

Automating Python Testing with GitHub Actions

A comprehensive guide to setting up a Continuous Integration (CI) pipeline for Python projects using GitHub Actions to automate testing workflows

Table of Contents

1. [Overview](#)
2. [Step 1: Setting Up CI](#)
3. [Understanding the `.yaml` File](#)
4. [Step 2: Writing Python Code and Tests](#)
5. [Step 3: Viewing CI Results](#)
6. [Successful Test Example](#)
7. [Conclusion](#)
8. [Further Reading](#)

1. Overview

A **Continuous Integration (CI) pipeline** automates the process of integrating code changes frequently, running tests, and verifying the code. GitHub Actions is a tool built into GitHub that allows automating workflows, such as testing code after each commit.

This guide explains how to set up GitHub Actions to run Python tests in the repository. We will simulate a **failed test**, followed by steps to correct it for a **successful test**.

Learn more about CI pipelines:

- [What is Continuous Integration?](#)
- [GitHub Actions Documentation](#)

Understanding YAML Files

A YAML (`.yaml`) file, which stands for "YAML Ain't Markup Language," is a human-readable data serialization format commonly used for configuration files and data exchange between different programming languages. Its syntax is clean and easy to read, utilizing indentation to denote hierarchy instead of braces or brackets, which makes it intuitive for users.

Key Features of YAML

- **Human-Readable:** The format is straightforward, making it accessible for developers and non-developers alike.
- **Hierarchical Structure:** Indentation is used to define relationships and structure, reflecting the nested nature of the data.
- **Data Serialization:** YAML can represent complex data types, including lists, dictionaries, and scalar values.

2. Step 1: Setting Up CI

1. **Navigate to the repository** and click on the **Actions** tab.
2. Select **New Workflow** and click on **Set up a workflow yourself**.
3. Name the workflow `ci.yml` and place it in the `.github/workflows/` directory.
4. Paste the following YAML configuration, which will run tests using Python:

For the Error Test Configuration

File Name: `error_observation.yml`

Add this with the following commit message:

- **Commit Message:** "Adding YAML files for error test script configurations"
- **Detailed Commit:** Added YAML files for error test script configuration.

```
# Name of the workflow
name: Error CI Pipeline

# This workflow is designed to test the test_script_error.py,
# which intentionally contains errors.
# This allows us to verify how the CI/CD pipeline handles test failures.

# Define the events that trigger the workflow
on:
  # Trigger the workflow on pushes to the 'main' branch
  push:
    branches:
      - main
  # Trigger the workflow on pull requests targeting the 'main' branch
  pull_request:
    branches:
      - main

# Define the jobs that will run as part of this workflow
jobs:
  build:
    # Specify the operating system for the job
    runs-on: ubuntu-latest
```

```

# Define the steps to execute in this job
steps:
  # Step 1: Checkout the code from the repository
  - name: Checkout Code
    uses: actions/checkout@v2
    # This action checks out the repository under
    # $GITHUB_WORKSPACE, so the workflow can access it.

  # Step 2: Set up the Python environment
  - name: Set Up Python Environment
    uses: actions/setup-python@v2
    with:
      # Specify the version of Python to use for the environment
      python-version: "3.9" # Replace with the desired Python version

  # Step 3: Install project dependencies
  - name: Install Project Dependencies
    run: |
      # Upgrade pip to ensure we have the
      # latest package management tools
      python -m pip install --upgrade pip
      # Install pytest, a testing framework
      # that we'll use to run our test scripts
      pip install pytest

  # Step 4: Execute tests using pytest
  - name: Run Tests
    run: |
      # Execute the tests defined in the test_script_error.py script.
      # This script is expected to have errors,
      # allowing us to verify the CI's error handling.
      pytest test_script_error.py
      # Run tests in the specified test script

```

Output: This will run all the test_script_error, and since we have have error case, it will display 'Failed'(Red Cross ❌).

