Software Build Automation Tools

Laboratory Exercises 6

Subject
Advanced Git Workflows and GitHub Actions

Author:
PhD Hubert Zarzycki

Introduction

Version control is an essential aspect of modern software development, enabling teams to collaborate efficiently, track changes, and automate key processes. Git, combined with GitHub, provides a powerful ecosystem for managing code and integrating automation.

This laboratory session focuses on advanced Git workflows and GitHub Actions for automating development tasks. Understanding advanced branching strategies, automate release management, and integration of code quality tools into a CI/CD pipeline are essential skills for modern developers.

By the end of this session, you will:

- Understand advanced Git workflows such as *Git Flow* and *Trunk-based development*.
- Automate versioning and release management using *GitHub Actions*.
- Integrate *code quality tools* such as *SonarQube*, *Black*, *and Flake8* into a CI/CD pipeline.
- Improve code quality with *automated linting and formatting*.

1. Setting Up the Development Environment

1.1. Creating a Python Project in PyCharm

- Open *PyCharm* (or MS Visual Code) and create a new project.
- Select *Python as the project type* and ensure a *virtual environment (Virtualenv)* is created.
- Inside the project, create a new directory named `src/` and add a Python file (*`main.py`*).

1.2. Setting Up Git in PyCharm

To track changes and collaborate, the project should be connected to a *Git repository*.

- Open PyCharm and navigate to *VCS > Enable Version Control Integration*.
- Select *Git* and click *OK*.
- Open the terminal in PyCharm and initialize a Git repository:

```
git init
```

- Create a *.gitignore* file and add common Python exclusions:
  - with Windows Command Prompt (in PyCharm)

```
echo. > .gitignore
```

  - or in Linux terminal (Linux/macOS/Git Bash in Windows):

```
touch .gitignore
```

- Then paste the following rules and save the file::

```
__pycache__/
*.pyc
.venv/
.idea/
```

- Once the `.gitignore` have been created we need to add it to the repository:

```
git add .gitignore
git commit -m "Added .gitignore file"
git push origin main  # or other branch name
```

This will cause .gitignore to start working, ignoring the specified files and directories when adding them to your Git repository.

1.3. Connecting the Project to GitHub

- Open GitHub and create a *new repository* (without initializing README or .gitignore).
- Copy the repository URL.
- In PyCharm, open the terminal and run:

```
git remote add origin <repository_url>
git branch -M main
git push -u origin main
```

2. Exploring Advanced Git Workflows

2.1. Git Flow Strategy

Git Flow is a structured branching model for managing feature development and releases.

- Install *Git Flow*:

```
git flow init
```

- Create a new feature branch:

```
git flow feature start new-feature
```

- Develop and commit changes:

```
git add .
git commit -m "Implemented new feature"
```

- Finish the feature:

```
git flow feature finish new-feature
git push origin develop
```

2.2. Trunk-Based Development

Trunk-based development simplifies Git workflows by using *short-lived feature branches* and frequent merging into the `main` branch.

- Create a feature branch:

```
git checkout -b feature-quickfix
```

- Make changes and commit:

```
git commit -am "Quick fix for bug"
```

- Merge directly to `main`:

```
git checkout main
git merge feature-quickfix
git push origin main
```

3. Automating Release Management with GitHub Actions

3.1. Creating a GitHub Actions Workflow

GitHub Actions enables *continuous integration and deployment (CI/CD)* directly in GitHub repositories.

- Inside the project, create the directory `.github/workflows/`.
- Add a new workflow file: `.github/workflows/release.yml`
- Define a workflow to automate versioning:

```
name: Release Automation

permissions:
     contents: write
     pull-requests: write

on:
  push:
    branches:
      - main

jobs:
  release:
```

```
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.9'

      - name: Install dependencies
        run: |
          pip install --upgrade setuptools wheel

      - name: Generate release tag
        id: versioning
        run: |
          VERSION=$(date +'%Y.%m.%d')
          echo "RELEASE_VERSION=$VERSION" >> $GITHUB_ENV

      - name: Create GitHub release
        uses: softprops/action-gh-release@v1
        with:
          tag_name: ${{ env.RELEASE_VERSION }}
          generate_release_notes: true
```

- Commit and push the workflow file:

```
git add .github/workflows/release.yml
git commit -m "Added GitHub Actions for release automation"
git push origin main
```

3.2. Verifying Release Automation

- Go to *GitHub -> Actions* and check if the workflow executed.
- A new *release tag* should be created in the repository.
- View the *release notes* in the GitHub repository under the *Releases* tab.

