Qualité du Code Source - Bachelor CSI

Christophe Brun

Campus Saint-Michel IT

21 septembre 2023

https://github.com/St-Michel-IT/qualite-code-source

Table des matières

1. Programme du module
2. Généralités
3. Bonnes pratiques
4. Tests unitaires
5. Versionning (GIT)
6. Plateforme d'intégration et de livraisons continues
7. Conclusion

Compétences :

- Maîtriser la création et l'exécution de tests unitaires avec un framework de tests unitaires.
- Mettre en place une démarche d'amélioration de la qualité du code.
- Utiliser une plateforme d'intégration et de livraison continues

Qualité du Code Source

Le programme officiel des 3 jours du module

1. Les tests unitaires
   - Intégration des tests unitaires dans un projet
   - Assertions simples, interprétation des messages de retour
   - Gestion des exceptions
   - Tests utilisant des jeux de données
2. Bonnes pratiques
   - Formatage du code source (indentation, CamelCase)
   - Nomenclature du code
   - Génération de la documentation
   - Organisation du code d'un projet
3. Versionning (GIT)
   - Mise en place d'une plateforme d'intégration continue (GitLab CI)
   - Conteneurisation d'une application (API)
   - Configuration d'un pipeline de tests
4. Plateforme d'intégration et de livraisons continues
   - Mise en place d'un serveur d'intégration continue
   - Gestion de des tâches
   - Automatisation des tests unitaires et d'intégration
   - Génération et interprétation de rapports
   - Déploiement de la version validée

Evaluation

- 60 % sur le projet développé au cours du module. Basée en partie sur les commits des développements pour comprendre facilement l'évolution du code.
  - 15 % sur les bonnes pratiques
  - 15 % sur le testing
  - 15 % sur le versioning
  - 15 % sur l'intégration continue
- 40 % sur une évaluation écrite finale

Intervenant sur le module Qualité du Code Source

Christophe Brun, conseil en développement informatique

- 1ere année d'intervenant à Saint-Michel 🤩.
- 7 ans de conseil en développement au sein d'SSII .
- 7 ans de conseil en développement à mon compte PapIT.
- Passionné !

UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR

université de BORDEAUX

Pourquoi le génie logiciel en général ?

- Éviter les bugs
- Prévenir les bugs des futurs développeurs, i.e., la maintenance
- Performance, i.e., la réduction des coûts
- Parce que maintenant on peut

Qu'est-ce qu'un bug informatique ?

- Historiquement, un insecte dans un calculateur d'Harvard en 1946 [1]
- Martin Hopkins d'IBM en 1969 dit "Programmers call their errors "bugs" to preserve their sanity; that number of "mistakes" would not be psychologically acceptable!"[2]
- Comportement contraire à la spécification d'un logiciel (point de vue développeur).
- Ou comportement inattendu d'un logiciel (point de vue des autres).
- Peut-être connu et maîtrisé. L'impact est mineur et sa correction n'est pas prioritaire
- Peut-être inconnu et hors de contrôle

[1] INRIA, Bug, https://aconit.inria.fr/omeka/items/show/523.html

[2] NATO SCIENCE COMMITTEE, SOFTWARE ENGINEERING TECHNIQUES, http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF

Qu'est-ce qu'un bug informatique ?

- Bug dans les têtes, comme le bug de l'an 2000. Lié au format de date YY-MM-DD. Pas de gros soucis connus mais jusqu'à 80% d'annonces d'embauche en plus dans les SSII françaises...
- Le pilote automatique Tesla, "Certain 2016-2023 Model S, Model X,2017-2023 Model 3, and 2020-2023 Model Y vehicles equipped with Full Self-Driving Beta (FSD Beta) software or pending installation"[3]
- Le pilote automatique du Boeing 737 MAX après 2 crashes, "Boeing agrees with the FAA's decision and request, and is working on the required software"[4]
- WhatsApp et la vie privée, zero day RCE découvertes régulièrement, comme en 2022[5]

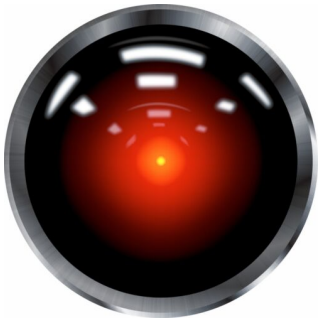
[3] Tesla, https://www.tesla.com/en_eu/support/annual-and-recall-service

[4] Boeing, conférence de presse, https://theaircurrent.com/aviation-safety/faa-and-boeing-initially-disagreed-on-severity-of-catastrophic-737-max-sof

[5] Sophos, CVE-2022-36934 et CVE-2022-27492, integer overflow et underflow, https://nakedsecurity.sophos.com/2022/09/27/whatsapp-zero-day-exploit-news-scare-what-you-need-to-know/

Qu'est-ce qu'un bug informatique ?

Bug de 2038, le 19 janvier 2038 à 3 h 14 min 7 s soit le 1 janvier 1970 plus `0b10**31`, i.e., 32bit en secondes



Celui qui reste de la S.F., où l'humain perd le contrôle sur l'IA

Très divers mais le plus souvent, il est "entre la chaise et le clavier..."

Combien coûte un bug informatique ?

- Plus il est trouvé tôt dans la vie du logiciel, moins il coûte.
  Donc, on teste le plus possible pour ne pas trouver de bug en production (e.g., recall, DeFi).
  C'est le "Shift left" !
- En développement on paie le correctif, en production son impact
- Postulat du génie logiciel, c'est que le correctif est moins coûteux que l'impact
- Le coût de l'impact dépend du domaine, mais peut être très élevé, comme dans la DeFi, cf. https://rekt.news/fr/

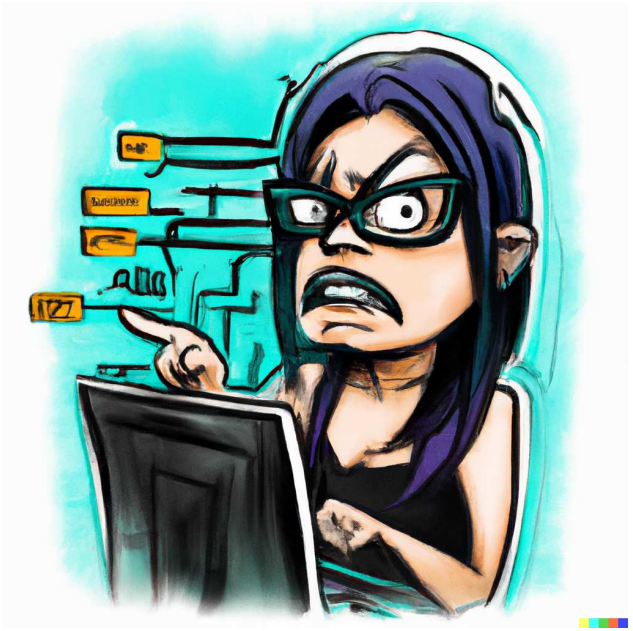
Capacités du génie logiciel moderne ?

- Nous
- Les connaissances acquises dans ce module :
  - Bonnes pratiques de développement
  - Testing automatique
  - Bonnes pratiques de développement
  - Outils collaboratifs de développement
  - CI/CD, (CI = Continuous Integration (cf. Mise en place d'un serveur d'intégration continue), CD = Continous Delivery (cf. Déploiement de la version validée))

Bonnes pratiques, sommaire

- Langages modernes comme Java et Python

- Parfois générales ou propres à une convention, un langage
- Les outils
- Pas de syntaxe 🚫
- Pas d'architecture 🚫



Bonnes pratiques, pourquoi?

- Les nouvelles applications et les développements en cours et à venir
- Le correctif
- La maintenance évolutive
- Le code legacy, même si elles étaient moins formalisées, le bon sens a toujours existé !

Bonnes pratiques, l'approche de Python

- Python a une syntaxe éloignée des langages de son temps qui sont toutes déclinées de celle du C. Les syntaxes de C, C++, C#, Java et même JavaScript
- Le pari d'une nouvelle syntaxe était risqué, mais G. Van Rossum souhaite une meilleure lisibilité
- "code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code. As PEP 20 says, **Readability counts**"[6]

[6] Guido Van Rossum, https://peps.python.org/pep-0008/

Bonnes pratiques, l'approche de Python

```
In [1]: import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Bonnes pratiques, l'approche de Python

En cas d'oubli, le mantra de Python A.K.A The Zen of Python, est dans l'interpréteur !

Le bon sens paysan fait consensus dans tous les langages de programmation !

Bonnes pratiques, Python et PEP

- PEP est la convention de codage recommandée par Python, comme PEP8, le Style Guide for Python Code, à lire SVP
- Définit les espaces utiles à la lisibilité et les inutiles
- Snake_case pour les variables
- Définit les casses et la déclaration :
  - Attribut et méthode privé/publique :

```
class Case:
    public = 0

Case.public

Out[3]: 0

class Case:
    __private = 0

In [13]: Case.__private
-----------------------------------------

AttributeError Traceback (most recent
     call last)
Cell In [13], line 1
----> 1 Case.__private

AttributeError: type object 'Case' has
     no attribute '__private'
```

Bonnes pratiques, Python et PEP

- Déclaration des constantes en upper case avec underscore, comme la plupart des langages :

```
In [17]: PLANCK_CONSTANT = 6.62607015*(10^-34) # Upper case with underscore
      for constants

In [18]: PLANCK_CONSTANT = 42 # But constant those not really exists in
     Python

In [19]: PLANCK_CONSTANT
Out[19]: 42
```

- Déclaration des classes en CamelCase et fonctions en snake_case :

```
In [20]: class UnNomEnCamelCase:
...: pass
...:

In [21]: def un_verbe_au_moins_en_snake_case():
...: pass
...:
```

Bonnes pratiques, Python et PEP

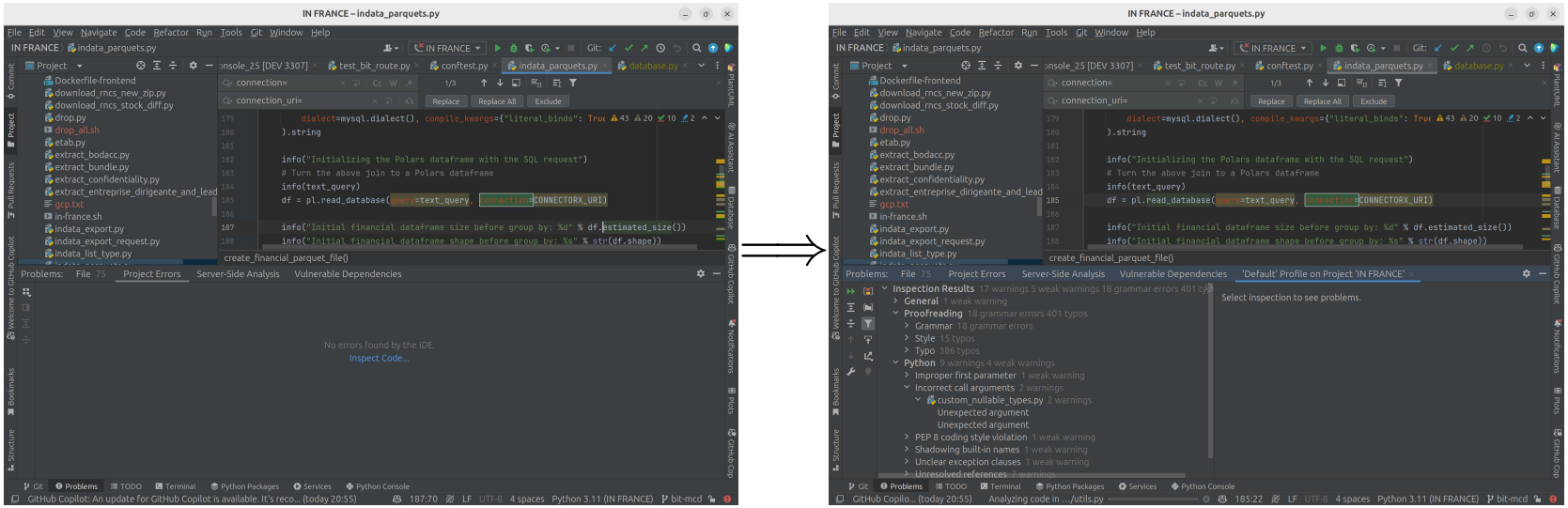
- Les commentaires, en ligne si possible :

