Informatique des expériences

Du détecteur à la mesure

Roman Le Montagner - IJCLab

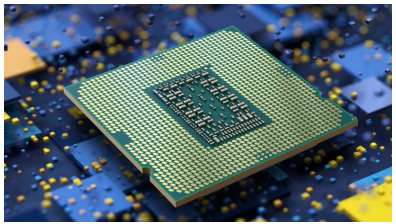
Plan du cours

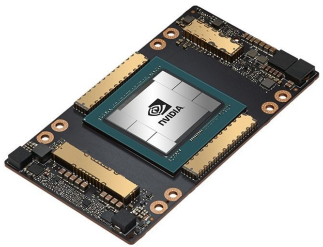
I. Fink, un logiciel pour le traitement automatique d'alerte astronomique
II. Qu'est ce qu'un logiciel ?
III. Qualité logicielle
IV. Architecture logicielle
V. Tests
VI. Documentations et Normes
VII. Gestion de Version
VIII. Integration Continue / Déploiement Continue
IX. Packaging & Publication
X. Reproducibilité

Le numérique

Pourquoi numériser ?

- Stockage fiable (**reproducibilité**, code correcteur, transmission simple)
- Partage des données (**intéropérabilité**, formats standard)
- Débruitage et correction (**calibration**, prétraitement, compression)
- Continuité et traçabilité des mesures (**version**, archivage)
- Traitement automatique (**algorithmes**, IA, filtrage, mesures statistiques)





Un logiciel pour le traitement automatique d’alerte astro.

Quelques chiffres …

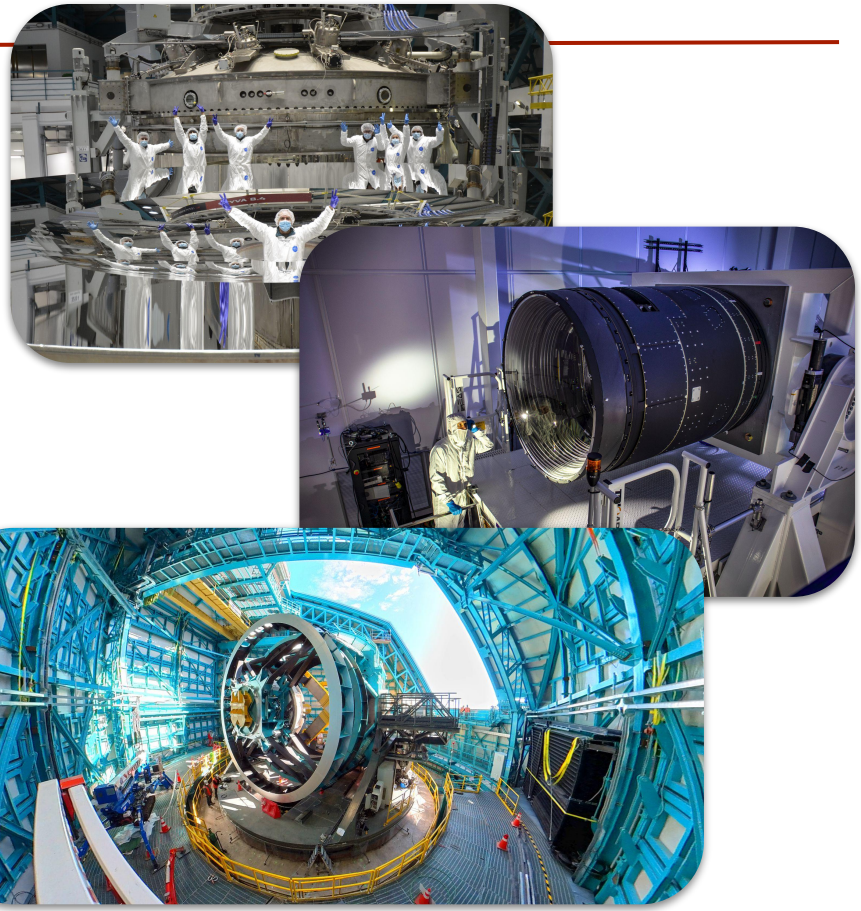
Champ de vue: 3.5 degrés
- 7 fois la Lune !

Diamètre mirroir primaire: 8.4 m
- 5 fois la taille d'un humain moyen

Nombre de pixel caméra: 3.2 Gpixels
- 1000 fois un smartphone

Volume de donnée nocturne: 20TB/nuit
- 20 disque dur standard !

Flux d'alerte nocturne: 1TB/night
- 10,000,000 notifications sur vos smartphone



Rubin/LSST Flux de donnée

Observatoire Rubin (2025+)

- **20TB d'images / nuit**
- **1TB d'alerts / nuit**: x100-x1000 au dessus des flux actuelles
- *Everything matters a priori*

Now

Raw Data
Sequential 30s image, 20TB/night

60s

Prompt Data Product
Difference Image Analysis
Alerts: up to 10 million per night

24h

Prompt Products DataBase
Images, Object and Source catalogs from DIA
Orbit catalog for ~6 million Solar System bodies

Year

Annual Data Release
Accessible via the LSST Science Platform & LSST Data Access Centers.

End

Final 10yr Data Release
Images: 5.5 million x 3.2 Gpx
Catalog: 15PB, 37 billion objects

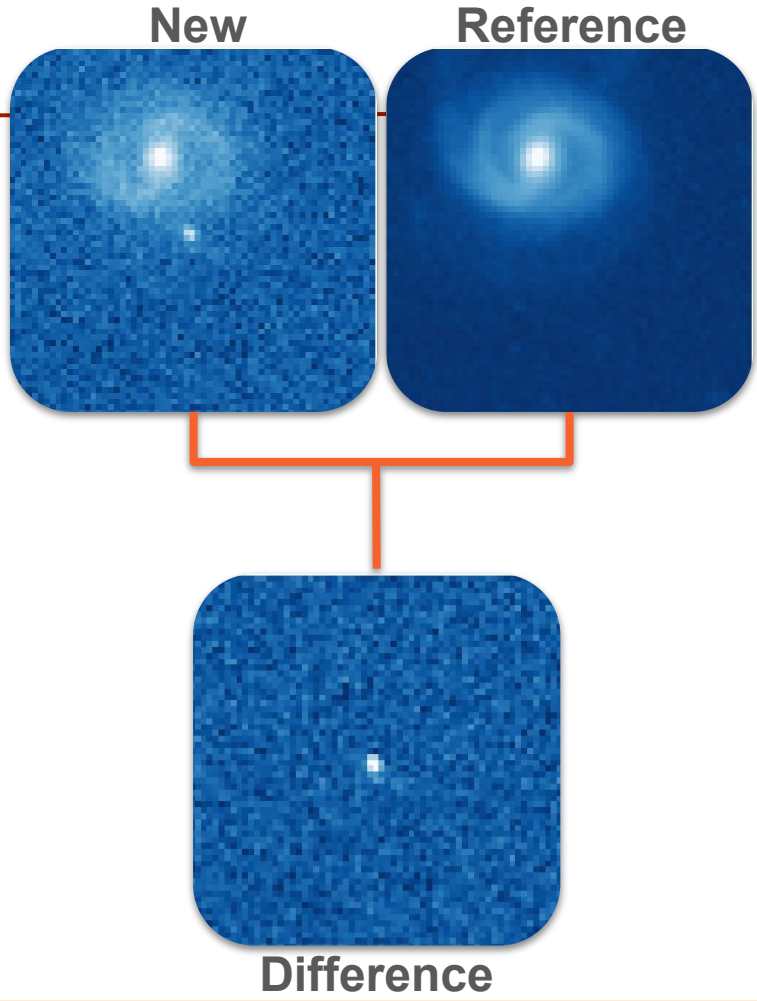
Contenu des alertes

Alertes basées sur **l'analyse d'images différentielles** (DIA)

Chaque alerte contient :

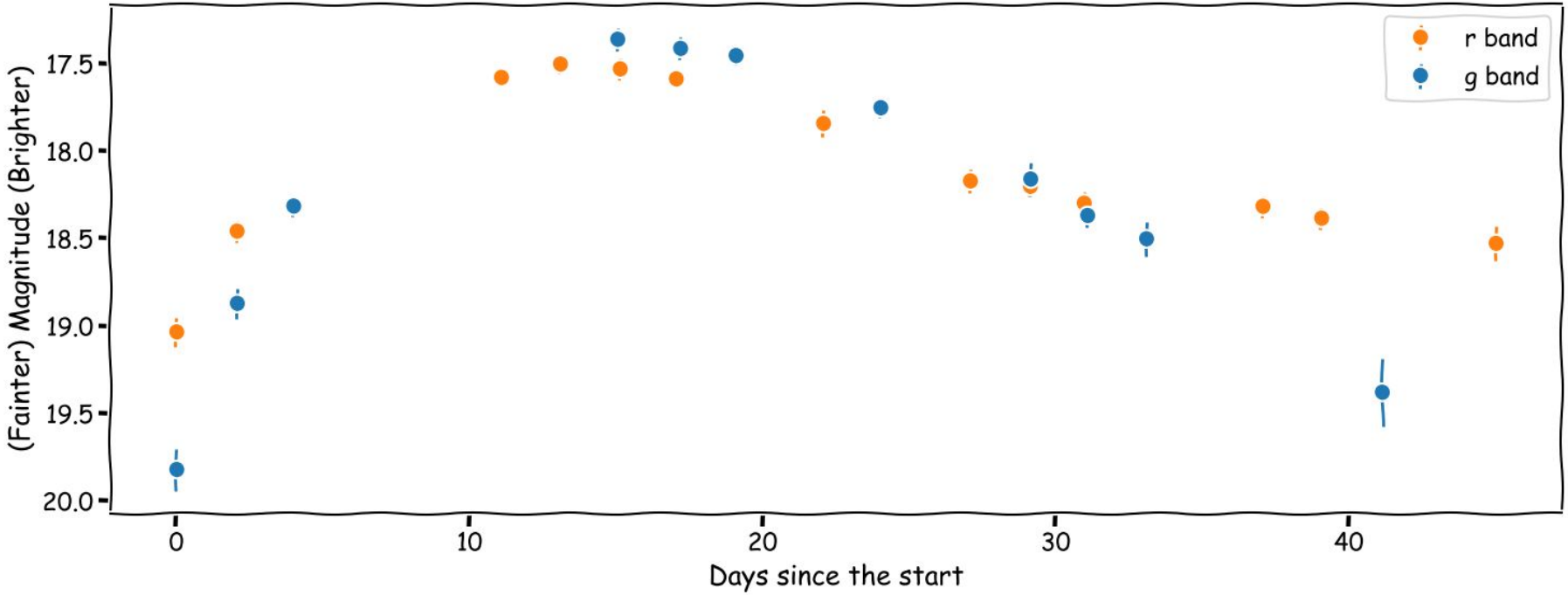
- Des informations sur la nouvelle détection (estimation de magnitude, position, …)
- Des informations sur les sources voisines (Gaia, Pan-STARRS)
- L'historique des observations à cette position
- De petites images autour de la détection (60×60 pixels)

More information: https://github.com/lsst/alert_packet



Courbe de lumière (Supernovae de type Ia)

Observations à intervalles irréguliers. Cadence variable d'une bande à l'autre.

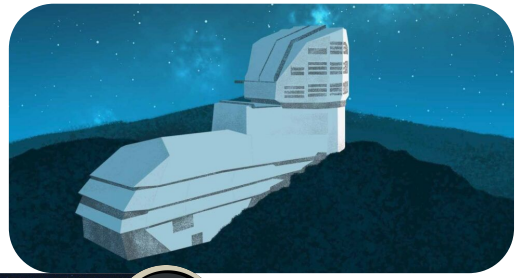


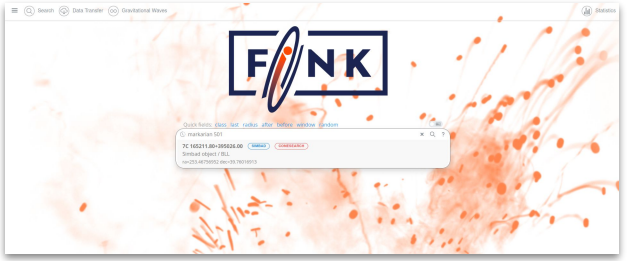
Fink in a nutshell

https://fink-broker.org

Fink est un *broker* au service de la communauté scientifique, chargé d'ingérer, de classifier, de filtrer et de redistribuer les alertes à la communauté scientifique.

- À partir de 2025 : plus de **70 collaborateurs**, répartis dans **15 pays**.
- Services déployés sur de grands clouds OpenStack (UPSaclay & CC-IN2P3).
- *scalable* jusqu'à plusieurs millions d'alertes par nuit.
- En fonctionnement **24h/24** et **7j/7** depuis 2019, au service de plus de **100 utilisateurs uniques par jour** (scientifiques, système de suivi et amateurs).





Fink : Infrastructure informatique

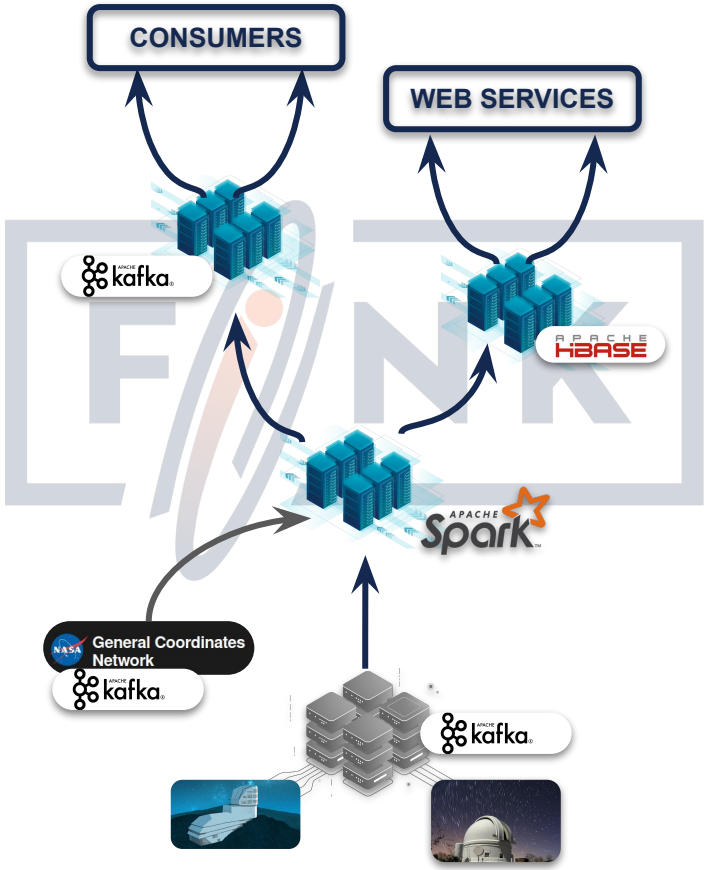
Déploiement de Fink sur des plateformes cloud (UPSaclay, CC-IN2P3).

Utilisation de système distribué :

- Calcul (Spark), Base de donnée (HBase), Streaming (Kafka), Stockage (Ceph et HDFS)
- Orchestration : Mesos et Kubernetes.
- Autoscaling en fonction de la charge.

ML : PyTorch, TensorFlow, scikit-learn (remarque : CPU uniquement !).

Versionnage du code, des modèles et des données.



Le bloc de base: Le logiciel

Logiciel : Une suite d'instructions interprétable par une machine et un jeu de données nécessaire aux opérations.

Trois éléments clé:

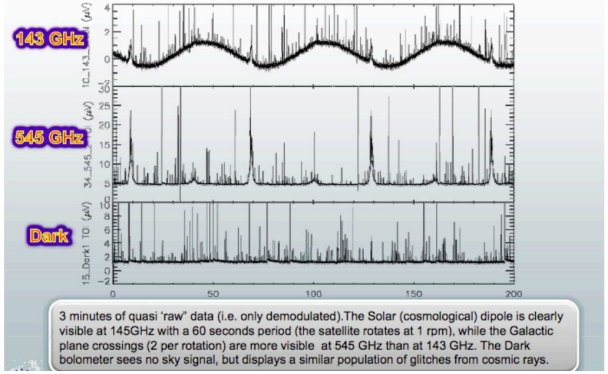
- Processus : ce que le logiciel fait
- Données : ce que le logiciel manipule
- Interfaces : comment on interagit avec le logiciel

Le bloc de base: Le logiciel

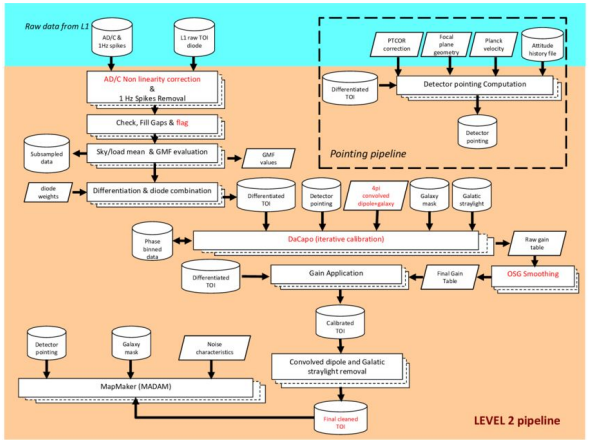
Logiciel : Une suite d'instructions interprétable par une machine et un jeu de données nécessaire aux opérations.

Données → **Traitement** → **Produit scientifique**

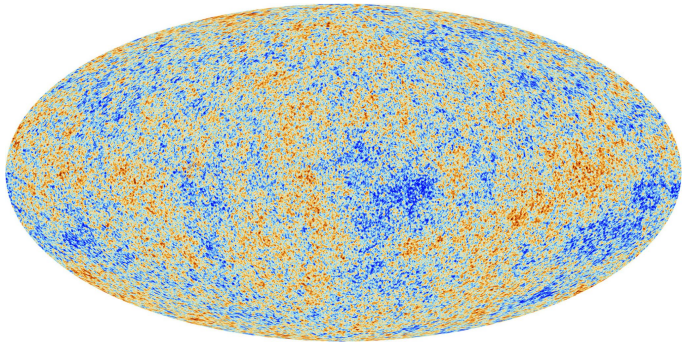
(nearly) raw data stream



Natoli, P. (2013). *Planck data processing and map-making* [Présentation]. Vietnam School of Physics, IN2P3. http://vietnam.in2p3.fr/2013/Cosmology/transparencies/Natoli1.pdf



Planck Collaboration, et al. *Planck 2018 Results. II. Low Frequency Instrument Data Processing*. 2020. https://www.researchgate.net/publication/345607185_Planck_2018_results_II_Low_Frequency_Instrument_data_processing



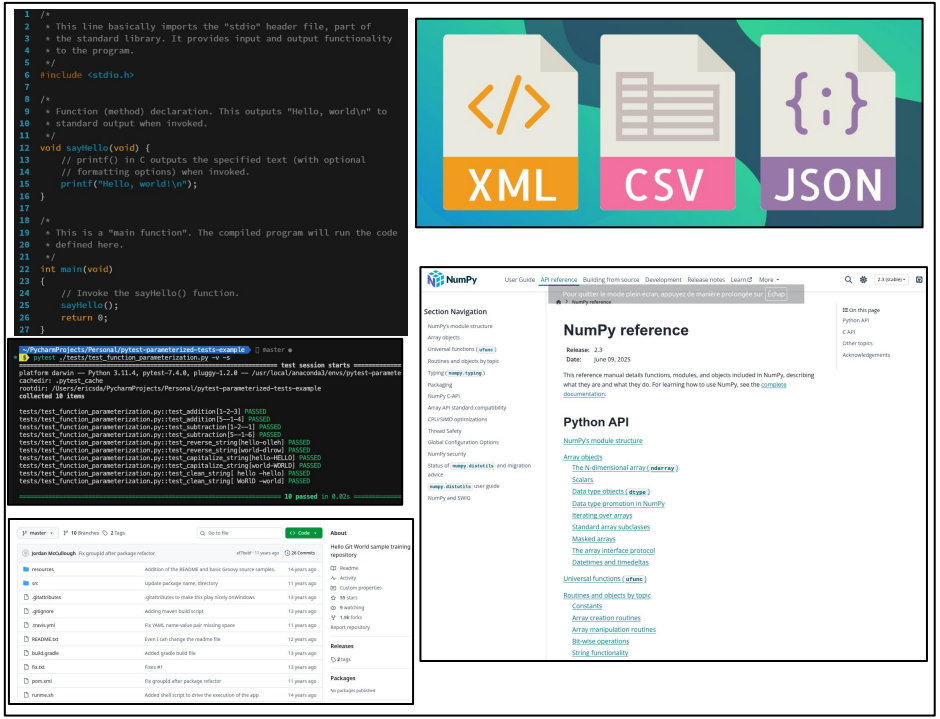
European Space Agency. *Planck CMB Map*. 2013. ESA, https://www.esa.int/ESA_Multimedia/Images/2013/03/Planck_CMB

Programme vs Logiciel

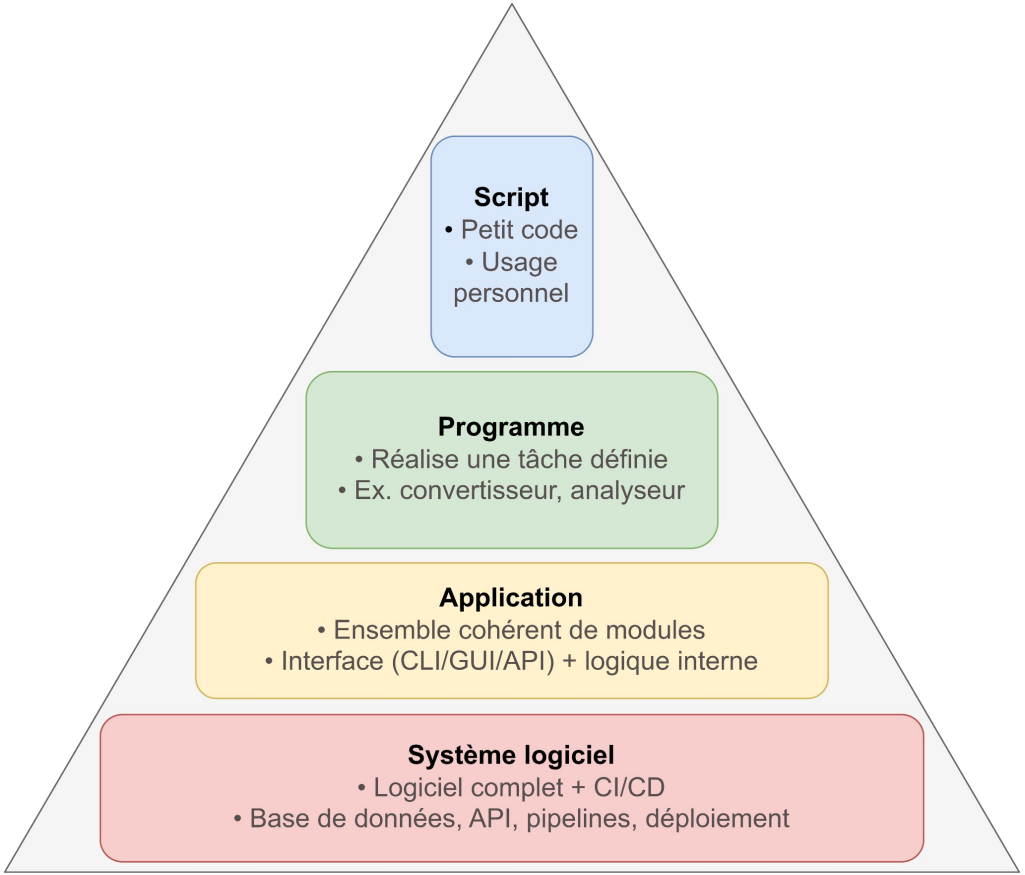
Programme
un fichier exécutable ou un script

```c
/*
 * This line basically imports the "stdio" header file, part of
 * the standard library. It provides input and output functionality
 * to the program.
 */
#include <stdio.h>

/*
 * Function (method) declaration. This outputs "Hello, world\n" to
 * standard output when invoked.
 */
void sayHello(void) {
    // printf() in C outputs the specified text (with optional
    // formatting options) when invoked.
    printf("Hello, world!\n");
}

/*
 * This is a "main function". The compiled program will run the code
 * defined here.
 */
int main(void)
{
    // Invoke the sayHello() function.
    sayHello();
    return 0;
}
```

