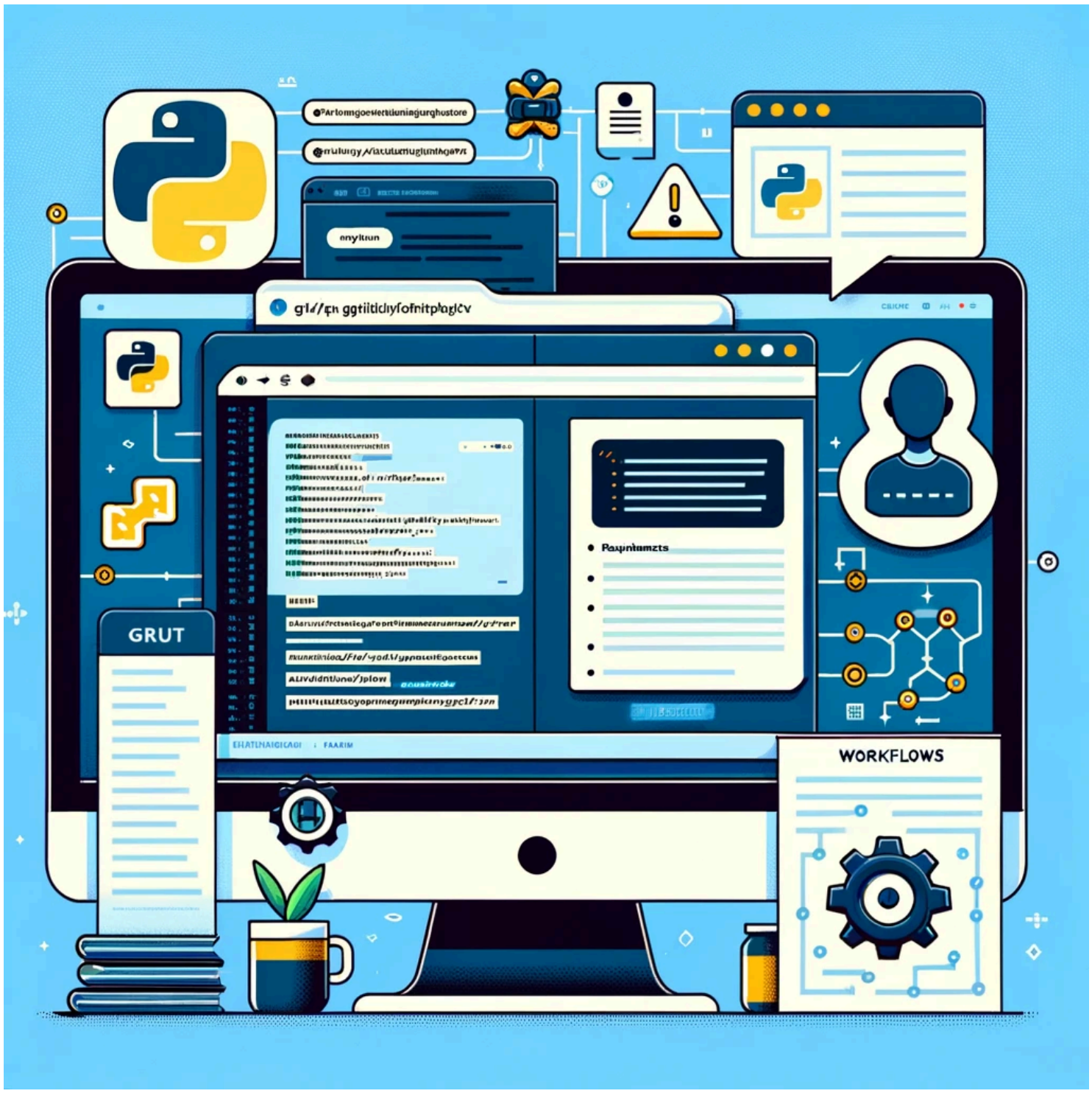
Setting Up CI GitHub Workflows for a New Python Project

← Previous Post

Next Post →

Setting Up CI GitHub Workflows for a New Python Project



GitHub Workflows are a powerful tool for automating software development processes. This post delves into the practical steps of setting up these workflows in a Python project to streamline your development cycle and ensure quality and efficiency. Here, we'll only cover the "CI" portion of "CI/CD". CI, or Continuous Integration, is a useful concept for ensuring that each developer's code is regularly checked against a standard set of tools to ensure a minimum bar of quality for the projct.

Understanding GitHub Actions

Before diving into setting up workflows, it's crucial to understand GitHub Actions, the foundation of GitHub Workflows. Actions are individual tasks that can run commands against your code. You can combine multiple actions into a job and multiple jobs into a workflow, which can be triggered by GitHub events like push, pull requests, or scheduled events.