```
In [22]: is_even = lambda x: x % 2 == 0 # Il est clair que l'on commente
     cette ligne mais c'est long

In [23]: # Retourne True pour un chiffre pair

In [24]: is_even = lambda x: x % 2 == 0
```

- Favoriser l'indentation même si la syntaxe en ligne est valide, après les ":" des conditions et des déclarations :

```
def is_even(x):return x % 2 == 0 # Pas biennnn

def is_even(x):
    return x % 2 == 0 # Biennnn
```

Bonnes pratiques, l'IDE, une aide précieuse

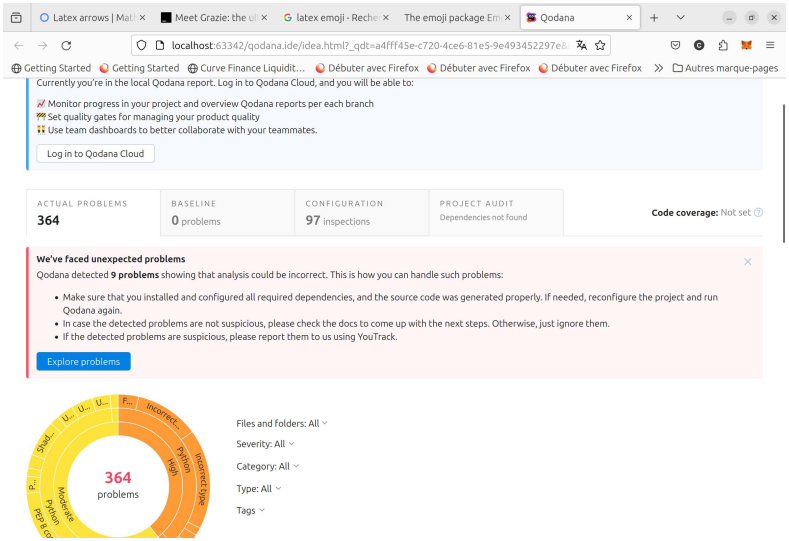


Dans la fenêtre dédiée aux divers warnings et erreurs, après analyse du project, les IDEs modernes détaillent de multiples types d'erreurs et warnings. Ils peuvent même les corriger automatiquement.

Une liste non exhaustive des IDEs ouverts :

- VS Code (Avec Black formatter et Pylint)
- Pycharm pour Python
- Eclipse pour Java
- IntelliJ pour Java
- Zed (`https://zed.dev/`, sur Mac OS et Linux pour l'instant

Bonnes pratiques, l'IDE, intégration à des outils tiers

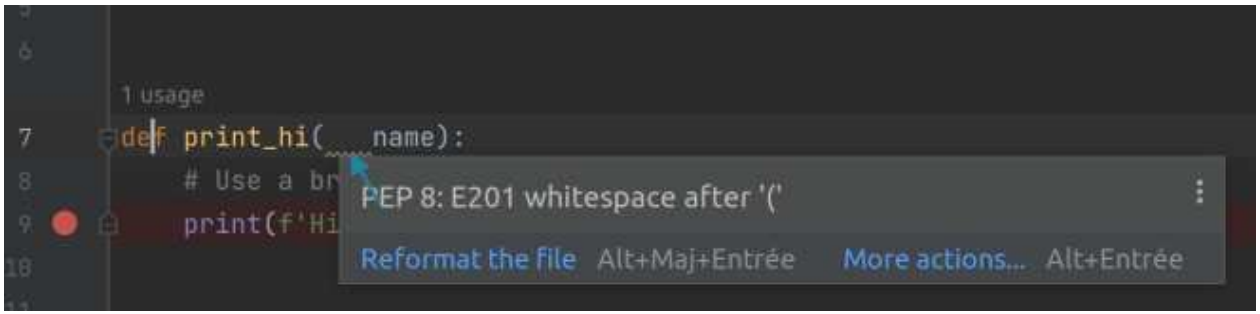


Certains plugins peuvent se connecter à des applications dédiées à l'analyse des sources comme SonarQube 🇨🇭, Qodana, etc. Ils permettent une analyse plus complète et d'enregistrer des indicateurs permettant de suivre l'évolution dans le temps de la qualité des sources du projet.

Bonnes pratiques, l'IDE, lire les indications

- VS Code ou Pycharm ont toutes ces fonctionnalités modernes
- Dans la carte, sur le côté on voit rapidement les soucis sur tout un fichier

- Dans le code source en cours d'édition



Bonnes pratiques, l'approche de Java

- Développé au début des années 90 chez Sun Microsystems, racheté par Oracle, actuellement le principal développeur
- "Its syntax is similar to C and C++, but it omits many of the features that make C and C++ complex, confusing, and unsafe."[7]
- Plus safe au niveau de la mémoire car il a un ramasse miette et plus simple à coder. Mais la syntaxe doit être familière pour les développeurs et donc s'inspirer des langages de l'époque C et C++

[7] Oracle, Java Virtual Machine Specification, https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-1.html

Bonnes pratiques, l'approche de Java

Oracle, Java code conventions[8] :

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

[8] Oracle, Java code conventions, https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html

Bonnes pratiques, l'approche de Java du nommage

Oracle, Java naming convention (identique pour JavaScript)[9] :

Packages	`com.sun.eng` `edu.cmu.cs.bovik.cheese`
Classes	`class Raster;` `class ImageSprite;`
Interfaces	`interface RasterDelegate;` `interface Storing;`
Methods	`run();` `runFast();`
Variables	`char c;` `float myWidth;`
Constants	`static final int MIN_WIDTH = 4;`

[9] Oracle, Java code conventions, https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

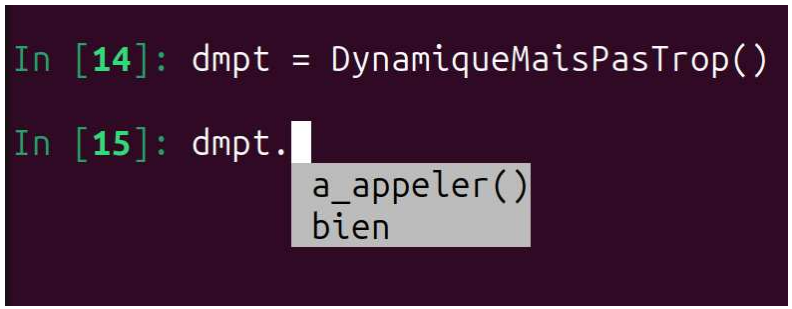
Bonnes pratiques, ne pas abuser des fonctionnalités du typage dynamique

Bonnes pratiques, ne pas abuser des fonctionnalités du typage dynamique

Les attribues d'un objet peuvent être définis dynamiquement mais cela empêche d'explorer la structure d'un objet lors du développement :

```
class DynamiqueMaisPasTrop:
    def __init__(self):
        self.bien = 0 # bien sera toujours visible dans l'IDE
    def a_appeler(self):
        self.pas_bien = 0 # Ne sera pas visible avant l'exécution de a_appeler
```

Tous les IDE seront perdus et l'attribue `pas_bien` ne sera pas visible avant l'exécution de la fonction `a_appeler`, comme ici IPython :



Tous les attribues doivent être définis dans le constructeur. PEP le recommande en Python par exemple. Cf. l'exemple à ne pas suivre d'SQLAlchemy

Bonnes pratiques, les différences entre Java et Python

- Rien à voir avec Python
- Java est typé statiquement, Python dynamiquement
- Majoritairement du CamelCase
- Pas du tout de snake_case
- Majoritairement du CamelCase
- Pas du tout de snake_case
- Toutes les JVM ont une compatibilité ascendante !
- Quand Python a une nouvelle bonne idée, elle peut être intégrée, on verra plus tard la compatibilité...e.g., la célèbre rupture de compatibilité Python 2 à 3

Bonnes pratiques, les flottants et leurs pièges

[10] Floating Point Arithmetic: Issues and Limitations, https://docs.python.org/3/tutorial/floatingpoint.html

[11] What Every Computer Scientist Should Know About Floating-Point Arithmetic, https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

Bonnes pratiques, les flottants et leurs pièges

Peu de nombres réels peuvent être représentés avec exactitude en informatique [10,11]:

```
In [8]: 0.2 + 0.1 == 0.3 # Should be True
Out[8]: False

In [9]: 2/10 + 1/10 == 3/10 # Should be True
Out[9]: False

In [10]: 1/2 + 4/16 + 1/16 == 8/16 + 4/16 +1/16 # Should be True
Out[10]: True
```

De l'arithmétique simple peut donc être fausse !

[10] Floating Point Arithmetic: Issues and Limitations, https://docs.python.org/3/tutorial/floatingpoint.html

[11] What Every Computer Scientist Should Know About Floating-Point Arithmetic, https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

Bonnes pratiques, les flottants et leurs pièges

Les réels sont exacts s'ils peuvent être représentés par une fraction sous la forme :

$$x/2^y$$

Sinon, le nombre réel est représenté par la fraction la plus approchante.
Python a une méthode d'introspection des flottants :

```
In [15]: from math import log, isclose

In [16]: (1/3).as_integer_ratio()
Out[16]: (6004799503160661, 18014398509481984)

In [17]: numerateur, denominateur = (0.33333333333333333333333333333333333333).as
    ...: _integer_ratio() # Le numerateur est 6004799503160661

In [18]: log(denominateur, 2)
Out[18]: 54.0
In [18]: isclose(1/3, numerateur / 2**54) # True if the closest fraction
Out[18]: True
```

La fraction la plus proche du flottant de valeur 1/3 est :

$$6004799503160661/2^{54}$$

Bonnes pratiques, les flottants et leurs pièges et leurs solutions

Une des solutions, est de toujours utiliser des entiers quand c'est possible!
Pour des devises, utiliser des entiers pour les centimes.
Voir même faire tous les calculs en centimes et ne convertir en euros par exemple, qu'à l'affichage.

