Software Build Automation Tools

Laboratory Exercises 4

Subject
Version Control Management in PyCharm Projects for Python

Author:
PhD Hubert Zarzycki

The scope of laboratories

Introduction

Version Control Management in PyCharm Projects for Python enable developers to track changes, collaborate effectively, and ensure the integrity of their codebase. Git, a distributed version control system, has become the industry standard, and platforms like GitHub simplify collaboration by hosting repositories online. PyCharm, a robust Integrated Development Environment (IDE) designed specifically for Python, integrates seamlessly with Git and GitHub, offering tools for version control and facilitating Continuous Integration (CI) and Continuous Deployment (CD).

This document introduces to version control management using Git in PyCharm. It also outlines the steps for creating a Python project in PyCharm, integrating it with a Git repository, and incorporating CI/CD practices. Exercises are provided to solidify the concepts, with the expectation that students will submit a report or source code as their final deliverable.

1. **Version Control Systems (VCS):**

A VCS tracks changes made to files over time. Git, a distributed VCS, allows multiple developers to work on the same project without overwriting each other's changes. Key operations include:

- Commit: Recording changes to the repository.
- Branching: Creating separate lines of development.
- Merging: Integrating changes from different branches.
- Pull and Push: Synchronizing local and remote repositories.

2. **PyCharm and Git Integration:**

PyCharm provides built-in support for Git, allowing developers to manage repositories, commit changes, and resolve conflicts directly within the IDE. Integration with GitHub enhances collaboration, making it easy to share code and manage pull requests.

- Continuous Integration (CI)
  CI is a practice where developers frequently integrate code into a shared repository, followed by automated testing to identify issues early.

- Continuous Deployment (CD):
  CD automates the deployment of changes, ensuring the application is always in a deployable state. Tools like GitHub Actions and Jenkins are commonly used for CI/CD workflows.

3. Setting Up a Python Project in PyCharm

- Install PyCharm
  - Download PyCharm from the official [JetBrains website](https://www.jetbrains.com/pycharm/).
  - Install the Community or Professional edition (Professional includes advanced tools for CI/CD).
- Create a New Project
  - Launch PyCharm and select "New Project."
  - Choose a location for the project and a virtual environment (Python interpreter).
  - Click "Create" to initialize the project.
- Set Up Git in PyCharm
  - Navigate to `File > Settings > Version Control > Git.`
  - Specify the path to the Git executable (install Git if not already done).
  - Test the connection by clicking "Test."
- Initialize a Git Repository
  - Open the project and right-click on the project directory in the Project pane.
  - Select `Git > Enable Version Control Integration.`
  - Choose "Git" from the dropdown menu. This initializes a local Git repository.
- Connect to GitHub
  - Navigate to `File > Settings > Version Control > GitHub`.
  - Authenticate with a GitHub account using a personal access token.
  - Link the project to a remote repository on GitHub:
    - Navigate to `VCS > Git > Push` and select `Define Remote`.
    - Specify the GitHub repository URL.
- Commit and Push Changes
  - Make changes to the project (e.g., create a `main.py` file).
  - Navigate to `VCS > Commit` to stage changes.
  - Add a commit message and click "Commit."
  - Push changes to GitHub using `VCS > Git > Push`.

4. CI/CD Integration in PyCharm

- Configure CI with GitHub Actions
  - Navigate to the GitHub repository and click on the "Actions" tab.
  - Select a Python workflow template and customize it.
  - Save the configuration in a `.github/workflows/main.yml` file in the repository.

Example Workflow Configuration of yaml:

```
name: Python CI
on:
  push:
    branches:
      - main
  pull_request:
    branches:
```

```
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: '3.9'
    - name: Install dependencies
      run: |
        pip install -r requirements.txt
    - name: Run tests
      run: |
        pytest
```

- Automate Deployment
  - Configure a deployment workflow in GitHub Actions or a CI/CD tool like Jenkins.
  - Specify deployment targets (e.g., Heroku, AWS).
- Monitor Pipelines
  - PyCharm provides integration with CI tools, allowing developers to monitor pipelines directly within the IDE.
  - Install the "Cloud Code" plugin in PyCharm for advanced CI/CD management.

5. Example Git Operations in PyCharm

- Creating a Branch:
  - Navigate to `VCS > Git > Branches`.
  - Click "New Branch" and provide a name.
- Switching Branches:
  - Open the Branches menu and select a branch to switch to.
- Merging Branches:
  - Navigate to `VCS > Git > Merge Changes`.
  - Select the source and target branches.
- Creating a Pull Request (via GitHub):
  - Push the branch to GitHub.
  - Open the repository on GitHub and click "New Pull Request."

Laboratory Tasks

The aim is to practice version control in PyCharm while exploring CI/CD integration.

1. For the tasks create a report including:
   - Authors' names
   - Repository link
   - Steps performed during the lab and observations.

- Screenshots and code snippets of key operations (checkout, commit, update, merge, create a branch, etc.).
- Description of any challenges faced and solutions applied.
- Final state of the repository.

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

2. Create a new Python project in PyCharm.
3. Set up a Git repository and connect it to GitHub.
4. Make changes to the project (e.g., create a `main.py` file). Than Commit and Push Changes.
5. Implement Git Workflows
   - Create a new branch and make changes (e.g., add a new Python file).
   - Switch branches.
   - Commit the changes and merge them into the main branch.
   - Push a branch to GitHub. Creating a Pull Request
6. Configure CI/CD
   - Create a `.github/workflows/main.yml` file for GitHub Actions.
   - Implement a CI pipeline that installs dependencies and runs tests.

7. Please submit to the e-learning platform 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.
Collins-Sussman, B., Fitzpatrick, B. W., & Pilato, C. M. (2008). *Version Control with Subversion*. O'Reilly Media.
Mahler, A. (2015). *Practical Subversion*. Apress.
Pérez, R. M. (2013). *Apache Subversion 1.7 Quick Reference Card*. Packt Publishing.
GIT Documentation - *https://git-scm.com/doc*
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*
Jenkins CI/CD Documentation - https://www.jenkins.io/doc/