Software Development and Ground segments

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

How to develop in the context of a ground segment

Nicolas Dagoneau - nicolas.dagoneau@cea.fr

CEA Paris-Saclay

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).

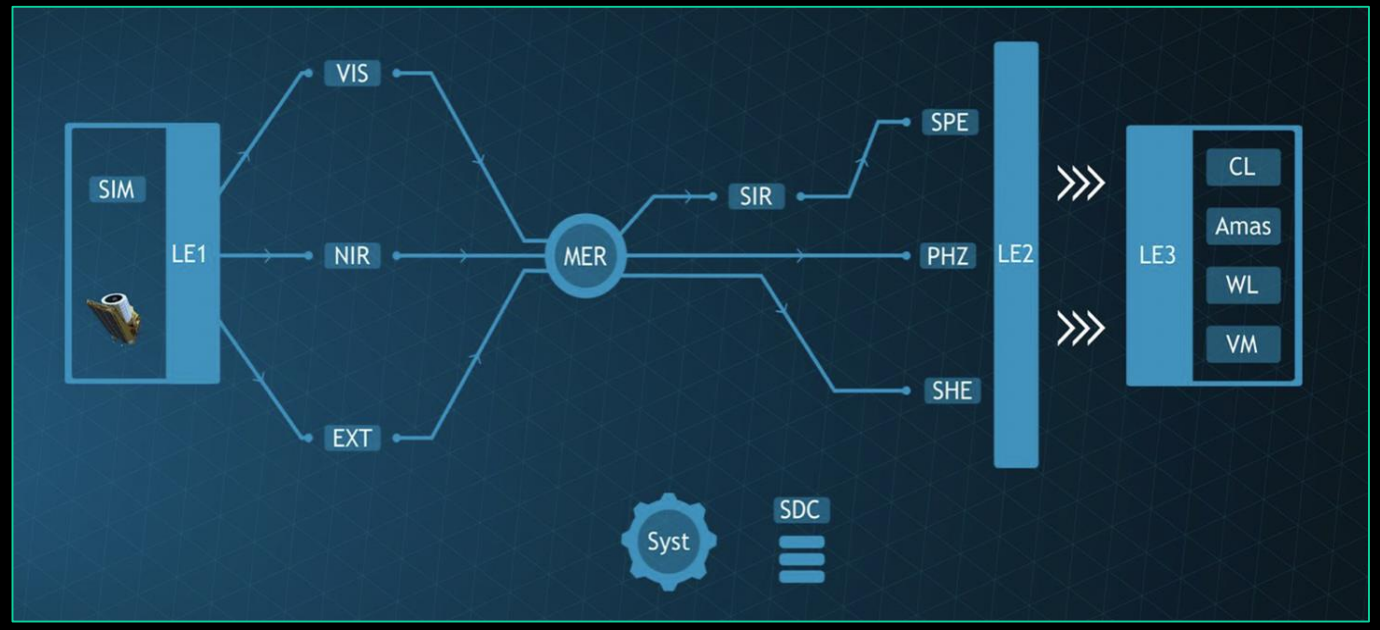
Why

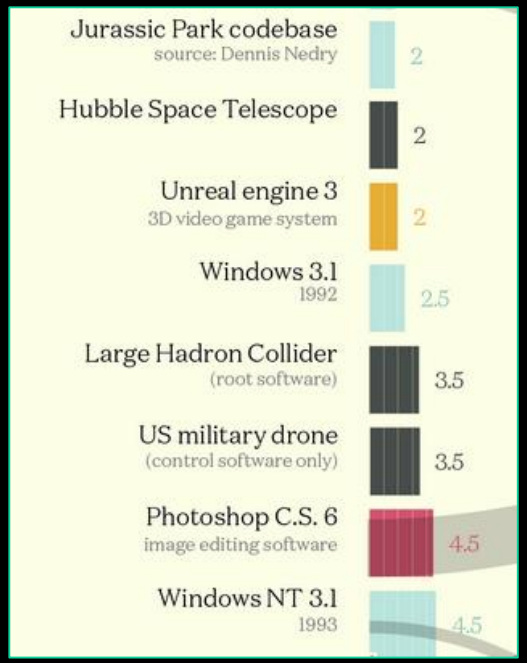
- During my PhD, the code I wrote was mainly python files (modules) + some jupyter notebooks (always changing) in different directories: not always backed-up, not documented, very few constrains on code quality, few or no test at all & difficult to maintain and to share.