```
class EuroCents: # Des centimes et de l'arithmétique sans flottant !
    def __init__(self, cents):
        self.cents = cents
    def __repr__(self):
        return str(self.cents / 100) + " €"
    def __add__(self, val):
        return EuroCents(self.cents + val.cents)

montant = EuroCents(5) + EuroCents(5)
print(montant) # >>> 0.1 €
```

Sinon, utiliser des librairies avec des types et fonctions dédiées aux nombres réels.

Bonnes pratiques, les autres langages

- JavaScript n'a pas de coding style officiel publié par l'ECMA International
- Certaines sociétés ou communautés ont publié des coding styles populaires :
  - Google JavaScript Style Guide
  - Airbnb JavaScript Style Guide
  - JavaScript Standard Style (n'a de standard que le nom)
  - Idiomatic
- PHP a PSR, c'est l'acronyme PHP Standard Recommendation
- Pour tout langage, suivre la convention officiel ou une mainstream

Bonnes pratiques, pour tous

- Certaines pratiques issues du bon sens et de l'expérience dans le développement logiciel s'appliquent à tous les langages. C'est plus essentiel que de connaître la dernière architecture logiciel à la mode
- Un best seller, la bible, d'où viennent certaines bonnes pratiques de ce cours, Clean Code: A Handbook of Agile Software Craftsmanship par Robert C. Martin
- Ou CODER PROPREMENT par Robert C. Martin...

Bonnes pratiques, pour tous

The Total Cost of Owning a Mess[12] :

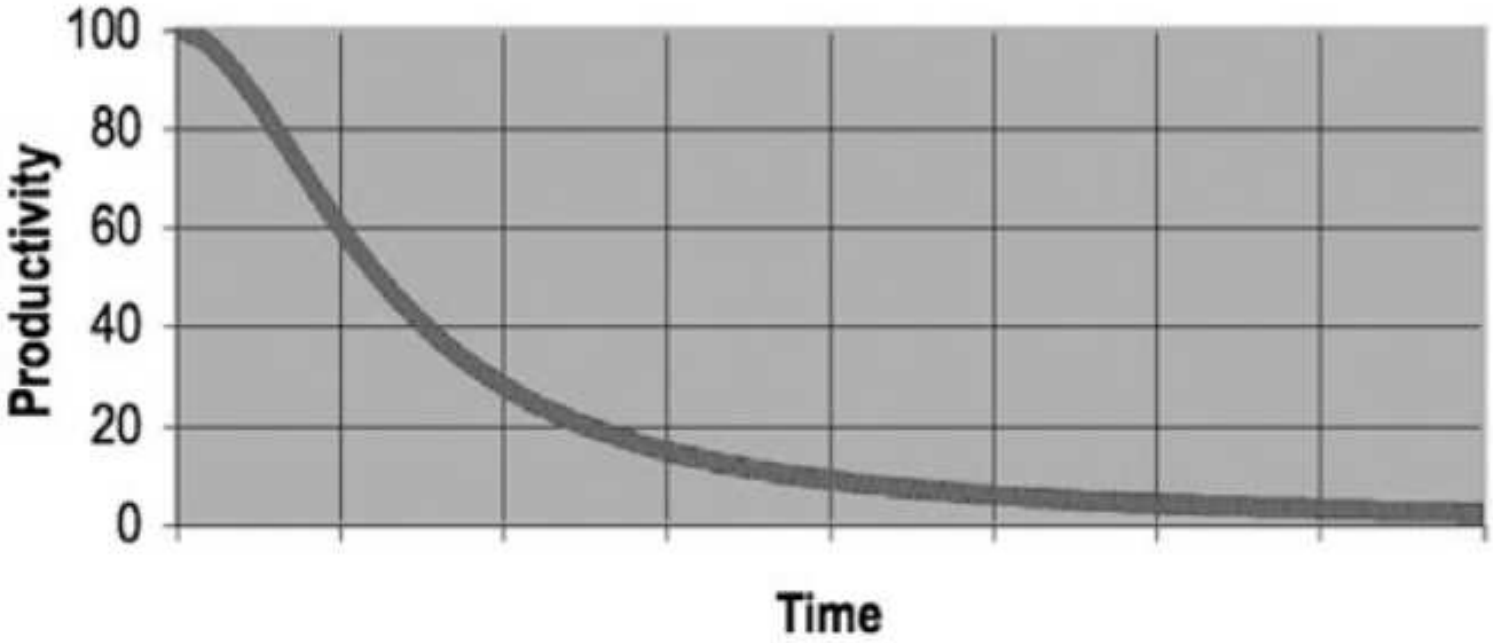


Figure 1-1
Productivity vs. time

[12] Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship

Bonnes pratiques, le fichier source pour tous

- Une ligne fait 80 caractères de long maximum si possible (Java, Python, Clean Code)
- Penser aux vertical rulers des IDE, à 80, 100, 120 caractères par exemple
- Héritage de la carte perforée mais surtout pour faire rentrer facilement la ligne dans un écran/terminal
- Indenter tout, les déclarations de classe ou de fonction, les conditions même quand le langage ne l'oblige pas comme Java

Bonnes pratiques, le fichier source pour tous

Créer des fichiers sources par classe ou fonction, avec un nombre de lignes le plus réduit possible.

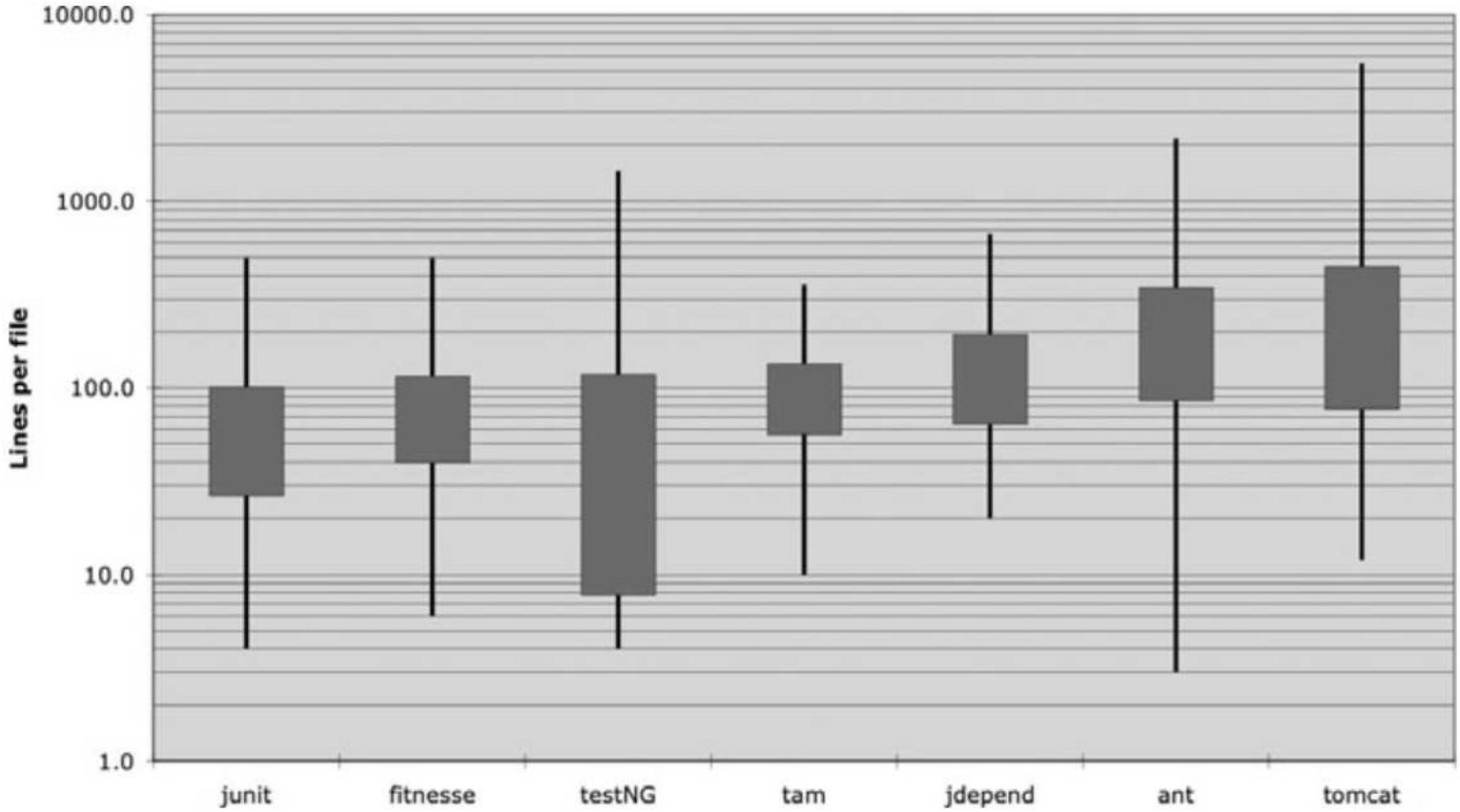


Figure 5-1

File length distributions LOG scale (box height = sigma)

Bonnes pratiques, le fichier source pour tous

```
private static BigInteger getSIRET(
String url,
String user,
String password,
Integer SIREN,
Float latitude,
Float longitude
) {
BigInteger distance = BigInteger.valueOf(0);
try {
Connection con;
con = DriverManager.getConnection(
        url, user, password
);
PreparedStatement stmt = con.prepareStatement( s: "SELECT\n"
        + "etab.siret,\n"
        + "ST_DISTANCE_SPHERE(POINT(etab.longitude, etab.latitude), POINT(?, ?)) AS distance\n"
        + "FROM\n"
        + "etab\n"
        + "WHERE\n"
        + "siren = ?\n"
        + "HAVING distance IS NOT NULL\n"
        + "ORDER BY distance ASC\n"
        + "LIMIT 1;");
stmt.setFloat( i: 1, longitude);
stmt.setFloat( i: 2, latitude);
stmt.setInt( i: 3, SIREN);
ResultSet rs = stmt.executeQuery();
            while (rs.next())
                distance = new BigInteger(rs.getString( i: 1));
```

```
private static BigInteger getSIRET(                                    ⚠ 3  ✓ 18
        String url,
        String user,
        String password,
        Integer SIREN,
        Float latitude,
        Float longitude
) {
    BigInteger distance = BigInteger.valueOf(0);
    try {
        Connection con;
        con = DriverManager.getConnection(
                url, user, password
        );
        PreparedStatement stmt = con.prepareStatement( s: "SELECT\n"
                + "etab.siret,\n"
                + "ST_DISTANCE_SPHERE(POINT(etab.longitude, etab.latitude), POINT(?, ?)) AS dista
                + "FROM\n"
                + "etab\n"
                + "WHERE\n"
                + "siren = ?\n"
                + "HAVING distance IS NOT NULL\n"
                + "ORDER BY distance ASC\n"
                + "LIMIT 1;");
        stmt.setFloat( i: 1, longitude);
        stmt.setFloat( i: 2, latitude);
        stmt.setInt( i: 3, SIREN);
        ResultSet rs = stmt.executeQuery();
        while (rs.next())
            distance = new BigInteger(rs.getString( i: 1));
```

Bonnes pratiques, le naming pour tous

- Donner aux variables des noms qui ont du sens
- Des noms qui doivent révéler vos intentions
- Correcte dans la langue dans laquelle vous développez
- Prononçable
- Que l'on peut rechercher dans les sources, pas de variable d'1, 2, 3 lettres

```
int d; // elapsed time in days
```

😡

```
int elapsedTimeInDays;
```

💕

Bonnes pratiques, le naming pour tous

- Éviter la désinformation
- L'information sur le type doit être précise et exacte
- Pas de L, de I ni de O

