

Placement Empowerment Program
Cloud Computing and DevOps Centre

Use Git Hooks for Automation: *Create a Git hook that automatically runs tests before committing code to your repository.*

Name: Monisha J R

Department: CSE

Introduction

This Proof of Concept (PoC) demonstrates the use of Git hooks to automate the process of running tests before committing code to a repository. Specifically, it uses the pre-commit hook, which triggers test execution via pytest for a Python project. The goal is to ensure that only code that passes the specified tests is committed to the version control system, thus maintaining code quality and preventing faulty code from being pushed to the repository.

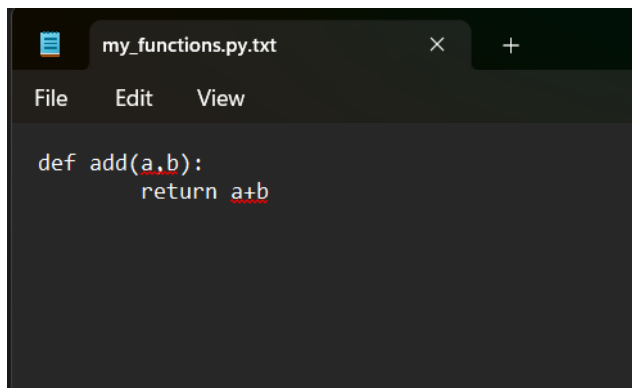
Objectives

- Automate testing by setting up a pre-commit hook to trigger tests before commits.
- Integrate pytest, a powerful testing framework, to run automated tests on the Python code.
- Ensure only error-free, validated code is committed, preventing bugs and improving code stability.
- Learn how to utilize Git hooks for automating routine tasks in the development lifecycle.
- Implement a seamless, automatic testing process that helps developers focus on writing code rather than performing manual checks.

Step by Step Overview

1. Create a sample python file

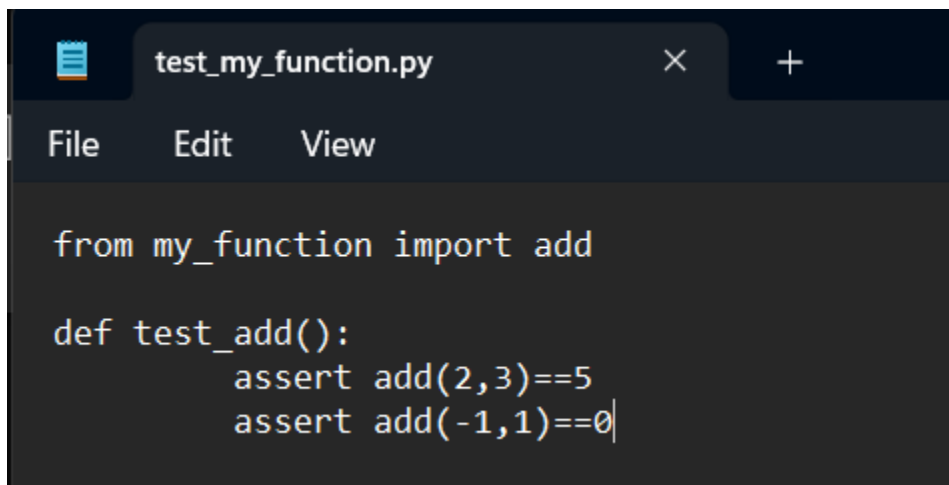
Create a folder in Desktop named my-python-project and Open Notepad Write the Python function to add two numbers and Save the file as *my_function.py* in your folder.

A screenshot of a Notepad window with a dark theme. The title bar shows 'my_functions.py.txt'. The menu bar has 'File', 'Edit', and 'View'. The code content is:

```
def add(a,b):  
    return a+b
```

2. Create test function

Open Notepad again to create the test file and Write the test case code to test the add function and Save the test file as **test_my_function.py** in the same folder where my_function.py is saved.

