# Software Development

> How to bring your (python) codes to the next level

Nicolas Dagoneau - nicolas.dagoneau@cea.fr

Transient Universe 2023 - Cargèse

## Todo before the hands-on

- Create an account on GitHub
- Ensure you can work with python (hands-on are tested with 3.10)
- Install required packages

I recommend working in a conda environment that is isolated from yours.

# Introduction

## About me

- PhD on Svom (2017-2020): image processing for the ECLAIRs onboard trigger, study GRBs detection (ultra-long GRBs).
- Since 2021: working at CEA, computer division, on Svom and Euclid (science & development).

## Goal of this presentation

- Show you how to turn bunch of python files into package, ready to be shared, tested and documented.
- This is not about data analysis.
- Based on my own experience (hence biased).

## Why?

During my PhD, the code I wrote was mainly python files (modules) + some jupyter notebooks (always changing) in different directories: not always backed-up, not documented, very few constrains on code quality, few or no test at all → difficult to maintain and to share (ie. not FAIR).

## Tools

- Use an integrated development environment: pycharm, VSCode...
- Define package: setuptools
- Implement tests (in parallel to the development): pytest, using `assert`
- Code coverage by the tests: coverage
- Write the documentation: sphinx/ReadTheDocs
- Autoformat code: black
- Analyse code: pylint, ruff, flake8...
- Push to git: github, gitlab

## Package structure

Basic package structure.

```
├── pyproject.toml
├── README.md
├── requirements.txt
├── doc
├── src
│   └── cargese
│       ├── gcn_requester.py
│       ├── __init__.py
│       └── tools.py
├── tests
│   ├── test_gcn_requester.py
│   └── test_tools.py
```

## Project configuration in *pyproject.toml*

```toml
[project]
name = "cargese"
version = "0.0.1"
authors = [
    {name = "Nicolas Dagoneau", email = "nicolas.dagoneau@cea.fr"},
]
description = "Tutorial package for Transient Universe 2023 school in Cargese"
readme = "README.md"
dependencies = [
    "requests",
    "pandas",
    'importlib-metadata; python_version<"3.8"',
]

[project.scripts]
gcn-requester = "cargese.scripts.cargese_gcn_requester:main"

[tool.coverage.run]
omit = ["*/scripts/*"]
```

## Write tests

Tests should be simple, short, easy to understand and allow to cover all cases in the code (*if*, *else*, *for*, raised exceptions...). They use `assert`.

### Example:

```python
def test_timestamp_to_datetime():
    utc_date = tools.timestamp_to_datetime(0)
    assert utc_date == datetime.datetime(1970, 1, 1)
```

## Install, test, build documentation

```
black src/
ruff check src/  # pylint src/
pip install .
pytest tests/
coverage run --source src/ -m pytest
coverage report
cd doc && make html # or other format
```

## All together with `make`

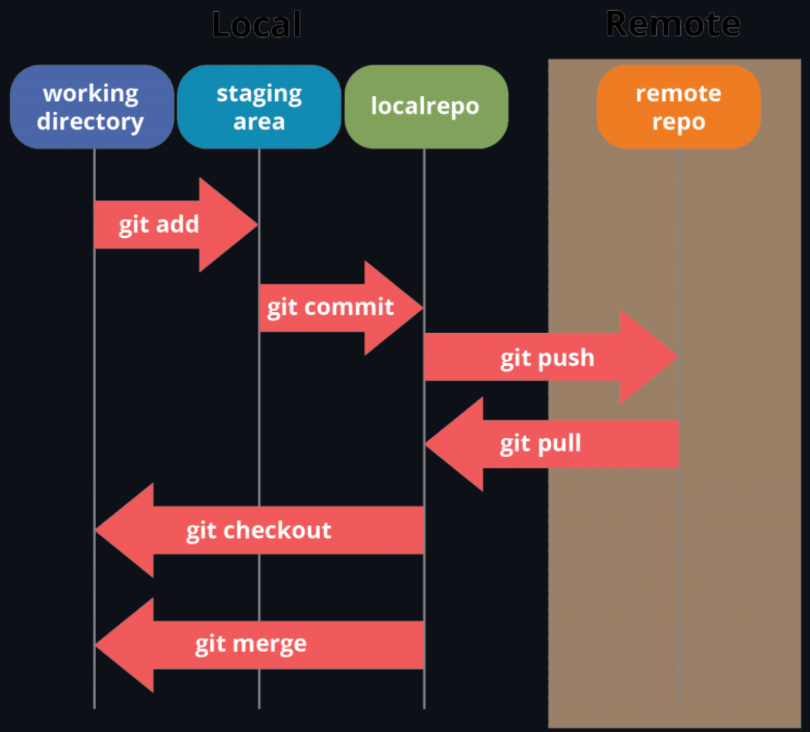
*Makefile:*: `make install`, `make test`...

```
.PHONY: all
all: install test sphinx coverage

install:
    @pip install .

test:
    @pytest tests

sphinx:
    @make -C doc/ html

coverage:
    @coverage run --source src/cargese -m pytest
    @coverage report
```

# A few words about git

Manage code versions, back-up, improve team development: git-guide, git-branching

```
git add new_class.py tests/test_new_class.py
git commit -m "Implement new class"
git push
```



## Continuous integration: push, build, test, deploy

You can build wathever you want (eg. building pdf for PhD manuscript).

### Running in a distant repository

Jobs (install, checks, tests, ...) are described in yaml files.

- On github, it works with *actions*, stored in `.github/workflows`.
- On gitlab, it works with `.gitlab-ci.yml`

## Lets practice

- Fork github.com/dagnic/cargese-TS2023-dev
- (Create conda env: `conda create -n cargese python=3.10`)
- Install requirements: `pip install -r requirements.txt`

### Exercise

Implement a new method/class, install, add tests and run them, generate documentation, (use `make`) push and check that jobs succeed!

## Few configurations

- Activate github pages on `gh-pages` branch (https://github.com/<user>/<project>/settings/pages). Documentation is here: https://<user>.github.io/<project>/
- Add your repository to coveralls.io

## To go further away

- For other languages (eg. C++), you could create bindings to access C++ classes/methods via python: pybind11, swig.
- Create your own dashboard to plot results using plotly/dash
- Licence for software distribution: that's something you have to consider if you want to share your package within the public domain.
- Publish package to PyPI: twine
- Things can alway be improved: find a balance
- Code design and factoring is also an important job
- Version number update: bump2version
- Changelog

… "Be kinder to your future self" *(ruff)*