```
Queue<Integer> ageList;
```
😡

```
Queue<Integer> ages;
```
💕

Bonnes pratiques, le naming pour tous

- Une variable instance de classe est un nom et un ou plusieurs qualificatifs différenciant cette instance
- Les noms des méthodes et fonctions contiennent au moins un verbe sinon une phrase succincte donc avec verbe également
- Simple à mémoriser
- Un mot seul par concept. Par exemple en anglais retourner une value pourrait utiliser :
  - Return
  - Fetch
  - Get
  - Retrieve
  - ...
- Ne pas dupliquer d'information en répétant un contexte déjà présent dans les niveaux supérieurs. E.g., dans un package dont le nom est celui de la compagnie il est inutile de répéter ce nom dans les classes

Bonnes pratiques, les fonctions pour tous

- Une bonne fonction est petite
- La plus petite possible
- Doit avoir la responsabilité d'une seule action, celle décrite dans son nom
- Pure ou avec effet secondaire mais pas les deux
- S'inspirer de la programmation fonctionnelle pour créer des fonctions pures le plus atomique possible

Bonnes pratiques, les fonctions pour tous

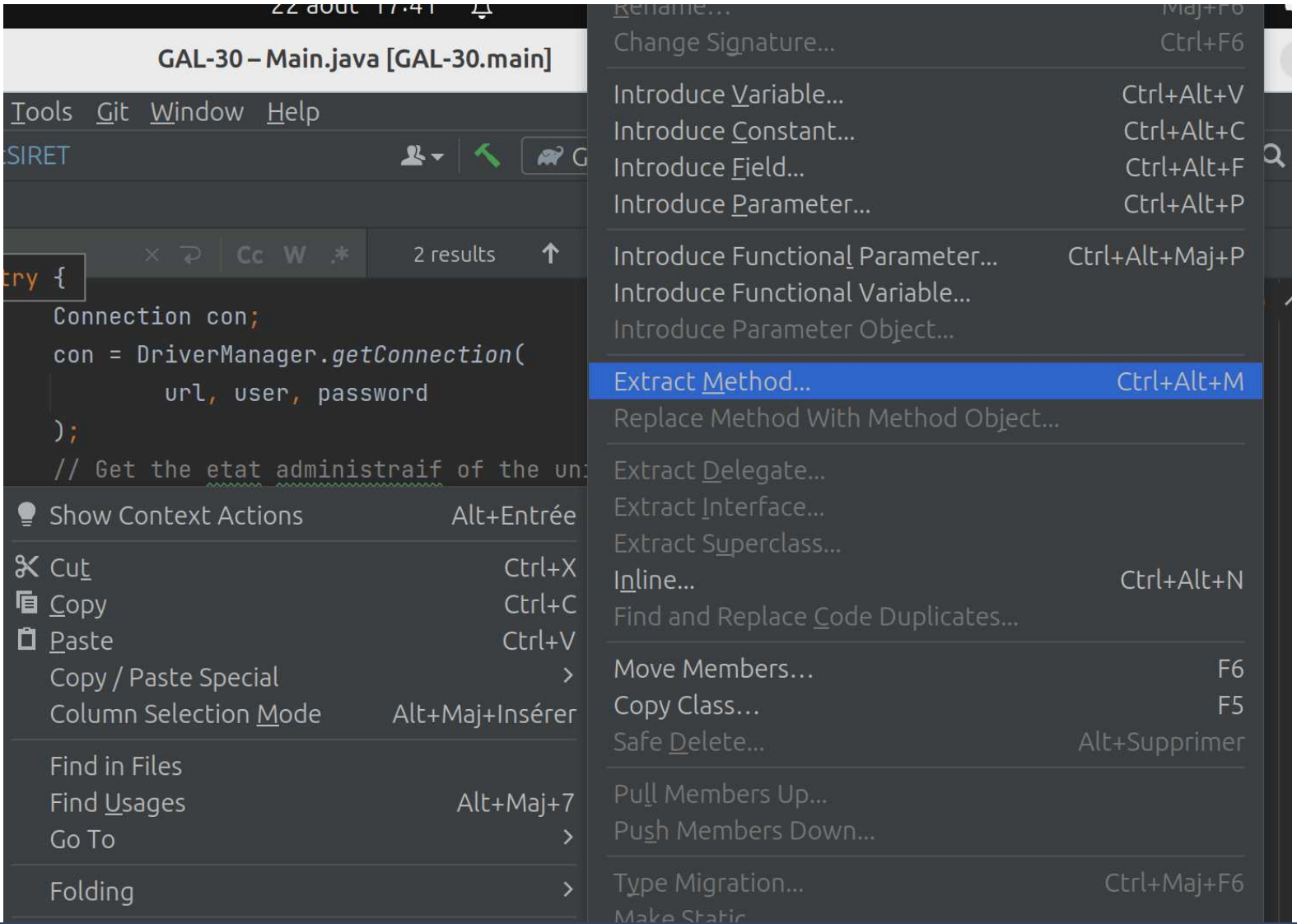
```
In [35]: def is_prime(x):
    ...:     for divisor in range(2, x):
    ...:         if x % divisor == 0:
    ...:             return False
    ...:     return True # Pure, nothing else but returning
    ...:

In [36]: def is_prime(x):
    ...:     for divisor in range(2, x):
    ...:         if x % divisor == 0:
    ...:             print(x + " is a prime number") # A side effect
    ...:             return False
    ...:     return True
    ...:

In [37]: def is_prime(x):
    ...:     for divisor in range(2, x):
    ...:         if x % divisor == 0:
    ...:             return False
    ...:     return True # Pure, nothing else but returning
    ...:

In [38]: def print_is_prime(x): # Only responsible of printing, i.e., the side effect
    ...:     return str(x) + (" is" if is_prime(x) is True else " is not") + " a prime number"
    ...:

In [39]: print_is_prime(11)
Out[39]: '11 is a prime number'

In [40]: print_is_prime(9)
Out[40]: '9 is not a prime number'
```

Bonnes pratiques, les fonctions pour tous

- Atomique, i.e., les fonctions ne font qu'une chose
- Si vous avez plusieurs sections ou plusieurs commentaires, c'est un signe que votre fonction fait plus qu'une chose
- Si possible un seul argument, éventuellement 2, maximum 3
- Pour réduire à un argument, pensez à transformer un ensemble d'arguments en classe avec ses attribues pour un langage OO, ou une struct (C, Rust), variable groupée (CoBOL)

Bonnes pratiques, les fonctions pour tous

Les IDE modernes ont le plus souvent une fonction pour créer automatiquement une fonction (e.g., intelliJ)



Bonnes pratiques, les fonctions pour tous

- Pas de modification de l'argument passé par référence
- La fonction a une action sur le(s) argument(s) et donne une valeur de sortie à retourner
- A droite, l'argument passé `outer_list` est modifié mais cela se comprend difficilement
- En plus, cela dépend du type de l'argument passé en Java comme en Python, voir ci-dessous

```
In [26]: def try_to_change_list_contents(the_list):
    ...:     print('got', the_list)
    ...:     the_list.append('four')
    ...:     print('changed to', the_list)
    ...:

In [27]: outer_list = ['one', 'two', 'three']
    ...:
    ...: print('before, outer_list =', outer_list)
    ...: try_to_change_list_contents(outer_list)
    ...: print('after, outer_list =', outer_list)
before, outer_list = ['one', 'two', 'three']
got ['one', 'two', 'three']
changed to ['one', 'two', 'three', 'four']
after, outer_list = ['one', 'two', 'three', 'four']

In [28]: outer_list
Out[28]: ['one', 'two', 'three', 'four']
```

```
In [29]: def add_one(x):
    ...:     x += 1
    ...:

In [30]: an_integer = 1

In [31]: add_one(an_integer)

In [32]: an_integer
Out[32]: 1
```

Bonnes pratiques, les fonctions pour tous

- Command Query Separation, A.K.A CQRS (Command and Query Responsibility Segregation), une fonction à action sur une base de donnée, API, c'est une Command. Quand on requête une donnée en base ou dans une API c'est une Query
- Cela permet également d'avoir une fonction avec moins de responsabilités

Bonnes pratiques, les commentaires pour tous

- Sont un complément aux commentaires structurés de l'autodocumentation comme Sphinx ou Javadoc (cf. la suite du cours)
- Sur la même ligne si cela ne fait pas une ligne trop longue
- Expliquer ses intentions
- Commentaires spéciaux comme les TODO (reste à faire) et NoQA (à tester) sont très utiles
- Ne pas commenter du code, le VCS est là pour créer l'historique du code
- Ne jamais partir du principe que les autres vont lire le commentaire et se contenter d'expliquer les problèmes à éviter dans les commentaires. C'est trop dangereux, il faut corriger l'algorithme

```
# En cas de plantage ne pas relancer ce script, en cas de plantage dans la fonct
# ion update_the_balance, cela envoie l'argent 2 fois dans la fonction
# send_the_money
import .send_the_money, .update_the_balance
send_the_money()
update_the_balance()
```

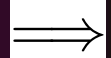
Bonnes pratiques, les classes pour l'OOP, héritage ou composition

```
In [9]: class Corde:
   ...:     diametre_mm: int
   ...:     longueur_m: int
   ...:     def __init__(self, diametre_mm, longueur_m):
   ...:         self.diametre_mm = diametre_mm
   ...:         self.longueur_m = longueur_m
   ...:

In [10]: class CordeDynamique(Corde): # Je suis une corde, j'hérite de Corde
    ...:     longueur_max_m: int # J'ai une longueur maximum étirée, c'est un attribut
    ...:     def __init__(self, diametre_mm, longueur_m):
    ...:         Corde.__init__(self, diametre_mm, longueur_m)
    ...:         self.longueur_max_m = self.longueur_m + self.longueur_m * 0.1
    ...:

In [11]: ma_corde_dynamique = CordeDynamique(9, 60)

In [12]: ma_corde_dynamique.longueur_m
Out[12]: 60

In [13]: ma_corde_dynamique.longueur_max_m
Out[13]: 66.0
```

Bonnes pratiques, les classes pour l'OOP

L'héritage évite les if/else et switch/case

```
In [28]: class Bird:
    ...:     # ...
    ...:     def getSpeed(self):
    ...:         if self.type == EUROPEAN:
    ...:             return self.getBaseSpeed()
    ...:         elif self.type == AFRICAN:
    ...:             return self.getBaseSpeed() * 2
    ...:         elif self.type == NORWEGIAN_BLUE:
    ...:             return self.getBaseSpeed() * 0.1
    ...:         else:
    ...:             raise Exception("Should be unreachable")
```