A screenshot of a Notepad window with a dark theme. The title bar shows 'test_my_function.py'. The menu bar has 'File', 'Edit', and 'View'. The code content is:

```
from my_function import add  
  
def test_add():  
    assert add(2,3)==5  
    assert add(-1,1)==0
```

3. Run the python file

Open GitBash and Navigate to the folder where the files are saved.

```
PS C:\Users\user> cd simplepythoncode
```

Install pytest using pip by running the following command: *pip install pytest*.

Now that we have the function and test cases, let's run pytest to check if everything works.

```
C:\Users\user\OneDrive\Desktop\simplepythoncode>python -m pytest --version
pytest 8.3.4

C:\Users\user\OneDrive\Desktop\simplepythoncode>python -m pytest test_my_function.py -v
===== test session starts =====
platform win32 -- Python 3.13.0, pytest-8.3.4, pluggy-1.5.0 -- C:\Program Files\Python313\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\user\OneDrive\Desktop\simplepythoncode
collected 1 item

test_my_function.py::test_add PASSED [100%]

===== 1 passed in 0.04s =====
C:\Users\user\OneDrive\Desktop\simplepythoncode>
```

4. Run Git

- Run the following command to initialize Git in your project folder : *git init*

```
C:\Users\user\OneDrive\Desktop\simplepythoncode>git init
Initialized empty Git repository in C:/Users/user/OneDrive/Desktop/simplepythoncode/.git/

C:\Users\user\OneDrive\Desktop\simplepythoncode>
```

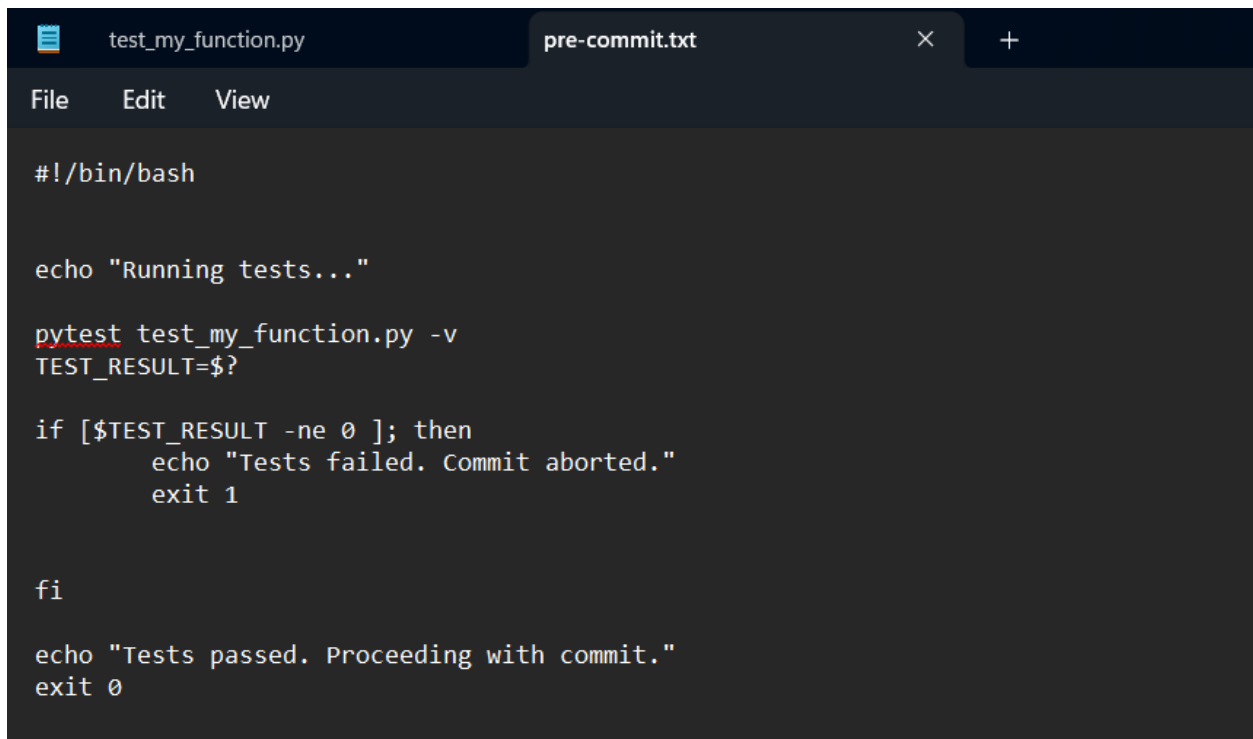
Copy the sample file to a new file called pre-commit (no .sample extension):

