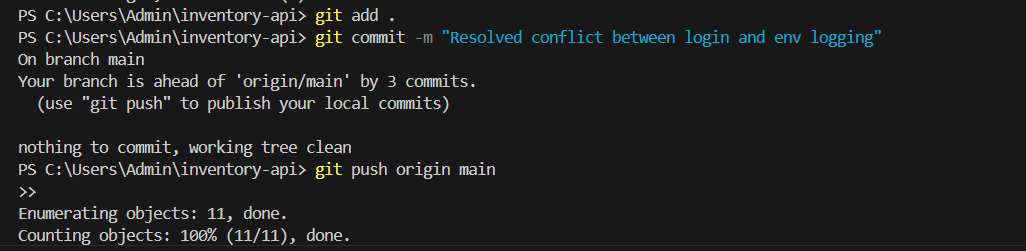
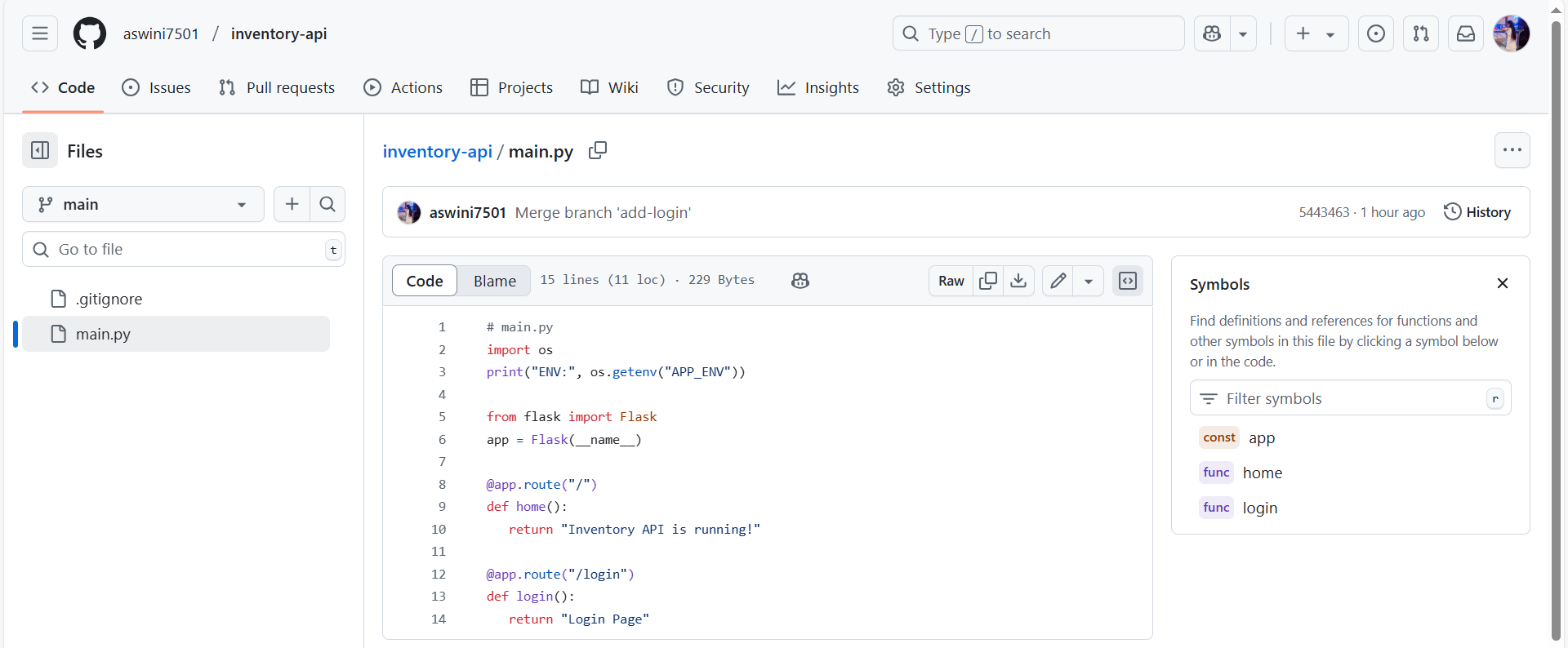
Then commit:

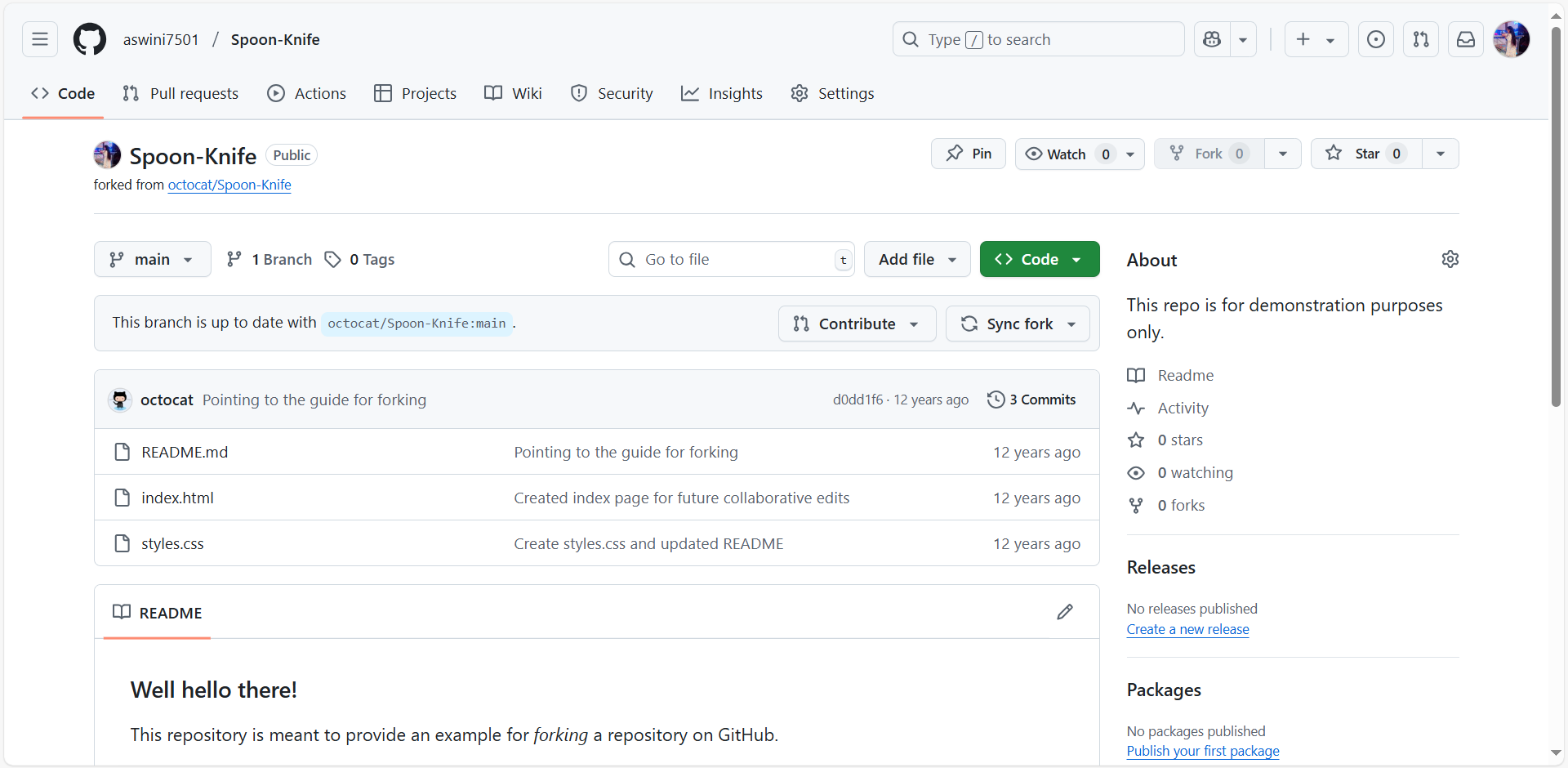
git commit -am "Added environment logging"