```
In [29]: class Bird:
    ...:     # ...
    ...:     def getBaseSpeed(self):
    ...:         pass
    ...:
    ...: class European(Bird):
    ...:     def getSpeed(self):
    ...:         return self.getBaseSpeed()
    ...:
    ...:
    ...: class African(Bird):
    ...:     def getSpeed(self):
    ...:         return self.getBaseSpeed() * 2
    ...:
    ...:
    ...: class NorwegianBlue(Bird):
    ...:     def getSpeed(self):
    ...:         return self.getBaseSpeed() * 0.1
    ...:
```

Autres livres sur les bonnes pratiques

- Steven C. McConnell, Code Complete Second Edition

Génération de la documentation, les frameworks d'autodocumentation

- Pourquoi ne pas documenter un programme dans un simple fichier Word ?
- L'autodocumentation ajoute à un document de base (e.g., le README) la description du code présente dans des commentaires structurés du fichier source
- Des frameworks propres à un langage, d'autres pour tous les langages et chacun ou presque a son modèle de commentaires structurés
- Ceux pour tous les langages ont l'avantage de ne s'apprendre qu'une fois sinon on en apprend un pour chaque langage
- Celui spécifique à un langage aura plus de chance d'être connu des développeurs de la communauté
- Les officiels, à préférer et les autres issues de plus petites communautés

Génération de la documentation, Java et Javadoc

- Autodocumentation officielle des sources Java
- Chaque déclaration de classe ou fonction Java a un commentaire avec des tags du langage de balisage Javadoc
- Spécification Javadoc https://www.oracle.com/fr/technical-resources/articles/java/javadoc-tool.html

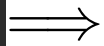
Order of Tags

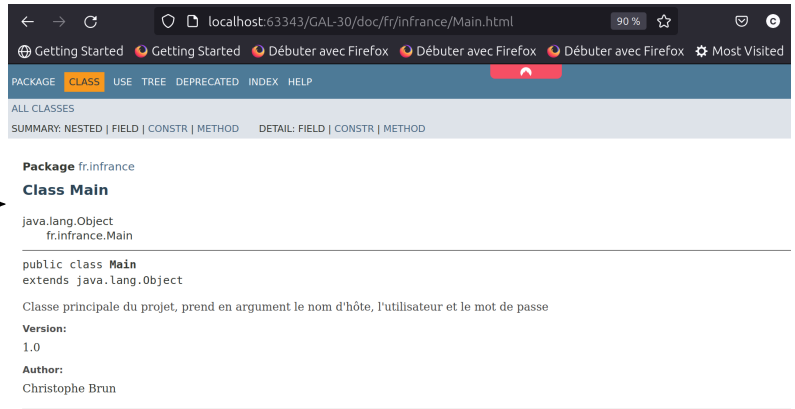
Include tags in the following order:

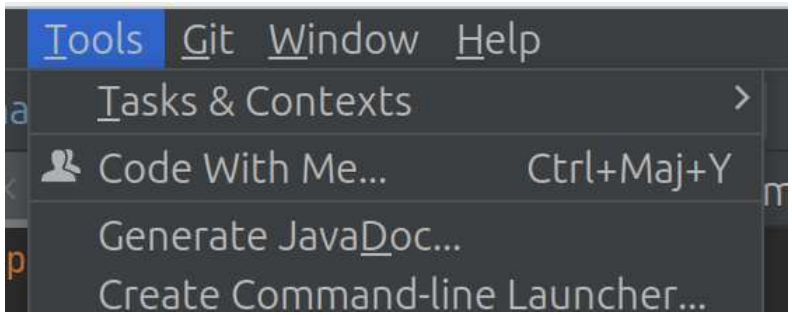
- `@author` (classes and interfaces only, required)
- `@version` (classes and interfaces only, required. See footnote 1)
- `@param` (methods and constructors only)
- `@return` (methods only)
- `@exception` (@throws is a synonym added in Javadoc 1.2)
- `@see`
- `@since`
- `@serial` (or @serialField or @serialData)
- `@deprecated` (see How and When To Deprecate APIs)

Génération de la documentation, Javadoc avec un l'IDE, IntelliJ

```java
/**
 * Classe principale du projet, prend en argument le nom d'hôte, l'utilisateur
 * et le mot de passe
 *
 * @author Christophe Brun
 * @version 1.0
 */
 chrichri *
public class Main {
```





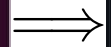


Génération de la documentation, Sphinx documentation avec PyCharm

- Langage de balisage est ReStructured Text, similaire au markdown mais différent (dommage...). Formatage très complet, permet de faire des UI riches. Voir https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html pour la définition
- Le README peut-être ajouté avant les commentaires du code. Il doit être au format RST (README.rst), Markdown (README.md), word, etc
- L'avantage des README Markdown et RST est qu'ils sont mis en forme par les serveurs comme GitHub, Gitlab...
- Toutes les documentations sont dans les docstrings, une structure propre à Python. Les docstrings sont dans l'attribue `__doc__` qui peut être appelé dans le shell Python avec la fonction `help`
- Documente les déclarations `def` et `class`
- Documente les modules avant le code

Génération de la documentation, Sphinx documentation avec PyCharm

```
In [2]: def is_odd(x):
   ...:     """
   ...:     Divide the parameter x by 2. If the modulo is
   ...:     different from 0, it returns True, otherwise False.
   ...:
   ...:        :param x: The number to test the parity
   ...:        :return: A boolean, True if off, else False
   ...:     """
   ...:     return x % 2 != 0
   ...:

In [3]: is_odd.__doc__
Out[3]: '\n    Divide the parameter x by 2. If the modulo is\n    different from
 0, it returns True, otherwise False.\n    \n       :param x: The number to test
 the parity\n       :return: A boolean, True if off, else False\n    '

In [4]: help(is_odd)
```



README.rst

L'INPI a communiqué cela le 03/04/2023:

Madame, Monsieur,

En tant que licencié aux données du RNE nous vous informons de la fermeture des anciennes applications de diffusion des données du RNCS le 30 juin 2023.

Les API et le serveur FTP RNCS ne seront plus accessibles, les données étant désormais disponibles sur les nouvelles applications API et SFTP du RNE au format JSON et PDF.

Nous vous rappelons par ailleurs que les quotas de téléchargement sont de 10 Go ou 10 000 appels par jour.

Vous pouvez retrouver l'ensemble des documentations techniques dans l'onglet « espace open data » du portail DATA INPI.

Qui est partant pour faire une version de Enthic avec le FTP du RNE?

Projet en pause

Ce projet est en pause faute de temps et d'énergie. Je me concentre maintenant sur un projet similaire, IN FRANCE, plus complet qui utilise ces mêmes données pour étudier l'économie Française. Merci à tous ceux qui ont collaboré. N'hésitez pas à me christophe.brun@papit.fr pour toute question.

Project on hold

This projet is on hold due to a lack of time and energy. I am focusing on IN FRANCE project, a similar one, more complete, studying the French economy. Thank you all contributor. Don't hesitate to contact me at christophe.brun@papit.fr.

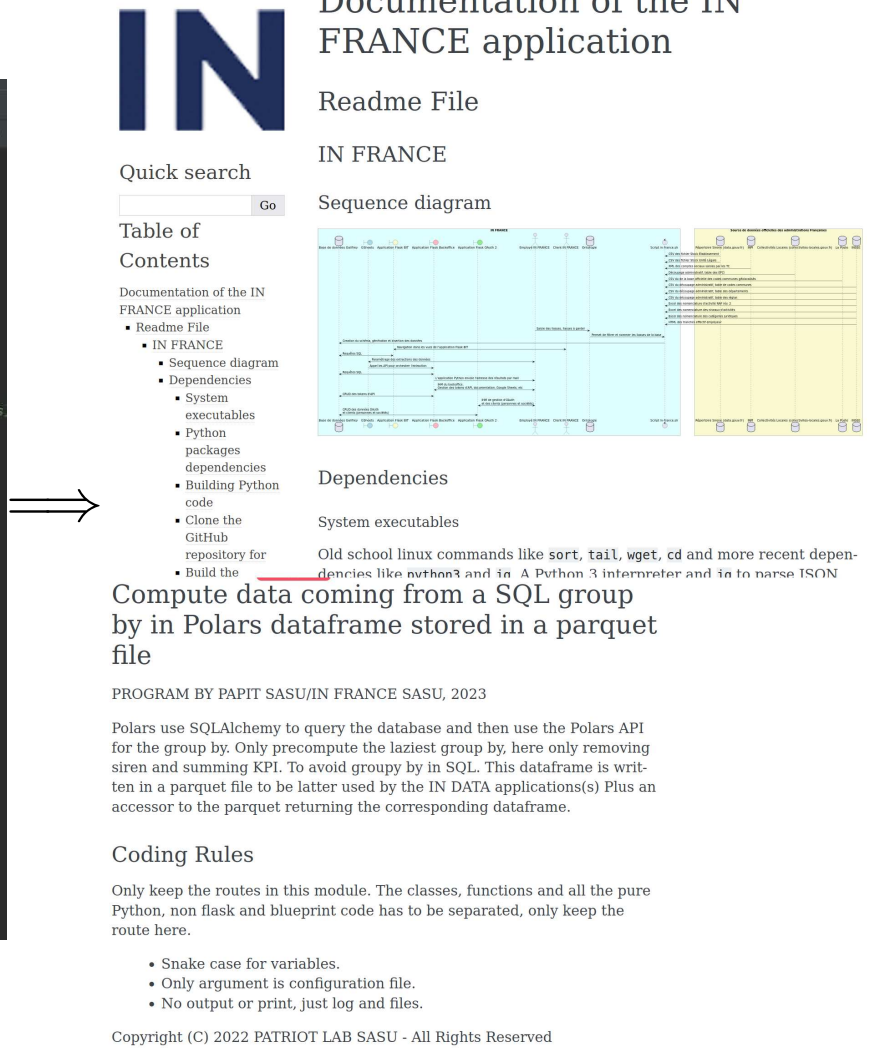
French societies accountability extraction and treatment by PapIT

Project that treats data from opendata-rncs.inpi.fr. They contain xml files of the account declaration of all french societies. The overall project is meant to be low-code and open source. Aim to provide ethical indicators on companies. Information media is a MySQL database, CSV files, web visualisation and a swagger API. The search

Génération de la documentation, Python et Sphinx documentation

README plus commentaires en RST des modules et des déclarations.

```
ta_management.py × In-france.sh × extract_bundle.py × configuration.json × bit_forms.py ×
# -*- coding: utf-8 -*-
"""
================================================================================
Compute data coming from a SQL group by in Polars dataframe stored in a parquet file
================================================================================

PROGRAM BY PAPIT SASU/IN FRANCE SASU, 2023

Polars use SQLAlchemy to query the database and then use the Polars API for the group
by. Only precompute the laziest group by, here only removing siren and summing
KPI. To avoid groupy by in SQL.
This dataframe is written in a parquet file to be latter used by the IN DATA applications(s)
Plus an accessor to the parquet returning the corresponding dataframe.

Coding Rules
============

Only keep the routes in
this module. The classes, functions and all the pure Python, non flask and
blueprint code has to be separated, only keep the route here.

   - Snake case for variables.
   - Only argument is configuration file.
   - No output or print, just log and files.


Copyright (C) 2022 PATRIOT LAB SASU - All Rights Reserved
"""
import ...

