Complete Guide to a Professional Project Workflow

Social Networks Course - Fall 2025

September 29, 2025

Contents

1 Introduction

This guide has been prepared to standardize the process of completing course projects and to familiarize you with the professional workflow used in the software development industry. Following these steps will not only help you achieve a better grade but will also prepare you for the job market.

2 Part 1: Prerequisites and System Setup

Before you begin, you must install the necessary tools and prepare your system.

2.1 Installing Core Tools

- **Python:** Download and install the latest stable version from the official python.org website. **Important:** During installation, make sure to check the box for `Add Python to PATH`.
- **Git:** The version control tool that is the foundation of our work. Download and install it from the git-scm.com website.
- **Code Editor (VS Code):** We recommend using Visual Studio Code, which integrates well with Git.

2.2 Initial Git Configuration

After installing Git, open a terminal (or Git Bash on Windows) and run the following commands with your own information. This information will be recorded in your commits.

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

2.3 Creating a Python Virtual Environment

Always create your Python projects within a virtual environment to keep their libraries and dependencies separate from other projects.

1. In your terminal, navigate to your project folder.
2. Run the following command to create a virtual environment:

   ```
   python -m venv venv
   ```

3. To activate it:
   - On Windows: `venv`
   - On macOS/Linux: `source venv/bin/activate`

3 Part 2: The Complete Workflow from Issue to Merge

This process is the beating heart of a standard project.

3.1 Step 1: Create and Clone a Repository

First, create a private repository on GitHub named `Social-Network-Fall-2025`. Then, copy its URL and clone it to your system with the following command:

```
git clone <YOUR_REPOSITORY_URL>
```

3.2 Step 2: Define a Task with an Issue

Go to your repository page on GitHub and navigate to the `Issues` tab. Create a `New issue`.

- **Title:** Write a short and descriptive title for the task (e.g., "Implement sentiment analysis function").
- **Description:** Provide full details of the task, including expected inputs and outputs.

3.3 Step 3: Create a Branch Connected to the Issue

On the issue page you just created, find the **Create a branch** option in the right-hand menu. Click it. GitHub will automatically create a branch with a name related to the issue number and title.

3.4 Step 4: Fetch the New Branch and Start Coding

1. In VS Code, open the terminal and run the following command to fetch new information (including the new branch) from the server:

   ```
   git fetch origin
   ```

2. Now, switch to the new branch:

   ```
   git checkout <new-branch-name>
   ```

   Tip: In VS Code, you can also easily switch branches from the bottom-left corner.

3.5 Step 5: Committing with Professional Standards

Write your code. Whenever you reach a logical and stable point, create a commit. Commits should be small and atomic.

- **Commit Message Standard (Conventional Commits):** Your commit message should follow this structure:

  ```
  type(scope): description
  ```

  - **type:** Specifies the kind of change. Common types:
    - `feat:` A new feature
    - `fix:` A bug fix
    - `docs:` Documentation only changes
    - `style:` Code style changes (formatting, etc.)
    - `refactor:` A code change that neither fixes a bug nor adds a feature
    - `test:` Adding or correcting tests
  - **scope (optional):** The section of the codebase affected (e.g., 'data$_{l}$*oader*', '*api*').
  - **description:** A short, imperative summary of the change (e.g., "add function to parse user data").

- **Committing Steps:**

  ```
  # Add changed files to the staging area
  git add .

  # Commit with a standard message
  git commit -m "feat(parser): add function to parse user data"
  ```

3.6 Step 6: Synchronize with GitHub (Sync/Push)

After a few commits, send your changes to the GitHub server:

```
git push origin <your-branch-name>
```

3.7 Step 7: Create a Pull Request (PR)

When your work on the branch is complete, go to your GitHub page. GitHub will show you a prompt to create a **Pull Request**. Click it.

- Write a clear title and description for the PR. In the description, reference the issue this PR resolves (e.g., `Closes #1`).
- Ask your teammates (or in this course, the TAs) to **Review** your code.

3.8 Step 8: Review and Merge

After your code is approved, the project manager (or you) will merge it into the main branch (`main`).

- **Best Practice:** Use the **Squash and merge** option. This combines all your branch's commits into a single, clean commit on the `main` branch, keeping the project history readable.

4 Part 3: Documentation and Project Structure

4.1 The `README.md` File

This file is your project's storefront. It's the first thing people see when they visit your repository.

- **Markdown (.md) Basics:**
  - `# Heading 1`, `## Heading 2`
  - `*italic*` or `_italic_`
  - `**bold**` or `__bold__`
  - Lists with `*` or `-`
  - Code blocks: ``` ``pythòn ... `` ` ```
  - Γ*Links* : `[Link Text](URL)`

A good `README.md` should include:

1. Project title and a short description.
2. Installation and setup instructions (including prerequisites).
3. How to run the code (necessary commands).
4. An explanation of the file and folder structure.

4.2 The `.gitignore` File

This file tells Git which files and folders to ignore. This is crucial for preventing unnecessary, large, or sensitive files from being uploaded. A good example for a Python project:

```
# Python virtual environment
venv/

# Python cache files
__pycache__/
*.pyc

# Datasets - Do not upload large data files!
data/
*.csv
```

```
    *.json

    # IDE and OS files
    .vscode/
    .DS_Store

```

5 Part 4: Best Practices for Code and Testing

5.1 Code Structure: Functional vs. Object-Oriented

Your code should be well-structured. Choose one of the following paradigms (or a thoughtful combination):

- **Functional:** Functions performing specific tasks, suitable for data processing and linear workflows.
- **Object-Oriented:** Classes and objects encapsulating data and behavior, ideal for modeling complex systems.

Key takeaway: Maintain consistency in your chosen paradigm.

5.2 `.ipynb` Files (Jupyter Notebooks)

Notebooks are excellent for exploratory data analysis and visualization but are not intended for defining core library functions.

- **Recommendation:** Implement core logic in `.py` files and import them into your notebooks for analysis and presentation.

5.3 Testing Strategies for Reliability

Implement testing to ensure code reliability and correctness. Focus on these key areas:

- **Unit Tests:** Verify individual components in isolation.
- **Integration Tests:** Ensure that different parts of the system work together correctly.

Testing demonstrates a clear understanding of the code's logic and purpose. Use the `assert` statement or testing frameworks like `pytest` or `unittest` for more comprehensive test suites.

6 Part 5: Responsible Use of Artificial Intelligence (AI)

Using AI tools like ChatGPT or GitHub Copilot for learning and assistance is allowed, but not for blindly copying code.

- **Goal:** AI should be your assistant for understanding concepts, debugging, or suggesting structure, not a replacement for your own thinking.

- **Requirement:** If you use AI, you must create a section in your `README.md` titled "AI Usage" and include a link to the chat or a brief explanation of how it helped you. This is a matter of academic integrity.

7 Part 6: Useful Resources and Links

For deeper learning, refer to the following resources:

- **Git Tutorial:** The free online book Pro Git.
- **GitHub Guide:** The official GitHub Docs.
- **Conventional Commits Standard:** The official Conventional Commits website to learn how to write standard commit messages.
- **Markdown Guide:** A simple and quick Markdown Guide Cheat Sheet.
- **Python Documentation:** The official Python documentation, the best source for learning any part of the language.
- **LaTeX Tutorial:** To learn more about LaTeX, the Overleaf Learn website is an excellent starting point.