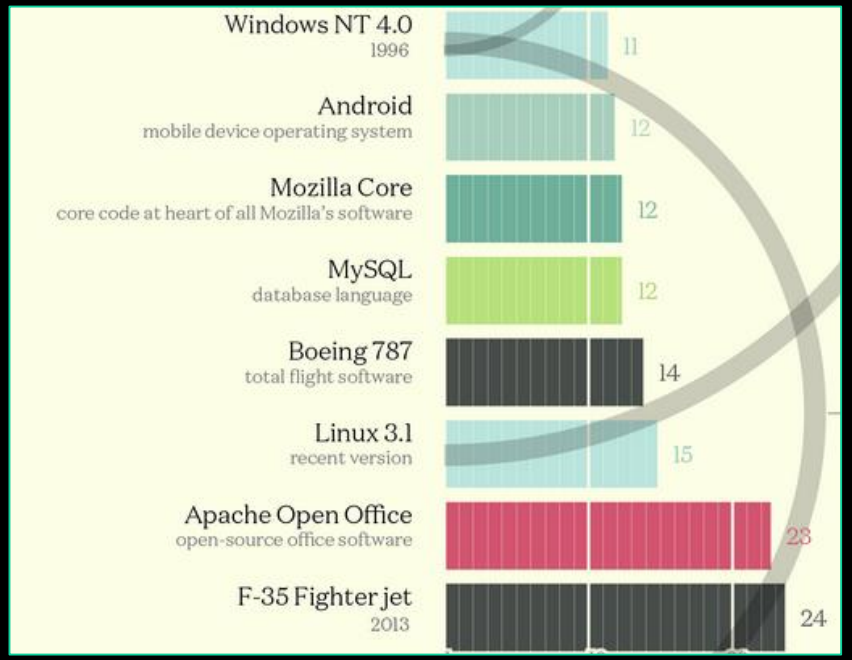
- You will probably write code with a team for a quite large project.

Ground segments: Euclid (ESA)

- 21 countries
- 285 institutes
- 1.2 million lines of code
- 8 computing center (CC-IN2P3 near Lyon for France)