cp .git/hooks/pre-commit.sample .git/hooks/pre-commit

```
C:\Users\user\OneDrive\Desktop\simplepythoncode>copy .git\hooks\pre-commit.sample .git\hooks\pre-commit
1 file(s) copied.
```

5. Open the hooks folder

Open Notepad and type the following code and save as pre-commit in Inside the .git folder, open the hooks folder:

A screenshot of a Notepad application window. The title bar shows two tabs: 'test_my_function.py' and 'pre-commit.txt'. The 'pre-commit.txt' tab is active. The menu bar includes 'File', 'Edit', and 'View'. The text area contains a bash script for a pre-commit hook. The script starts with a shebang line, followed by an echo statement, a pytest command, and a conditional check for test results. If tests fail, it echoes a message and exits with code 1. If tests pass, it echoes a message and exits with code 0.

```
#!/bin/bash

echo "Running tests..."

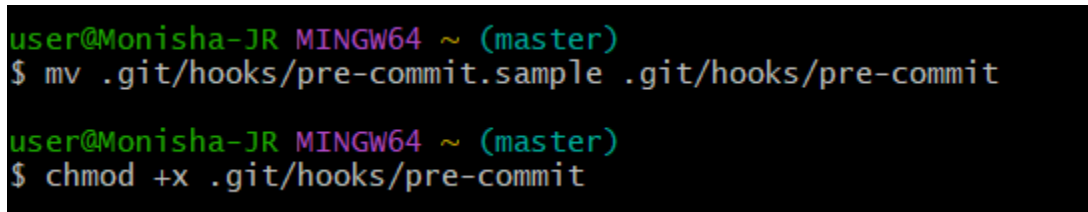
pytest test_my_function.py -v
TEST_RESULT=$?

if [ $TEST_RESULT -ne 0 ]; then
    echo "Tests failed. Commit aborted."
    exit 1
fi

echo "Tests passed. Proceeding with commit."
exit 0
```

6. Once the file is saved, you need to make it executable. You can do this by running the following command :

chmod +x .git/hooks/pre-commit

A screenshot of a terminal window. The prompt is 'user@Monisha-JR MINGW64 ~ (master)'. The user enters the command '\$ mv .git/hooks/pre-commit.sample .git/hooks/pre-commit'. The prompt changes to '\$' and the user enters the command '\$ chmod +x .git/hooks/pre-commit'.

```
user@Monisha-JR MINGW64 ~ (master)
$ mv .git/hooks/pre-commit.sample .git/hooks/pre-commit

user@Monisha-JR MINGW64 ~ (master)
$ chmod +x .git/hooks/pre-commit
```

7. Test

You can test it by staging and committing some changes:

- Stage your changes: ***git add .***
- Commit the changes: ***git commit -m "Testing pre-commit hook"***

```
user@Monisha-JR MINGW64 ~/OneDrive/Desktop/simplepythoncode (master)
$ cd /c/Users/user/OneDrive/Desktop/simplepythoncode

user@Monisha-JR MINGW64 ~/OneDrive/Desktop/simplepythoncode (master)
$ git add .

user@Monisha-JR MINGW64 ~/OneDrive/Desktop/simplepythoncode (master)
$ git commit -m "Testing pre-commit hook"
my_function.py:2: trailing whitespace.
+     return a+b
```

Outcome:

- Implement a pre-commit hook that automatically triggers test execution before every commit, ensuring that only valid code is committed.
- Integrate pytest with Git hooks to run automated unit tests, improving the overall code quality and preventing the introduction of bugs into the repository.
- Learn to automate common tasks using Git hooks, enhancing your understanding of version control automation in the development process.