Code Publishing Good Practice

for Open Research Scripts and Software @ ENAC

This document provides guidelines for publishing research scripts and software openly, following software architecture and open science best practices, according to the FAIR principles to ensure reproducibility of codes.

Publishing Open Source Code

Scaling Open Software: best practices

1. Share & version the source code

Must: Share the source code

Source code should be stored & published on a version control system, such as Git. Most platforms offer other functionalities (collaborative tools, bugs tracking, wiki, CI/CD..).

WHY? Allow collaboration in an async way at any time. Allow you to go back in time in your project.

TOOLS:

- GitHub (recommended)
- GitLab (self-hosted version)
- c4science (deprecated platform of EPFL).

Interactive scripts in Jupyter notebooks may be shared via nbviewer (open to the world), Noto EPFL (for EPFL community).

2. Document the project

Must: Document the project

README.md: The repository should always have one in the root folder:

- Contain info on how to install, run, deploy the code
- Credit authors and share license
- Describe input and output data
- Link to open data if relevant - include sample/dummy data in expected formats to test running the code.

Contributing.md

- Add a CONTRIBUTING.md at root (to help contributors onboarding)
- More infos on open source projects (guide on how to do a good open source project)
- Examples of github repository having a good contributing file: Atom editor **or** **VS Code**

3. List dependencies

Must: List dependencies

Dependency Management

Using a dependency management tool is crucial to ease project setup/installation.
Typical tools:

a. Python
   i. Pipenv: Pipfile
   ii. conda: environment.yml
   iii. pip: requirements.txt
b. JavaScript/TypeScript
   i. npm: package.json

An abstraction layer such as a Docker image ease the reproducibility (see below)

4. Document the code

Must: Document the project/code

Code should also be documented:

- Python
  - Docstring
- JavaScript/TypeScript
  - JSDoc

Documentation Generator

Once the code is documented, you can automate generating the documentation which come along with the application (usually in html format).

- Python: Sphinx, Read the Docs, Doxygen
- Other languages: Doxygen, Others

5. Choose a license

Licenses are a legal contract between authors and users of a creative work; the authors can grant different levels of permission and usability under specified conditions. The work is only open-source if it is subject to an open source license.

At EPFL, EPFL owns the original data & code, but the authors can use it for research and IP.

- https://choosealicense.com/
- Library RDM Guide on Licensing
- https://tldrlegal.com/
- https://www.epfl.ch/research/services/protect-intellectual-property/software-licenses/choose-the-right-license/

A license file, containing the full license, can be placed in the root folder. For common licenses, a license notice (short version of the license) is enough.

In addition, each script can include in header:

- License, authors, date, last update...
- You may streamline scripts headers with py-licenser.

6. Git Workflow

WHY? Be consistent to facilitate work between contributors

A Git workflow will help you to structure how you use Git (branches, pull/merge requests, …), you can adapt it for your use case, the point is to be consistent between all contributors.

TOOLS: Git Workflows: Gitflow / Gitlab flow / Github flow

Typical branch structure:

➔ main: always points to the latest release / version
➔ develop: integration/working branch
➔ feature/*: work on separate feature branches and merge them to develop

7. Project layout

Following the language and/or framework use a project initializer:

- JavaScript/TypeScript
  - Vue CLI
  - Angular CLI
- Python: there is no “official” project layout, but a general one would look like:

```
my_app/
│
├── my_app/
│   ├── __init__.py
│   ├── core.py
│   └── helpers.py
│
├── tests/
│   ├── test_basic.py
├── docs/
│   ├── conf.py
│   ├── index.rst
└── setup.py
├── .gitignore
├── LICENSE
├── README.md
├── requirements.txt
```

  - More details

8. Coding Conventions

Select a code style guide for each language and enforce them in the whole project:

Language	**Style Guide**	**Enforcement tool**
Python	PEP 8	Flake8 + black
JavaScript/TypeScript	Prettier	ESLint
C++	Google C++ Style Guide	Cpplint
VueJs	eslint vue	eslint-plugin-vue

Static Analysis Tool

Having consistent formatted code is good, but you can do better with a static code analysis tool: it can warn you about code duplications, security flaws, syntax errors, unreachable codes, ...

- Python
  - Pylint
- JavaScript
  - JSHint
- C++
  - Cppcheck
- Multiple languages
  - SonarQube

9. Software tests

Testing is key at each release of the software, to avoid breaking the code. (Read more about tests in Python).

There are many kinds of tests and the most commons are:

- **Functional Tests**: verify the output given some inputs
  - **Unit Tests**: test individual methods
  - **Integration Tests**: check behavior on a running application with other components (database, service, …)
- **Non-Functional Tests**: to determine breaking points
  - **Performance Tests**: check that the application responds in the expected time

Language	**Unit Tests**	**Integration Tests**
Python	unittest	
JavaScript/TypeScript	Mocha JEST	Cypress Nightwatch.js
Vue3	vitest	Cypress

It is a good practice to at least create some unit tests during the development instead of testing manually your application.

CI/CD

Instead of running the tests, code analysis and/or generating the documentation manually, this can be done automatically by a continuous integration pipeline.
Whenever you push some commits on the repository, a process can check if there is no regression in your application.

Depending on the chosen CI, some tools are included

Platform	**CI**	**Static Code Analysis**
GitHub	GitHub Actions	Code Scanning
GitLab	GitLab CI	Code Quality Code Security
Any Platforms	Jenkins Travis CI	SonarCloud

Security

At each release, ensure compliance with secure coding practices (e.g. input validation, proper error handling, password use in code, etc.) and take into consideration common application security vulnerabilities (e.g., code injection, etc.).

10. Packaging

Packaging

To distribute your application on a public package repository (such as PyPI) and make it easily installable by everyone, you will need to build and pack it in a distribution package.

- Python: Packaging Python Projects to PyPI
- JavaScript/TypeScript: Contributing packages to npm

Docker

Containerization with Docker helps reproducibility of the project. A container is like a virtual machine, but lighter and faster: https://www.docker.com/resources/what-container
Once you define the image, it can be run on any machine in the same way.

Docker Compose

With Docker Compose, you can define how a set of Docker containers starts.

Versioning

Publishing your application also means creating some releases. This helps the user to know which version it uses and what are the existing ones.
Each release should have a version number and it should follow the well known semantic versioning.

11. Distribution

User Support

User support is very important to building a community of users around your software. Choose an outlet to keep track of user's comments (and/or bug tracking), and point your users to it. General forums also work (StackOverflow…)

- Google groups
- GitHub Issues or Github discussions
- GitLab Issues

Communication & dissemination

Dissemination time/effort is not to be underestimated to achieve best impact of open software

- Website
- Communication strategy (community of users, who's the audience?)
- Publications: Open software journals such as JOSS.

Why Open Software (Open Science) ?

- Societal impact : scientific vulgarisation, research valorisation
- Academic career recognition: peers usage, citations...
- Innovation/private sector : a start-up out of your research software!

12. Resources

- Guide for Reproducible Research
- Code & Data mgt workshop, EPFL Library
- Funding opportunities for Open Scripts & Software (internal ENAC)