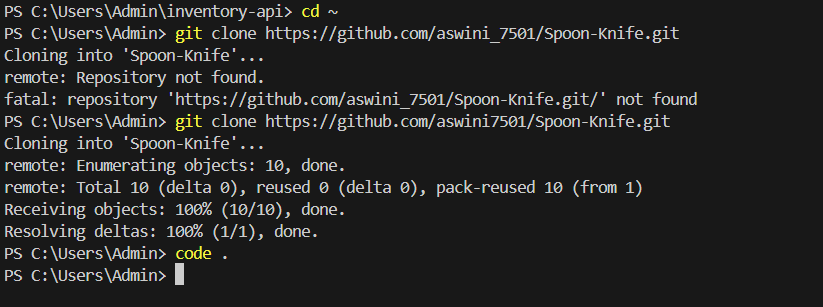
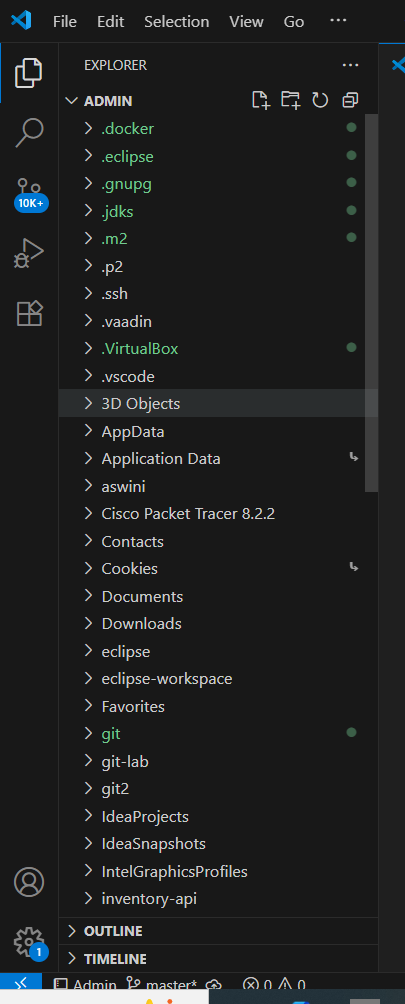


4. **Try to merge the feature branch:**  
  
git merge add-login  
  
You’ll now get a merge conflict!

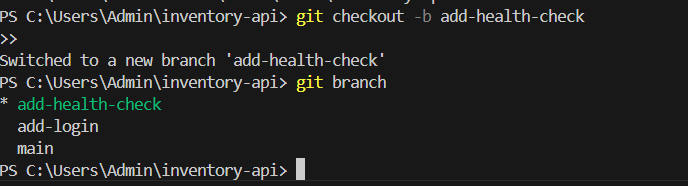
****5. **Open main.py in VSCode  
VSCode highlights the conflict:**  
  
<<<<<<< HEAD  
import os  
print("ENV:", os.getenv("APP\_ENV"))  
=======  
@app.route("/login")  
def login():  
   return "Login Page"  
>>>>>>> add-login  
  
**Manually resolve it to:**  
  
import os  
print("ENV:", os.getenv("APP\_ENV"))  
  
@app.route("/login")  
def login():  
   return "Login Page"