4. Integrating Code Quality Tools into CI/CD

4.1 Adding Code to the Repository

- If you are working with an existing repository, pull the latest version

```
git pull origin main  # or another branch name
```

- Update a Python file (`main.py`) with code that needs to be checked:

```
def hello_world():
print("Hello, world!")  # Incorrect indentation (missing spaces)
hello_world()
```

This code contains an intentional error (incorrect indentation), which the linter should detect.

4.2. Adding Black and Flake8 for Code Formatting

- Install `black` and `flake8`:

```
pip install black flake8
```

- Add a GitHub Actions workflow for *code linting*:

```
name: Code Quality Check

on:
  pull_request:
    branches:
      - main

jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.9'

      - name: Install dependencies
        run: pip install black flake8

      - name: Run Black
        run: black --check .

      - name: Run Flake8
        run: flake8 .
```

- Commit and push:

```
git add .github/workflows/lint.yml
git commit -m "Added GitHub Actions for code quality checks"
git push origin main
```

4.3 Checking the Execution in GitHub Actions

- Navigate to your repository on GitHub.
- Open the Actions tab.
- Find the workflow named Lint Code.
- Check whether the pipeline executed successfully.
- If errors exist in the code, the workflow will fail – click on the details to see the messages from Flake8 and Black.

4.4 Fixing Code Issues and Retesting

- Modify main.py to follow linting standards:

```
def hello_world():
```

```
    print("Hello, world!")  # Fixed indentation

hello_world()
```

- Add and push the corrected version:

```
git add main.py
git commit -m "Fixed indentation issue in main.py"
git push origin main
```

GitHub Actions will automatically trigger the workflow again, and this time it should pass without errors.

- Create a *pull request* to verify if the workflow runs correctly.

```
git pull origin main  # or another branch name
```

Laboratory Tasks

The aim is to practice version control in PyCharm (or MS Visual Code) while exploring CI/CD integration.

1. For the tasks create an archived project file as well as a report including:
   - Authors' names
   - Repository link
   - Steps performed during the lab and observations.
   - Screenshots and code snippets of key operations (e.g. Git setup, GitHub Actions workflows)
   - Description of any challenges faced and solutions applied.
   - Final state of repository.
   - A brief analysis of the results.

The report should be submitted via the e-learning platform as a DOC or PDF document and archived project files.

2. Implementing Git Flow in a Project
   - Initialize a *Git Flow* workflow.
   - Create a new feature branch and add a Python script.
   - Commit and merge the feature into develop.
   - Push all changes to GitHub.
3. Automating Releases with GitHub Actions
   - Set up a *GitHub Actions workflow* for automated releases.
   - Push a change to the main branch to trigger a release.
   - Verify that a *new release tag* appears in GitHub.
4. Implementing Code Quality Checks
   - Install *Black and Flake8*.
   - Set up *GitHub Actions* to check code formatting.
   - Create a *pull request* and verify that the checks pass.

5. Please submit to the e-learning platform an archived project file as well as a report summarizing the work, including challenges faced and solutions applied. Include a link to the repository.

Literature

Loeliger, J., & McCullough, M. (2012). *Version Control with Git: Powerful tools and techniques for collaborative software development.* O'Reilly Media.
Chacon, S., & Straub, B. (2014). *Pro Git.* Apress.
Rubalcaba, C. (2021). Git for Programmers*: Master Git for Effective Development and Deployment.* Apress.
Okken, B. (2017). Python Testing with pytest. Pragmatic Bookshelf.
Hambleton, A. (2020). Test-Driven Development with Python. O'Reilly Media.
Beck, K. (2003). Test-Driven Development: By Example. Addison-Wesley.GIT Documentation - *https://git-scm.com/doc*
Merkel, D. (2014). *Docker: The Future of Deployment*.
Burns, B. (2018). *Designing Distributed Systems*. O'Reilly Media.
GitHub Learning Lab - *https://lab.github.com/*
Atlassian GIT Tutorials - *https://www.atlassian.com/git/tutorials*
GitHub Actions Documentation - *https://docs.github.com/en/actions*
PyCharm Guide: *https://www.jetbrains.com/pycharm/guide/*
pytest Documentation: https://docs.pytest.org/
PyCharm & Docker Guide: *https://www.jetbrains.com/pycharm/guide/*