# Cannot import dbname from data_management because of the scheme unknown to
# mysql+pymysql
DBNAME = "mysql://%s:%s@%s/%s" % (
    config["MySQL"]["user"],
    config["MySQL"]["password"],
```



Documentation of the IN FRANCE application

Readme File

IN FRANCE

Quick search

Go

Table of Contents

Documentation of the IN FRANCE application
- Readme File
  - IN FRANCE
    - Sequence diagram
    - Dependencies
      - System executables
      - Python packages dependencies
      - Building Python code
      - Clone the GitHub repository for
      - Build the

Sequence diagram

Dependencies

System executables

Old school linux commands like `sort`, `tail`, `wget`, `cd` and more recent dependencies like `python3` and `jq` A Python 3 interpreter and `jq` to parse JSON

Compute data coming from a SQL group by in Polars dataframe stored in a parquet file

PROGRAM BY PAPIT SASU/IN FRANCE SASU, 2023

Polars use SQLAlchemy to query the database and then use the Polars API for the group by. Only precompute the laziest group by, here only removing siren and summing KPI. To avoid groupy by in SQL. This dataframe is written in a parquet file to be latter used by the IN DATA applications(s) Plus an accessor to the parquet returning the corresponding dataframe.

Coding Rules

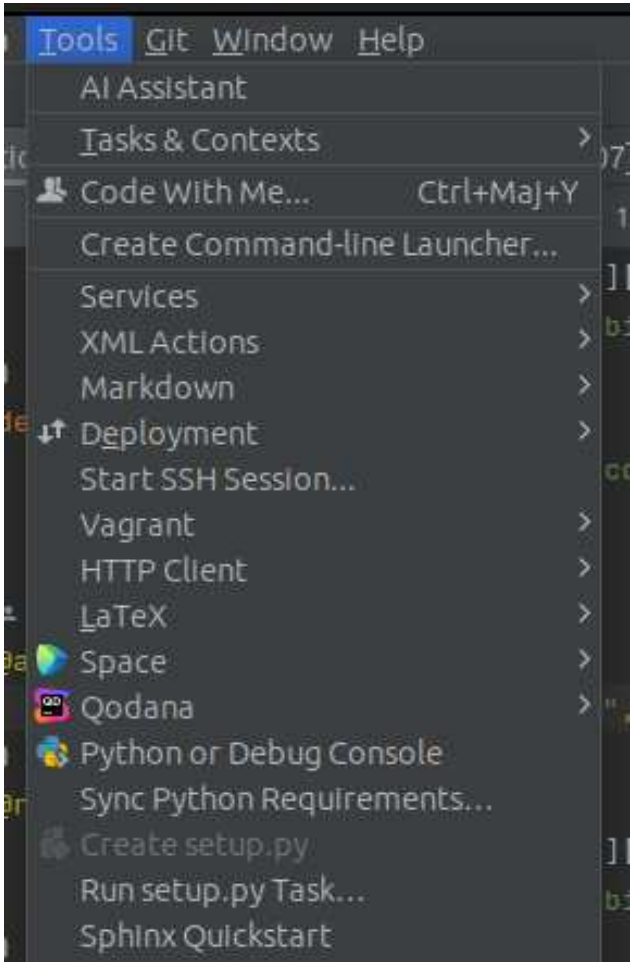
Only keep the routes in this module. The classes, functions and all the pure Python, non flask and blueprint code has to be separated, only keep the route here.

- Snake case for variables.
- Only argument is configuration file.
- No output or print, just log and files.

Copyright (C) 2022 PATRIOT LAB SASU - All Rights Reserved

Génération de la documentation, Python et Sphinx documentation

Fonction Sphinx Doc intégrée dans PyCharm pour la génération de la documentation.



Génération de la documentation, Doxygen pour tous les sources

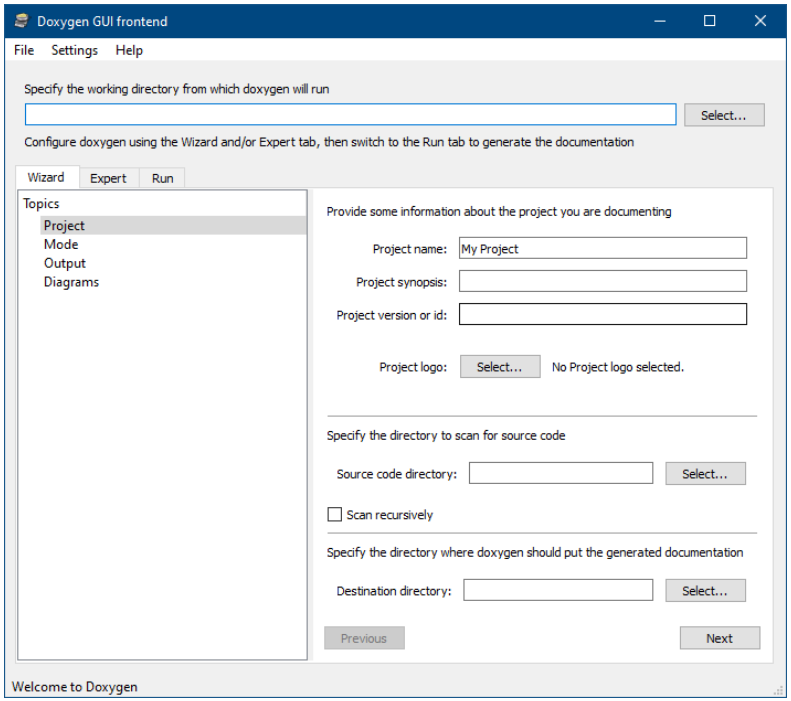
- “Generate documentation from source code”, c’est aussi générique que cela! 😇
- La liste des langages supportés est très longue, “C++, C, Objective-C, C#, PHP, Java, Python, IDL, Fortran, and to some extent D”
- Comme JavaDoc, des tags sont rajoutés dans les commentaires
- Peut fonctionner avec les tags JavaDoc si configuré pour

```
/**********************************************************************
 * A brief history of JavaDoc-style (C-style) banner comments.
 *
 * This is the typical JavaDoc-style C-style "banner" comment. It starts with
 * a forward slash followed by some number, n, of asterisks, where n > 2. It's
 * written this way to be more "visible" to developers who are reading the
 * source code.
 *
 * Often, developers are unaware that this is not (by default) a valid Doxygen
 * comment block!
 *
 * However, as long as JAVADOC_BLOCK = YES is added to the Doxyfile, it will
 * work as expected.
 *
 * This style of commenting behaves well with clang-format.
 *
 * @param theory Even if there is only one possible unified theory. it is just a
 *               set of rules and equations.
 **********************************************************************/
void javadocBanner( int theory );
```

Génération de la documentation, Doxygen pour tous les sources

- Configurable comme tous les frameworks d'autodocumentation :
  - Template
  - Type de document généré
  - Logo
  - ...
- En ligne de commande avec la commande doxygen

  `doxygen -g <config-file>`
- Dans un GUI, le Doxygen wizard
- Cf. https://www.doxygen.nl/



Organisation du code d'un projet

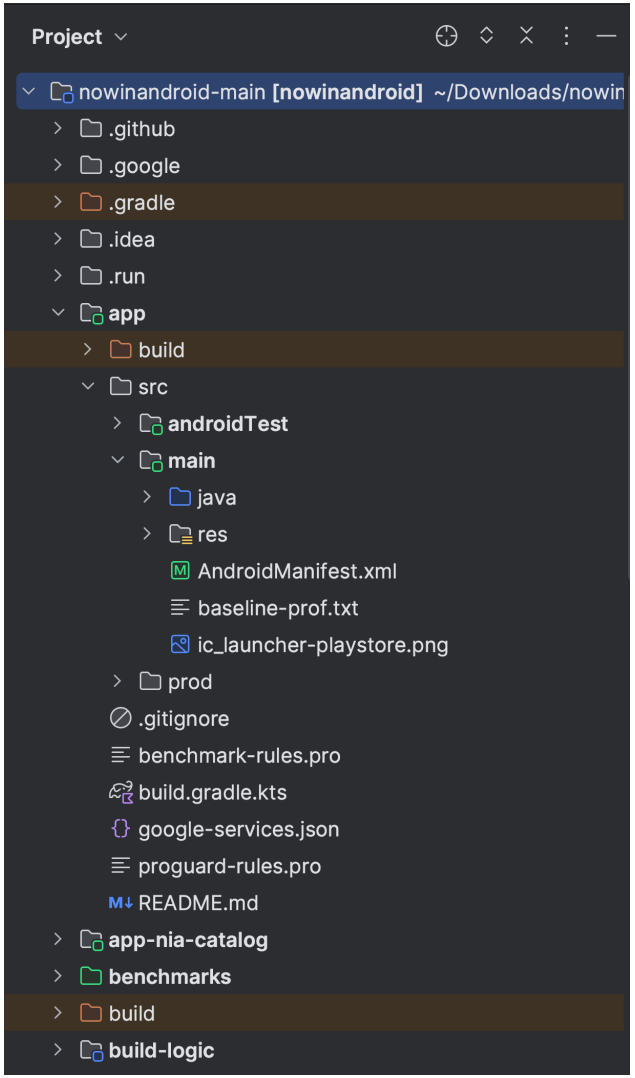
- Dépend du langage
- Dépend parfois du framework en plus
- Séparer les différents types de codes sources :
  - Code source de l'application
  - Tests du code source
  - Documentation
  - CI/CD
- Séparer les sources des dépendances externes
- Séparer les sources des fichiers générés (par la compilation, etc)
- Les fichiers temporaires et générés sont souvent dans un dossier caché (commençant par un point) pour être plus discret
- Pas toujours maîtrisable avec les dépendances :
  - Node JS, dans le `node_modules` du projet ou ailleurs en fonction de l'installation
  - Dans le répertoire de l'interpréteur pour Python
  - Dans un n'importe quel répertoire quand il n'y a pas de package manager comme en C/C++

Organisation du code d'un projet, e.g., Java « classique » et Gradle

- Tout est séparé
  - Les ressources statiques
  - Les sources
  - Les tests
  - Le code compilé en classes
  - Le code compilé en exécutable JAR
  - La configuration de Gradle (Gère les dépendance, les builds, etc)
  - La documentation
- Ce n'est qu'un exemple, une possibilité

Organisation du code d'un projet, e.g., Java Android et Gradle

- Tout est toujours séparé
- Différent du projet Java classique
- Des fichiers de configuration propres au développement Android
- Des fichiers de configuration propres au déploiement sur Playstore
- Ça dépend du projet, du framework, du déploiement, etc



Organisation du code d'un projet, Python, Solidity...

- Framework style MVC avec tout dedans
- Exemple d'application
  https://docs.djangoproject.com/en/4.2/intro/tutorial03
- Le nom des dossiers et fichiers réfèrent explicitement à la fonction du source

```
.
├── manage.py
├── mon_app
│   ├── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
└── project
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

- Solidity est le langage de la Machine Virtual Ethereum (A.K.A EVM)
- L'environnement de développement de base RemixIDE ne support pas les imports
- Tout est donc dans un seul source, à comparer une fois compilé au bytecode présent dans la blockchain (voir

Organisation du code d'un projet, Conclusion

- L'organisation diffère entre les technos et même dans un même langage
- Différence entre les technos compilées et interprétées
- La bonne pratique, commune dans toutes les technos est le naming des dossiers et fichiers qui explicite la fonction du source
- Explorer les documentations et ressources officielles de chaque techno pour la bonne pratique en vigueur (RTFM encore une fois)

Travaux dirigés, les sujets possibles

- Mettre en pratique tous les concepts vus dans ce cours
- Par groupe de 3 maximum
- Publier les résultats dans un repository sur une plateforme comme Github, Gitlab
- Envoyer l'adresse du repository à `christophe.brun@papit.fr`
- Mettre au propre votre projet perso
- Créer un soft inexistant simple. Par exemple comme ajouter des SRI inplace :
  - En Java
  - En C++
- Un projet open source déjà existant ne mettant pas en application les bonnes pratiques de ce cours. Raymond Hettinger a commencé par compléter des commentaires et la documentation de Python et conseille de faire de même pour être onboarder sur un projet OS
- Bien d'autres possibilités encore !