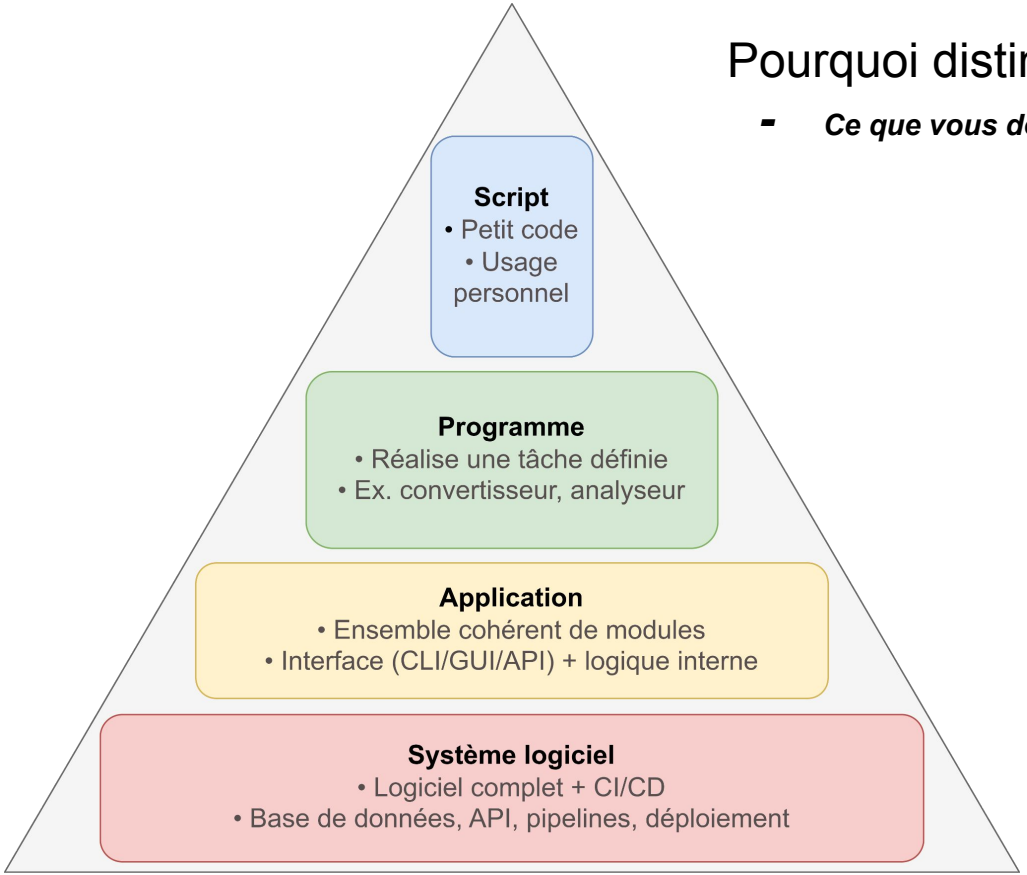
Logiciel
un ensemble **organisé**



Les différentes échelles



Les différentes échelles

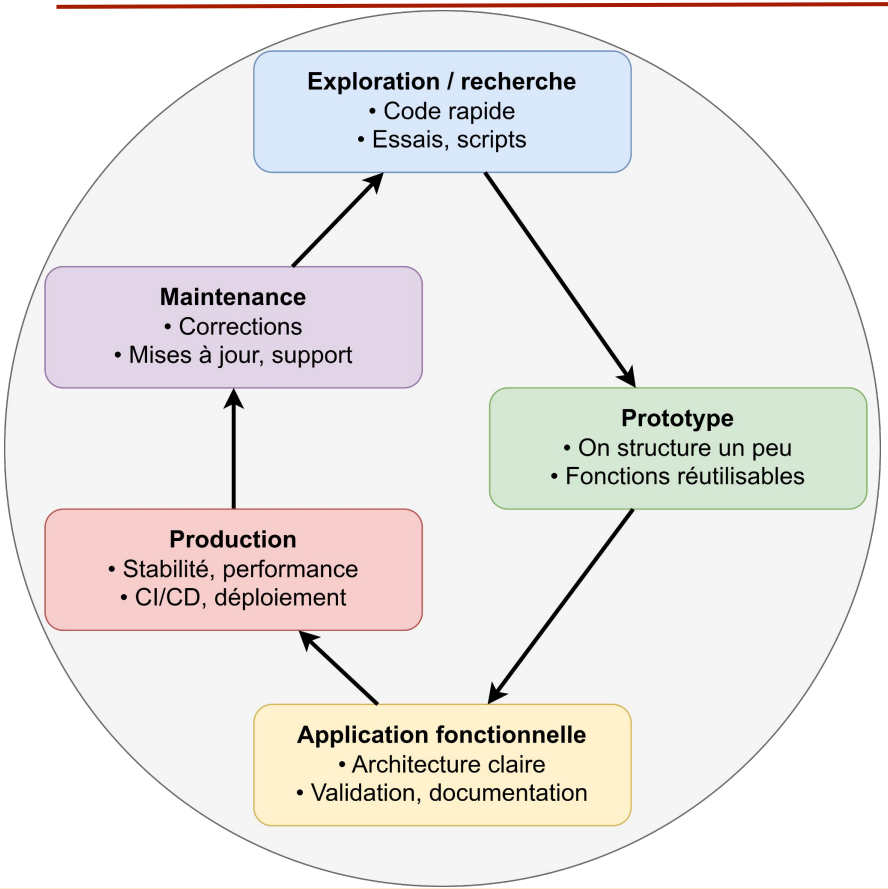


Pourquoi distinguer script / programme / application ?

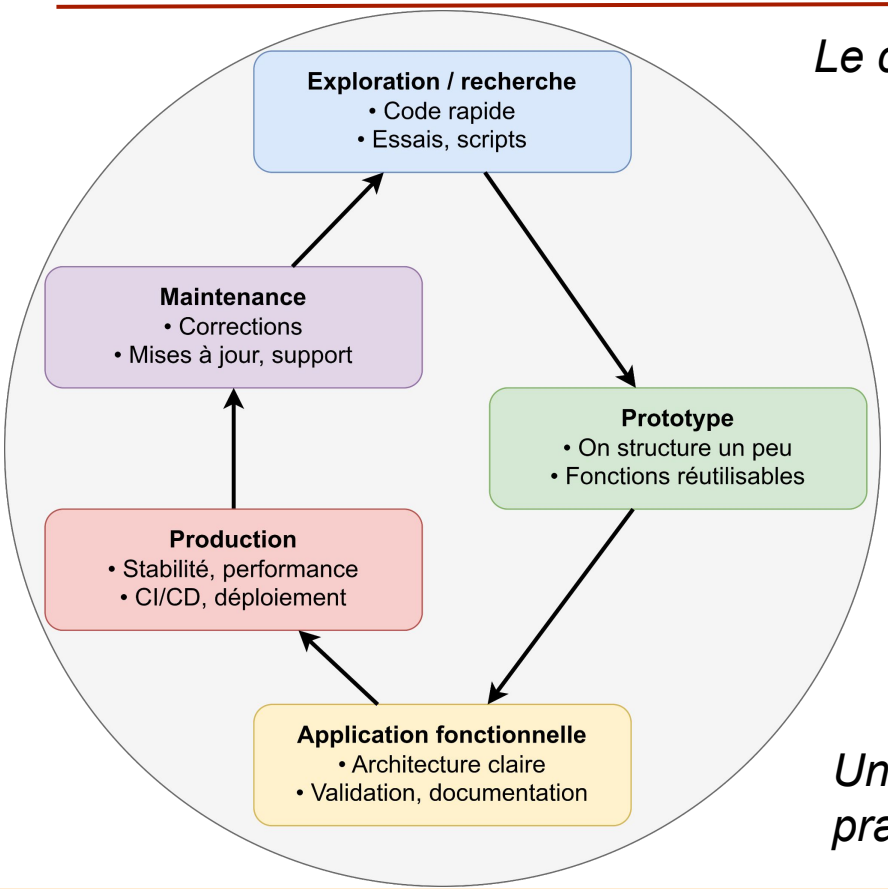
- ***Ce que vous développez n'est pas un script quand d'autres doivent l'utiliser.***

Niveau	Robustesse	Tests & Doc	Scénario d'utilisation
Script	minimal	quasi nulle	Usage local, non partagé
Programme	moyenne	basique	Petites équipes
Application	élevée	obligatoire	Utilisateurs multiples
Système logiciel	critique	complet (CI/CD)	Production, pipeline durable

Cycle de vie d’un logiciel



Cycle de vie d'un logiciel



Le cycle de vie d'un logiciel dépend de son contexte.

- **Cycle scientifique**
  - Rapidité
  - flexibilité
  - prototypage rapide
  - Priorités: résultats, exploration
- **Cycle industriel**
  - Stabilité
  - Qualité
  - Sécurité
  - Priorités: prédictibilité, longévité

Un logiciel scientifique mûrit et finit par adopter les pratiques industrielles lorsqu'il devient critique.

La qualité logicielle

- **Qu'est ce qu'un logiciel de qualité ?**

- Un logiciel de qualité est…
  - Fiable (ne casse pas à la moindre erreur)
  - Durable (évolue sans s'effondrer)
  - Compréhensible (lisible, clair)
  - Utilisable (adapté à ses utilisateurs)

La qualité logicielle

- **Qu'est ce qu'un logiciel de qualité ?**

- Un logiciel de qualité est…
  - Fiable (ne casse pas à la moindre erreur)
  - Durable (évolue sans s'effondrer)
  - Compréhensible (lisible, clair)
  - Utilisable (adapté à ses utilisateurs)

- Dans un projet scientifique, un logiciel doit :
  - survivre à plusieurs campagnes d'observation
  - s'adapter à de nouveaux instruments ou formats
  - être compris par les futurs doctorants / chercheurs
  - produire des résultats fiables et traçables

La qualité logicielle

- 6 aspect de qualité essentielles:

1. Robustesse
2. Maintenabilité
3. Performance / Scalabilité
4. Testabilité
5. Portabilité

Qualité logicielle - Robustesse

Entrées imparfaites

- Bruit
- donnée manquante
- formats partiels

Exemples:

- fichier de donnée incomplet
- série temporelle avec des NaN

Situations inattendues

- Exceptions
- erreurs I/O
- Situations imprévues

Exemples:

- division par zéro
- connexion réseau perdu
- mauvaise permission de fichier

Valeurs limites

- limites mathématiques
- conditions extrêmes

Exemples:

- valeur proche de zéro
- flux saturé
- horizon temporel très long

Le logiciel ne doit jamais surprendre.
Il doit résister aux erreurs.
Il doit échouer proprement.

Qualité logicielle - Maintenabilité

```python
import math, sys

def run(x):
    r = []
    for v in x:
        if v is None or v == "":
            continue
        v = float(v)
        v = (v * 0.00123) + 0.42
        if v < 0:
            v = 0
        r.append(v)
    s = 0
    for v in r:
        s += v
    m = s / len(r)
    s2 = 0
    for v in r:
        s2 += (v - m) ** 2
    s2 = math.sqrt(s2 / len(r))
    if s2 > 0.1:
        print("ALERT", m, s2)
    else:
        print("OK", m, s2)

if __name__ == "__main__":
    data = []
    for a in sys.argv[1:]:
        data.append(a)
    run(data)

```

Qualité logicielle - Maintenabilité

```python
import math, sys

def run(x):
    r = []
    for v in x:
        if v is None or v == "":
            continue
        v = float(v)
        v = (v * 0.00123) + 0.42
        if v < 0:
            v = 0
        r.append(v)
    s = 0
    for v in r:
        s += v
    m = s / len(r)
    s2 = 0
    for v in r:
        s2 += (v - m) ** 2
    s2 = math.sqrt(s2 / len(r))
    if s2 > 0.1:
        print("ALERT", m, s2)
    else:
        print("OK", m, s2)

if __name__ == "__main__":
    data = []
    for a in sys.argv[1:]:
        data.append(a)
    run(data)
```

```
$ python bad.py
Traceback (most recent call last):
  File "/home/roman/Documents/Work/Mission/Roscoff_11_2025/bad.py", line 30, in <module>
    run(data)
  File "/home/roman/Documents/Work/Mission/Roscoff_11_2025/bad.py", line 16, in run
    m = s / len(r)
        ~~^~~~~~~~
ZeroDivisionError: division by zero
$
```

Qualité logicielle - Maintenabilité

```python
import math, sys

def run(x):
    r = []
    for v in x:
        if v is None or v == "":
            continue
        v = float(v)
        v = (v * 0.00123) + 0.42
        if v < 0:
            v = 0
        r.append(v)
    s = 0
    for v in r:
        s += v
    m = s / len(r)
    s2 = 0
    for v in r:
        s2 += (v - m) ** 2
    s2 = math.sqrt(s2 / len(r))
    if s2 > 0.1:
        print("ALERT", m, s2)
    else:
        print("OK", m, s2)

if __name__ == "__main__":
    data = []
    for a in sys.argv[1:]:
        data.append(a)
    run(data)
```

```
$ python bad.py
Traceback (most recent call last):
  File "/home/roman/Documents/Work/Mission/Roscoff_11_2025/bad.py", line 30, in <module>
    run(data)
  File "/home/roman/Documents/Work/Mission/Roscoff_11_2025/bad.py", line 16, in run
    m = s / len(r)
        ~~^~~~~~~~
ZeroDivisionError: division by zero
$
```

```
$ python bad.py abc
Traceback (most recent call last):
  File "/home/roman/Documents/Work/Mission/Roscoff_11_2025/bad.py", line 30, in <module>
    run(data)
  File "/home/roman/Documents/Work/Mission/Roscoff_11_2025/bad.py", line 8, in run
    v = float(v)
        ^^^^^^^^
ValueError: could not convert string to float: 'abc'
$
```

Qualité logicielle - Maintenabilité

```python
import math, sys

def run(x):
    r = []
    for v in x:
        if v is None or v == "":
            continue
        v = float(v)
        v = (v * 0.00123) + 0.42
        if v < 0:
            v = 0
        r.append(v)
    s = 0
    for v in r:
        s += v
    m = s / len(r)
    s2 = 0
    for v in r:
        s2 += (v - m) ** 2
    s2 = math.sqrt(s2 / len(r))
    if s2 > 0.1:
        print("ALERT", m, s2)
    else:
        print("OK", m, s2)

if __name__ == "__main__":
    data = []
    for a in sys.argv[1:]:
        data.append(a)
    run(data)
```

```
$ python bad.py abc
Traceback (most recent call last):
  File "/home/roman/Documents/Work/Mission/Roscoff_11_2025/bad.py", line 30, in <module>
    run(data)
  File "/home/roman/Documents/Work/Mission/Roscoff_11_2025/bad.py", line 8, in run
    v = float(v)
        ^^^^^^^^
ValueError: could not convert string to float: 'abc'
$
```

```
$ python bad.py nan
OK nan nan
$
```

https://www.destroyallsoftware.com/talks/wat

Qualité logicielle - Maintenabilité

```python
import math, sys

def run(x):
    r = []
    for v in x:
        if v is None or v == "":
            continue
        v = float(v)
        v = (v * 0.00123) + 0.42
        if v < 0:
            v = 0
        r.append(v)
    s = 0
    for v in r:
        s += v
    m = s / len(r)
    s2 = 0
    for v in r:
        s2 += (v - m) ** 2
    s2 = math.sqrt(s2 / len(r))
    if s2 > 0.1:
        print("ALERT", m, s2)
    else:
        print("OK", m, s2)

if __name__ == "__main__":
    data = []
    for a in sys.argv[1:]:
        data.append(a)
    run(data)
```

```
$ python bad.py nan
OK nan nan
$
```

```
$ python bad.py 50 60 70
OK 0.4938 0.010042907945411034
$
```

```
$ python bad.py 50 60 300
ALERT 0.5881 0.14214647375154965
$
```

Qualité logicielle - Maintenabilité

```python
from __future__ import annotations

import math
from dataclasses import dataclass
from typing import Iterable, List


# ==== Configuration & constantes ====

ADC_GAIN = 0.00123      # V / ADU
ADC_OFFSET = 0.42       # V
STD_ALERT_THRESHOLD = 0.10  # V


# ==== Modèle de donnée ====

@dataclass
class RawSample:
    """Une mesure brute lue depuis la chaîne d'acquisition."""
    value: float


@dataclass
class CalibratedSample:
    """Une mesure convertie en unités physiques (volts)."""
    volts: float

```

Qualité logicielle - Maintenabilité

```python
from __future__ import annotations

import math
from dataclasses import dataclass
from typing import Iterable, List


# ==== Configuration & constantes ====

ADC_GAIN = 0.00123      # V / ADU
ADC_OFFSET = 0.42       # V
STD_ALERT_THRESHOLD = 0.10  # V


# ==== Modèle de donnée ====

@dataclass
class RawSample:
    """Une mesure brute lue depuis la chaîne d'acquisition."""
    value: float


@dataclass
class CalibratedSample:
    """Une mesure convertie en unités physiques (volts)."""
    volts: float
```

```python
# ==== Fonctions métier ====

def parse_raw_values(args: Iterable[str]) -> List[RawSample]:
    """Transforme une liste de chaînes en valeurs brutes numériques.

    Ignore les entrées vides ou invalides au lieu de planter.
    """
    samples: List[RawSample] = []
    for arg in args:
        if not arg:
            continue
        try:
            samples.append(RawSample(float(arg)))
        except ValueError:
            # On loguerait un warning dans un vrai logiciel
            continue
    return samples


def calibrate(sample: RawSample) -> CalibratedSample:
    """Convertit une mesure brute en tension calibrée.

    La loi de calibration est : V = G * ADU + offset.
    """
    volts = ADC_GAIN * sample.value + ADC_OFFSET
    volts = max(volts, 0.0)  # on évite les valeurs négatives
    return CalibratedSample(volts=volts)


def mean_and_std(values: Iterable[float]) -> tuple[float, float]:
    """Calcule moyenne et écart-type d'une séquence de réels."""
    data = list(values)
    if not data:
        raise ValueError("Impossible de calculer des statistiques sur une liste vide.")
    mean = sum(data) / len(data)
    var = sum((x - mean) ** 2 for x in data) / len(data)
    std = math.sqrt(var)
    return mean, std
```

Qualité logicielle - Maintenabilité

```
102
103 def analyse_samples(raw_args: Iterable[str]) -> None:
104     """Pipeline complet : parsing, calibration, statistiques, décision."""
105     raw = parse_raw_values(raw_args)
106     calibrated = [calibrate(s).volts for s in raw]
107
108     mean, std = mean_and_std(calibrated)
109
110     status = "ALERT" if std > STD_ALERT_THRESHOLD else "OK"
111     print(f"{status} | mean={mean:.3f} V | std={std:.3f} V")
112
113
114 # ==== Point d'entrée ====
115
116 if __name__ == "__main__":
117     import sys
118
119     analyse_samples(sys.argv[1:])
120
```

Qualité logicielle - Maintenabilité

```python

def analyse_samples(raw_args: Iterable[str]) -> None:
    """Pipeline complet : parsing, calibration, statistiques, décision."""
    raw = parse_raw_values(raw_args)
    calibrated = [calibrate(s).volts for s in raw]

    mean, std = mean_and_std(calibrated)

    status = "ALERT" if std > STD_ALERT_THRESHOLD else "OK"
    print(f"{status} | mean={mean:.3f} V | std={std:.3f} V")


# ==== Point d'entrée ====

if __name__ == "__main__":
    import sys

    analyse_samples(sys.argv[1:])

```

```
$ python good.py
Traceback (most recent call last):
  File "/home/roman/Documents/Work/Mission/Roscoff_11_2025/good.py", line 119, in <module>
    analyse_samples(sys.argv[1:])
  File "/home/roman/Documents/Work/Mission/Roscoff_11_2025/good.py", line 108, in analyse_samples
    mean, std = mean_and_std(calibrated)
                ^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/roman/Documents/Work/Mission/Roscoff_11_2025/good.py", line 96, in mean_and_std
    raise ValueError("Impossible de calculer des statistiques sur une liste vide.")
ValueError: Impossible de calculer des statistiques sur une liste vide.
$
```

Qualité logicielle - Maintenabilité

```python

def analyse_samples(raw_args: Iterable[str]) -> None:
    """Pipeline complet : parsing, calibration, statistiques, décision."""
    raw = parse_raw_values(raw_args)
    calibrated = [calibrate(s).volts for s in raw]

    mean, std = mean_and_std(calibrated)

    status = "ALERT" if std > STD_ALERT_THRESHOLD else "OK"
    print(f"{status} | mean={mean:.3f} V | std={std:.3f} V")


# ==== Point d'entrée ====

if __name__ == "__main__":
    import sys

    analyse_samples(sys.argv[1:])

```

```
$ python good.py 50 60 70
OK | mean=0.494 V | std=0.010 V
$
```

Qualité logicielle - Maintenabilité

```python

def analyse_samples(raw_args: Iterable[str]) -> None:
    """Pipeline complet : parsing, calibration, statistiques, décision."""
    raw = parse_raw_values(raw_args)
    calibrated = [calibrate(s).volts for s in raw]

    mean, std = mean_and_std(calibrated)

    status = "ALERT" if std > STD_ALERT_THRESHOLD else "OK"
    print(f"{status} | mean={mean:.3f} V | std={std:.3f} V")


# ==== Point d'entrée ====

if __name__ == "__main__":
    import sys

    analyse_samples(sys.argv[1:])

```

- code lisible
- Documentation
- Typage
- Modulaire
- Dépendances claires

Un logiciel maintenable peut évoluer sans réécriture complète.

Qualité logicielle - Performance

Performance : Aller vite sur un dataset donné

- Temps d'exécution raisonnable
- Optimisations CPU/GPU, I/O, allocations
- Bons choix algorithmiques (éviter $O(N^2)$ si possible)

Exemples: *Extraire les particules d'un évènement en quelque millisecondes*

Qualité logicielle - Performance / Scalabilité

Performance : Aller vite sur un dataset donné
- Temps d'exécution raisonnable
- Optimisations CPU/GPU, I/O, allocations
- Bons choix algorithmiques (éviter $O(N^2)$ si possible)

Exemples: *Extraire les particules d'un évènement en quelque millisecondes*

Exemples: *Gérer 100 kHz, 1 MHz, 100 MHz jusqu'à 1 GHz d'événements*

Scalabilité : Performance stable quand le volume croit
- Supporter un volume de données ×1000.
- Gestion mémoire, streaming, parallélisation

Qualité logicielle - Testabilité

Un logiciel doit être conçu pour être testé dès le départ.

Qualité logicielle - Testabilité

Un logiciel doit être conçu pour être testé dès le départ.

Comment rendre un logiciel testable ?

- interfaces claires
- Composants indépendants
- Résultats déterministes
- Pas d'effets cachés (I/O, états globales)

```
81
82  def calibrate(sample: RawSample) -> CalibratedSample:
83      """Convertit une mesure brute en tension calibrée.
84
85      La loi de calibration est : V = G * ADU + offset.
86      """
87      volts = ADC_GAIN * sample.value + ADC_OFFSET
88      volts = max(volts, 0.0)  # on évite les valeurs négatives
89      return CalibratedSample(volts=volts)
90
```

Qualité logicielle - Testabilité

Un logiciel doit être conçu pour être testé dès le départ.

Comment rendre un logiciel testable ?

- interfaces claires
- Composants indépendants
- Résultats déterministes
- Pas d'effets cachés (I/O, états globales)

Comment on teste ?

- Tests unitaire
- Tests d'intégration

```
81
82  def calibrate(sample: RawSample) -> CalibratedSample:
83      """Convertit une mesure brute en tension calibrée.
84
85      La loi de calibration est : V = G * ADU + offset.
86      """
87      volts = ADC_GAIN * sample.value + ADC_OFFSET
88      volts = max(volts, 0.0)  # on évite les valeurs négatives
89      return CalibratedSample(volts=volts)
90
```

```
103 def analyse_samples(raw_args: Iterable[str]) -> None:
104     """Pipeline complet : parsing, calibration, statistiques, décision."""
105     raw = parse_raw_values(raw_args)
106     calibrated = [calibrate(s).volts for s in raw]
107
108     mean, std = mean_and_std(calibrated)
109
110     status = "ALERT" if std > STD_ALERT_THRESHOLD else "OK"
111     print(f"{status} | mean={mean:.3f} V | std={std:.3f} V")
```

Qualité logicielle - Testabilité

Un logiciel doit être conçu pour être testé dès le départ.

Comment rendre un logiciel testable ?

- interfaces claires
- Composants indépendants
- Résultats déterministes
- Pas d'effets cachés (I/O, états globales)

Comment on teste ?