Initial Setup: Creating Your Workflow File

The first step is to create a `.github/workflows` directory in your Python library's repository. Inside this directory, you'll create your workflow files, which are written in YAML. Let's start with a basic workflow file named `python-ci.yml`.

```yaml
name: Python CI

on:
  push:

jobs:
  ci:

    # Ideally, you'll pin this to a specific version of ubuntu like
    # "ubuntu-22.04", but "ubuntu-latest" will ensure that you'll always pull
    # the latest version that GitHub supports:
    # https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners/about-github-hosted-runners#supported-runners-and-hardware-resources
    runs-on: ubuntu-latest

    steps:

    # Check out the code using v4 of the "actions/checkout" action:
    # https://github.com/actions/checkout
    - name: Checkout
      uses: actions/checkout@v4

    # Set the environment up to use Python. Here, I'm explicitly setting
    # the Python version to 3.10, but you'll probably want to run this against
    # multiple different versions. More on that later on.
    # https://github.com/actions/setup-python
    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: '3.10'

    # Install Python dependencies. This step doesn't run any actions but rather
    # executes a couple of shell commands.
    - name: Install dependencies
      run: |
        python3 -m pip install --upgrade pip
        pip3 install -r requirements.txt

    # Additional steps will be added here
```

This snippet sets up a workflow that triggers whenever code is pushed to the repository. It also specifies a job named `ci` that will be executed whenever this workflow triggers. Here, we're just setting the runner up to execute steps that will be described below.

Integrating Testing Frameworks

Testing is a critical part of maintaining a high-quality Python library. Let's integrate pytest, a popular testing framework for Python.

First, ensure `pytest` is included in your project's dependencies, usually listed in `requirements.txt` or `setup.py`. Then, add this step to run tests.

```
    - name: Test with pytest
      run: pytest
```

When this step executes, it will run the `pytest` utility against your project. If the test run fails, then the workflow will error out, and you should receive an email saying as much. This is how GitHub workflows are intended to work as you'd want to be notified if code that you push to the repository doesn't meet the basic expectations of the project.

Code Quality and Linting

Maintaining code quality is another essential aspect. Tools like flake8 or black can be used for linting and formatting. Here's how you can add `flake8` to your workflow:

```
    - name: Lint with flake8
      run: flake8 . --count  --show-source --statistics
```

This step ensures your code adheres to style guidelines and helps identify potential errors early. Here's an example of how to use `black`:

```
    - name: Check formatting using black
      run: black --check .
```

By checking that a change meets the project's formatting requirements, you will avoid format nitpicks in the code review process. In addition, it will make it easier for newcomers to contribute if the code formatting is standardized. It should also be said that this step requires that the user format their code using black before pushing to the repository in order to pass the check:

```
black .
```

Running against multiple Python versions simultaneously

This section is concerned with running the above steps against multiple different Python versions. To acomplish this, we'll be using the matrix strategy. Here are the edits that you'd make to the job that we've been working on:

```yaml
...

jobs:
  ci:
    strategy:
      max-parallel: 3
      matrix:
        python-version: ['3.10', '3.11', '3.12']

...

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: ${{ matrix.python-version }}

...
```

Using this strategy, GitHub will execute not one but three jobs simultaneously. It will execute your steps against Python versions `3.10`, `3.11`, and `3.12`. If any of these versions fail, then the workflow will fail.

Continuous Integration Best Practices

When setting up GitHub Workflows, follow these best practices:

- **Keep Your Workflows Simple and Focused:** Each workflow should have a clear purpose, whether it's running tests, linting, or deploying.
- **Use Matrix Builds for Testing Across Multiple Environments:** This is useful if your library should be compatible with multiple Python versions or operating systems.
- **Regularly Update Your Actions:** Actions are constantly evolving. Ensure you're using the latest versions for security and feature enhancements.
- **Document Your Workflow:** Make sure that the purpose of each job and step in your workflow is clear to anyone reading your repository.

Final Thoughts

GitHub Workflows bring a robust set of tools to automate and enhance the quality of your Python library development process. For further reading on GitHub Actions and Workflows, visit the GitHub Actions Documentation. Also, exploring community actions on the GitHub Marketplace can provide additional capabilities to your workflows.

Remember, the key to successful automation is continuous improvement. Regularly review and update your workflows to adapt to new challenges and incorporate best practices. With these foundations, you're well-equipped to build high-quality, secure Python projects with confidence.

Share this post

by Michael Renken

February 6, 2024

Share this post