L'histoire du testing

- ISTQB, l'International Software Testing Qualifications Board a été fondée en 1998[13]
- Dans son livre Extreme Programming Explained, Kent Beck parle de Test-first Programming en 1999
- Les standards modernes datent de la fin des années 90, début des années 2000
- JUnit, le framework de test le plus utilisé en Java date de 2002[14]
- Début du BDD (Behavior-Driven Development) avec JBehave, censé remplacer JUnit en utilisant le behavior au lieu de test, date de 2003[15]

[13] ISTQB, About Us, https://www.istqb.org/about-us/who-we-are

[14] Steven J Zeil, Unit Testing Frameworks, https://www.cs.odu.edu/~zeil/cs350/latest/Public/junit/index.html

[15] Cucumber, https://cucumber.io/docs/bdd/history/

Qu'est-ce qu'un test ?

Définition de l'International Software Testing Qualifications Board et IBM

L'ISTQB défini les termes suivants dans son glossaire[16] :
Test : Un ensemble d'un ou plusieurs cas de tests.
Cas de test : Un ensemble de conditions préalables, de données d'entrée, d'actions (le cas échéant), de résultats attendus et de postconditions, élaboré sur la base des conditions de test.

Selon IBM[17] :
"Le test logiciel est le processus qui consiste à évaluer et à vérifier qu'un produit ou une application logicielle fait ce qu'il ou elle est censé(e) faire."

---

[16] ISTQB, Glossaire des termes utilisés en tests de logiciels, https://www.cftl.fr/wp-content/uploads/2018/10/Glossaire-des-tests-logiciels-v3_2F-ISTQB-CFTL-1.pdf

[17] IBM, Qu'est-ce que le test logiciel ?, https://www.ibm.com/fr-fr/topics/software-testing

Qu'est-ce qu'un test unitaire ?

Définition de la taverne du testeur[18]

Il obéit au principe "F.I.R.S.T." :

- **Fast :** S'exécute rapidement et est donc automatisé
- **Isolated :** Est indépendant des facteurs externes et des autres tests
- **Repeatable :** Isole les bugs automatiquement
- **Self-validating (autonome) :** N'est pas ambiguë (pas sujet à interprétation, ne demande pas une action manuelle pour vérifier le résultat)
- **Timely (tôt) :** Écrit en même temps que le code (même avant en TDD)

Si le test échappe à un de ces principes, il n'est probablement pas unitaire.

Pour isoler les bugs automatiquement, un test unitaire doit tester le code unitairement, et isolement. Il doit donc être écrit dans le but de tester la logique d'une ligne de code, et quasiment si un test unitaire échoue vous pouvez connaitre la ligne de code incriminée.

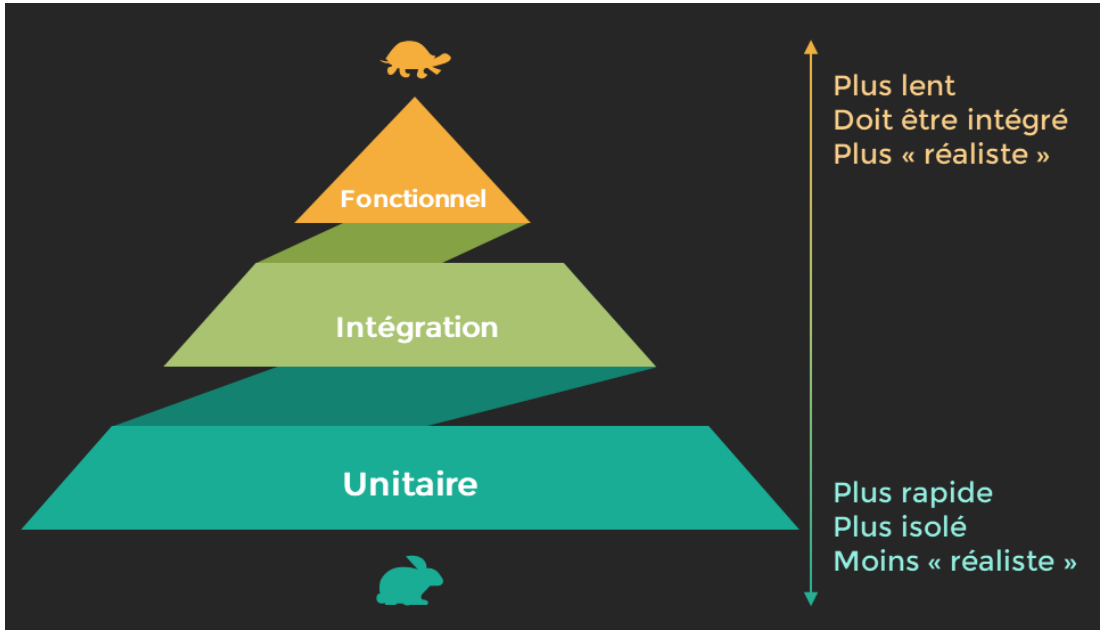
[18] Mais c'est quoi un test unitaire ?, https://lataverneduteste ur.fr/2018/04/11/mais-cest-quoi-un-test-unitaire/

Testing, les frameworks de test unitaire

- UnitTest, le framework de test standard de Python
- PyTest, un framework pour tout faire en Python
- JUnit, le framework de test le plus utilisé en Java
- PHPUnit, le framework de test standard de PHP
- Google Test, pour le C++
- Jest pour JS, Babel, pour TypeScript, Node, React, Angular, Vue et plus encore
- etc

La Pyramide des tests

Différents types de tests peuvent être identifiés. La classification la plus classique étant la suivante[19] :



Mais d'autres valent le coup d'être découvertes...

[19] OPENCLASSROOMS, Testez votre code Java pour réaliser des applications de qualité, https://openclassrooms.com/fr/courses/6100311-testez-votre-code-java-pour-realiser-des-applications-de-qualite/6616481-decouvrez-les-tests-dintegration-et-les-tests-fonctionnels

Testing

- Un programme a pour entrée le standard input, et sorties les standard output et standard error, ce sont les standard streams
- Un programme se termine avec un code retour, le plus souvent un entier différent de 0 en cas de problème
- Les frameworks de tests retournent 3 statuts, OK, FAILED et ERROR
- Les OK si aucune erreur n'a lieu et FAILED en cas d'erreur d'assertion

```
chrichri@chrichri-HKD-WXX:~/Documents/Campus St Michel IT$ cat test_stdout_stderr.py
import sys

def test_hello():
    print("hello testing")
    print("stderr during testing", file=sys.stderr)
    assert False
chrichri@chrichri-HKD-WXX:~/Documents/Campus St Michel IT$ python3 -m pytest test_stdout_stderr.py  1> /dev/null
chrichri@chrichri-HKD-WXX:~/Documents/Campus St Michel IT$ python3 -m pytest test_stdout_stderr.py  2> /dev/null
============================================= test session starts ==============================================
platform linux -- Python 3.11.4, pytest-7.2.1, pluggy-1.0.0+repack
rootdir: /home/chrichri/Documents/Campus St Michel IT
collected 1 item

test_stdout_stderr.py F                                                                                  [100%]

=================================================== FAILURES ===================================================
_____________________________________________________ test_hello _____________________________________________________

    def test_hello():
        print("hello testing")
        print("stderr during testing", file=sys.stderr)
>       assert False
E       assert False

test_stdout_stderr.py:6: AssertionError
----------------------------------------------- Captured stdout call -----------------------------------------------
hello testing
----------------------------------------------- Captured stderr call -----------------------------------------------
stderr during testing
=========================================== short test summary info ============================================
FAILED test_stdout_stderr.py::test_hello - assert False
============================================== 1 failed in 0.10s ===============================================
chrichri@chrichri-HKD-WXX:~/Documents/Campus St Michel IT$ echo $?
1
```

Brief sur les outils de test logiciel

- Les frameworks de test retournent 3 statuts, OK, FAILED et ERROR .
- OK si aucune erreur n'a lieu et FAILED en cas d'erreur d'assertion, `AssertionError` en Python et Java.
- Pas de standard universellement accepté mais un format domine, le XML JUnit. Un XML le définit par la grammaire https://windyroad.com.au/dl/Open%20Source/JUnit.xsd.
- Vient de l'écosystème Java avec le package du même nom, JUnit, mais est présent dans la plupart des frameworks de test comme PHPUnit, Pytest.
- L'interopérabilité entre les outils est permise par cette XSD commune, mais pas plus de contrainte. Par exemple, l'interopérabilité entre les frameworks de test et les outils de CI/CD .
- Certains outils, souvent des solutions propriétaires, rechignent toujours à exporter un JUnit contenant les résultats de test pour fermer leur environnement.

L'assertion error

En Python

L'instruction `assert` vérifie une condition. Si la condition est vraie, cela ne fait rien et votre programme continue simplement à s'exécuter. Mais si la condition d'assertion est fausse, elle lève une exception `AssertionError`.

```
assert 23 % 2 == 0, "Le restant de la division est différent de 0"
```

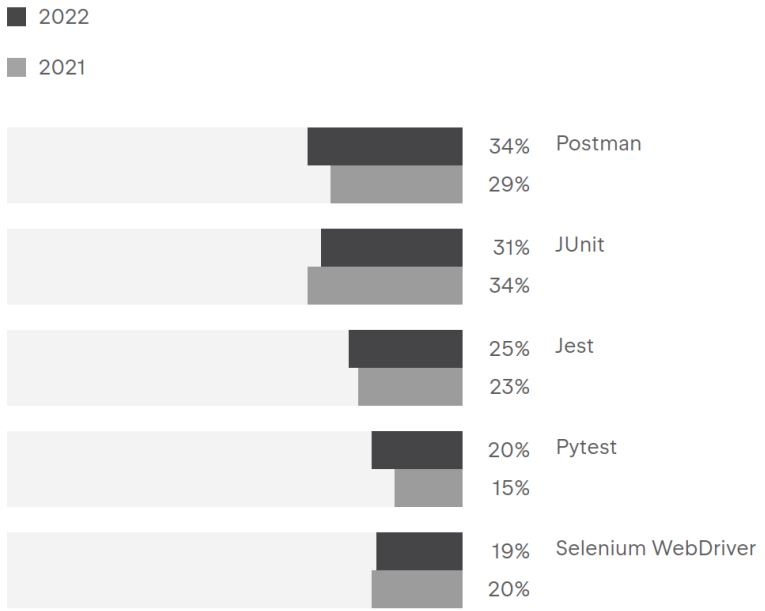
Étant toujours fausse, le programme crash.

```
chrichri@chrichri-HKD-WXX:~$ python3 -c 'assert 23 % 2 == 0, "Le restant de la
    division est différent de 0"'
Traceback (most recent call last):
  File "<string>", line 1, in <module>
AssertionError: Le restant de la division est différent de 0
```

Pourquoi Pytest pour le testing

"pytest is a mature full-featured Python testing tool that helps you write better programs.[20]"

Pytest est le framework de test en Python le plus utilisé selon les sondages de JetBrains[21]. Seuls JUnit et Pytest sont pour tous usages, les autres sont orientés web.



[20] pytest: helps you write better programs , https://docs.pytest.org

[21] JetBrains, Which test frameworks , https://www.jetbrains.com/lp/devecosystem-2022/testing/

- Un module de test Pytest est un module Python préfixé par `test_`.
- Toutes les méthodes préfixées par `test_` sont exécutées par Pytest. Qu'elles soient dans une classe ou non. Les autres méthodes ne sont exécutées uniquement si les tests les appellent.