- Tests unitaire
- Tests d'intégration
- Jeu de données de référence
- CI automatique (forges)

```
81
82  def calibrate(sample: RawSample) -> CalibratedSample:
83      """Convertit une mesure brute en tension calibrée.
84
85      La loi de calibration est : V = G * ADU + offset.
86      """
87      volts = ADC_GAIN * sample.value + ADC_OFFSET
88      volts = max(volts, 0.0)  # on évite les valeurs négatives
89      return CalibratedSample(volts=volts)
90
```

```
103 def analyse_samples(raw_args: Iterable[str]) -> None:
104     """Pipeline complet : parsing, calibration, statistiques, décision."""
105     raw = parse_raw_values(raw_args)
106     calibrated = [calibrate(s).volts for s in raw]
107
108     mean, std = mean_and_std(calibrated)
109
110     status = "ALERT" if std > STD_ALERT_THRESHOLD else "OK"
111     print(f"{status} | mean={mean:.3f} V | std={std:.3f} V")
```

Qualité logicielle - Testabilité

Un logiciel doit être conçu pour être testé dès le départ.

Comment rendre un logiciel testable ?

- interfaces claires
- Composants indépendants
- Résultats déterministes
- Pas d'effets cachés (I/O, états globales)

Comment on teste ?

- Tests unitaire
- Tests d'intégration
- Jeu de données de référence
- CI automatique (forges)

```
81
82  def calibrate(sample: RawSample) -> CalibratedSample:
83      """Convertit une mesure brute en tension calibrée.
84
85      La loi de calibration est : V = G * ADU + offset.
86      """
87      volts = ADC_GAIN * sample.value + ADC_OFFSET
88      volts = max(volts, 0.0)  # on évite les valeurs négatives
89      return CalibratedSample(volts=volts)
90
```

```
103 def analyse_samples(raw_args: Iterable[str]) -> None:
104     """Pipeline complet : parsing, calibration, statistiques, décision."""
105     raw = parse_raw_values(raw_args)
106     calibrated = [calibrate(s).volts for s in raw]
107
108     mean, std = mean_and_std(calibrated)
109
110     status = "ALERT" if std > STD_ALERT_THRESHOLD else "OK"
111     print(f"{status} | mean={mean:.3f} V | std={std:.3f} V")
```

Reproducibilité

Qualité logicielle - Portabilité

Problèmatique : Architecture matérielle et logiciel hétérogène

- Linux, macOS, Windows
- environement local, serveur, cluster HPC
- matériel CPU/GPU (AMD, NVIDIA, INTEL, ARM…)

Qualité logicielle - Portabilité

Problèmatique : Architecture matérielle et logiciel hétérogène

- Linux, macOS, Windows
- environement local, serveur, cluster HPC
- matériel CPU/GPU (AMD, NVIDIA, INTEL, ARM…)

Garantir les mêmes résultats

- dépendances identiques
  - python
  - système
- mêmes configurations
- Gestions d'environnement
  - venv
  - conda
  - pdm

Garantir le lancement du logiciel

- Documentation claire
- Virtualisation
  - Docker
  - Singularity
- portabilité CPU/GPU
  - NumPy ↔ CuPy
  - Pytorch / tensorflow
  - SYCL / oneAPI / Kokkos

Qualité logicielle - Résumé

Robustesse — résister aux entrées imparfaites et aux situations imprévues

Maintenabilité — pouvoir évoluer sans tout casser

Performance & **Scalabilité** — rester efficace, même quand la charge augmente

Testabilité — garantir la vérification et la constance des résultats

Portabilité — fonctionner partout, du laptop au cluster HPC

Penser l’architecture logicielle

Architecture = organisation intelligente du logiciel / code

Sans architecture :

- maintenabilité et testabilité compromise
- Ajout / mise à jour du code difficile
- fichier principal gigantesque (plusieurs milliers de lignes)
- dépendances entremêlées

3 principes importants en architecture logicielle

1. Séparation des responsabilités

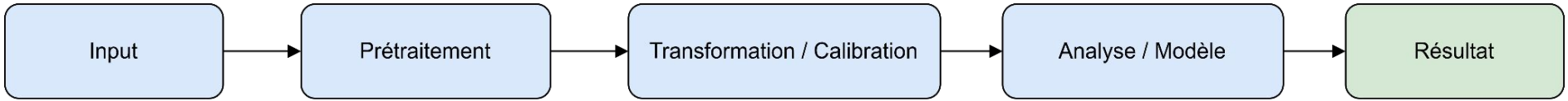
Définir les briques du logiciel

- I/O → Lecture / écriture des données
- Modèle → représentation des données
- Processing → transformer les données
- Analyse → algorithme, traitement sur les données
- Résultats → plots, visualisation des données

3 principes importants en architecture logicielle

1. Séparation des responsabilités
2. **Flux logique des données (dataflow)**

logiciel scientifique → Pipeline, une suite d'étape



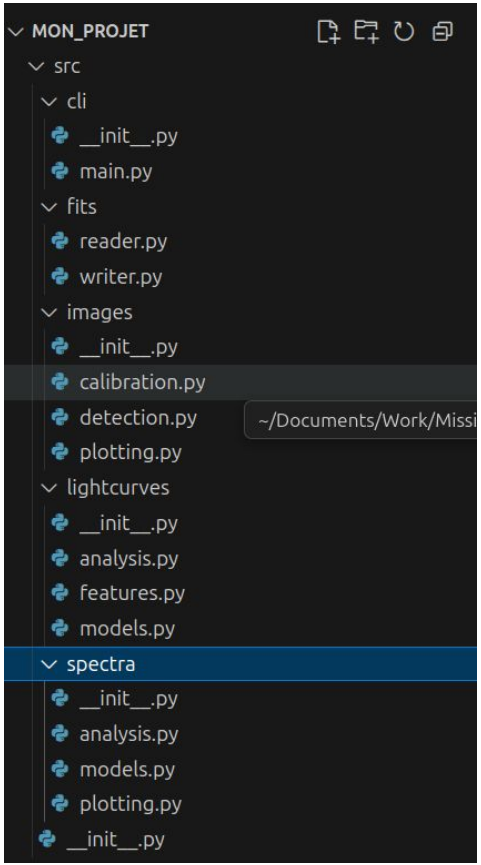
- facile à tester, étape par étape
- lisibilité immédiate

3 principes importants en architecture logicielle

1. Séparation des responsabilités
2. Flux logique des données (dataflow)
3. **Modularité** & encapsulation

   module = bloc de code cohérent, autonome

   servant un objectif via une interface claire



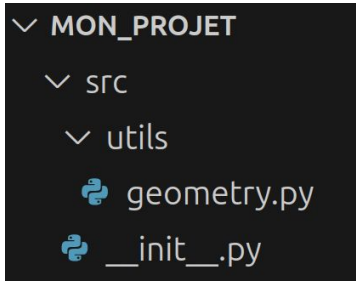
3 principes importants en architecture logicielle

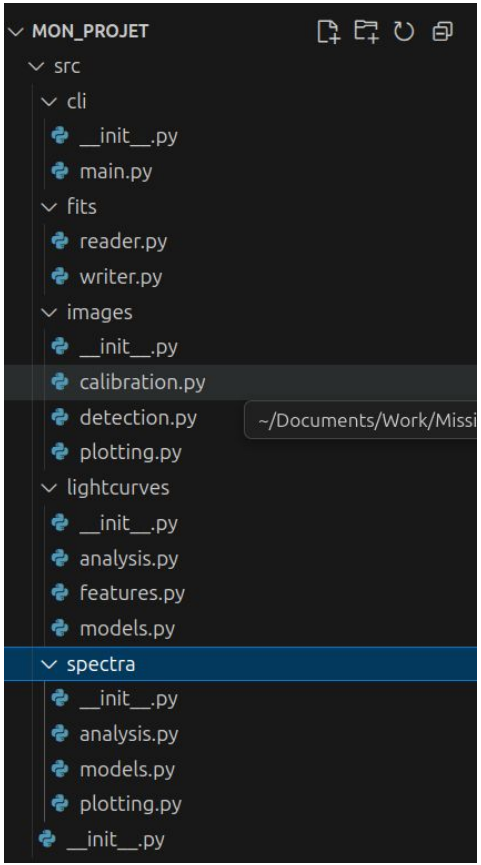
1. Séparation des responsabilités
2. **Flux logique des données (dataflow)**
3. **Modularité** & encapsulation

   module = bloc de code cohérent, autonome

   servant un objectif via une interface claire

Eviter les pièges: “utils/”, “helpers/”, “misc/”, “core/”
Exemple: Une fonction “Convertir RA/Dec en radians”





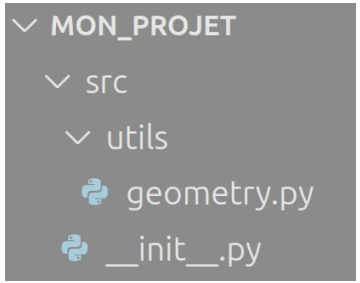
3 principes importants en architecture logicielle

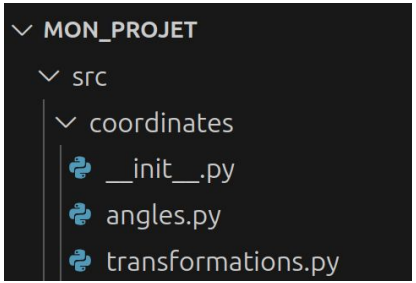
1. Séparation des responsabilités
2. **Flux logique des données (dataflow)**
3. **Modularité** & encapsulation

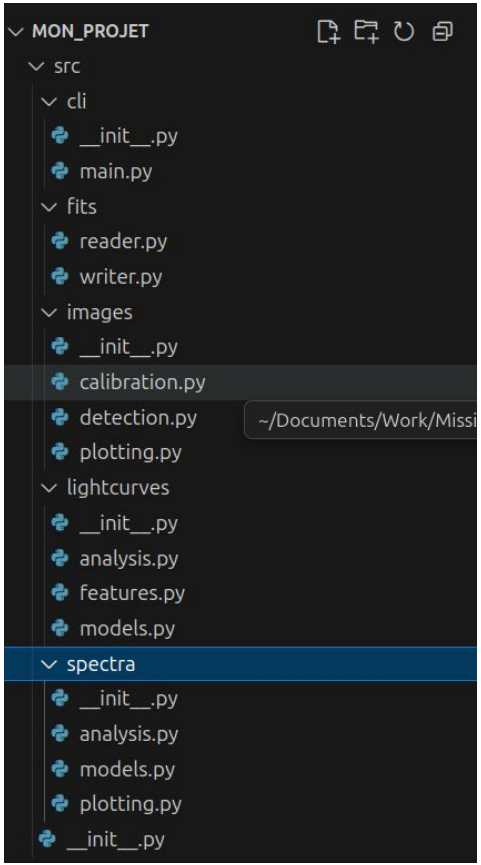
   module = bloc de code cohérent, autonome

   servant un objectif via une interface claire

Eviter les pièges: “utils/”, “helpers/”, “misc/”, “core/”
Exemple: Une fonction “Convertir RA/Dec en radians”







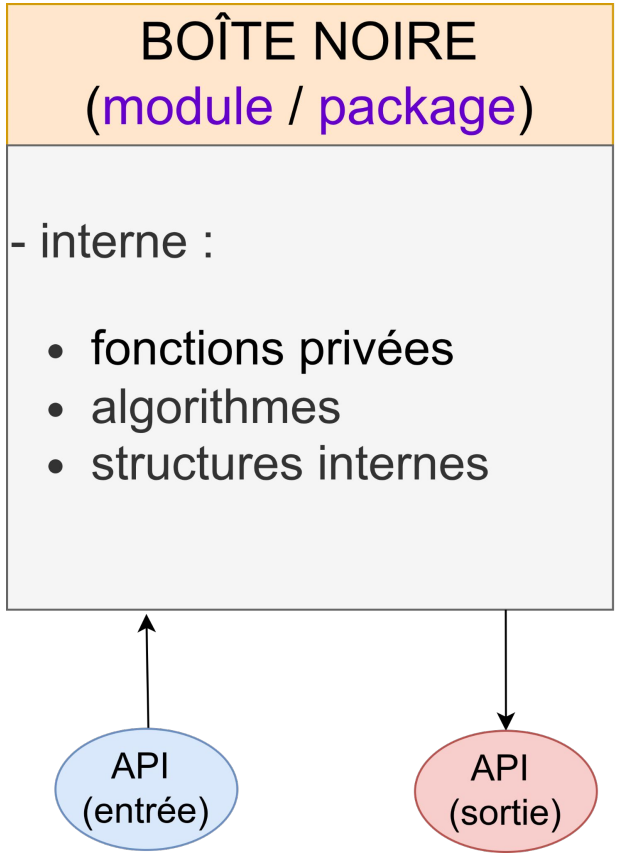
3 principes importants en architecture logicielle

1. Séparation des responsabilités
2. **Flux logique des données (dataflow)**
3. Modularité **& encapsulation**

encapsulation = cacher la complexité interne

et exposer uniquement le nécessaire

⟶ API : Application Programming Interface



Tests : fondations

Pourquoi tester ?

- détections d'erreur → éviter effets cascades
- confiance dans le code → refactoring safe
- documentation exécutable → « exemples vérifiés »
- non-régression → reproductibilité scientifique

Type de tests

Tests unitaires

permet de tester des fonctions isolé → *rapide*

```python
def parse_raw_values(args: Iterable[str]) -> List[RawSample]:
    """Transforme une liste de chaînes en valeurs brutes numériques.

    Ignore les entrées vides ou invalides au lieu de planter.
    """
    samples: List[RawSample] = []
    for arg in args:
        if not arg:
            continue
        try:
            samples.append(RawSample(float(arg)))
        except ValueError:
            # On loguerait un warning dans un vrai logiciel
            continue
    return samples
```

Type de tests

Tests unitaires

permet de tester des fonctions isolé → *rapide*

```python
def parse_raw_values(args: Iterable[str]) -> List[RawSample]:
    """Transforme une liste de chaînes en valeurs brutes numériques.

    Ignore les entrées vides ou invalides au lieu de planter.
    """
    samples: List[RawSample] = []
    for arg in args:
        if not arg:
            continue
        try:
            samples.append(RawSample(float(arg)))
        except ValueError:
            # On loguerait un warning dans un vrai logiciel
            continue
    return samples
```

```python
def test_parse_valid_strings(self):
    args = ["1.0", "2.5", "3"]
    samples = parse_raw_values(args)

    assert len(samples) == 3
    assert isinstance(samples[0], RawSample)
    assert [s.value for s in samples] == [1.0, 2.5, 3.0]
```

```python
def test_parse_ignores_empty_and_invalid(self):
    args = ["", "  ", "abc", "42", "NaN", "3.14"]
    samples = parse_raw_values(args)

    values = [s.value for s in samples]

    assert len(samples) == 3
    assert values[0] == 42.0
    assert math.isnan(values[1])
    assert values[2] == 3.14
```

Type de tests

Tests intégration

permet de tester plusieurs modules → *API globale*

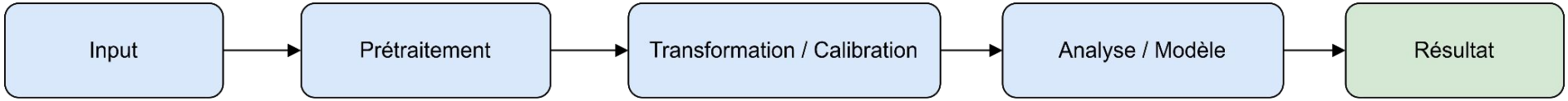
```python
def parse_raw_values(args: Iterable[str]) -> List[RawSample]:
def calibrate(sample: RawSample) -> CalibratedSample:
def mean_and_std(values: Iterable[float]) -> tuple[float, float]:
```

```python
class TestIntegrationPipeline:
    def test_full_pipeline_nominal(self):
        """
        Test d'intégration "heureux" :
        - on part de chaînes de caractères
        - on parse en RawSample
        - on calibre en CalibratedSample
        - on calcule moyenne et écart-type sur les volts
        """

        # 1) Entrée simulée (ex: valeurs ADC en argument de ligne de commande)
        raw_args = ["100", "110", "120"]

        # 2) Parsing → RawSample
        raw_samples = parse_raw_values(raw_args)
        assert [s.value for s in raw_samples] == [100.0, 110.0, 120.0]

        # 3) Calibration → CalibratedSample
        calibrated = [calibrate(s) for s in raw_samples]
        volts = [c.volts for c in calibrated]

        # 4) Statistiques sur les volts
        mean, std = mean_and_std(volts)

        # 5) Calcul attendu "à la main" en utilisant les mêmes constantes
        adus = [100.0, 110.0, 120.0]
        expected_volts = [max(ADC_GAIN * v + ADC_OFFSET, 0.0) for v in adus]
        expected_mean = sum(expected_volts) / len(expected_volts)
        expected_var = sum((v - expected_mean) ** 2 for v in expected_volts) / len(
            expected_volts
        )
        expected_std = math.sqrt(expected_var)

        assert volts == pytest.approx(expected_volts)
        assert mean == pytest.approx(expected_mean)
        assert std == pytest.approx(expected_std)
```

Type de tests

Tests End-to-End

variante du test d'intégration (scénario utilisateur)

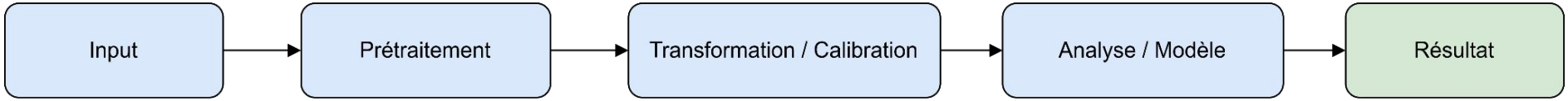



```python
# On récupère le chemin absolu du fichier à tester (good.py)
SCRIPT_PATH = os.path.join(os.path.dirname(__file__), "..", "good.py")


class TestCLIEndToEnd:
    def test_cli_ok_low_dispersion(self):
        """
        Reproduit :
        $ python good.py 120 123 125
        """
        result = subprocess.run(
            [sys.executable, SCRIPT_PATH, "120", "123", "125"],
            capture_output=True,
            text=True,
            check=True,
        )
        assert result.stdout.strip() == "OK | mean=0.571 V | std=0.003 V"
        assert result.stderr == ""
```

Type de tests

Tests End-to-End

variante du test d'intégration (scénario utilisateur)




```python
# On récupère le chemin absolu du fichier à tester (good.py)
SCRIPT_PATH = os.path.join(os.path.dirname(__file__), "..", "good.py")


class TestCLIEndToEnd:
    def test_cli_ok_low_dispersion(self):
        """
        Reproduit :
        $ python good.py 120 123 125
        """
        result = subprocess.run(
            [sys.executable, SCRIPT_PATH, "120", "123", "125"],
            capture_output=True,
            text=True,
            check=True,
        )
        assert result.stdout.strip() == "OK | mean=0.571 V | std=0.003 V"
        assert result.stderr == ""
```

```
$ python good.py 120 123 125
OK | mean=0.571 V | std=0.003 V
$
```

Type de tests

Property based testing (https://hypothesis.readthedocs.io/en/latest/)

*Tester des **invariants** : on génère des dizaines d'entrées aléatoires pour s'assurer que les lois mathématiques restent vraies pour toutes les valeurs.*

```
32  # ============================
33  # Stratégies Hypothesis
34  # ============================
35
36  finite_floats = st.floats(
37      min_value=-1e6,
38      max_value=1e6,
39      allow_nan=False,
40      allow_infinity=False,
41  )
42
43  non_empty_float_lists = st.lists(
44      finite_floats,
45      min_size=1,
46      max_size=50,
47  )
```

Type de tests

Property based testing (https://hypothesis.readthedocs.io/en/latest/)

*Tester des **invariants** : on génère des dizaines d'entrées aléatoires pour s'assurer que les lois mathématiques restent vraies pour toutes les valeurs.*

```python
# ============================
# Stratégies Hypothesis
# ============================

finite_floats = st.floats(
    min_value=-1e6,
    max_value=1e6,
    allow_nan=False,
    allow_infinity=False,
)

non_empty_float_lists = st.lists(
    finite_floats,
    min_size=1,
    max_size=50,
)
```

```python
@given(non_empty_float_lists)
def test_mean_and_std_basic_invariants(values):
    """
    Propriétés de base :
    - l'écart-type est toujours >= 0
    - la moyenne est (à flot près) dans l'intervalle [min, max]
    """
    mean, std = mean_and_std(values)

    assert std >= 0.0

    mn = min(values)
    mx = max(values)

    # La moyenne d'une liste finie est dans [min, max], à une petite dérive près
    assert mn - MEAN_BOUNDS_TOL <= mean <= mx + MEAN_BOUNDS_TOL
```

Type de tests

Property based testing (https://hypothesis.readthedocs.io/en/latest/)

*Tester des **invariants** : on génère des dizaines d'entrées aléatoires pour s'assurer que les lois mathématiques restent vraies pour toutes les valeurs.*

```python
# ============================
# Stratégies Hypothesis
# ============================

finite_floats = st.floats(
    min_value=-1e6,
    max_value=1e6,
    allow_nan=False,
    allow_infinity=False,
)

non_empty_float_lists = st.lists(
    finite_floats,
    min_size=1,
    max_size=50,
)
```

```python
@given(non_empty_float_lists, finite_floats)
def test_mean_and_std_translation_invariance(values, shift):
    """
    Propriété : translation.
    mean(x + c) = mean(x) + c
    std(x + c)  = std(x)

    On exclut les séquences constantes pour ce test, car
    le cas "std ~ 0" est déjà testé séparément.
    """
    # évite les listes constantes (tous les éléments identiques)
    assume(len(set(values)) > 1)

    shifted = [x + shift for x in values]

    mean1, std1 = mean_and_std(values)
    mean2, std2 = mean_and_std(shifted)

    # moyenne : tolérance relative + absolue
    assert mean2 == pytest.approx(
        mean1 + shift, rel=MEAN_REL_TOL, abs=MEAN_ABS_TOL
    )

    # std : doit rester ~ la même, à une petite erreur flottante près
    assert std2 == pytest.approx(
        std1, rel=STD_REL_TOL, abs=STD_ABS_TOL
    )
```

Type de tests

Property based testing (https://hypothesis.readthedocs.io/en/latest/)

*Tester des **invariants** : on génère des dizaines d'entrées aléatoires pour s'assurer que les lois mathématiques restent vraies pour toutes les valeurs.*

```python
# ============================
# Stratégies Hypothesis
# ============================

finite_floats = st.floats(
    min_value=-1e6,
    max_value=1e6,
    allow_nan=False,
    allow_infinity=False,
)

non_empty_float_lists = st.lists(
    finite_floats,
    min_size=1,
    max_size=50,
)
```

```python
@given(non_empty_float_lists, finite_floats)
def test_mean_and_std_scaling(values, scale):
    """
    Propriété : homothétie.
    mean(a * x) = a * mean(x)
    std(a * x) = |a| * std(x)

    On exclut les listes constantes pour cette propriété, car on a
    un test spécifique pour ce cas (std ~ 0).
    """
    # On évite les séquences constantes pour ce test
    assume(len(set(values)) > 1)

    scaled = [scale * x for x in values]

    mean1, std1 = mean_and_std(values)
    mean2, std2 = mean_and_std(scaled)

    assert mean2 == pytest.approx(
        scale * mean1, rel=MEAN_REL_TOL, abs=MEAN_ABS_TOL
    )
    assert std2 == pytest.approx(
        abs(scale) * std1, rel=STD_REL_TOL, abs=STD_ABS_TOL
    )
```