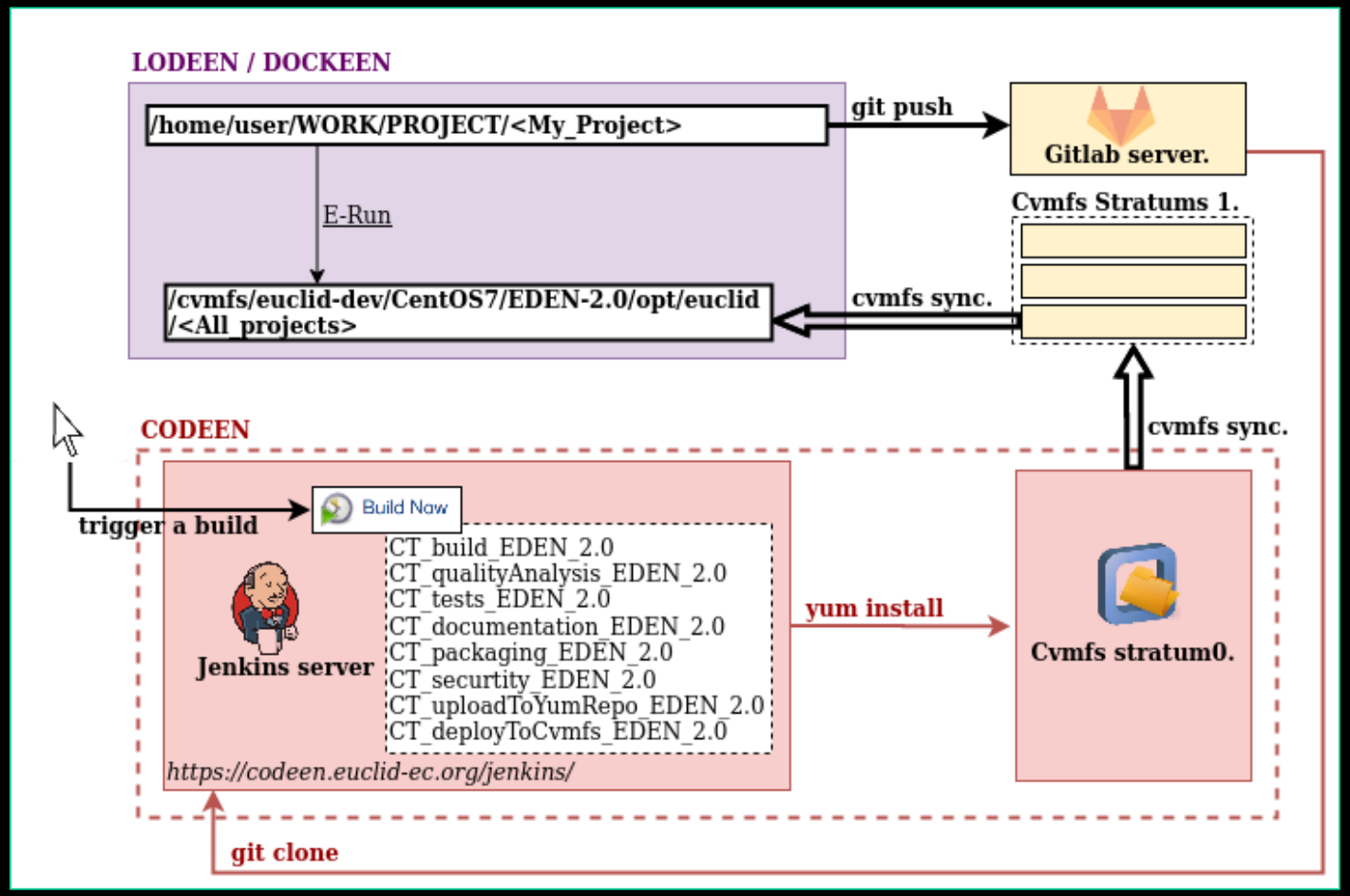



Jurassic Park codebase
source: Dennis Nedry
2
Hubble Space Telescope
2
Unreal engine 3
3D video game system
2
Windows 3.1
1992
2.5
Large Hadron Collider
(root software)
3.5
US military drone
(control software only)
3.5
Photoshop C.S. 6
image editing software
4.5
Windows NT 3.1
1993
4.5


Windows NT 4.0
1996
11
Android
mobile device operating system
12
Mozilla Core
core code at heart of all Mozilla's software
12
MySQL
database language
12
Boeing 787
total flight software
14
Linux 3.1
recent version
15
Apache Open Office
open-source office software
23
F-35 Fighter jet
2013
24


Microsoft Office 2013
45
1
10
20
30
40
50
60
70
80
Large Hadron Collider
total code
125%
50
Windows Vista
2007
50
Microsoft Visual Studio 2012
50
Facebook
(including backend code)
62
US Army Future Combat System
fast battlefield network system (aborted)
63
Debian 5.0 codebase
free, open-source operating system
68
Mac OS X "Tiger"
v 10.4
86

Ground segments: Euclid (ESA)



- Common development environment (OS, libs, …) for developers and computing centers
- Shared network repository where the code is installed

Ground segments: SVOM (CNES/CNSA)

Different approach:

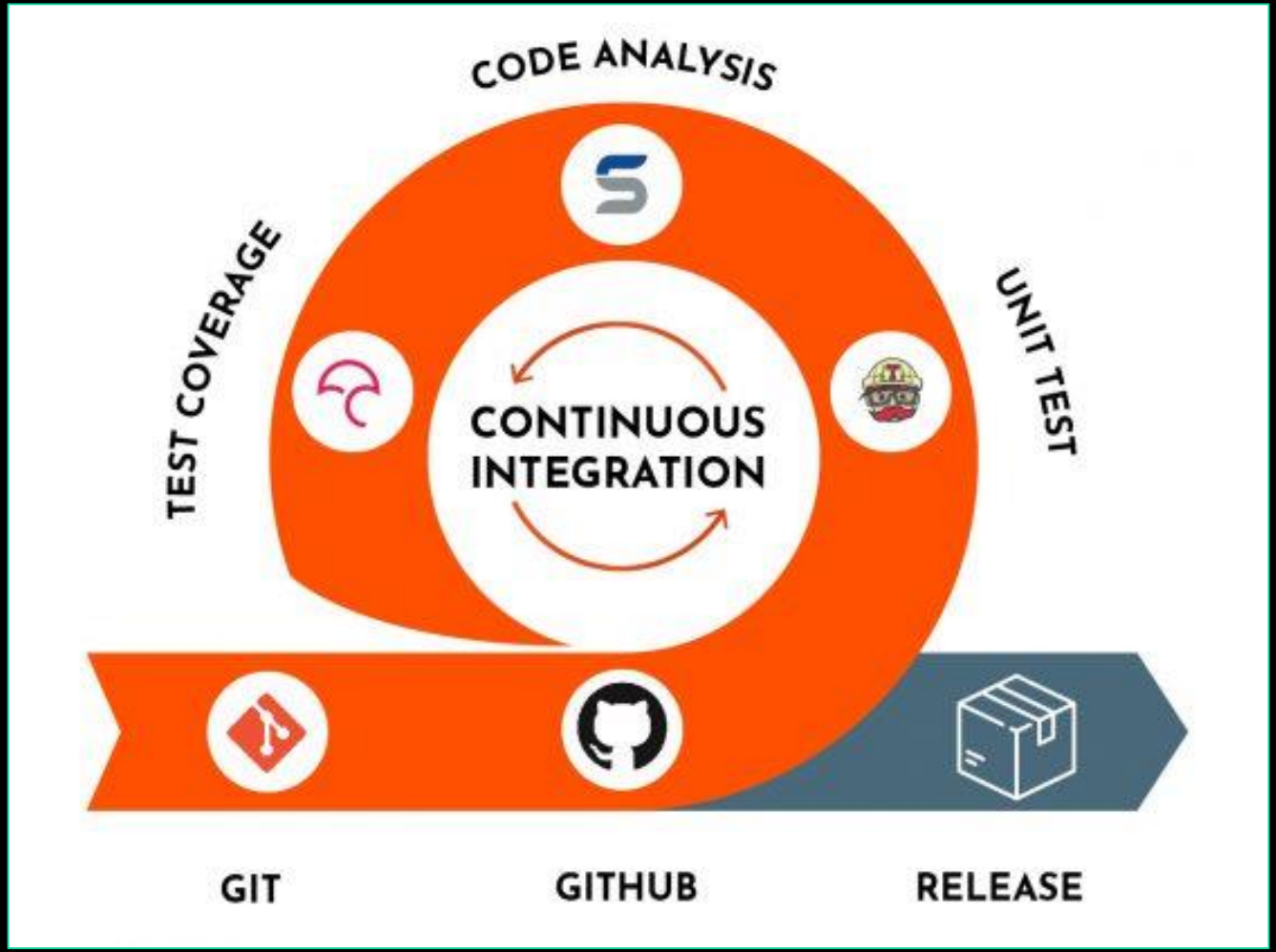
- Development in user preferred environment
- Code installed in Docker images
- Docker images stored in a cloud registry
- Services use code within docker images to achieve a task

Metrics example (SVOM)

CNES metric	Target	Sonarqube metric
Cyclomatic number	<=25	Cyclomatic complexity of functions should not be too high (or 'Methods should not be too complex' for Java)
Simplified cyclomatic number	<=20	Cognitive complexity of methods should not be too high
Number of lines of code	<=100	Functions should not have too many lines of code
Number of nesting	<=7	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply
Comment rate	>20%	Comments (%)
Copy/paste rate in each component	<15%	Duplicated Lines (%)
Technical debt for reused code	<5%	Technical Debt Ratio
Number of bugs	0	Bugs
Coverage Level	>80%	Line Coverage (%)

Continuous integration

When tests, analysis or coverage failed, no deployment in production!



Code analysis

- Check readability, maintainability, duplications, coding style, errors, …
- See python guidelines PEP 8

```
def my_example_method(k):
    for i in range(10):
        print(j)
    return i
```

What are the problems here?

Code analysis

- Check readability, maintainability, duplications, coding style, errors, …
- See python guidelines PEP 8

```
def my_example_method(k):
    for i in range(10):
        print(j)
    return i
```

```
$ pylint test.py
test.py:9:0: C0305: Trailing newlines (trailing-newlines)
test.py:5:0: C0116: Missing function or method docstring (missing-function-docstring)
test.py:7:14: E0602: Undefined variable 'j' (undefined-variable)
test.py:5:22: W0613: Unused argument 'k' (unused-argument)
------------------------------------------------------------------
Your code has been rated at 0.00/10
```

Code analysis

- Check readability, maintainability, duplications, coding style, errors, …
- See python guidelines PEP 8

```
def my_example_method(k: int):
    """
    An example method

    :param k: number of iteration
    :return: last iteration index
    """
    for i in range(k):
        j = i+1
        print(j)
    return i
```

Your code has been rated at 10.00/10

Write unit 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.

