Modern Software Development with Git, GitHub & CI/CD

In brief

Course langage: English

Presentation

Prerequisites

- Basic Python programming skills (variables, functions, classes, modules).
- Familiarity with using a command-line terminal. Basic understanding of software development concepts.
- A GitHub account and a laptop with internet access.
- No prior experience with Git, Docker, CI/CD, or collaborative software development is required

Learning objectives

This course introduces students to the complete software development lifecycle used in modern software engineering environments. Beyond programming itself, students will learn how professional development teams collaborate, ensure software quality, automate validation and deployment processes, and deliver reliable software products using contemporary DevOps practices. Through a project-based learning approach, participants will experience the full workflow of a collaborative software project, including issue tracking, version control, code review, automated testing, technical documentation, containerization, and continuous integration/continuous deployment (CI/CD). Particular emphasis is placed on teamwork, reproducibility, maintainability, code quality, and industry-standard development practices.

By the end of the course, students will be able to:

- Use Git effectively for version control and source code management.
- Collaborate on software projects using GitHub workflows, including issues, branches, pull requests, and code reviews.
- Apply modern software engineering practices in a team-based development environment.
- Design and implement automated tests using PyTest.
- Produce clear and maintainable technical documentation using docstrings and Markdown.
- Build and deploy Continuous Integration (CI) pipelines using GitHub Actions.
- Containerize applications using Docker.

- Resolve merge conflicts and integrate contributions from multiple developers.
- Use AI-assisted development tools, such as GitHub Copilot, in a responsible and effective manner.
- Contribute to a shared software project following professional software development workflows and best practices."

Description of the programme

- Day 1 begins with a morning of lectures covering Git and the GitHub collaborative workflow. Students will leave Session 1 knowing how to commit, branch, open issues, submit pull requests, and conduct code reviews, even before writing a single line of project code. The final 20 minutes introduce the data challenge and assign students to one of two teams, each of which is responsible for a different module on the same shared codebase. The afternoon begins with a 30-minute introduction to GitHub Copilot, followed immediately by hands-on work. Both teams implement the shared foundation layer and their own module via parallel feature branches and pull requests.
- Day 2 focuses on software quality and automation. The morning session covers testing (Pytest) and documentation (Docstrings and Markdown). Students will write tests and document their module, all via parallel pull requests (PRs) in the same GitHub workflow they already know. In the afternoon, they bring the full picture together with CI/CD using GitHub Actions: a test pipeline on every push, API docs auto-deployed to GitHub Pages, and a basic Docker image pushed to the GitHub Container Registry. The course concludes with the integration climax: both teams open pull requests to the shared upstream repository. They encounter merge conflicts where their independently evolved code diverges in the same files and negotiate a unified codebase, just as real-life projects do at scale.

Throughout both days, students work on a single, shared project, building on each concept as they go. Each lecture is immediately followed by an integrated live demo and hands-on session. By the end of the course, each student will have contributed real commits to a shared GitHub repository, including automated tests, published documentation, a Docker image, and a resolved merge conflict. This will serve as a concrete portfolio artifact reflecting professional development practices.

Prerequisites: Python programming basics and a laptop with internet access. Tools used: Git, GitHub, VS Code, GitHub Copilot, PyTest, Docker, and GitHub Actions.

Generic central skills and knowledge targeted in the discipline

This course contributes to the development of the following Centrale graduate competencies:

Scientific and Technical Skills

- Master modern software engineering methodologies.
- Apply best practices in software quality assurance.
- Understand automation and DevOps principles.
- Design reproducible and maintainable software systems.

Digital and Data Skills

- Use collaborative digital development platforms.
- Manage source code and software lifecycle processes.
- Employ AI-assisted development tools critically and effectively.

Project and Innovation Skills

- Work effectively within multidisciplinary technical teams.

- Manage collaborative development workflows.
- Conduct peer reviews and integrate feedback.
- Contribute to shared technical deliverables.

Professional Skills

- Communicate technical information through documentation.
- Practice collaborative problem solving.
- Develop autonomy in software development environments.
- Understand professional software development standards used in industry.

How knowledge is tested

Assessment is based on continuous evaluation throughout the practical project.

Students are assessed according to the following criteria:

Assessment Criterion	Weight
Effective use of Git and GitHub workflows	20%
Quality of code contributions and pull requests	20%
Implementation of automated tests	15%
Quality of technical documentation	15%
Implementation of the CI/CD pipeline	15%
Participation in collaborative integration and merge conflict resolution	15%

Expected deliverables include:

- A history of Git contributions (commits) and pull requests within a shared repository.
- An automated test suite developed using PyTest.
- Technical documentation for the project.
- A continuous integration workflow implemented with GitHub Actions.
- A Docker image published to a container registry.
- A final integrated version of the software repository resulting from the contributions of all teams.

The assessment evaluates both the technical quality of the deliverables and the students' ability to apply professional collaborative software development practices.

Bibliography

Version Control & Git

- Chacon, S., Straub, B. (2023). *Pro Git* (2nd Edition). Apress. Available online: Pro Git Book

Software Engineering

- Martin, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
- Hunt, A., Thomas, D. *The Pragmatic Programmer* (20th Anniversary Edition). Addison-Wesley.

Testing

- Okken, B. *Python Testing with Pytest*. Pragmatic Bookshelf.

Documentation

- Di Pierro, M. *Documentation Best Practices for Software Projects*.

DevOps & CI/CD

- Humble, J., Farley, D. *Continuous Delivery*. Addison-Wesley.
- Kim, G., Humble, J., Debois, P., Willis, J. *The DevOps Handbook*. IT Revolution.

Docker

- Poulton, N. *Docker Deep Dive*. Independently Published.

Official Documentation

- Git Documentation
- GitHub Documentation
- GitHub Actions Documentation
- PyTest Documentation
- Docker Documentation

Teaching team

Julien ZOUBIAN

Total des heures	**10h**
MN	10h