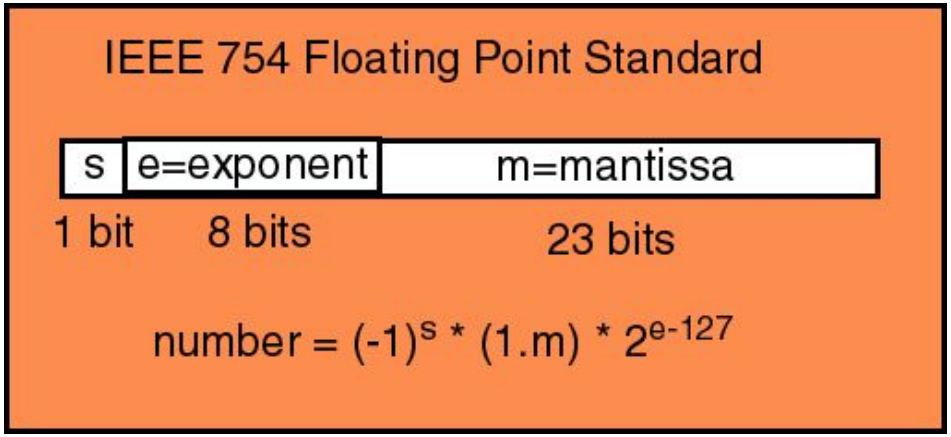
Aparté : Tests et flottant

```python
# ============================

# Tolérance absolue pour les comparaisons sur la moyenne (et limites min/max)
MEAN_ABS_TOL = 1e-9
# Tolérance relative pour les comparaisons de moyenne
MEAN_REL_TOL = 1e-9

# Tolérance absolue pour les comparaisons sur l'écart-type
STD_ABS_TOL = 1e-9
# Tolérance relative pour les comparaisons sur l'écart-type
STD_REL_TOL = 1e-9

# Tolérance spécifique pour "mean dans [min, max]"
MEAN_BOUNDS_TOL = 1e-9
```

```python
    # moyenne : tolérance relative + absolue
    assert mean2 == pytest.approx(
        mean1 + shift, rel=MEAN_REL_TOL, abs=MEAN_ABS_TOL
    )

    # std : doit rester ~ la même, à une petite erreur flottante près
    assert std2 == pytest.approx(
        std1, rel=STD_REL_TOL, abs=STD_ABS_TOL
    )
```

Aparté : Tests et flottant

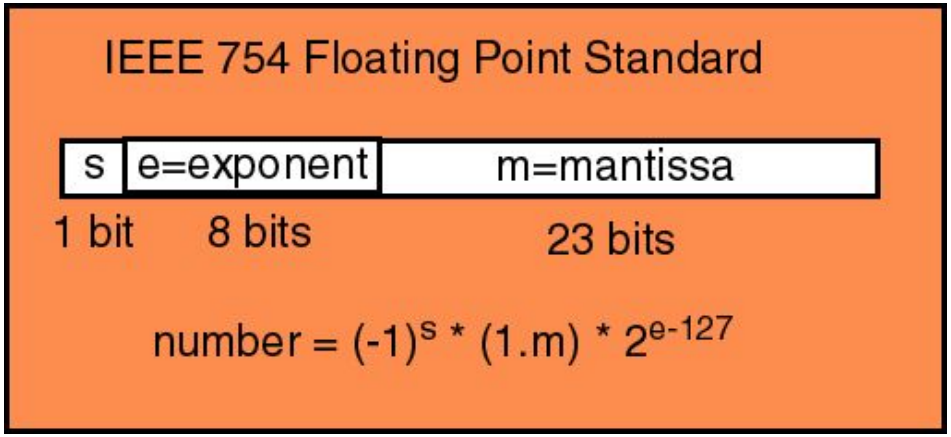
Nombre flottant : Représentation en base 2 (Norme IEEE 754) $\mathbb{F} \subsetneq \mathbb{R}$




```
In [1]: (1e20 + -1e20) + 1
```

Aparté : Tests et flottant

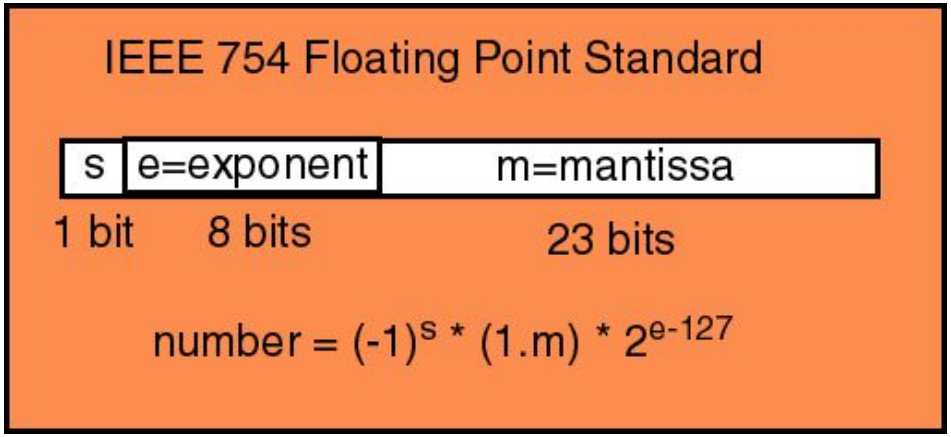
Nombre flottant : Représentation en base 2 (Norme IEEE 754) $\mathbb{F} \subsetneq \mathbb{R}$




```
In [1]: (1e20 + -1e20) + 1
Out[1]: 1.0
```

Aparté : Tests et flottant

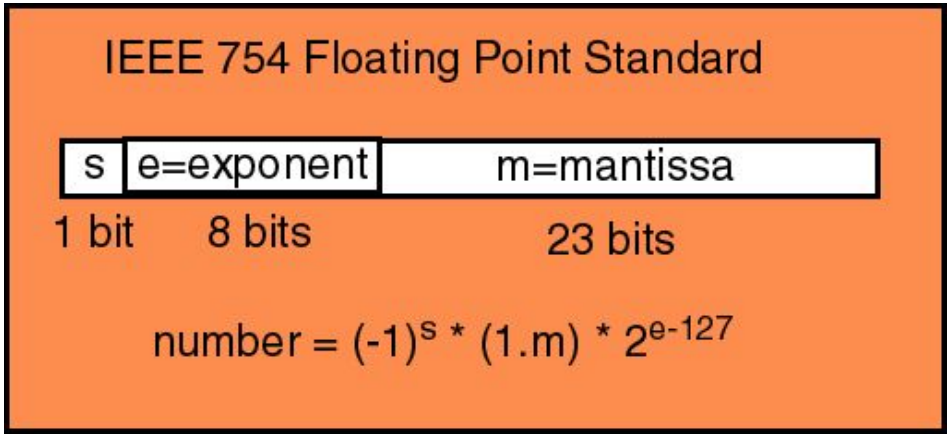
Nombre flottant : Représentation en base 2 (Norme IEEE 754) $\mathbb{F} \subsetneq \mathbb{R}$




```
In [1]: (1e20 + -1e20) + 1
Out[1]: 1.0

In [2]: 1e20 + (-1e20 + 1)
```

Aparté : Tests et flottant

Nombre flottant : Représentation en base 2 (Norme IEEE 754) $\mathbb{F} \subsetneq \mathbb{R}$



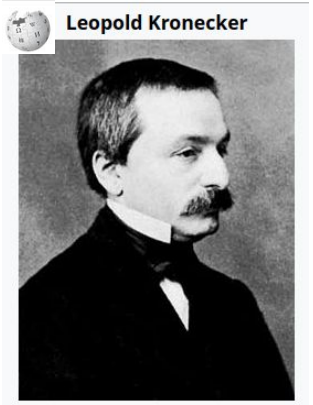

```
In [1]: (1e20 + -1e20) + 1
Out[1]: 1.0

In [2]: 1e20 + (-1e20 + 1)
Out[2]: 0.0
```

Aparté : Tests et flottant

Nombre flottant : Représentation en base 2 (Norme IEEE 754) $\mathbb{F} \subsetneq \mathbb{R}$

- Exemple : `0.1 + 0.2 != 0.3`
- Non-associativité : `(a + b) + c ≠ a + (b + c)`
- Non-distributivité : `a * (b + c) ≠ a*b + a*c`

```
In [1]: (1e20 + -1e20) + 1
Out[1]: 1.0

In [2]: 1e20 + (-1e20 + 1)
Out[2]: 0.0
```



Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk
"God made the integers, all else is the work of man"

- https://cc-fr.eu/webinars-replay/
- https://www.youtube.com/watch?v=JWe74pqFXWE

Aparté : Tests et flottant

Nombre flottant : Représentation en base 2 (Norme IEEE 754)

$\mathbb{F} \subsetneq \mathbb{R}$

- Exemple : `0.1 + 0.2 != 0.3`
- Non-associativité : `(a + b) + c ≠ a + (b + c)`
- Non-distributivité : `a * (b + c) ≠ a*b + a*c`

Exemples d'erreur d'arrondi très connu

- Missiles Patriot, 1991 : erreur ~ 500 m après 100h d'erreur d'arrondie
- Vancouver Stock Exchange, 1982 : erreur de 52 % après deux ans, 524.811$ à la place de 1098.892$

- https://cc-fr.eu/webinars-replay/
- https://www.youtube.com/watch?v=JWe74pqFXWE

Aparté : Tests et flottant

Cursed Cancellation

⇒ higher degree polynomials can degrade high resolutions (*cf* RUMP)

$$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

with $x = 77617$, $y = 33096$ (coprime integers)

[S.M. RUMP, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

- https://cc-fr.eu/webinars-replay/
- https://www.youtube.com/watch?v=JWe74pqFXWE

Aparté : Tests et flottant

Cursed Cancellation

⇒ higher degree polynomials can degrade high resolutions (*cf* RUMP)

$$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

with $x = 77617$, $y = 33096$ (coprime integers)

[S.M. RUMP, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float`: $P = -6.33825300e + 29$

Name	Common name	Radix	Digits[c]	Decimal digits[d]
binary32	Single precision	2	24	7.22

- https://cc-fr.eu/webinars-replay/
- https://www.youtube.com/watch?v=JWe74pqFXWE

Aparté : Tests et flottant

Cursed Cancellation

⇒ higher degree polynomials can degrade high resolutions (*cf* RUMP)

$$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

with $x = 77617$, $y = 33096$ (coprime integers)

[S.M. RUMP, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float:` $P = -6.33825300e + 29$
`double:` $P = -1.1805916207174113e + 021$

Name	Common name	Radix	Digits[c]	Decimal digits[d]
binary32	Single precision	2	24	7.22
binary64	Double precision	2	53	15.95

- https://cc-fr.eu/webinars-replay/
- https://www.youtube.com/watch?v=JWe74pqFXWE

Aparté : Tests et flottant

Cursed Cancellation

⇒ higher degree polynomials can degrade high resolutions (*cf* RUMP)

$$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

with $x = 77617$, $y = 33096$ (coprime integers)

[S.M. RUMP, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float:`	$P = -6.33825300e + 29$
`double:`	$P = -1.1805916207174113e + 021$
`long double:`	$P = +5.76460752303423489188e + 17$

Name	Common name	Radix	Digits[c]	Decimal digits[d]
binary32	Single precision	2	24	7.22
binary64	Double precision	2	53	15.95

- https://cc-fr.eu/webinars-replay/
- https://www.youtube.com/watch?v=JWe74pqFXWE

Aparté : Tests et flottant

Cursed Cancellation

⇒ higher degree polynomials can degrade high resolutions (*cf* RUMP)

$$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

with $x = 77617$, $y = 33096$ (coprime integers)

[S.M. RUMP, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float:`	$P = -6.33825300e + 29$
`double:`	$P = -1.1805916207174113e + 021$
`long double:`	$P = +5.764607523034234891 88e + 17$
`quad:`	$P = +1.172603940053178631858834904520 18380$

Name	Common name	Radix	Digits[c]	Decimal digits[d]
binary32	Single precision	2	24	7.22
binary64	Double precision	2	53	15.95
binary128	Quadruple precision	2	113	34.02

- https://cc-fr.eu/webinars-replay/
- https://www.youtube.com/watch?v=JWe74pqFXWE

Aparté : Tests et flottant

Cursed Cancellation

⇒ higher degree polynomials can degrade high resolutions (*cf* RUMP)

$$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

with $x = 77617$, $y = 33096$ (coprime integers)

[S.M. RUMP, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float:`	$P = -6.33825300e + 29$
`double:`	$P = -1.1805916207174113e + 021$
`long double:`	$P = +5.764607523034234891 88e + 17$
`quad:`	$P = +1.17260394005317863185883490452018380$
`fp16:`	$P =$ `NaN`

Name	Common name	Radix	Digits[c]	Decimal digits[d]
binary16	Half precision	2	11	3.31
binary32	Single precision	2	24	7.22
binary64	Double precision	2	53	15.95
binary128	Quadruple precision	2	113	34.02

- https://cc-fr.eu/webinars-replay/
- https://www.youtube.com/watch?v=JWe74pqFXWE

Aparté : Tests et flottant

Cursed Cancellation

⇒ higher degree polynomials can degrade high resolutions (*cf* RUMP)

$$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

with $x = 77617$, $y = 33096$ (coprime integers)

[S.M. RUMP, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float:`	$P = -6.33825300e + 29$
`double:`	$P = -1.1805916207174113e + 021$
`long double:`	$P = +5.764607523034234891 88e + 17$
`quad:`	$P = +1.17260394005317863185883490452018380$
`fp16:`	$P =$ `NaN`
exact:	$P \approx -0.82739605994682136814116509547981629 2$
	$P = -\frac{54767}{66192}$

Name	Common name	Radix	Digits[c]	Decimal digits[d]
binary16	Half precision	2	11	3.31
binary32	Single precision	2	24	7.22
binary64	Double precision	2	53	15.95
binary128	Quadruple precision	2	113	34.02

- https://cc-fr.eu/webinars-replay/
- https://www.youtube.com/watch?v=JWe74pqFXWE

Métrique de test

Couverture de Code:

- mesure de la **proportion du code exécutée** par les tests.

Métrique de test

Couverture de Code:

- mesure de la **proportion du code exécutée** par les tests.

Pourquoi c'est utile ?

- Identifie les **zones non testées** (donc potentiellement risquées).
- Aide à **prioriser l'écriture de tests**.
- Soutient la **robustesse** et **maintenance** du logiciel.

```
$ pip install pytest-cov
$ pytest --cov=mon_package
```

Métrique de test

Codecov

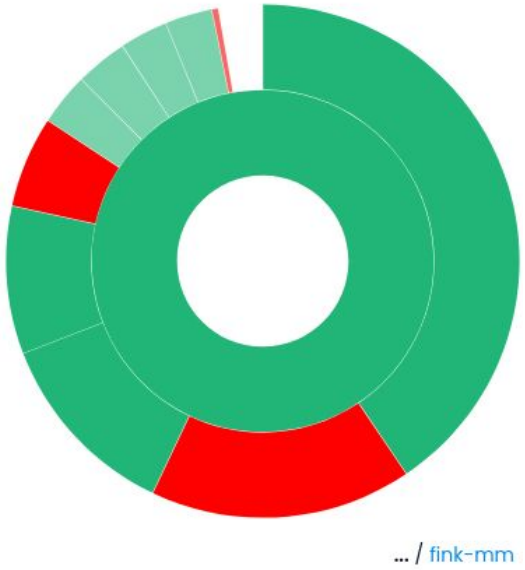
Couverture de Code:

- mesure de la **proportion du code exécutée** par les tests.

Code tree | File list | fink-mm / fink_mm

Search for files

Files ↑	Tracked lines	Covered	Partial	Missed	Coverage %
..					
distribution	77	71	0	6	92.21%
gcn_stream	136	69	0	67	50.74%
observatory	363	343	0	20	94.49%
test	133	79	0	54	59.40%
utils	273	244	0	29	89.38%
__init__.py	3	3	0	0	100.00%
conftest.py	148	148	0	0	100.00%
fink_mm_cli.py	24	3	0	21	12.50%
init.py	62	56	0	6	90.32%
ztf_join_gcn.py	153	141	0	12	92.16%
Subtotal	1372	1157	0	215	



Métrique de test

Codecov

Couverture de Code

```
144
145     elif topic in TOPICS_FORMAT["json"]:
146         try:
147             df = gr.parse_json_alert(gcn, logger, logs, is_test)
148         except Exception:
149             logger.error(
150                 "error while reading the json notice\n\n\tgcn: {}".format(gcn),
151                 exc_info=1,
152             )
153             return gcn_tracking
154
155     else:
156         logger.error(
157             "error while parsing the gcn file:\n\ttopic: {}\n\tgcn: {}".format(
158                 topic, gcn
159             )
160         )
161         raise Exception("bad gcn file format")
162
163     try:
164         if df is None:
165             if logs:  # pragma: no cover
166                 logger.info("The gcn is not a real observation")
167             return gcn_tracking
168
169         triggerId = df["triggerId"].values[0]
170         timejd = df["triggerTimejd"].values[0]
171         current_id_mask = gcn_tracking["triggerId"] == triggerId
172         if current_id_mask.any():
173             current_track = gcn_tracking.loc[current_id_mask]
174             updated_tag = f"update_{current_track['nb_received'].values[0]}"
175             df["gcn_status"] = updated_tag
176             gcn_tracking.loc[current_id_mask, "nb_received"] += 1
177         else:
178             gcn_tracking.loc[len(gcn_tracking)] = [triggerId, timejd, 0]
179             updated_tag = "initial"
180             df["gcn_status"] = updated_tag
181
```

Métrique de test

Couverture de Code :

⚠️ fort taux de couverture ≠ tests de qualité.

- On peut avoir 95% de couverture avec des tests inutiles.
- **Couverture de code = Indicateur**

Métrique de test

Couverture de Code :

⚠️ fort taux de couverture ≠ tests de qualité.

- On peut avoir 95% de couverture avec des tests inutiles.
- **Couverture de code = Indicateur**

Métrique de qualité de test

- **Mutation Testing Score**

Injection **automatique** de fautes dans le code (mutation)

- >= → >
- + → -
- return x → return 0

Un test de qualité doit **échouer** sur ces mutations → il « tue » le mutant.

Métrique de test

Couverture de Code :

⚠️ fort taux de couverture ≠ tests de qualité.

- On peut avoir 95% de couverture avec des tests inutiles.
- **Couverture de code = Indicateur**

Métrique de qualité de test

- **Mutation Testing Score**

Injection **automatique** de fautes dans le code (mutation)

- >= → >
- + → -
- return x → return 0

Un test de qualité doit **échouer** sur ces mutations → il « tue » le mutant.

$ pip install mutmut
ou
$ pip install cosmic-ray

Documentation & Normes

Pourquoi documenter ?

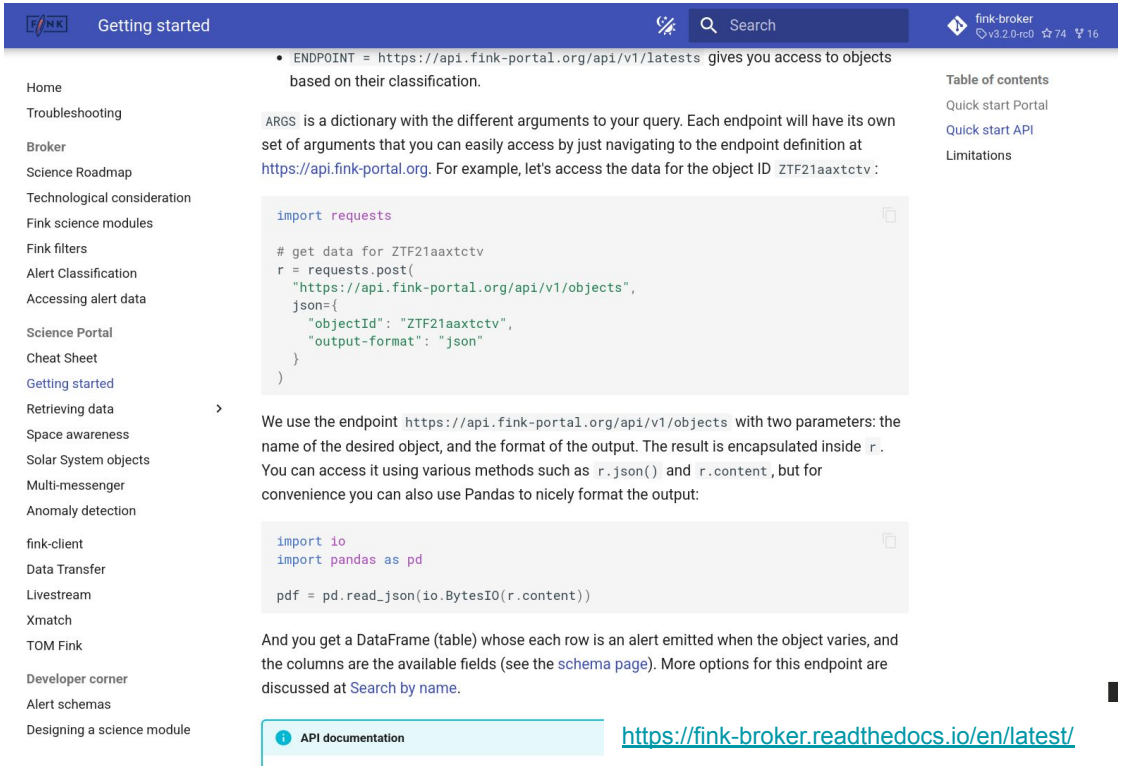
- Transmettre le fonctionnement
- Réduire les erreurs et les malentendus
- Assurer la reproducibilité scientifique
- Aider les nouveaux arrivants
- Améliorer les tests

Contribue à la **robustesse**, **maintenabilité** et la **testabilité**

Documentation & Normes

Documentation utilisateur vs développeur

- Documentation utilisateur
  - Installation, exécution
  - Tutoriels / How-to
  - Exemples concrets

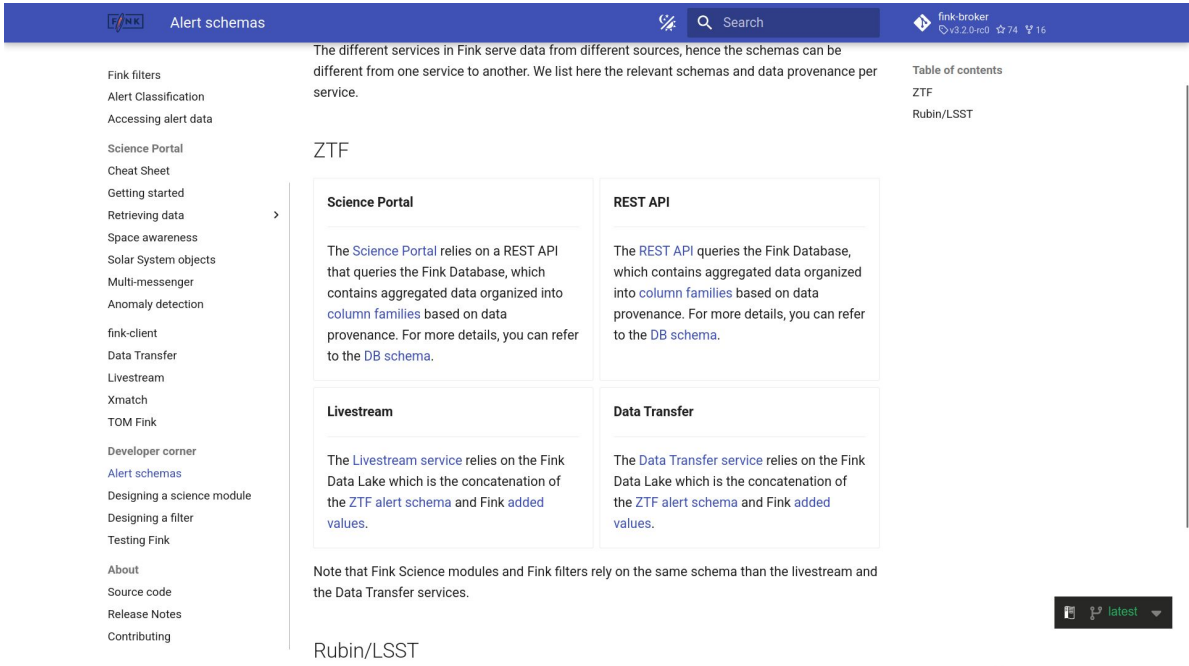


https://fink-broker.readthedocs.io/en/latest/

Documentation & Normes

Documentation utilisateur vs développeur

- Documentation développeur
  - Architecture (modules)
  - API (privée)
  - Comment contribuer



https://fink-broker.readthedocs.io/en/latest/

Documentation & Normes

Standards et automatisation

- Markdown

README.md

```
<p align="center">
  <img src="https://raw.githubusercontent.com/FusRoman/Outfit/main/outfit_logo.jpeg" alt="Outfit Logo" width="160"/>
</p>

<div align="center">
<h1>Outfit</h1>

A fast, safe, and extensible Rust library for **managing astrometric observations** and **determining preliminary orbits** of small bodies. Outfit reads common observation formats (MPC 80-column, ADES XML, Parquet), performs **initial orbit determination (IOD)** with the **Gauss method**, manages observers (topocentric geometry), and interfaces with **JPL ephemerides** for accurate state propagation.
</div>

<p align="center">
  <a href="https://crates.io/crates/outfit">
    <img src="https://img.shields.io/crates/v/outfit.svg" alt="Crates.io"/>
  </a>
  <a href="https://docs.rs/outfit">
    <img src="https://docs.rs/outfit/badge.svg" alt="Docs.rs"/>
  </a>
  <a href="LICENSE">
    <img src="https://img.shields.io/badge/license-CeCILL--C-blue.svg" alt="License: CeCILL-C"/>
  </a>
  <a href="https://github.com/FusRoman/Outfit/actions">
    <img src="https://github.com/FusRoman/Outfit/actions/workflows/ci.yml/badge.svg" alt="CI"/>
  </a>
  <a href="https://codecov.io/gh/FusRoman/Outfit">
    <img src="https://codecov.io/gh/FusRoman/Outfit/branch/main/graph/badge.svg" alt="codecov"/>
  </a>
  <a href="https://www.rust-lang.org/">
    <img src="https://img.shields.io/badge/MSRV-1.82%2B-orange" alt="MSRV"/>
  </a>
</p>

> **Why Outfit?**
> Modern asteroid pipelines need a library that is **fast (Rust)**, **reproducible**, and **easy to integrate** in data-intensive workflows (batch files, Parquet, CI/benchmarks). Outfit re-implements classic OrbFit IOD logic with a **memory-safe**, **modular** design and production-grade ergonomics (features, docs, tests, benches). It is built to:
> - ingest **large datasets** efficiently (columnar Parquet, batch APIs);
> - run **deterministic IOD** with controlled noise and repeatable seeds;
> - interface **cleanly** with JPL ephemerides (e.g., DE440);
> - provide a **clean API** that composes well across projects.

---

## Table of Contents

- [Features](#features)
- [Installation](#installation)
- [Quick Start](#quick-start)
- [Data Formats](#data-formats)
- [Initial Orbit Determination](#initial-orbit-determination)
- [Observers & Reference Frames](#observers--reference-frames)
```

README License

Outfit

A fast, safe, and extensible Rust library for **managing astrometric observations** and **determining preliminary orbits** of small bodies. Outfit reads common observation formats (MPC 80-column, ADES XML, Parquet), performs **initial orbit determination (IOD)** with the **Gauss method**, manages observers (topocentric geometry), and interfaces with **JPL ephemerides** for accurate state propagation.