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

Coverage

```python
def get_trig_type(trig):
    """
    Get the trigger type
    :param trig: integer ATC_trigIMT
    :return: trigger type
    """
    if trig == 1:
        return f"IMT [{trig}]"
    if trig == 0:
        return f"CRT [{trig}]"
    return trig
```

Not all the cases are covered by the tests. Coverage < 50 % for this method.

Sonarqube

Security

0 Open issues

A

0 H | 0 M | 0 L

Reliability

6 Open issues

A

6 H | 0 M | 0 L

Maintainability

89 Open issues

A

6 H | 50 M | 33 L

Accepted issues

0

Valid issues that were not fixed

Coverage

85.1%

On **4.9k** lines to cover.

Duplications

0.2%

Required ≤ 15.0%

On **15k** lines.

Documentation

- Written in reStructuredText or Markdown and converted to html by sphinx.
- Uses method docstrings to generate package documentation.
- Uses program parameters to generate CLI (command line interface) documentation.
- Can use various themes such as ReadTheDocs.

```
usspytools
==========

This is the *usspytools* package documentation. To ready about the distribution and the installation of the package, please refer to
 the `Ground section <https://fsc.svom.org/documentation/eclairs-lv/USS-Ground/index.html>`_ of the `ECLAIRs ASW User Manual
 <https://fsc.svom.org/documentation/eclairs-lv/index.html>`_.

.. toctree::
   :maxdepth: 2
   :caption: Contents:

   ui
   cli
   api


Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

```
pyDecodeCcsds.py
================

.. argparse::
   :filename: ../usspytools/scripts/pyDecodeCcsds.py
   :func: get_parser
   :prog: pyDecodeCcsds.py
```

```
Reference API
*************

.. automodapi:: usspytools.pages
  :no-inheritance-diagram:
  :include-all-objects:

.. automodapi:: usspytools.tools
  :no-inheritance-diagram:
  :include-all-objects:

.. automodapi:: usspytools.science
  :no-inheritance-diagram:
  :include-all-objects:
```

Documentation

usspytools

v7.0

Search docs

CONTENTS:

ETC trigger UI

CLI

Reference API

/ usspytools

usspytools

This is the *usspytools* package documentation. To ready about the distribution and the installation of the package, please refer to the Ground section of the ECLAIRs ASW User Manual.

Contents:

- ETC trigger UI
  - Interface home page
  - Troubleshooting
- CLI
  - pyDecodeCcsds.py

get_time_utc

`usspytools.tools.get_time_utc(time_svom: int) → Time`

Parameters: **time_svom** – SVOM time

Returns: astropy Time

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 (jupyter-black)
- Analyse code: pylint, ruff, flake8...
- Push to git: github, gitlab

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/*"]
```