For the Successful Test Configuration

File Name: correct_observation.yml

Add this with the following commit message:

- **Commit Message:** "Added YAML files for successful test script configuration."
- **Detailed Commit:** Adding YAML files for successful test script configurations

```

# Name of the workflow
name: Correct CI Pipeline

```

```

# This workflow is designed to run tests on the corrected
# test_script_corrected.py
# It verifies that the code functions as expected without errors.

# Define the events that trigger the workflow
on:
  # Trigger the workflow on pushes to the 'main' branch
  push:
    branches:
      - main
  # Trigger the workflow on pull requests targeting the 'main' branch
  pull_request:
    branches:
      - main

# Define the jobs that will run as part of this workflow
jobs:
  build:
    # Specify the operating system for the job
    runs-on: ubuntu-latest

    # Define the steps to execute in this job
    steps:
      # Step 1: Checkout the code from the repository
      - name: Checkout Code
        uses: actions/checkout@v2
        # This action checks out the repository under
        # $GITHUB_WORKSPACE, so the workflow can access it.


      # Step 2: Set up the Python environment
      - name: Set Up Python Environment
        uses: actions/setup-python@v2
        with:
          # Specify the version of Python to use for the environment
          python-version: "3.9" # Replace with the desired Python version

      # Step 3: Install project dependencies
      - name: Install Project Dependencies
        run: |
          # Upgrade pip to ensure we have the
          # latest package management tools
          python -m pip install --upgrade pip
          # Install pytest, a testing framework
          # that we'll use to run our test scripts
          pip install pytest

      # Step 4: Execute tests using pytest
      - name: Run Tests
        run: |
          # Execute the tests defined in the test_script_error.py script.
          # Run the tests defined in the corrected test_script_corrected.py script.
          # This script is expected to
          # contain passing tests to ensure that

```

```
# the code is functioning correctly.
pytest test_script_corrected.py
# Execute tests in the specified test script
```

Output: This will run all the test_script_corrected, and since we have do not have error case, it will display 'Passed'(Green Tick ).

This configuration will trigger tests each time code is pushed to the `main` branch or when a pull request is created.

This specifies the particular tests we want to run. If we want to execute all tests, we can simply run `pytest` , and it will automatically run all test cases.

File Name: `normal_all_workflow.yml`

Add this with the following commit message:

- **Commit Message:** "Adding YAML files for normal(all) test script configurations"
- **Detailed Commit:** Adding YAML files for normal(all) test script configurations.

```
# Name of the workflow
name: Normal CI Pipeline

# This workflow is designed to test all the test cases.
# This allows us to verify how the
# CI/CD pipeline handles all test cases.

# Define the events that trigger the workflow
on:
  # Trigger the workflow on pushes to the 'main' branch
  push:
    branches:
      - main
  # Trigger the workflow on pull requests targeting the 'main' branch
  pull_request:
    branches:
      - main

# Define the jobs that will run as part of this workflow
jobs:
  build:
    # Specify the operating system for the job
    runs-on: ubuntu-latest

    # Define the steps to execute in this job
    steps:
      # Step 1: Checkout the code from the repository
      - name: Checkout Code
        uses: actions/checkout@v2
```

```


# This action checks out the repository under
# $GITHUB_WORKSPACE, so the workflow can access it.

# Step 2: Set up the Python environment
- name: Set Up Python Environment
  uses: actions/setup-python@v2
  with:
    # Specify the version of Python to use for the environment
    python-version: "3.9" # Replace with the desired Python version

# Step 3: Install project dependencies
- name: Install Project Dependencies
  run: |
    # Upgrade pip to ensure we have the
    # latest package management tools
    python -m pip install --upgrade pip
    # Install pytest, a testing framework
    # that we'll use to run our test scripts
    pip install pytest

# Step 4: Execute tests using pytest
- name: Run Tests
  run: |
    # Execute the tests defined in the test_script_error.py script.
    # This will run all the test cases in the test suite
    pytest

```

Output: This will run all the tests, and since we have one error case, it will display an error for that while showing 'Passed'(Green Tick ) for the rest.