Why Outfit?
Modern asteroid pipelines need a library that is **fast (Rust)**, **reproducible**, and **easy to integrate** in data-intensive workflows (batch files, Parquet, CI/benchmarks). Outfit re-implements classic OrbFit IOD logic with a **memory-safe**, **modular** design and production-grade ergonomics (features, docs, tests, benches). It is built to:

- ingest **large datasets** efficiently (columnar Parquet, batch APIs);
- run **deterministic IOD** with controlled noise and repeatable seeds;
- interface **cleanly** with JPL ephemerides (e.g., DE440);
- provide a **clean API** that composes well across projects.

Table of Contents

- Features
- Installation
- Quick Start
- Data Formats
- Initial Orbit Determination
- Observers & Reference Frames
- Cargo Feature Flags
- Performance & Reproducibility
- Roadmap
- Contributing
- License
- Acknowledgements

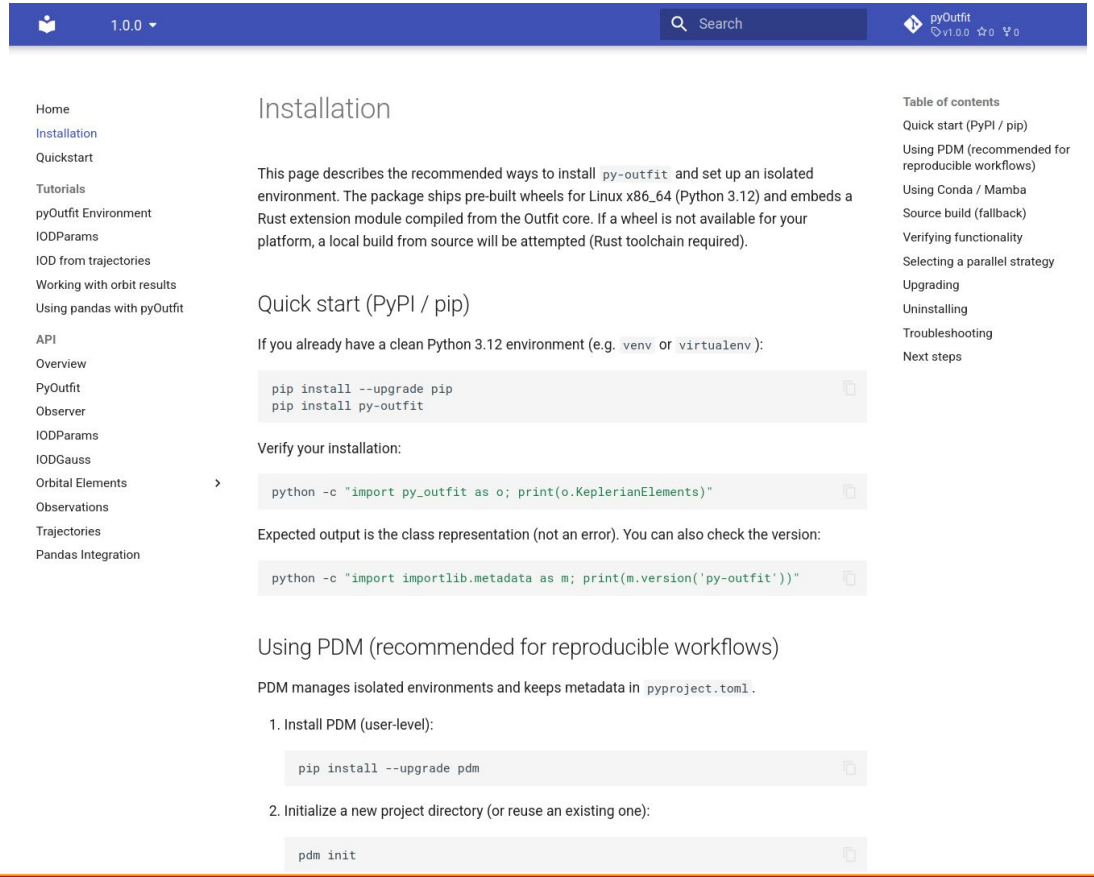
Features

- **Observation I/O**
  - MPC **80-column** files
  - **ADES XML** files
  - **Parquet** batches for high-throughput pipelines
- **Observer management**
  - Lookup by **MPC code**
  - Topocentric geometry (geocentric & heliocentric positions, AU, J2000)
- **Initial Orbit Determination**
  - **Gauss method** on observation triplets

Documentation & Normes

Standards et automatisation

- Markdown (https://squidfunk.github.io/mkdocs-material/)



Documentation & Normes

Standards et automatisation

- Markdown
- Type hints + docstring

```python
def mean_and_std(values: Iterable[float]) -> tuple[float, float]:
    """
    Calcule la moyenne et l'écart-type (population) d'une séquence de réels.

    La moyenne arithmétique est définie par :

    $$
    \\bar{x} = \\frac{1}{N} \\sum_{i=1}^{N} x_i
    $$

    et l'écart-type (définition *population*) par :

    $$
    \\sigma = \\sqrt{\\frac{1}{N} \\sum_{i=1}^{N} (x_i - \\bar{x})^2}
    $$

    Parameters
    ----------
    values : Iterable[float]
        Séquence de valeurs numériques (par exemple une liste ou un
        générateur de flottants).

    Returns
    -------
    mean : float
        Moyenne arithmétique des valeurs.
    std : float
        Écart-type (population) des valeurs.

    Raises
    ------
    ValueError
        Si la séquence fournie est vide.
    """
    data = list(values)
    if not data:
```

mean_and_std ¶

`mean_and_std(values: Iterable[float]) -> tuple[float, float]`

Calcule la moyenne et l'écart-type (population) d'une séquence de réels.

La moyenne arithmétique est définie par :

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

et l'écart-type (définition *population*) par :

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2}$$

PARAMETER	DESCRIPTION
`values`	Séquence de valeurs numériques (par exemple une liste ou un générateur de flottants). **TYPE:** `Iterable[float]`

RETURNS	DESCRIPTION
`mean`	Moyenne arithmétique des valeurs. **TYPE:** `float`
`std`	Écart-type (population) des valeurs. **TYPE:** `float`

RAISES	DESCRIPTION
`ValueError`	Si la séquence fournie est vide.

Documentation & Normes

Standards et automatisation

- Markdown
- Type hints + docstring + tests

```
$ pytest --doctest-modules good.py -vv -x
============================================================
platform linux -- Python 3.12.10, pytest-9.0.1, pluggy-1.6.0 -
cachedir: .pytest_cache
hypothesis profile 'default'
rootdir: /home/roman/Documents/Work/Mission/Roscoff_11_2025
configfile: pyproject.toml
plugins: hypothesis-6.148.2
collected 1 item

good.py::good.mean_and_std PASSED

============================================================
```

```
    Examples
    --------
    Exemple simple avec quelques valeurs :

    >>> mean, std = mean_and_std([1.0, 2.0, 3.0])
    >>> round(mean, 6)
    2.0
    >>> round(std, 6)
    0.816497

    Fonctionne aussi avec un générateur :

    >>> data = (float(i) for i in range(4))  # 0, 1, 2, 3
    >>> mean, std = mean_and_std(data)
    >>> round(mean, 6)
    1.5
    >>> round(std, 6)
    1.118034

    Une séquence constante donne un écart-type nul :

    >>> mean, std = mean_and_std([5.0, 5.0, 5.0])
    >>> mean
    5.0
    >>> std
    0.0

    Une liste vide provoque une erreur :

    >>> mean_and_std([])
    Traceback (most recent call last):
        ...
    ValueError: Impossible de calculer des statistiques sur une liste vide.
    """
```

Documentation & Normes

Standards et automatisation

- Markdown
- Type hints + docstring + tests

```
$ pytest --doctest-modules good.py -vv -x
============================================================
platform linux -- Python 3.12.10, pytest-9.0.1, pluggy-1.6.0 -
cachedir: .pytest_cache
hypothesis profile 'default'
rootdir: /home/roman/Documents/Work/Mission/Roscoff_11_2025
configfile: pyproject.toml
plugins: hypothesis-6.148.2
collected 1 item

good.py::good.mean_and_std PASSED

============================================================
```

Examples:

Exemple simple avec quelques valeurs :

```
>>> mean, std = mean_and_std([1.0, 2.0, 3.0])
>>> round(mean, 6)
2.0
>>> round(std, 6)
0.816497
```

Fonctionne aussi avec un générateur :

```
>>> data = (float(i) for i in range(4))  # 0, 1, 2, 3
>>> mean, std = mean_and_std(data)
>>> round(mean, 6)
1.5
>>> round(std, 6)
1.118034
```

Une séquence constante donne un écart-type nul :

```
>>> mean, std = mean_and_std([5.0, 5.0, 5.0])
>>> mean
5.0
>>> std
0.0
```

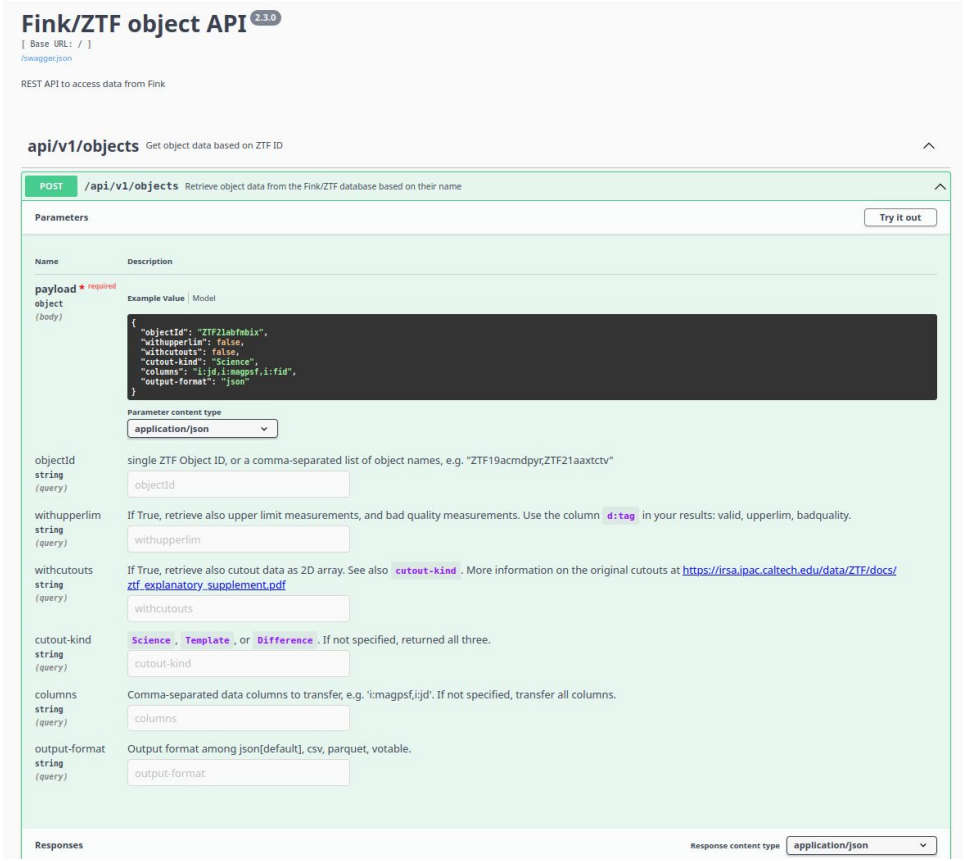
Une liste vide provoque une erreur :

```
>>> mean_and_std([])
Traceback (most recent call last):
    ...
ValueError: Impossible de calculer des statistiques sur une liste vide.
```

Documentation & Normes

Standards et automatisation

- Markdown
- Type hints + docstring
- OpenApi/Swagger (https://swagger.io/specification/)



Documentation & Normes

Normes et Conventions

rendre le code lisible, cohérent et compréhensible par tous

Documentation & Normes

Normes et Conventions

rendre le code lisible, cohérent et compréhensible par tous

- Standard de style
  - PEP8 (Python)
  - style, indentation, imports, nommage, lisibilité

Documentation & Normes

Normes et Conventions

rendre le code lisible, cohérent et compréhensible par tous

- Standard de style
  - PEP8 (Python)
  - style, indentation, imports, nommage, lisibilité

```python
import sys,os

def add(a,b):return a+b

def main():
    result=add(1,2)
    print("Result is:",result)

if __name__=="__main__":
    main()
```

Documentation & Normes

Normes et Conventions

rendre le code lisible, cohérent et compréhensible par tous

- Standard de style
  - PEP8 (Python)
  - style, indentation, imports, nommage, lisibilité

```python
import sys,os

def add(a,b):return a+b

def main():
    result=add(1,2)
    print( "Result is:",result )

if __name__=="__main__" :
 main()
```

```
$ flake8 bad_format.py
bad_format.py:1:1: F401 'sys' imported but unused
bad_format.py:1:1: F401 'os' imported but unused
bad_format.py:1:7: E271 multiple spaces after keyword
bad_format.py:1:12: E401 multiple imports on one line
bad_format.py:1:12: E231 missing whitespace after ','
bad_format.py:3:1: E302 expected 2 blank lines, found 1
bad_format.py:3:4: E271 multiple spaces after keyword
bad_format.py:3:12: E231 missing whitespace after ','
bad_format.py:3:15: E231 missing whitespace after ':'
bad_format.py:5:1: E302 expected 2 blank lines, found 1
bad_format.py:6:11: E225 missing whitespace around operator
bad_format.py:6:17: E231 missing whitespace after ','
bad_format.py:7:11: E201 whitespace after '('
bad_format.py:7:26: E231 missing whitespace after ','
bad_format.py:7:35: E202 whitespace before ')'
bad_format.py:9:1: E305 expected 2 blank lines after class or function definition, found 1
bad_format.py:9:12: E225 missing whitespace around operator
bad_format.py:9:24: E203 whitespace before ':'
bad_format.py:10:2: E111 indentation is not a multiple of 4
$ ^C
```

Documentation & Normes

Normes et Conventions

rendre le code lisible, cohérent et compréhensible par tous

- Standard de style
  - PEP8 (Python)
  - style, indentation, imports, nommage, lisibilité

```python
import sys,os

def add(a,b):return a+b

def main():
    result=add(1,2)
    print( "Result is:",result )

if __name__=="__main__" :
    main()
```

```python
def add(a, b):
    return a + b


def main():
    result = add(1, 2)
    print(f"Result is: {result}")


if __name__ == "__main__":
    main()
```

```
$ flake8 good_format.py
$
```

Documentation & Normes

Normes et Conventions

rendre le code lisible, cohérent et compréhensible par tous

- Standard de style
  - PEP8 (Python)
  - style, indentation, imports, nommage, lisibilité
- Outils automatique
  - **Linters** : pylint, flake8… → détectent erreurs, incohérences, complexité inutile.
  - **Formateurs** : black, isort → formattent automatiquement le code selon les conventions.

Ce qu’on a…

- Une architecture logicielle
  - Modules & interfaces
- Des tests
  - Unitaire
  - Integration
- Une documentation
  - Utilisateur
  - Développeur

Ce qui manque

- Une architecture logicielle
  - Modules & interfaces
- Des tests
  - Unitaire
  - Integration
- Une documentation
  - Utilisateur
  - Développeur

- **Traçabilité des changements**
- **Travail collaboratif**
- **Automatisation**
- **Publication**

Gestion de versions

Pourquoi versionner ?

Problèmatique:

- Comment tracer les changements ?
- Comment revenir en arrière après un bug ?
- Comment maintenir une base stable tout en développant des changements ?
- Comment travailler en équipe ?
- Qui a changé quoi ?
- Comment reproduire des résultats ?

Gestion de versions

Un outil: Git

- **Historique complet** des modifications
  comprendre l'évolution, diagnostiquer un bug
- **Sécurité**
  pouvoir revenir à un état sain à n'importe quel moment
- **Expérimentation** sans risque
  branches indépendantes, essais, protos
- **Collaboration** structurée
  éviter les conflits, revoir le code, fusionner proprement
- **Reproductibilité** scientifique
  associer un résultat à une version exacte (avec un tag)

Gestion de versions

- **Gestion de l'historique**

proto.py

```python
import math, sys

def run(x):
    r = []
    for v in x:
        if v is None or v == "":
            continue
        v = float(v)
        v = (v * 0.00123) + 0.42
        if v < 0:
            v = 0
        r.append(v)
    s = 0
    for v in r:
        s += v
    m = s / len(r)
    s2 = 0
    for v in r:
        s2 += (v - m) ** 2
    s2 = math.sqrt(s2 / len(r))
    if s2 > 0.1:
        print("ALERT", m, s2)
    else:
        print("OK", m, s2)

if __name__ == "__main__":
    data = []
    for a in sys.argv[1:]:
        data.append(a)
    run(data)
```

```
$ git init
astuce: Utilisation de 'master' comme nom de la branche initiale. Le nom de la branche
astuce: par défaut peut changer. Pour configurer le nom de la branche initiale
astuce: pour tous les nouveaux dépôts, et supprimer cet avertissement, lancez :
astuce:
astuce:         git config --global init.defaultBranch <nom>
astuce:
astuce: Les noms les plus utilisés à la place de 'master' sont 'main', 'trunk' et
astuce: 'development'. La branche nouvellement créée peut être rénommée avec :
astuce:
astuce:         git branch -m <nom>
Dépôt Git vide initialisé dans /home/roman/Documents/Work/Mission/Roscoff_11_2025/detmes/.git/
$ git status
Sur la branche master

Aucun commit

rien à valider (créez/copiez des fichiers et utilisez "git add" pour les suivre)
$
```

Gestion de versions

git

- **Gestion de l'historique**

proto.py

```python
import math, sys

def run(x):
    r = []
    for v in x:
        if v is None or v == "":
            continue
        v = float(v)
        v = (v * 0.00123) + 0.42
        if v < 0:
            v = 0
        r.append(v)
    s = 0
    for v in r:
        s += v
    m = s / len(r)
    s2 = 0
    for v in r:
        s2 += (v - m) ** 2
    s2 = math.sqrt(s2 / len(r))
    if s2 > 0.1:
        print("ALERT", m, s2)
    else:
        print("OK", m, s2)

if __name__ == "__main__":
    data = []
    for a in sys.argv[1:]:
        data.append(a)
    run(data)
```

```
$ git status
Sur la branche master

Aucun commit

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
        proto.py

aucune modification ajoutée à la validation mais des fichiers non suivis sont
$ git add proto.py
$ git status
Sur la branche master

Aucun commit

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)
        nouveau fichier : proto.py

$
```

Gestion de versions

git

- **Gestion de l'historique**

proto.py

```python
import math, sys

def run(x):
    r = []
    for v in x:
        if v is None or v == "":
            continue
        v = float(v)
        v = (v * 0.00123) + 0.42
        if v < 0:
            v = 0
        r.append(v)
    s = 0
    for v in r:
        s += v
    m = s / len(r)
    s2 = 0
    for v in r:
        s2 += (v - m) ** 2
    s2 = math.sqrt(s2 / len(r))
    if s2 > 0.1:
        print("ALERT", m, s2)
    else:
        print("OK", m, s2)

if __name__ == "__main__":
    data = []
    for a in sys.argv[1:]:
        data.append(a)
    run(data)
```

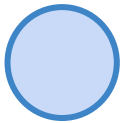
```
$ git commit -m "prototype de la chaine de traitement"
[master (commit racine) c6e71f6] prototype de la chaine de traitement
 1 file changed, 30 insertions(+)
 create mode 100644 proto.py
$ git status
Sur la branche master
rien à valider, la copie de travail est propre
$ git log
commit c6e71f6cba32451d87cd72b6dfef40bc4c0bdf8b (HEAD -> master)
Author: Roman <roman.lemontagner@gmail.com>
Date:   Fri Nov 21 14:31:40 2025 +0100

    prototype de la chaine de traitement
$
```

Gestion de versions

git

- **Gestion de l'historique**

e71229



Prototype de la chaine de traitement

Gestion de versions

git

- **Gestion de l'historique**

= un commit

Commande :

$ **git init** (inititalise le repo git)

$ **git status** (check l'état du repo)

$ **git add** (ajout des modifications à la validation)

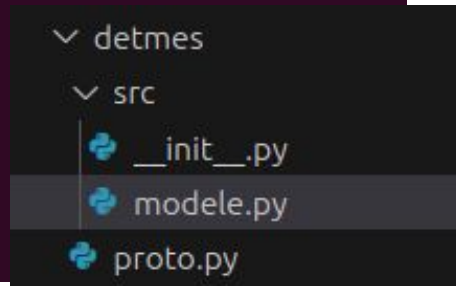
$ **git commit** (valide les modifications)

Gestion de versions

- **Gestion de l'historique**

```python
from dataclasses import dataclass

# ==== Modèle de donnée ====


@dataclass
class RawSample:
    """
    Représente une mesure brute lue depuis la chaîne d'acquisition.

    Parameters
    ----------
    value : float
        Valeur numérique brute, typiquement lue en unités ADU
        (Analog-to-Digital Units).
    """

    value: float


@dataclass
class CalibratedSample:
    """
    Représente une mesure convertie en unités physiques (volts).

    Parameters
    ----------
    volts : float
        Valeur de la mesure après calibration, exprimée en volts.
    """

    volts: float
```

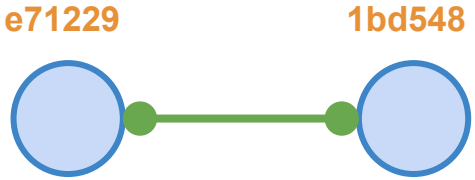
```
$ git status
Sur la branche master
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
        src/

aucune modification ajoutée à la validation mais des fichiers non suivis
$ git add src
$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git restore --staged <fichier>..." pour désindexer)
        nouveau fichier : src/__init__.py
        nouveau fichier : src/modele.py

$ git commit -am "ajout des modèles de données"
[master 0e6e614] ajout des modèles de données
 2 files changed, 32 insertions(+)
 create mode 100644 src/__init__.py
 create mode 100644 src/modele.py
$ git status
Sur la branche master
rien à valider, la copie de travail est propre
$
```



