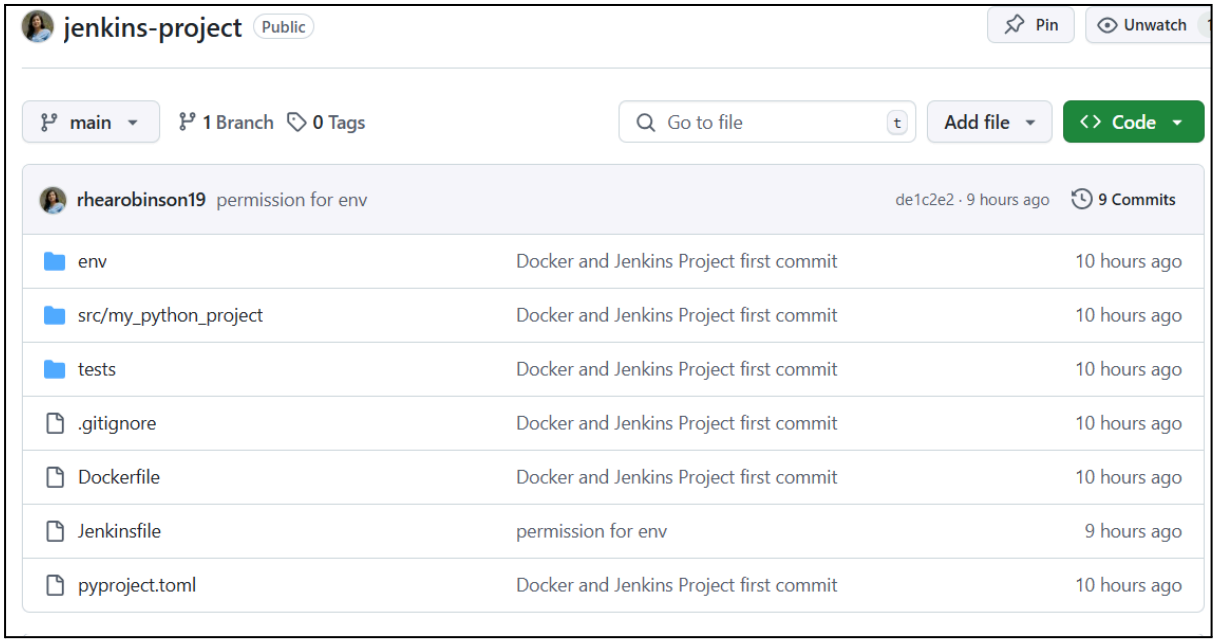
SRE TRAINING (DAY 13) - PYTHON JENKINS CI/CD PIPELINE PROJECT

✅ Project Objective:

- Builds a Python wheel package
- Runs automated unit tests
- Creates and runs a Docker container
- Deploys the application automatically upon successful pipeline execution

STAGE 1 - Project Initialization:

- Created a dedicated project directory for the Python application.
- Initialized a Git repository to track project changes and version control



STAGE 2- Python Project Configuration:

- Created a `pyproject.toml` file to define build system requirements using **hatchling** as the build backend.
- Specified project metadata (name, version, dependencies).

pyproject.toml × Jenkinsfile

my_python_project > pyproject.toml

```toml
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[project]
name = "my_python_project"
version = "0.1.0"
dependencies = []

[project.scripts]
myapp = "my_python_project.main:main"

[tool.hatch.build.targets.wheel]
packages = ["my_python_project"]
```

STAGE 3 - Adding Source Code & Test Cases:

- Implemented the main Python function that outputs a message.
- Wrote unit tests using Python's unittest framework to verify the main function's output

pyproject.toml main.py × test_main.py

my_python_project > src > my_python_project > main.py

```python
def main():
    print("Hello from Jenkins CI/CD Pipeline!")

if __name__ == "__main__":
    main()
```

pyproject.toml main.py test_main.py ×

my_python_project > tests > test_main.py

```python
import unittest
from my_python_project.main import main
import io
import sys

class TestMain(unittest.TestCase):
    def test_main(self):
        captured_output = io.StringIO()
        sys.stdout = captured_output
        main()
        sys.stdout = sys.__stdout__
        self.assertEqual(captured_output.getvalue().strip(), "Hello from Jenkins CI/CD Pipeline!")

if __name__ == "__main__":
    unittest.main()
```

STAGE 4 - Dockerization of the Project:

- Created a Dockerfile to containerize the Python application.

Configured Docker to install the wheel file and run the application using the defined entry point.

pyproject.toml | Dockerfile

my_python_project > Dockerfile

```
FROM python:3.9-slim

WORKDIR /app

COPY dist/*.whl .
RUN pip install *.whl

CMD ["myapp"]
```

STAGE 5 - Jenkins Pipeline Configuration:

- **Created a Jenkinsfile to define pipeline stages:**
  - **Checkout**: Pulled the latest code from the Git repository.
  - **Build Wheel**: Built the Python wheel file.
  - **Test**: Installed pytest and executed automated tests.
  - **Build Docker Image**: Created a Docker image from the Dockerfile.
  - **Deploy:** Stopped any existing Docker container and deployed the new one.

pyproject.toml | Dockerfile | Jenkinsfile | main.py | test_main.py

my_python_project > Jenkinsfile

```
pipeline {
    agent any

    stages {
        stage('Clone Repository') {
            steps {
                // Clean workspace before starting
                cleanWs()

                // Echo the current directory for verification
                sh 'echo "Current directory: $PWD"'

                // Clone the repository into workspace instead of Jenkins home
                sh '''
                    if [ -d "my_python_project" ]; then
                        echo "Directory already exists, removing it"
                        rm -rf my_python_project
                    fi

                    git clone https://github.com/jineshranawatcode/my_python_project.git

                    echo "Repository cloned successfully!"
                    ls -la my_python_project
                '''
            }
        }

        stage('Verify Clone') {
            steps {
                // Verify in the workspace where we have permissions
                sh '''
                    cd my_python_project
                    ls -la
                    git status
                '''
            }
```

Key Issue in Jenkins Pipeline:

- **Problem:** *Externally-managed-environment* error while installing Python packages inside the Jenkins virtual environment.
- **Resolution:** Used the `--break-system-packages` flag as a workaround.

STAGE 6 - Docker Image Build & Deployment:

- Built the Docker image from the application's wheel file.
- Stopped any previously running Docker containers before deploying the new version.

```
root@RheaAlisha:/home/rhearobinson23/jenkins-project# docker images
REPOSITORY                                TAG       IMAGE ID       CREATED        SIZE
my-python-app                             latest    09f1e4853562   10 hours ago   132MB
my_python_project_image                   latest    5085d3e9d19d   11 hours ago   132MB
python-docker-project_web                 latest    e7840c9166eb   2 days ago     139MB
rhearobinson23/python-docker-project_web  v1        e7840c9166eb   2 days ago     139MB
python-docker-project                     latest    05983e13e361   2 days ago     139MB
rhearobinson23/python-docker-project      v1        05983e13e361   2 days ago     139MB
```

```
root@RheaAlisha:/home/rhearobinson23/jenkins-project# docker build -t my-python-project:latest .
2025/02/26 16:08:35 in: []string{}
2025/02/26 16:08:35 Parsed entitlements: []
[+] Building 0.1s (1/1) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 2B
```

STAGE 7 - Setting up pipeline and building in Jenkins

- Installed Required Jenkins Plugins - Pipeline, Git, Docker pipeline
- Created a New Pipeline Project
- Connect to Your Git Repository
- Add Jenkinsfile to Your Project
- Run the Pipeline (Build)