3. Understanding the `ym1` File

The `ym1` file is the configuration file for GitHub Actions. Here's a breakdown of the key components:

1. **name: Error CI Pipeline** or **name: Correct CI Pipeline** :

This sets the name of the workflow, which will appear in the Actions tab of the GitHub repository.

2. **on:** :

This section defines the events that trigger the workflow. In this example:

- `push` will run the workflow when new code is pushed to the `main` branch.
- `pull_request` will trigger the workflow when a pull request is created for the `main` branch.

3. **jobs:** :

This section defines a set of steps to be executed. In this example, we have one job named `build`.

4. `runs-on` :

This specifies the virtual machine environment for the job. Here, `ubuntu-latest` is used, indicating that the job will run on the latest Ubuntu OS.

5. Steps:

- `actions/checkout@v2` :

This step checks out the repository's code so the pipeline can access it.

- `actions/setup-python@v2` :

This step installs Python on the runner. The `python-version` field specifies which version of Python to install.

- `Install dependencies` :

This step installs the required dependencies for the project using the following commands:

```
python -m pip install --upgrade pip
pip install pytest
```

These commands update `pip` (Python's package installer) and install the `pytest` package, which is used for running the tests.

- `Run tests` :

- For `error_observation.yml` tests:

- This step runs the tests in the `test_script_error.py` file using the `pytest` framework.

- For `correct_observation.yml` tests:

- This step runs the tests in the `test_script_corrected.py` file using the `pytest` framework.

4. Step 2: Writing Python Code and Tests

Here, we will create the following Python files:

1. Create `script.py` :

```
# script.py

# Function to add two numbers
def add(a, b):
    # Return the sum of a and b
    return a + b
```

```
# Function to subtract one number from another
def subtract(a, b):
    # Return the result of a minus b
    return a - b
```

This Python script contains functions for adding and subtracting two numbers.

2. Create `test_script_error.py` :

```
# test_script_error.py

# Import the functions to be tested from script.py
from script import add, subtract

# Function to test the add function
def test_add():
    # Check if adding 3 and 2 equals 5
    assert add(3, 2) == 5 # This will pass

# Function to test the subtract function
def test_subtract():
    # Check if subtracting 2 from 5 equals 3
    assert subtract(5, 2) == 4 # This will fail
```

This Python script contains unit tests for the `add` and `subtract` functions from `script.py`.

Add this with commit message: "Adding scripts and test script with an error"

Detailed commit: Added scripts and test script with an error to observe the working of the GitHub Actions workflow.

In this scenario, the `test_subtract()` will fail, causing the CI pipeline to report an error.

5. Step 3: Viewing CI Results

1. Once we push the code, GitHub Actions will automatically run the CI pipeline.
2. To view the results:
 - Go to the **Actions** tab in the GitHub repository.
 - Click on the workflow run to see detailed logs and the status of each step.

6. Successful Test Example

To fix the failing test, modify `test_script_corrected.py` as follows:


```
# test_script_corrected.py

# Import the functions to be tested from script.py
from script import add, subtract

# Function to test the add function
def test_add():
    # Check if adding 3 and 2 equals 5
    assert add(3, 2) == 5 # Passes if the sum is correct

# Function to test the subtract function
def test_subtract():
    # Check if subtracting 2 from 5 equals 3
    assert subtract(5, 2) == 3 # Passes if the result is correct
```

Add this with commit message: "Added corrected test script"

Detailed commit: Added corrected test script to see GitHub Actions workflow.

Push the update, and the CI pipeline will run again, passing the tests this time.

7. Conclusion

we have successfully set up GitHub Actions to run Python tests on each push or pull request. This automated pipeline will help ensure that the code is tested before being merged, improving its quality and reliability.

8. Further Reading:

- [CI/CD Best Practices](#) - Learn more about the best practices in CI/CD and how it integrates into the DevOps workflow.
- [Automated Testing Strategies](#) - Discover various testing strategies for automation using tools like PyTest, Selenium, and others.