Gestion de versions

git

- **Gestion de l'historique**



Prototype de la chaine de traitement

ajout des modèles de données

Gestion de versions

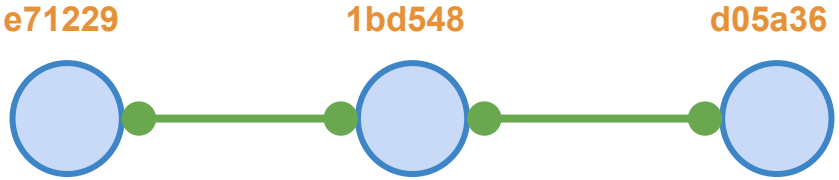
- **Gestion de l'historique**

```
$ rm proto.py
$ git status
Sur la branche master
Modifications qui ne seront pas validées :
  (utilisez "git add/rm <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git restore <fichier>..." pour annuler les modifications dans le répertoire de
        supprimé :        proto.py

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
        src/calibration.py
        src/input.py
        src/stats.py

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git commit -a")
$ git add proto.py src/
$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git restore --staged <fichier>..." pour désindexer)
        supprimé :        proto.py
        nouveau fichier : src/calibration.py
        nouveau fichier : src/input.py
        nouveau fichier : src/stats.py

$ git commit -m "suppression de proto.py, ajout du reste de la chaine de traitement"
[master 6648350] suppression de proto.py, ajout du reste de la chaine de traitement
 4 files changed, 135 insertions(+), 30 deletions(-)
 delete mode 100644 proto.py
 create mode 100644 src/calibration.py
 create mode 100644 src/input.py
 create mode 100644 src/stats.py
$ git status
Sur la branche master
rien à valider, la copie de travail est propre
$
```

Gestion de versions

- **Gestion de l'historique**

```
$ git log
commit 664835022a1d5bf5ba8834736002ac59086c24ea (HEAD -> master)
Author: Roman <roman.lemontagner@gmail.com>
Date:   Fri Nov 21 14:50:43 2025 +0100

    suppression de proto.py, ajout du reste de la chaine de traitement

commit 89aba1746ebfda0277877d49a20698507c4f36fc
Author: Roman <roman.lemontagner@gmail.com>
Date:   Fri Nov 21 14:47:21 2025 +0100

    ajout des modèles de données

commit 6b86786e4bb39d28c6153d1c0201fac0a4d6cc9d
Author: Roman <roman.lemontagner@gmail.com>
Date:   Fri Nov 21 14:46:40 2025 +0100

    prototype de la chaine de traitement
$
```



Prototype de la chaine de traitement

ajout des modèles de données

suppression de proto.py, ajout du reste de la chaine de traitement

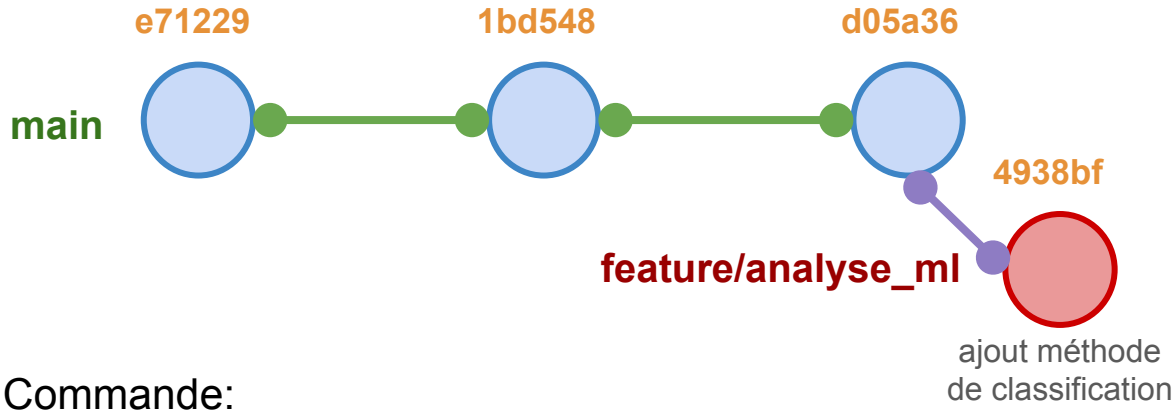
⚠️ Ne jamais laisser de fichier inutile dans un repo git → **Historique**

Gestion de versions

git

- **Expérimentation / Ajout**

  Gestion de branche : travailler sans casser la version stable



Commande:

$ **git switch -c feature/analyse_ml**

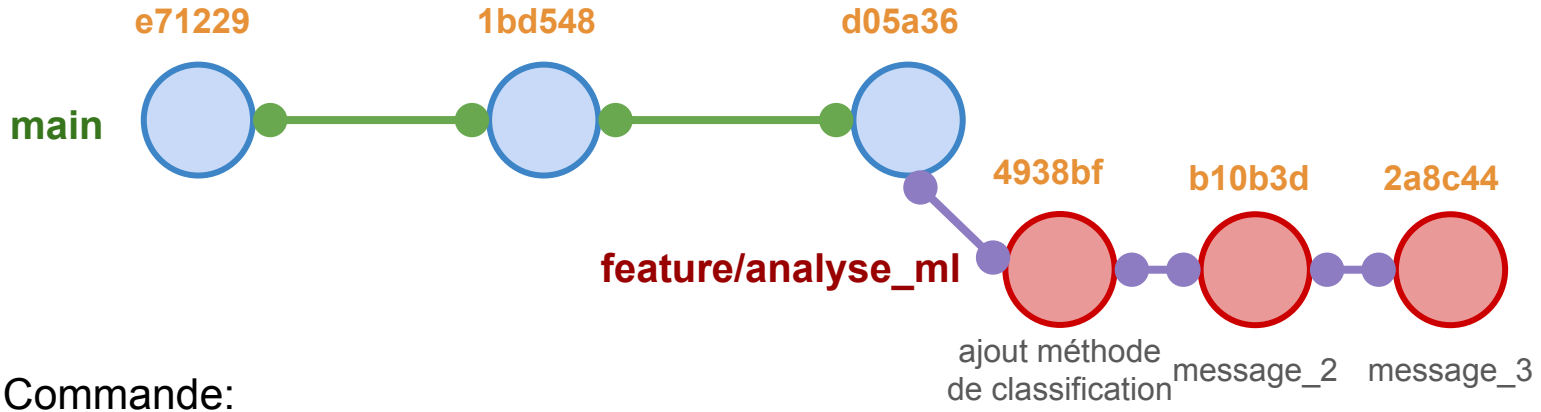
$ git commit -m “ajout méthode de classification”

Gestion de versions

git

- **Expérimentation / Ajout**

  Gestion de branche : travailler sans casser la version stable



Commande:

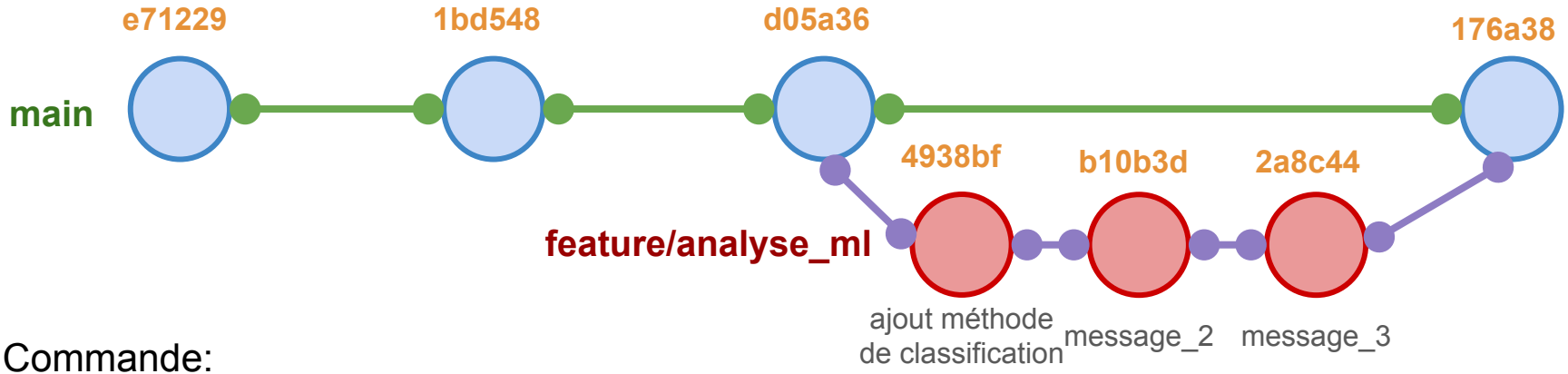
$ git commit -m “message_2”

$ git commit -m “message_3”

Gestion de versions

- **Expérimentation / Ajout**

  Gestion de branche : travailler sans casser la version stable



Commande:

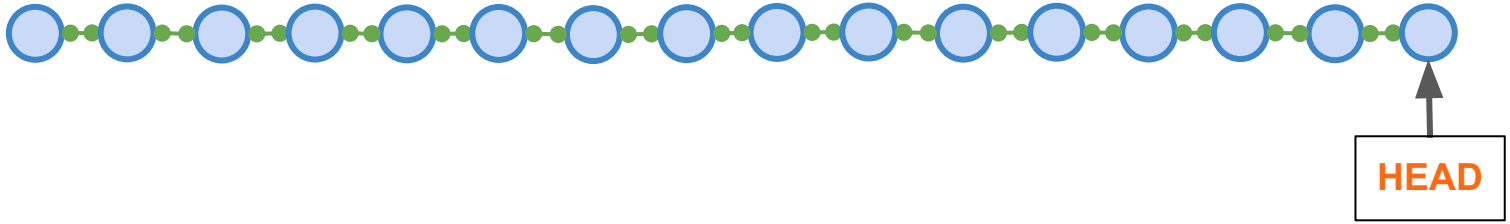
$ git switch main

$ git merge feature/analyse_ml

Gestion de versions

git

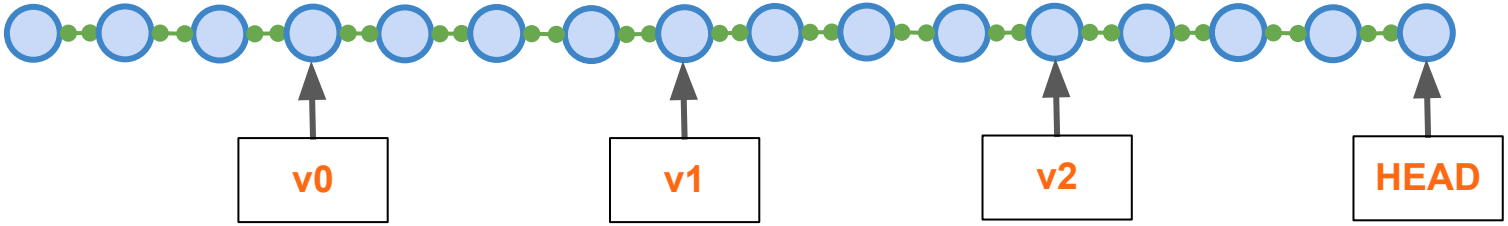
- **Versionning (Tag)**



Gestion de versions

git

- **Versionning (Tag)**

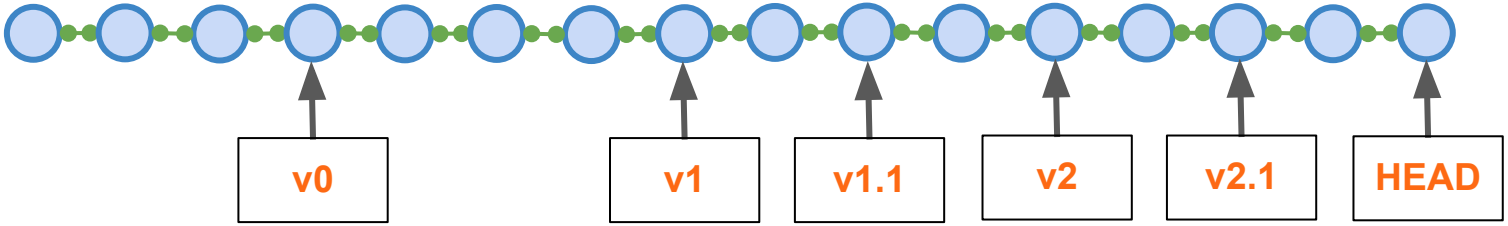


Une version (numéro et/ou nom) = une photo d'un logiciel

Gestion de versions

git

- **Versionning (Tag)**



- Versionnement sémantique: MAJEUR.MINEUR.CORRECTIF
  - Majeur : changement non rétrocompatible
  - Mineur : changement rétrocompatible
  - Correctif : correction d'anomalies rétrocompatible

https://semver.org/lang/fr/

Gestion de versions

- **Branche, Versionning, Release**

Quelque bonnes pratiques :

- On ne commit **jamais** dans la branche main

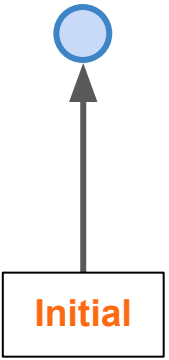
Gestion de versions

git

- **Branche, Versionning, Release**

Quelque bonnes pratiques :

- On ne commit **jamais** dans la branche main
- Organisation en release de version (branche release, branche feature)



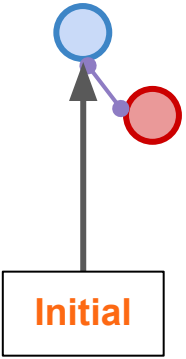
Gestion de versions

git

- **Branche, Versionning, Release**

Quelque bonnes pratiques :

- On ne commit **jamais** dans la branche main
- Organisation en release de version (branche release, branche feature)



main

release v1

Gestion de versions

git

- **Branche, Versionning, Release**

Quelque bonnes pratiques :

- On ne commit **jamais** dans la branche main
- Organisation en release de version (branche release, branche feature)

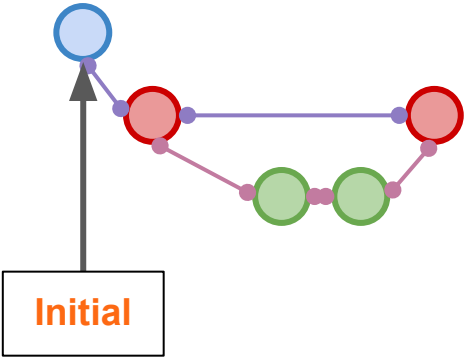
Gestion de versions

git

- **Branche, Versionning, Release**

Quelque bonnes pratiques :

- On ne commit **jamais** dans la branche main
- Organisation en release de version (branche release, branche feature)



main

release v1

feature/f1

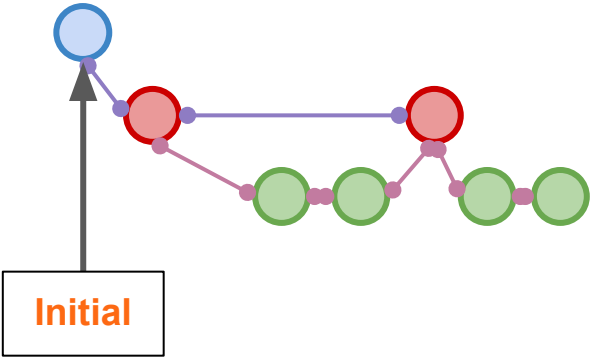
Gestion de versions

git

- **Branche, Versionning, Release**

Quelque bonnes pratiques :

- On ne commit **jamais** dans la branche main
- Organisation en release de version (branche release, branche feature)



main

release v1

feature/f2

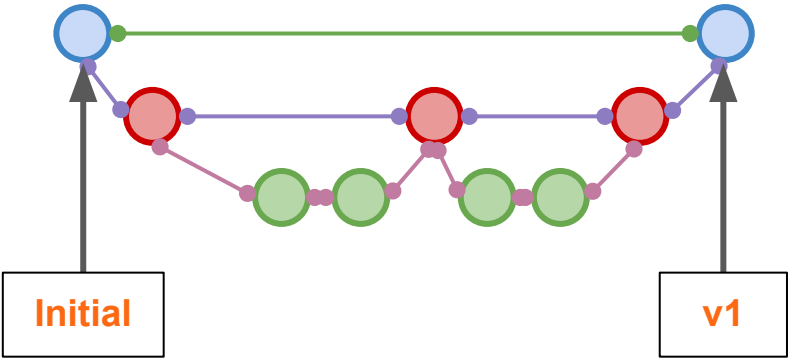
Gestion de versions

git

- **Branche, Versionning, Release**

Quelque bonnes pratiques :

- On ne commit **jamais** dans la branche main
- Organisation en release de version (branche release, branche feature)



main

release v1

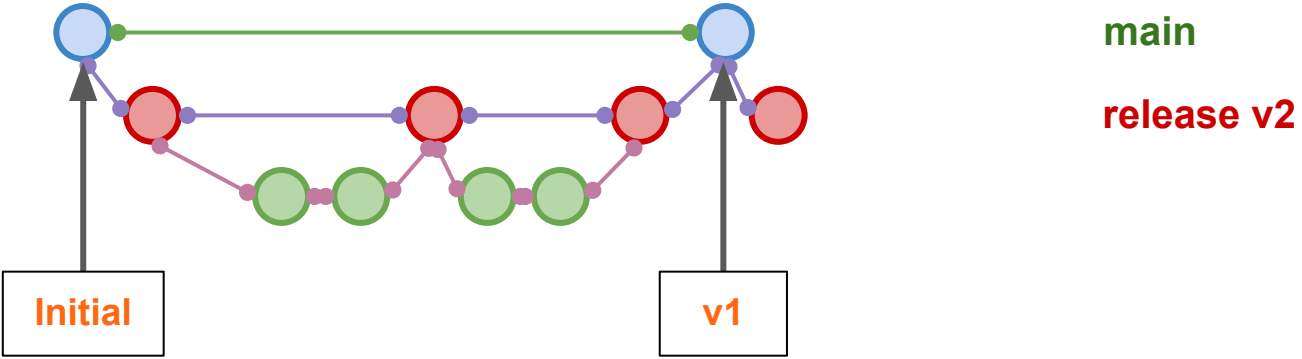
Gestion de versions

git

- **Branche, Versionning, Release**

Quelque bonnes pratiques :

- On ne commit **jamais** dans la branche main
- Organisation en release de version (branche release, branche feature)



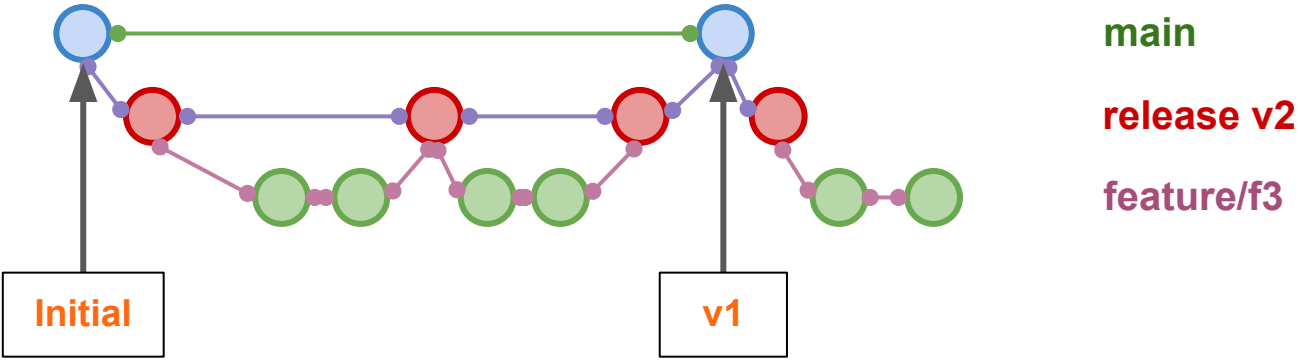
Gestion de versions

git

- **Branche, Versionning, Release**

Quelque bonnes pratiques :

- On ne commit **jamais** dans la branche main
- Organisation en release de version (branche release, branche feature)



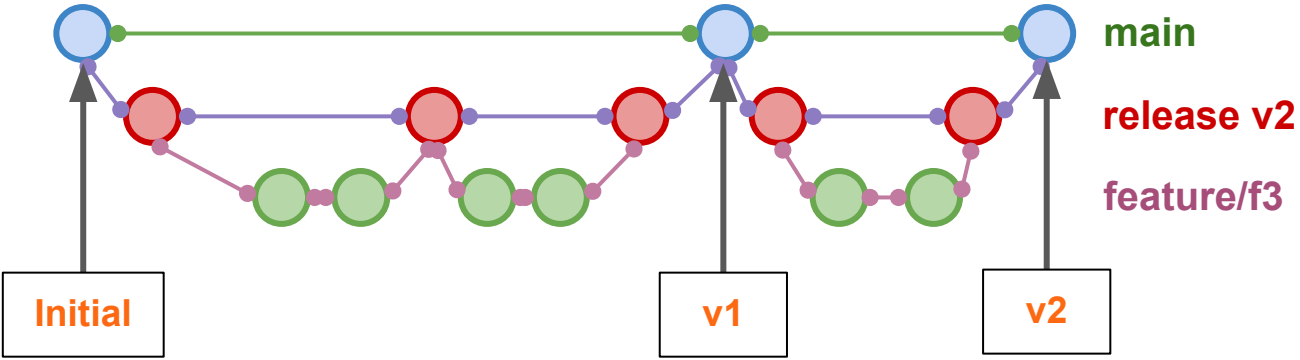
Gestion de versions

git

- **Branche, Versionning, Release**

Quelque bonnes pratiques :

- On ne commit **jamais** dans la branche main
- Organisation en release de version (branche release, branche feature)



Gestion de versions

quelque commandes :

- git init: intialise le repo
- git add : ajoute des modifs
- git commit : valide les modifs
- git switch : changer de branche
  - (ajout de l'option '-c' pour créer une branche)
- git merge : fusionner une branche
- git status : checker l'état du repo
- git log : voir l'historique

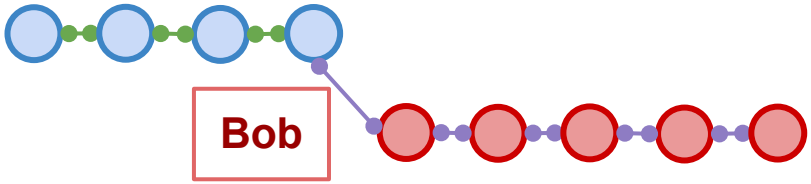
Conseil : Utiliser git même si je suis seul et sur un petit projet

Gestion de versions

Git + forge

- **Travail collaboratif**

  Gestion de branche : travailler en équipe



release

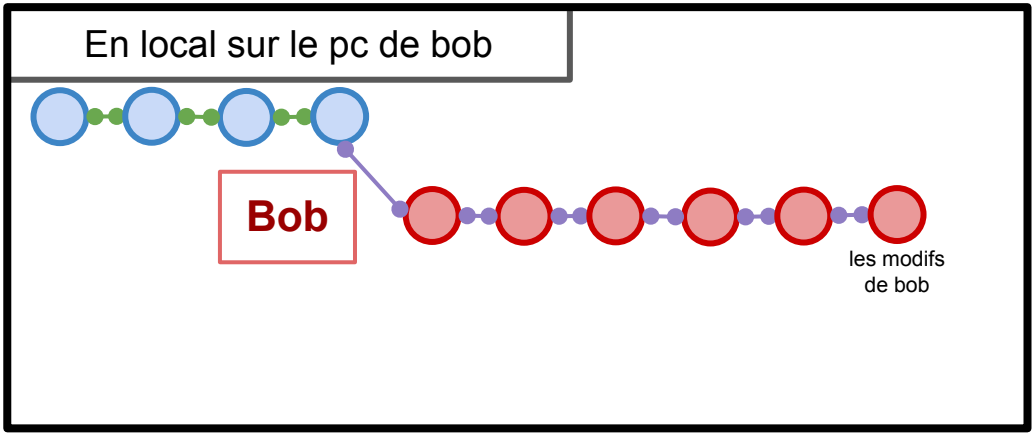
feature/f_bob

Gestion de versions

Git + forge

- **Travail collaboratif**

  Gestion de branche : travailler en équipe



release

feature/f_bob

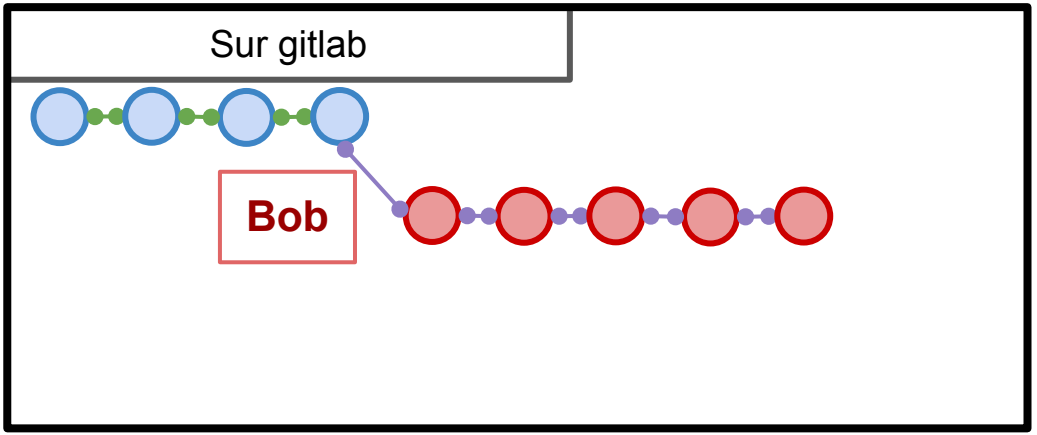
Commande:
$ git add “bob_fichier.py”
$ git commit -m “les modifs de bob”