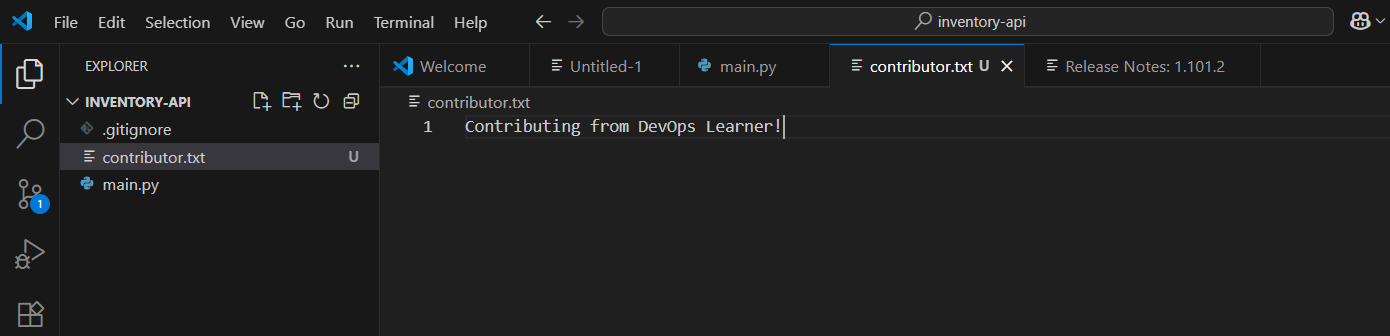
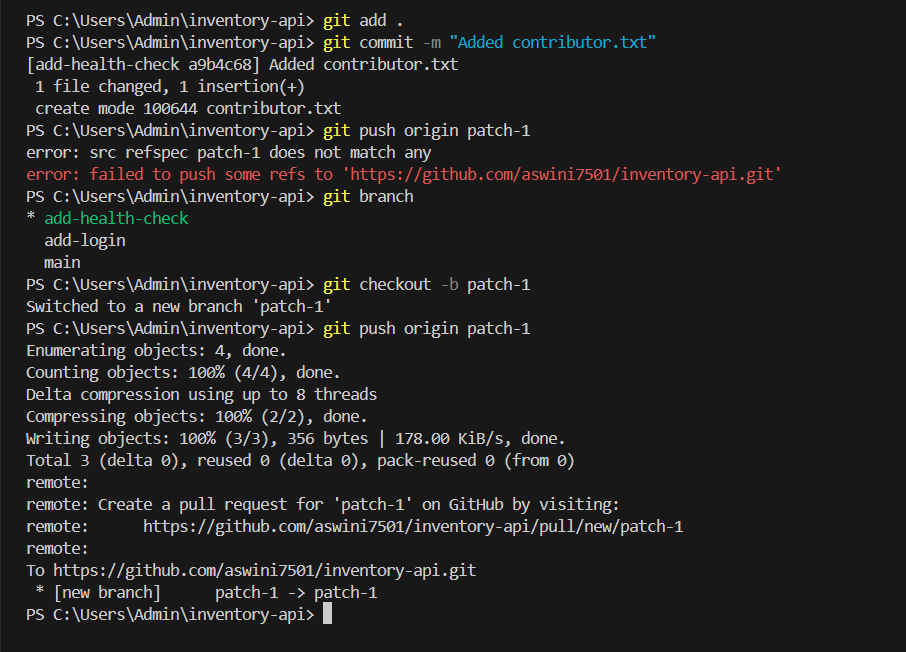
6. **Stage and complete the merge:**  
  
git add main.py  
git commit -m "Resolved conflict between login and env logging"  
  
  
  
  
  
**Lab 3: GitHub Collaboration – Forking & Pull Request Workflow  
  
Scenario:**You’re contributing to an open-source repo (e.g., a DevOps monitoring tool) and follow the standard fork-clone-change-PR cycle.  
 **Concepts Practiced:**Forking, Pull Request, GitHub web interface **Tools Used:** GitHub, Git CLI, VSCode  
**Steps:**1. **Fork the repo**Go to:<https://github.com/octocat/Spoon-Knife>Click Fork → your account.

****

2. **Clone your fork locally:**  
  
git clone <https://github.com/aswini_7501/Spoon-Knife.git>  
cd Spoon-Knife  
code .  
  


3. **Create a new branch and make a change:**  
  
git checkout -b patch-1

  
  
**Add a new file in VSCode:**  
  
echo "Contributing from DevOps Learner!" >> contributor.txt

  
  
**Then:**  
  
git add .  
git commit -m "Added contributor.txt"  
git push origin patch-1  
  
  
4. **Open a Pull Request**Go to your fork’s GitHub page → Click “Compare & Pull Request”  
Add a title, comment, and submit the PR to octocat/Spoon-Knife.  
  
  
  
**Optional Bonus: Visual Git Playground  
  
Try this for interactive branching:**<https://learngitbranching.js.org/>

**Summary of Real-World Applications:**

|  |  |  |
| --- | --- | --- |
| **Lab** | **Scenario** | **Skills Practiced** |
| Lab 1 | Push real code to GitHub | init, commit, remote, push |
| Lab 2 | Resolve merge conflict | branch, merge, conflict fixing |
| Lab 3 | Pull Request contribution | fork, clone, push, PR |