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

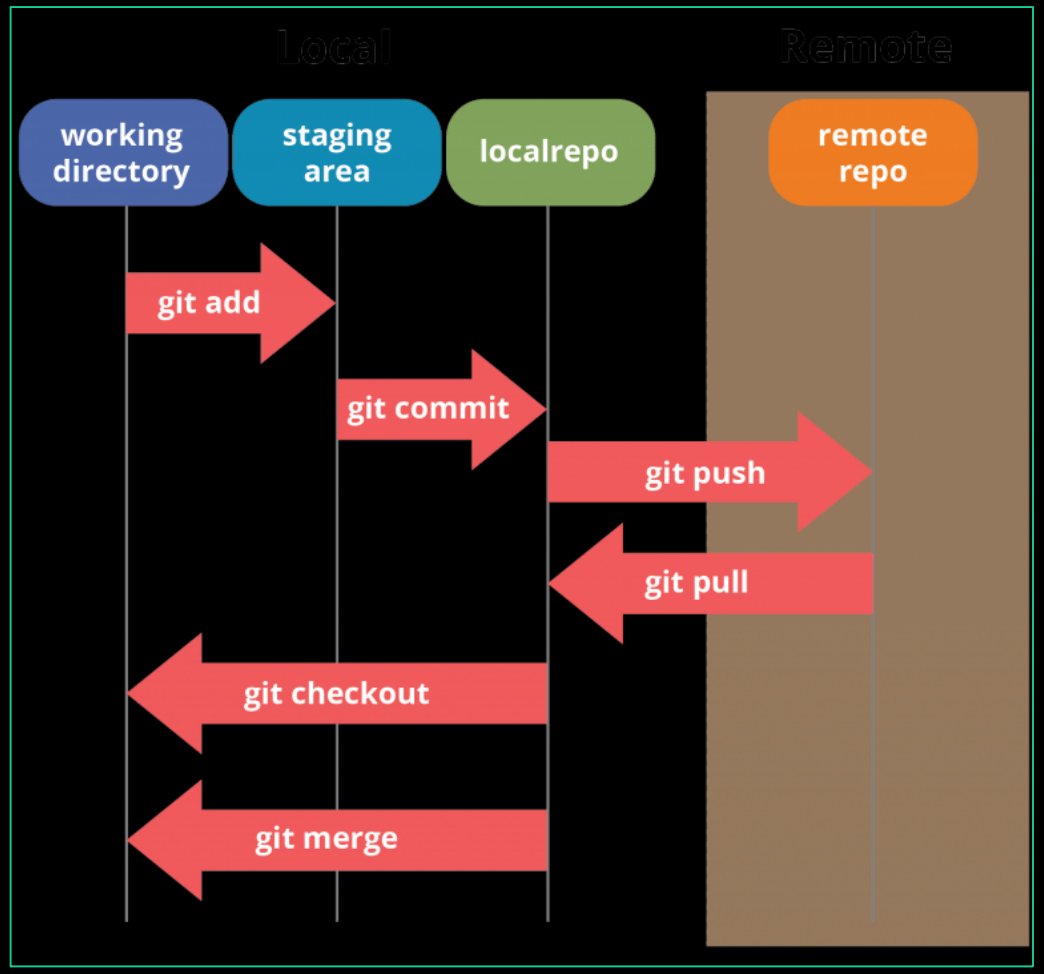
make install, make test, make all

```makefile
.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
```



Collaborative work with git: branches

A bad example…

May 20, 2024

[MR] - Code optimization authored 5 months ago	5f51b503
[MR] - Code optimization authored 5 months ago	04b9ef81
[MR] - Code optimization authored 5 months ago	f2b0e410
[MR] - Code optimization authored 5 months ago	4b8d8385
[MR] - Code optimization authored 5 months ago	
[MR] - Code optimization authored 5 months ago	
[MR] - Code optimization authored 5 months ago	
[MR] - Code optimization authored 5 months ago	
[MR] - Code optimization authored 5 months ago	

COMMENT	DATE
CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
ENABLED CONFIG FILE PARSING	9 HOURS AGO
MISC BUGFIXES	5 HOURS AGO
CODE ADDITIONS/EDITS	4 HOURS AGO
MORE CODE	4 HOURS AGO
HERE HAVE CODE	4 HOURS AGO
AAAAAAAA	3 HOURS AGO
ADKFJSLKDFJSDKLFJ	3 HOURS AGO
MY HANDS ARE TYPING WORDS	2 HOURS AGO
HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Continuous integration: push, build, test, deploy

You can build whatever you want (eg. building a personal webpage).

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

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 data using plotly/dash
- License 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 always 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)

Job to do (individual group or by 2)

- Create a github project (public or private, if private you can add me later @dagnic)
- Create a python project following the basic structure (pyproject.toml, requirements.txt, src, doc, tests)
- Publish several commits to implement something (~ 100 lines of code), eg.
  - A personal web page
  - A module with a CLI program to compute something with user inputs
  - A graphic interface (window, dashboard to display some data, eg. related to your GRB)
- Follow coding standards
- Implement tests and measure coverage
- Use actions on github (tests, quality)
- Serve sphinx documentation to https://<user>.github.io/<project>
- Advanced: add badges to the README with coverage and CI status

Requirements

- Ensure you can work with python (prefer Miniforge over Anaconda).
- Create an account on GitHub
- Activate github pages on gh-pages branch (https://github.com/<user>/<project>/settings/pages).
- Coverage can be analyze directly from github using coveralls.io (need public repository)

Modality

- Individual work or by group of 2
- Due date: December 15, 2024
- Deliverable: link to your github repository (invite me if private, @dagnic)
- I will check: rules with black, coverage, tests (all must pass), documentation (program, method, classes) deployed as a github page, I will try to run pip install locally on Ubuntu (it should work), git history with nice commits, …

The only valid metric…