Gestion de versions

Git + forge

- **Travail collaboratif**

  Gestion de branche : travailler en équipe



release

feature/f_bob

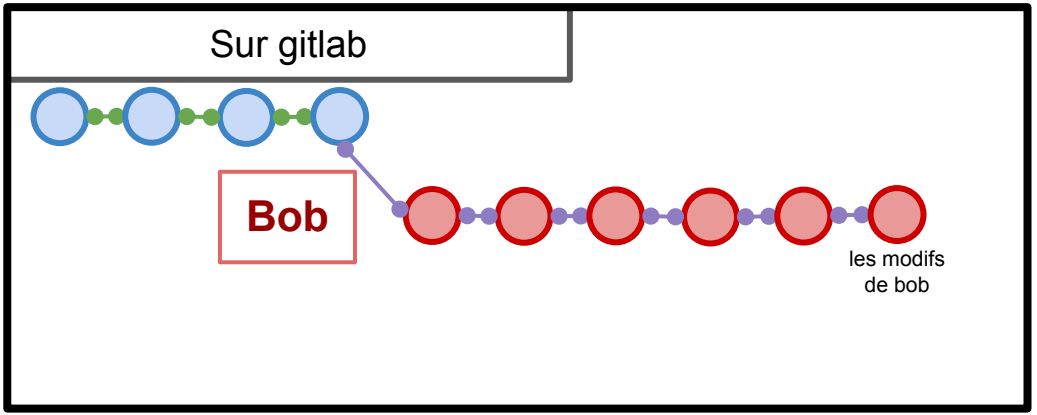
Commande:
$ git add “bob_fichier.py”
$ git commit -m “les modifs de bob”

Gestion de versions

Git + forge

- **Travail collaboratif**

  Gestion de branche : travailler en équipe



release

feature/f_bob

Commande:

$ git add “bob_fichier.py”

$ git commit -m “les modifs de bob”

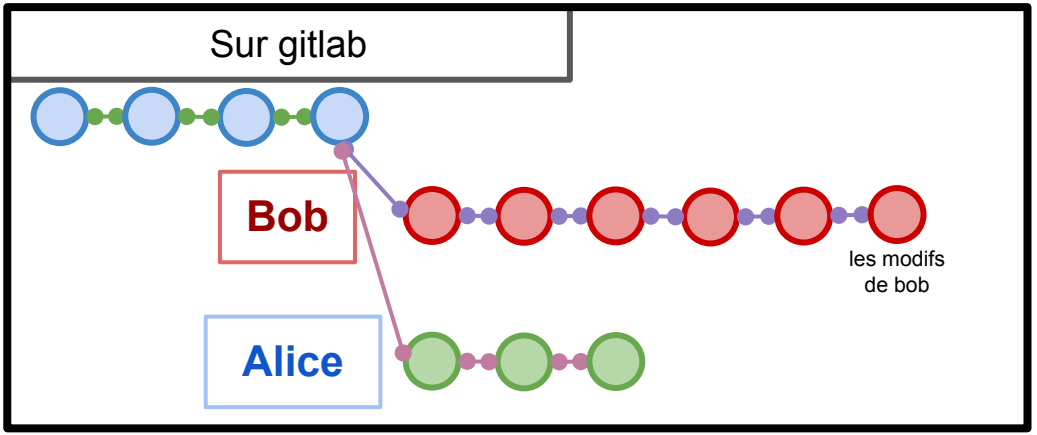
$ **git push**

Gestion de versions

Git + forge

- **Travail collaboratif**

Gestion de branche : travailler en équipe



release

feature/f_bob

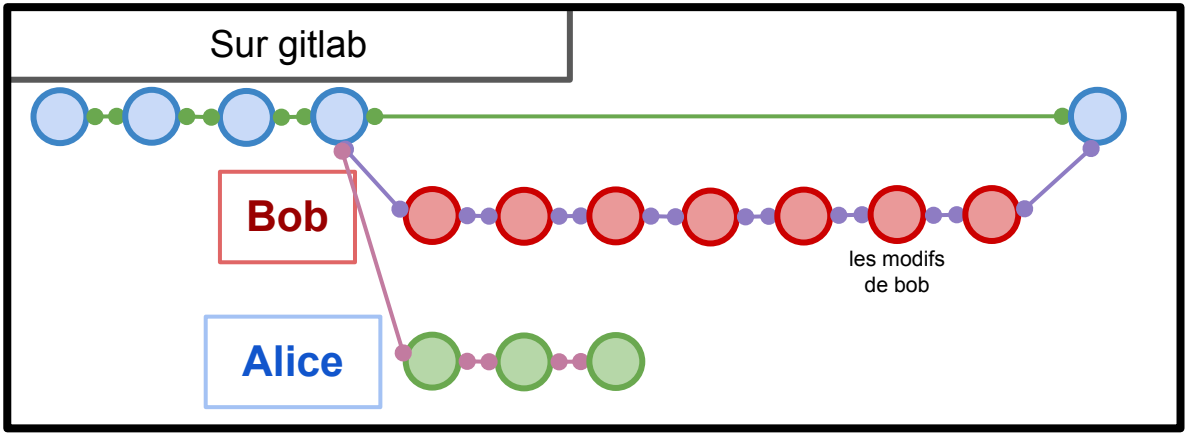
feature/f_alice

Gestion de versions

Git + forge

- **Travail collaboratif**

Gestion de branche : travailler en équipe



release

feature/f_bob

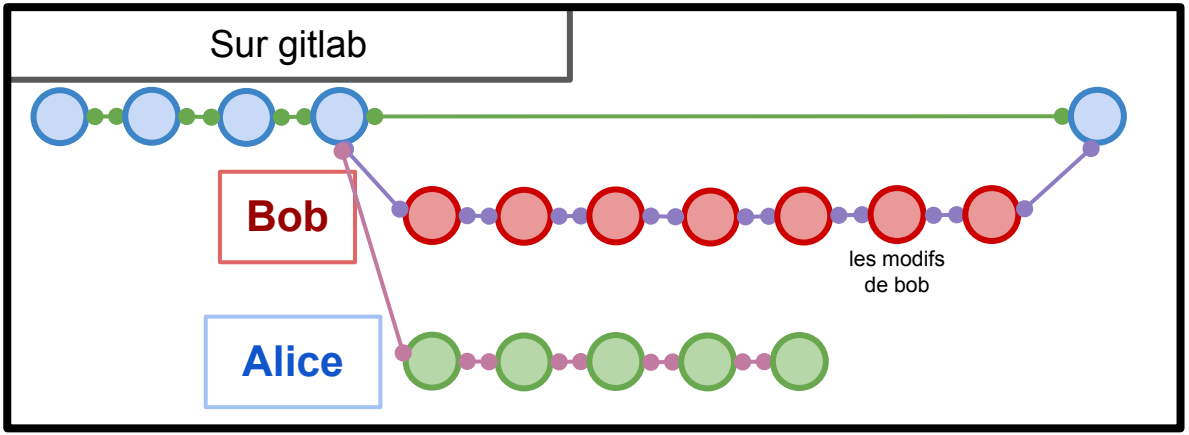
feature/f_alice

Gestion de versions

Git + forge

- **Travail collaboratif**

  Gestion de branche : travailler en équipe



release

feature/f_bob

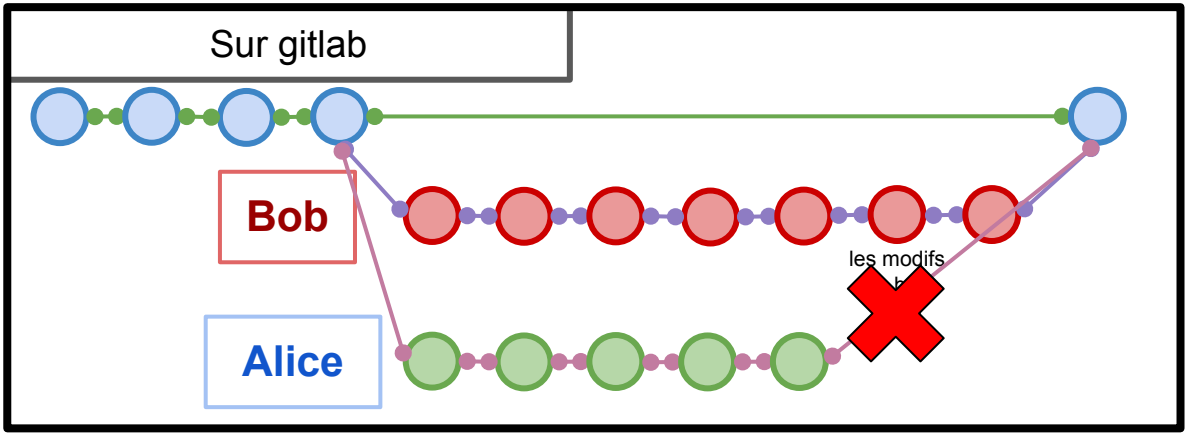
feature/f_alice

Gestion de versions

Git + forge

- **Travail collaboratif**

Gestion de branche : travailler en équipe



release

feature/f_bob

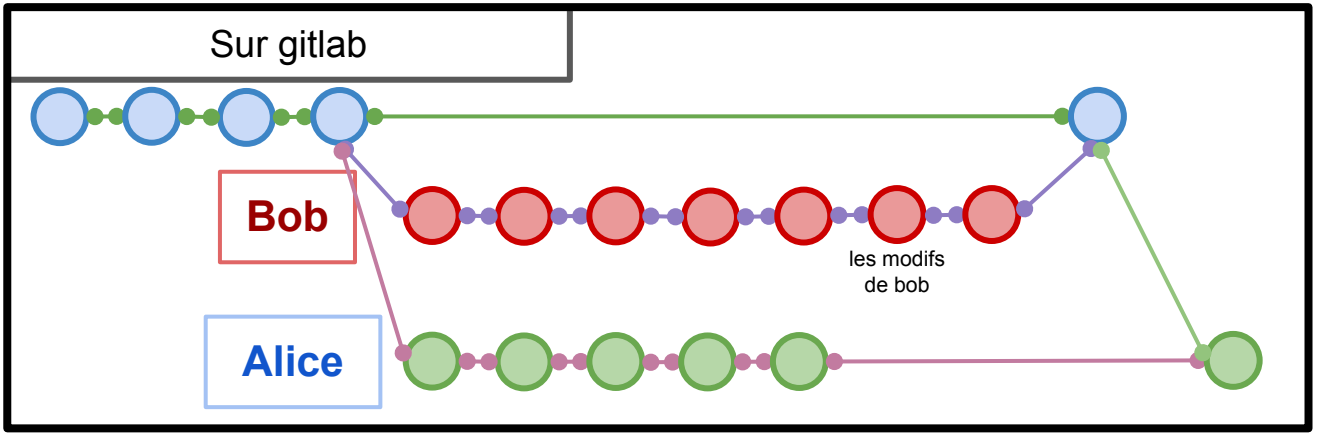
feature/f_alice

Gestion de versions

Git + forge

- **Travail collaboratif**

  Gestion de branche : travailler en équipe



release

feature/f_bob

feature/f_alice

Commande:

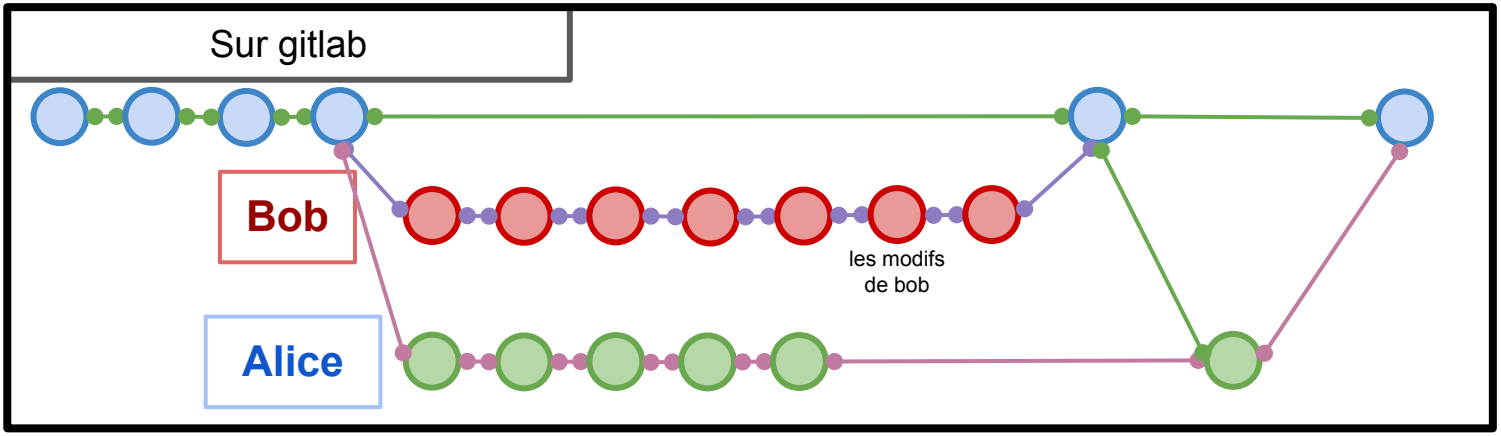
$ **git pull**

Gestion de versions

Git + forge

- **Travail collaboratif**

  Gestion de branche : travailler en équipe



release

feature/f_bob

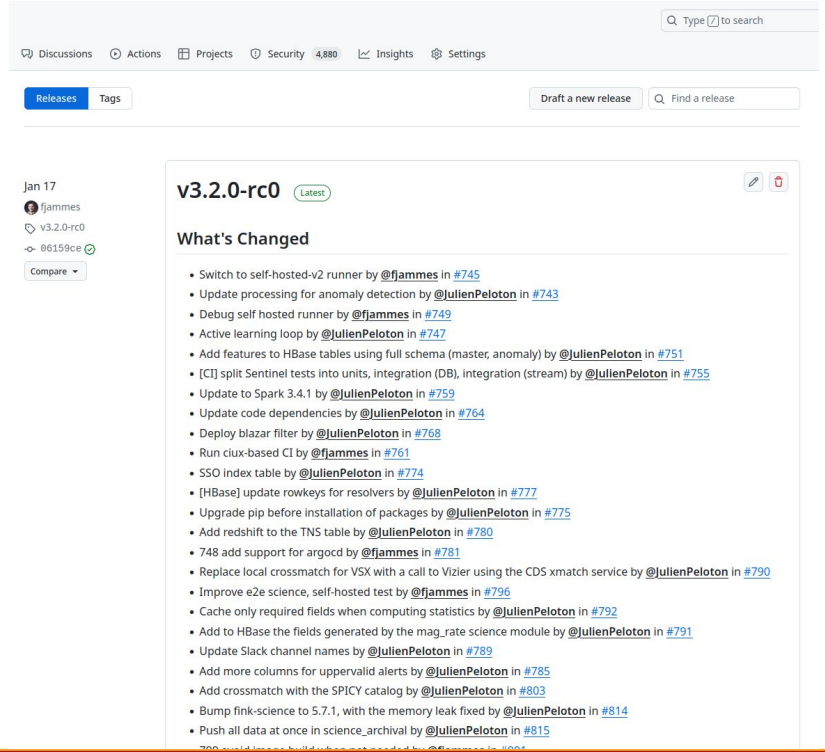
feature/f_alice

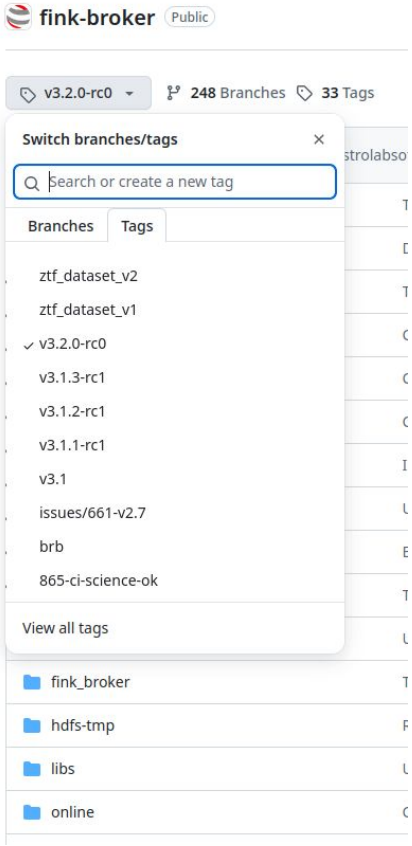
Commande:

$ **git push**

Forge

Versionning / Release





GitLab

GitHub

• Catch json decoding issues by @JulienPeloton in #938
• Drop images for the anomaly table by @JulienPeloton in #943
• 913 replace minio s3 with hdfs by @fjammes in #930

New Contributors

• @Knispel2 made their first contribution in #929

Full Changelog: `v3.1...v3.2.0-rc0`

Contributors

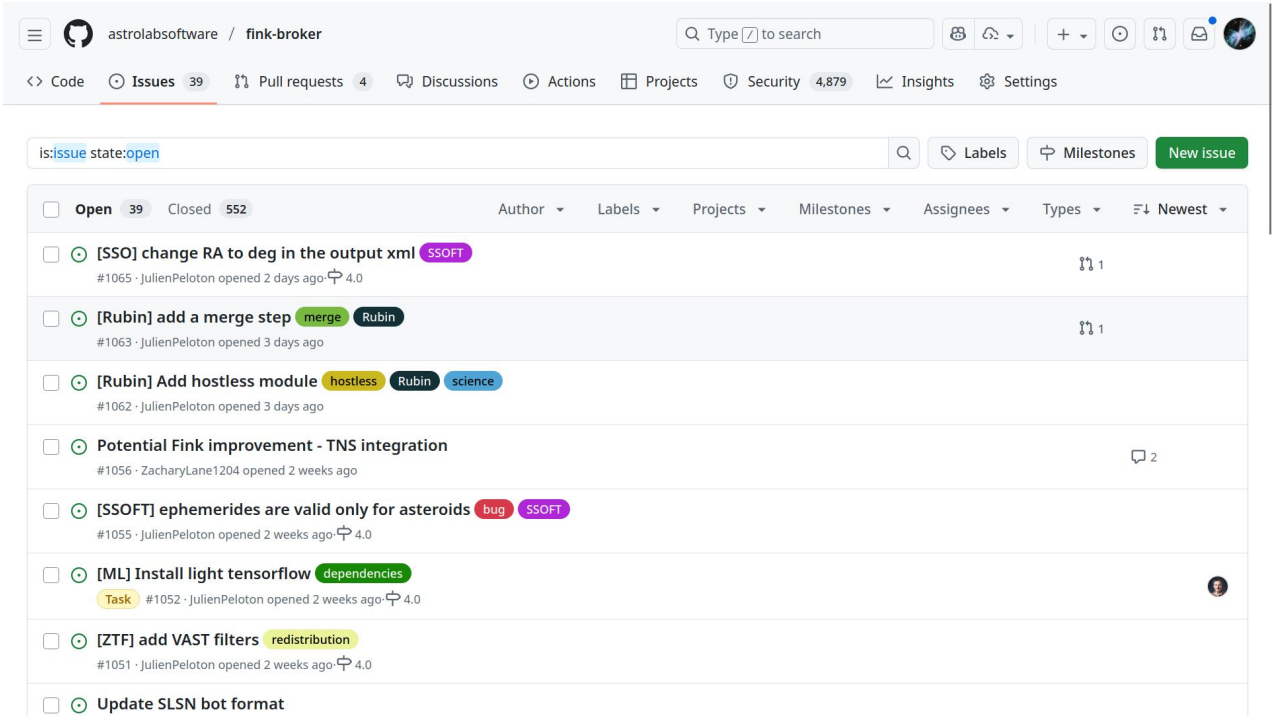
fjammes, anaismoller, and 3 other contributors

▾ **Assets** 2

Source code (zip)
Source code (tar.gz)

Forge

Gestion de ticket

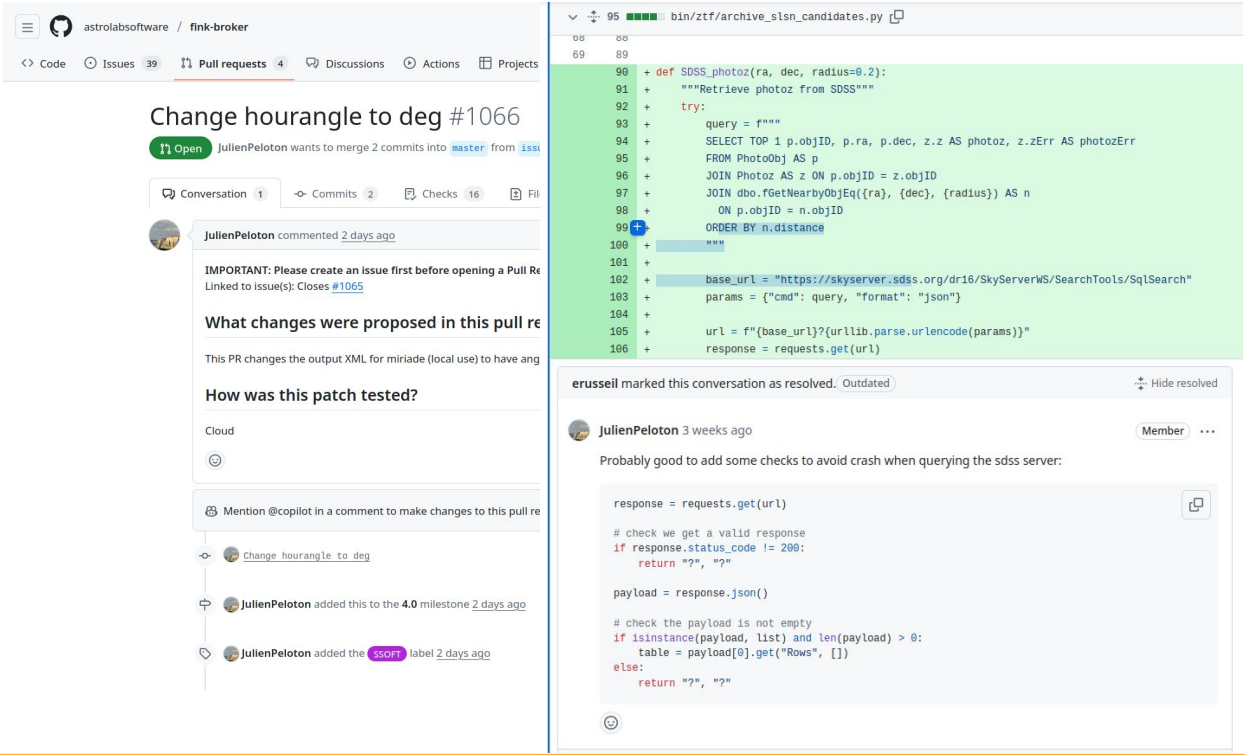


GitLab

GitHub

Forge

Revue de Code / Fusion de branche

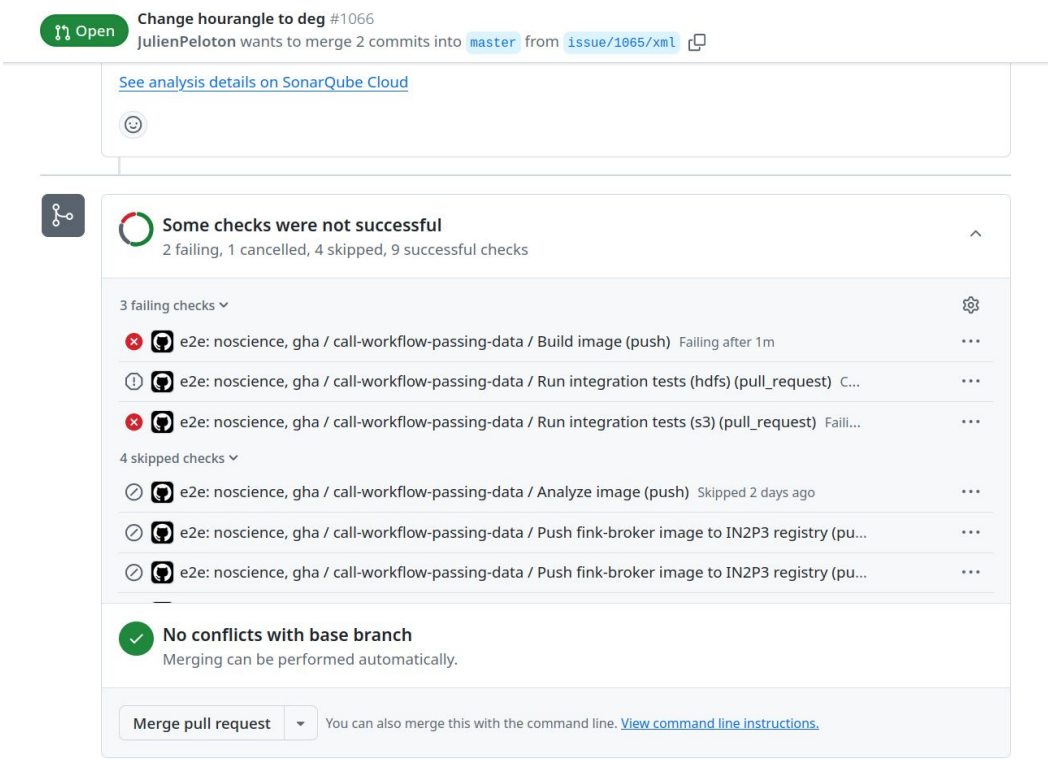


GitLab

GitHub

Forge

Revue de Code / Fusion de branche



GitLab

GitHub

Integration Continue (CI)

Maintenir un niveau de qualité constant au court du temps

- détection précoce des erreurs
- automatisation

Integration Continue (CI)

Maintenir un niveau de qualité constant au court du temps

- détection précoce des erreurs
- automatisation

CI = exécution automatique de tâches à chaque modification (commit/PR)

Integration Continue (CI)

Maintenir un niveau de qualité constant au court du temps

- détection précoce des erreurs
- automatisation

CI = exécution automatique de tâches à chaque modification (commit/PR)

Que peut t'on faire dans une CI ?

→ A peu près tout

Integration Continue (CI)

Maintenir un niveau de qualité constant au court du temps

- détection précoce des erreurs
- automatisation

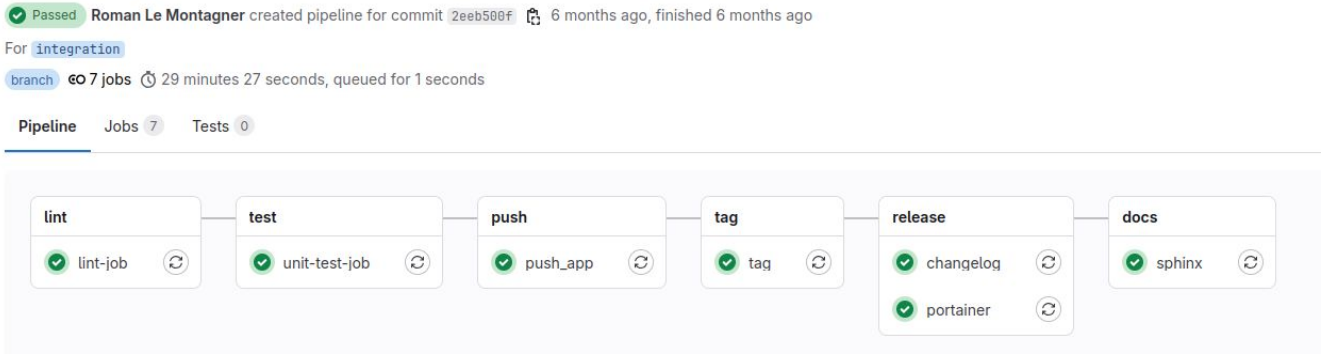
CI = exécution automatique de tâches à chaque modification (commit/PR)

Que peut t'on faire dans une CI ?

Mais en général

- Build
- Tests (couverture de code)
- lint

Integration Continue (CI)



Etape de la CI

- Execution sur **machine isolé**
- Chaque étape **indépendante**
- **S'arrête** dès qu'une étape échoue
- Résultat **visible** par tous

Integration Continue (CI)

.gitlab-ci.yml

```yaml
# gitlab-ci - GitLab CI Configuration file (ci.json)
stages:
  - lint
  - test

# Image officielle Python, simple et reproductible
image: python:3.11-slim

before_script:
  - python -m pip install --upgrade pip
  - pip install -r requirements.txt
  - pip install pytest pytest-cov black flake8 mypy

lint:
  stage: lint
  script:
    - black --check src/
    - flake8 src/
    - mypy src/

tests:
  stage: test
  script:
    - pytest --cov=src --cov-report=xml --cov-report=term
  artifacts:
    paths:
      - coverage.xml
    expire_in: 1 week
```

python-ci.yml

```yaml
name: Python CI

on:
  push:
  pull_request:

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repo
        uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: "3.11"

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
          pip install pytest pytest-cov black flake8 mypy

      - name: Lint
        run: |
          black --check src/
          flake8 src/
          mypy src/

      - name: Run tests
        run: |
          pytest --cov=src --cov-report=xml --cov-report=term
```

Integration Continue / Déploiement Continue (CI/CD)

CI = exécution automatique de tâches à chaque modification (commit/PR)
CD = déploiement automatique d'une version validée après la CI

Publication de :

- packages Python
- images Docker
- documentation
- applications web (API/Flask…)

Réduit les erreurs humaines et accélère les cycles

Integration Continue / Déploiement Continue (CI/CD)

CI = exécution automatique de tâches à chaque modification (commit/PR)

CD = déploiement automatique d'une version validée après la CI

Réduit les erreurs humaines et **accélère les cycles**

- Système en deux temps
  - Plateforme intégration
  - Plateforme production

Exemple:

- Intégration@Mésocentre UPSaclay - VirtualData
- Production@CC-IN2P3

Packaging & Publication

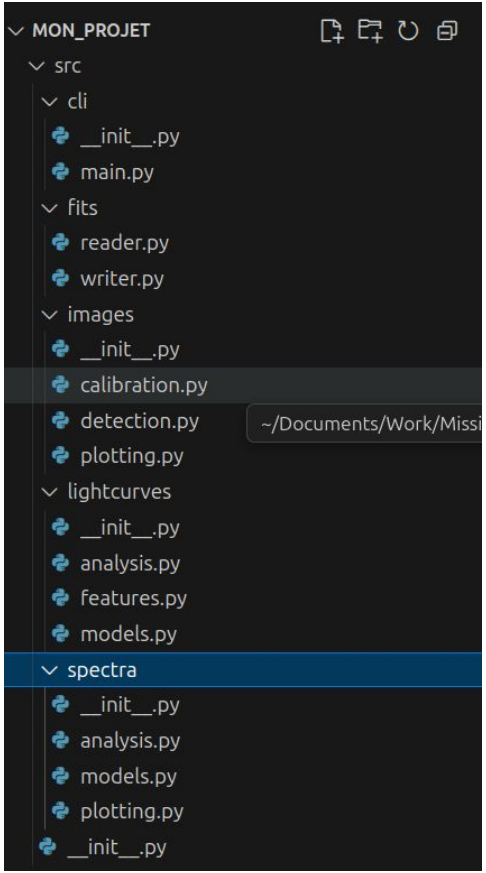
Pourquoi packager un logiciel ?

- Installation : `pip install mon_paquet`
- Partage : Pypi, Github release…

Packaging & Publication

Que contient un paquet ?

- le code (modules / paquets)



Packaging & Publication

Que contient un paquet ?

- le code (modules / paquets)
- un manifeste (pyproject.toml)

pyproject.toml > {} project > [ ] dependencies

```toml
[project]
name = "grandma-gcn"
version = "9.2.0"
description = "Automated ingestion, analysis, and distribution of gravitational wave (GW)
authors = [{ name = "Roman", email = "roman.lemontagner@gmail.com" }]
dependencies = [
    "gcn-kafka>=0.3.3",
    "tomli>=2.0.2",
    "pytz>=2025.2",
    "slack-sdk>=3.35.0",
    "fink-utils>=0.41.0",
    "typing-extensions>=4.13.2",
    "astropy>=7.0.2",
    "astropy-healpix>=1.1.2",
    "celery[redis]>=5.5.2",
    "dotenv>=0.9.9",
    "gwemopt @ git+https://github.com/FusRoman/old_gwemopt@v1.3.1",
    "ligo-skymap>=2.4.0",
    "yarl>=1.20.0",
    "spherical-geometry>=1.3.3",
    "sqlalchemy>=2.0.41",
    "psycopg[binary]>=3.2.9",
    "alembic>=1.16.2",
    "python-dotenv>=1.1.1",
    "flask>=3.1.1",
    "gunicorn>=23.0.0",
]
requires-python = ">=3.12"
readme = "README.md"
license = { text = "CeCILL-C" }

[build-system]
requires = ["pdm-backend"]
build-backend = "pdm.backend"


[tool.pdm]
distribution = true
```

Packaging & Publication

Que contient un paquet ?

- le code (modules / paquets)
- un manifeste (pyproject.toml)

Métadonnées indispensable:

- Nom du package
- version, description courte.
- Long description (README).
- Dépendances.
- Auteur(s)
- License

pyproject.toml > {} project > [ ] dependencies

```toml
[project]
name = "grandma-gcn"
version = "9.2.0"
description = "Automated ingestion, analysis, and distribution of gravitational wave (GW)
authors = [{ name = "Roman", email = "roman.lemontagner@gmail.com" }]
dependencies = [
    "gcn-kafka>=0.3.3",
    "tomli>=2.0.2",
    "pytz>=2025.2",
    "slack-sdk>=3.35.0",
    "fink-utils>=0.41.0",
    "typing-extensions>=4.13.2",
    "astropy>=7.0.2",
    "astropy-healpix>=1.1.2",
    "celery[redis]>=5.5.2",
    "dotenv>=0.9.9",
    "gwemopt @ git+https://github.com/FusRoman/old_gwemopt@v1.3.1",
    "ligo-skymap>=2.4.0",
    "yarl>=1.20.0",
    "spherical-geometry>=1.3.3",
    "sqlalchemy>=2.0.41",
    "psycopg[binary]>=3.2.9",
    "alembic>=1.16.2",
    "python-dotenv>=1.1.1",
    "flask>=3.1.1",
    "gunicorn>=23.0.0",
]
requires-python = ">=3.12"
readme = "README.md"
license = { text = "CeCILL-C" }

[build-system]
requires = ["pdm-backend"]
build-backend = "pdm.backend"


[tool.pdm]
distribution = true
```

Packaging & Publication

Que contient un paquet ?

- le code (modules / paquets)
- un manifeste (pyproject.toml)
- une licence

LICENSE

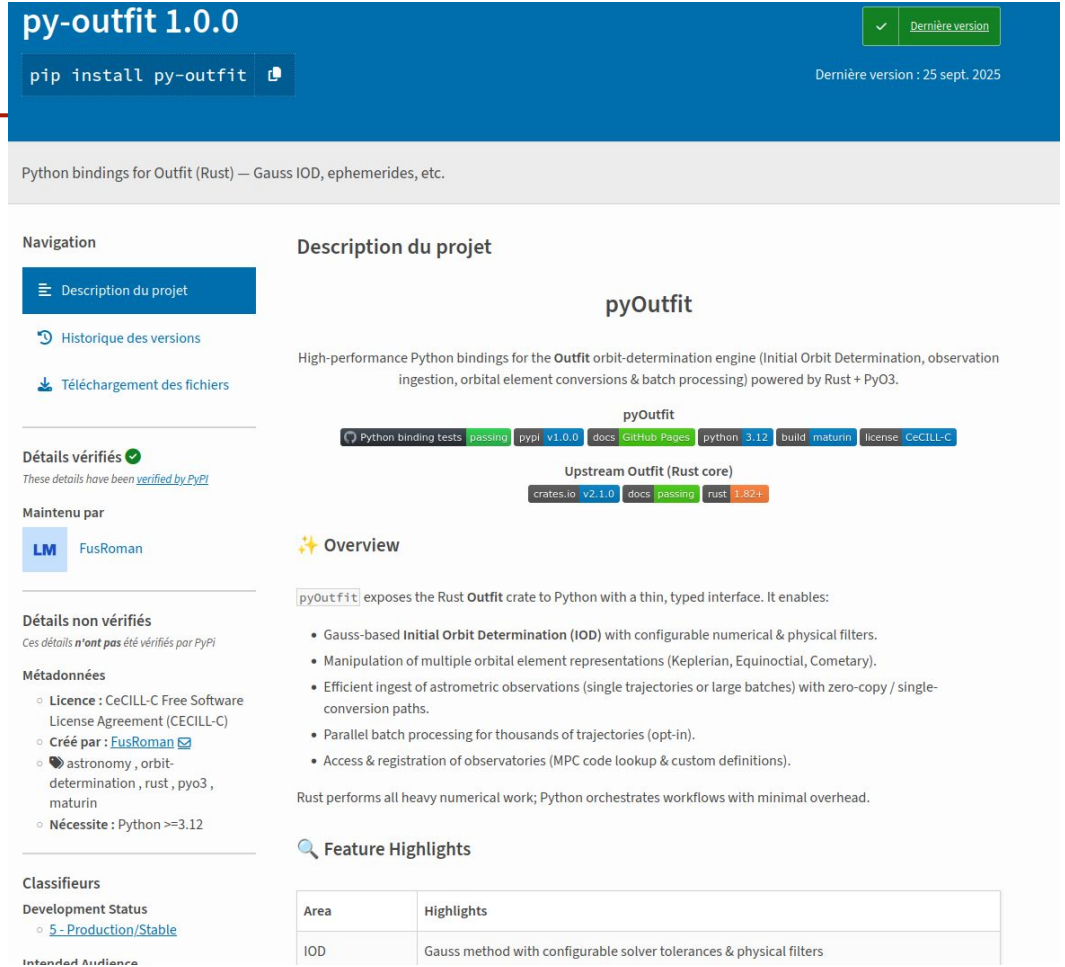
```
CeCILL-C FREE SOFTWARE LICENSE AGREEMENT     Chat Ctrl+L Edit Ctrl+I


    Notice

This Agreement is a Free Software license agreement that is the result
of discussions between its authors in order to ensure compliance with
the two main principles guiding its drafting:

    * firstly, compliance with the principles governing the distribution
      of Free Software: access to source code, broad rights granted to
      users,
    * secondly, the election of a governing law, French law, with which
      it is conformant, both as regards the law of torts and
      intellectual property law, and the protection that it offers to
      both authors and holders of the economic rights over software.

The authors of the CeCILL-C (for Ce[a] C[nrs] I[nria] L[ogiciel] L[ibre])
license are:

Commissariat à l'Energie Atomique - CEA, a public scientific, technical
and industrial research establishment, having its principal place of
business at 25 rue Leblanc, immeuble Le Ponant D, 75015 Paris, France.

Centre National de la Recherche Scientifique - CNRS, a public scientific
and technological establishment, having its principal place of business
at 3 rue Michel-Ange, 75794 Paris cedex 16, France.

Institut National de Recherche en Informatique et en Automatique -
INRIA, a public scientific and technological establishment, having its
principal place of business at Domaine de Voluceau, Rocquencourt, BP
105, 78153 Le Chesnay cedex, France.


    Preamble

The purpose of this Free Software license agreement is to grant users
```

Packaging & Publication

Que contient un paquet ?

- le code (modules / paquets)
- un manifeste (pyproject.toml)
- une licence
- les tests
- la documentation

Publication (Pypi)



Packaging & Publication

Que contient un paquet ?

- le code (modules / paquets)
- un manifeste (pyproject.toml)
- une licence
- les tests
- la documentation

Outils standard de gestion de paquet

- Poetry et PDM

Packaging & Publication

Que contient un paquet ?

- le code (modules / paquets)
- un manifeste (pyproject.toml)
- une licence
- les tests
- la documentation

Outils standard de gestion de paquet

- Poetry et **PDM**

- Gestion de dépendances
- build automatique
- Environnement local au paquet

Reproductibilité

Pourquoi la reproductibilité ?

- Pouvoir refaire l'expérience et obtenir les mêmes résultats.
- Faciliter la vérification, la correction et l'extension des travaux.
- Réduire l'effet "résultats miracles".
- Améliorer la fiabilité des pipelines numériques.

Reproductibilité

Comment garantir la reproductibilité ?

Appliquer ce qu'on a vu dans ce cours

- Bonne architecture logicielle
- Tests (automatisé CI)
- **Gestion de version**
  - Code **et donnée**
- Packaging & Publication

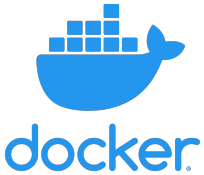
Reproductibilité

Comment garantir la reproductibilité ?

Appliquer ce qu'on a vu dans ce cours

- Bonne architecture logicielle
- Tests (automatisé CI)
- **Gestion de version**
  - Code **et donnée**
- Packaging & Publication

Conteneurisation (Docker/Singularity)



- CI/CD
- cloud
- microservice

- HPC
- No root
- Pipeline reproducible

Reproductibilité

Comment garantir la reproductibilité ?

```dockerfile
# Image de base Ubuntu
FROM ubuntu:22.04

# Mise à jour et installation de Python + pip
RUN apt-get update && apt-get install -y \
    python3 \
    python3-pip \
    && rm -rf /var/lib/apt/lists/*

# Installation d'un paquet Python d'exemple (requests)
RUN pip3 install requests

# Répertoire de travail
WORKDIR /app

# Copier ton projet dans le conteneur (optionnel)
COPY . /app

# Commande par défaut : lance un shell
CMD ["bash"]
```

Conteneurisation (Docker/Singularity)

- Un fichier recette de cuisine (Dockerfile)
  - OS (Linux, MacOs, Windows)
  - Dépendances système
  - Paquet python
- Execution rapide
  - docker build
  - docker run
- Plus léger qu'une VM
  - VM O(GB)
  - Conteneur O(MB)

Reproductibilité

Une note sur le calcul flottant (Norme IEEE 754)

- Garantie de résultat déterministe et unique
  - +, -, *, /
  - racine carré
  - FMA (Fused Multiply-Add : a + (b * c))

Reproductibilité

Une note sur le calcul flottant (Norme IEEE 754)

- Garantie de résultat déterministe et unique
  - +, -, *, /
  - racine carré
  - FMA (Fused Multiply-Add : a + (b * c))

- **Aucune garantie pour tout le reste**
  - fonction transcendante

- e^x, 2^x, 10^x
- $e^x - 1$, $2^x - 1$, $10^x - 1$
- $\ln x$, $\log_2 x$, $\log_{10} x$
- $\ln(1+x)$, $\log_2(1+x)$, $\log_{10}(1+x)$
- $\sqrt{x^2+y^2}$
- $1/\sqrt{x}$
- $(1+x)^n$ for $x \geq -1$ (named *compound* and used to compute an exponential growth, whose rate cannot be less than -1)[50]
- $x^{\frac{1}{n}}$
- x^n, x^y
- $\sin x$, $\cos x$, $\tan x$
- $\arcsin x$, $\arccos x$, $\arctan x$, $\operatorname{atan2}(y, x)$
- $\operatorname{sinPi} x = \sin \pi x$, $\operatorname{cosPi} x = \cos \pi x$, $\operatorname{tanPi} x = \tan \pi x$ (see also: Multiples of π)
- $\operatorname{asinPi} x = \frac{1}{\pi} \arcsin x$, $\operatorname{acosPi} x = \frac{1}{\pi} \arccos x$, $\operatorname{atanPi} x = \frac{1}{\pi} \arctan x$, $\operatorname{atan2Pi}(y, x) = \frac{1}{\pi} \operatorname{atan2}(y, x)$ (see also: Multiples of π)
- $\sinh x$, $\cosh x$, $\tanh x$
- $\operatorname{arsinh} x$, $\operatorname{arcosh} x$, $\operatorname{artanh} x$

reproductibilité ⟶ Science ouverte

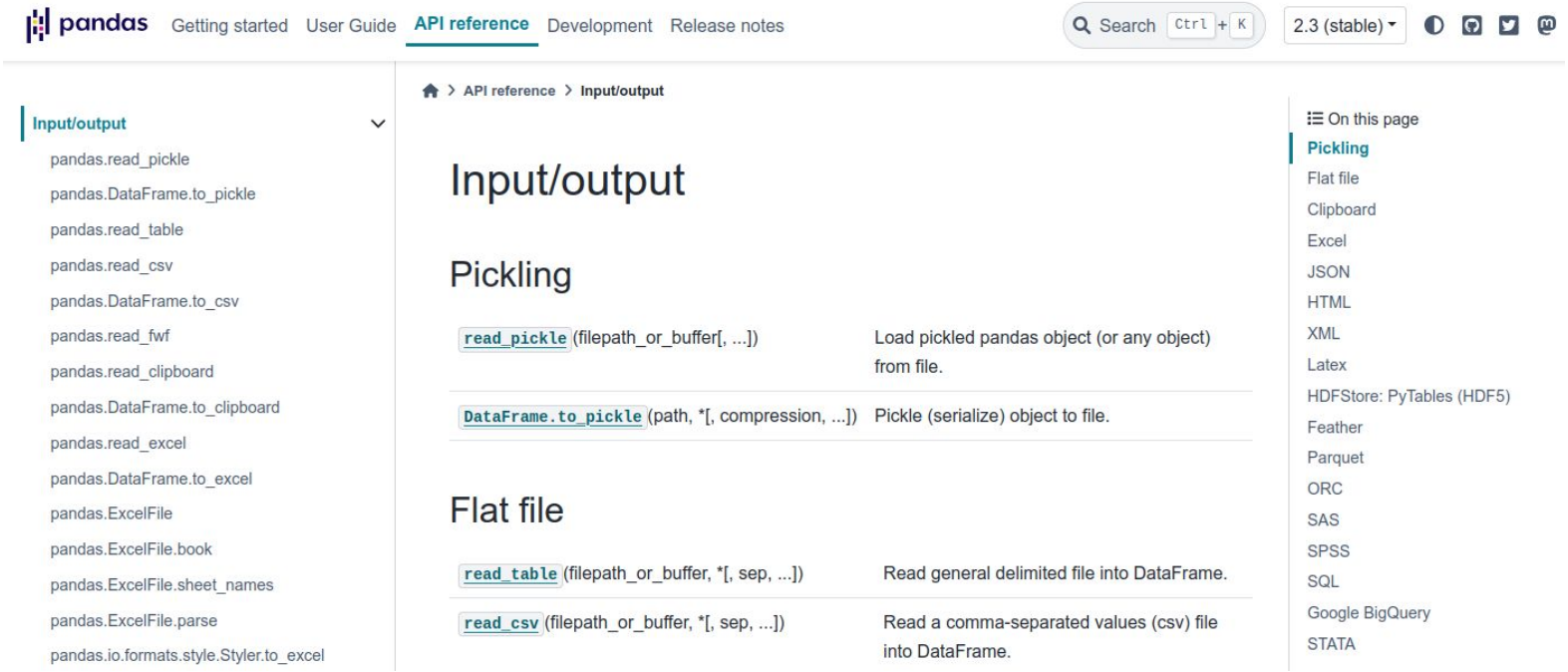
Informatique repose sur des codes, des données, des pipelines, des modèles et des infrastructures

- **Open Source** : code, simulateur, pipeline, notebooks
- **Open Data** : format de donnée ouvert, public, métadonnées complètes (provenance, calibration…)
- **Open methods** / **Open workflows** : Jupyter, DAG versionnés
- **Open access** : publications accessibles et lien vers forges (github, gitlab ouvert…)

Reproducibilité ⟶ Science ouverte

Initiative pour des données de qualité

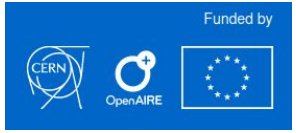
- FAIR (Findable, Accessible, Interoperable, Reusable)



Reproducibilité ⟶ Science ouverte

Initiative pour la science ouverte

- FAIR (Findable, Accessible, Interoperable, Reusable)
  - ⚠️ FAIR ≠ Open
- ZENODO (https://zenodo.org/)
  - développé par le CERN



  - archive durable de production scientifique *(jeux de données, logiciels / dépôts Git, notebooks, papiers, rapports, posters, slides, etc.)*
  - DOI permanent et de version

Conclusion

La conception d'un logiciel s'approche de celle d'un instrument

- Prototype
- Version stable
- Tests
- Normes et standard

On a aussi des outils

- Git, PDM, CI/CD

Conclusion

Des outils puissant mais dangereux

- **Large Language Model (LLM)**
- A utiliser quand on sait ce qu'on fait
  - Code / Algorithme (~)
  - Tests ✔
  - Refactoring ✔
  - Documentation ✔
  - écriture de CI ✔