Ici par exemple, la précédente assertion est intégrée dans une méthode nommée `test_divide` dans le module de test `test_division.py`.

```
def test_divide(): # Un test Pytest est préfixé par test_
    assert 23 % 2 == 0, "Le restant de la division est différent de 0."
```

Lancer avec la commande `python -m pytest test_divison.py`, Pytest affiche le rapport de test.

```
...
def test_divide(): # Un test Pytest est préfixé par test_
> assert 23 % 2 == 0, "Le restant de la division est différent de 0."
E AssertionError: Le restant de la division est différent de 0.
E assert (23 % 2) == 0

test_divison.py:2: AssertionError
...
```

De très utiles et nombreuses options peuvent compléter cette ligne de commande. Pour les découvrir, lancez `python -m pytest --help` et “RTFM”.

Les tests paramétriques sont des tests qui prennent un ensemble de paramètres en entrées.
Ils permettent de tester plusieurs cas avec un seul test.
Le rapport génère un cas de test par paramètre. Il est donc plus détaillé et permet de trouver les tests en échec plus facilement que si l'`assert` était dans une boucle, ce qui ne donne qu'un statut OK ou Fail dans le rapport.

```
import pytest
...
@pytest.mark.parametrize("dividend", range(100)) # Paramétrage du test
def test_divide_from_0_to_99(dividend): # Doit avoir un argument présent dans le
      paramétrage
   assert dividend % 2 == 0, "Le restant de la division est différent de 0."
```

Génère 100 cas de tests. Met en évidence les dividendes dont le restant de la division par 2 est différent de 0, ici 1 par exemple.

```
collecting ... collected 100 items

test_divison.py::test_divide_from_0_to_99[0] PASSED [ 1%]
test_divison.py::test_divide_from_0_to_99[1] FAILED [ 2%]
test_divison.py:7 (test_divide_from_0_to_99[1])
1~!= 0

Expected~:0
Actual ~:1
<Click to see difference>

dividend = 1

    @pytest.mark.parametrize("dividend", range(100)) # Paramétrage du test
    def test_divide_from_0_to_99(dividend): # Doit avoir un argument présent dans
          le paramétrage
> assert dividend % 2 == 0, "Le restant de la division est différent de 0."
E AssertionError: Le restant de la division est différent de 0.
E assert (1 % 2) == 0
```

Pytest

Plantage dans un test, quel statut ?

Ce test crash avant l'assert.

```
def test_fail_or_error(): # Une erreur donne un fail ou error?
    dividende = 23 / 0
    assert dividende % 2 == 0, \
        "Le restant de la division est différent de 0."
```

Malgré l'absence d'`assertionError`, le test est en échec et non en erreur.

```
test_divison.py::test_fail_or_error FAILED [100%]
test_divison.py:12 (test_fail_or_error)
def test_fail_or_error(): # Une erreur donne un fail ou error?
> dividende = 23 / 0
E ZeroDivisionError: division by zero

test_divison.py:14: ZeroDivisionError
```

Pour éviter les faux négatifs, un test doit tendre, tant que faire se peut, vers un `assert`. Pour cela pensez à utiliser les fonctions `parametrize` et `fixture`.

Cf. https://github.com/St-Michel-IT/testing/blob/main/test_customer_database.py

Un `return` ou un `yield` envoie l'objet au test dans l'état voulu.

```
@pytest.fixture
def customer_without_table():
    """..."""
    customer = Customer() # Avant le test, c'est le setup
    yield customer # Un yield évite de sortir de la fonction
    customer.con.close() # Après le test, le teardown
```

Le test n'a qu'une seule ligne, qu'un `assert` l'échec ne pourrait venir que de là.

```
def test_instantiation(customer_without_table):
    """..."""
    assert isinstance(customer_without_table.con, Connection)
```

En cas de plantage dans la fixture, i.e., avant ou après le test, la sanction serait erreur et non échec.
Le rapport reflétera fidèlement le test et sera exempt de faux négatif.

Pytest

Les fixtures, le moyen du pattern A.A.A[22]

Le pattern du A.A.A, Arrange, Act, Assert, est facile à implémenter avec les fixtures. Arrange et Act sont isolé dans la fixture, Assert seul est dans le test.

```
def customer_without_table():
    """...
    """
    customer = Customer() # Arrange et Act dans le setup avant yiel ou return
    yield customer
    customer.con.close() # Pas de nom en A, ils l'ont oublié?
```

Le test tend vers un `assert` quasi seul.

```
def test_instantiation(customer_without_table):
    """...
    """
    assert isinstance(customer_without_table.con, Connection) # Assert
```

Le B.D.D. suit le pattern A.A.A sous un autre nom : Given-When-Then. Le langage Gherkin utilise les étapes Given-When-Then pour spécifier les comportements dans les scénarios.

[22] Arrange-Act-Assert: A Pattern for Writing Good Tests, https://automationpanda.com/2020/07/07/arrange-act-assert-a-pattern-for-writing-good-tests/

Pytest

Le scope des fixtures

L'argument `scope` du décorateur `pytest.fixture` défini la durée de vie de la fixture. Il peut prendre 3 valeurs :

- `function` : La valeur par défaut si on ne met rien. La fixture est exécutée à chaque fonction de test `def test_...` qui l'appelle directement ou indirectement, donc les éventuels setup et teardown de cette dernière aussi.
- `module` : Une fois par module `test_....py`.
- `session` : Une seule fois durant la session de test quel que soit le nombre de modules et de tests exécutés.

```
@pytest.fixture(scope="function") # Scope de la fixture, par default function
def customer_without_table():
    """
    Connection to in memory database using the Customer class
```

Pour tester une API en étant sûr le token n'est pas périmé, on peut utiliser le scope `function` pour en avoir un nouveau à chaque appel.
Si préparer les conditions initiales, le setup, prend du temps, on évitera si possible de le répéter à chaque test avec les scopes `module`, `session`.

Pytest

Le `conftest.py`

Il n'a qu'une seule particularité, c'est d'être importé automatiquement par Pytest lorsqu'il est présent dans le dossier courant des tests.
Il permet de factoriser les fixtures et les paramétrages de tests utilisés dans plusieurs modules de tests, car ces derniers seront automatiquement disponibles sans même un import dans le module.

```
./
├── conftest.py
├── test_no_directory.py
├── unit
│   └── test_integration.py
└── functional
    └── test_functional.py
```

Pour tous ces tests, s'ils sont lancés depuis la racine avec la commande `pytest`, le `conftest.py` sera importé automatiquement.

PyCharm intègre par défaut Pytest comme lanceur de tests et fournit l'autocomplétion pour les fixtures du `conftest.py`.

Pytest

Le coverage

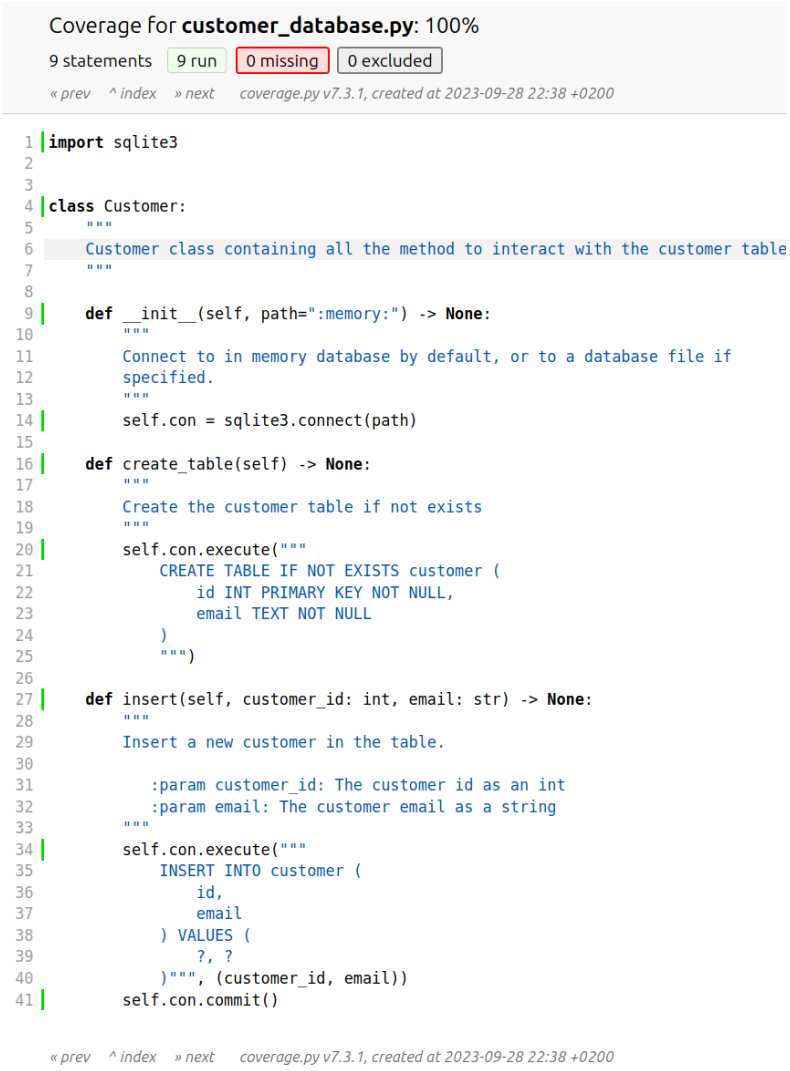
Pas de support natif du coverage par Pytest, il faut installer le plugin `pytest-cov`[23].
Pour appeler le plugin, il faut passer en argument le module à tester et le(s) test(s) de ce dernier.

```
pytest --cov=\textcolor{flatgreen}{customer_database} ./test_customer_database.
    py
...
rootdir: /home/chrichri/Documents/Campus-St-Michel-IT/testing
plugins: cov-4.1.0
collected 6 items

test_customer_database.py ...... [100%]

---------- coverage: platform linux, python 3.11.4-final-0 -----------
Name Stmts Miss Cover
----------------------------------------
customer_database.py 9 0 100%
----------------------------------------
TOTAL 9 0 100%
```

[23] Welcome to pytest-cov's documentation!,

Pytest

Le coverage



Coverage for **customer_database.py**: 100%

9 statements | 9 run | 0 missing | 0 excluded

« prev ^ index » next coverage.py v7.3.1, created at 2023-09-28 22:38 +0200

```
1  import sqlite3
2
3
4  class Customer:
5      """
6      Customer class containing all the method to interact with the customer table
7      """
8
9      def __init__(self, path=":memory:") -> None:
10         """
11         Connect to in memory database by default, or to a database file if
12         specified.
13         """
14         self.con = sqlite3.connect(path)
15
16     def create_table(self) -> None:
17         """
18         Create the customer table if not exists
19         """
20         self.con.execute("""
21             CREATE TABLE IF NOT EXISTS customer (
22                 id INT PRIMARY KEY NOT NULL,
23                 email TEXT NOT NULL
24             )
25             """)
26
27     def insert(self, customer_id: int, email: str) -> None:
28         """
29         Insert a new customer in the table.
30
31            :param customer_id: The customer id as an int
32            :param email: The customer email as a string
33         """
34         self.con.execute("""
35             INSERT INTO customer (
36                 id,
37                 email
38             ) VALUES (
39                 ?, ?
40             )""", (customer_id, email))
41         self.con.commit()
```

« prev ^ index » next coverage.py v7.3.1, created at 2023-09-28 22:38 +0200

L'option `--cov-report=html` permet de générer un rapport HTML plus détaillé qui met en lumière quelles parties du code sont couvertes par les tests et lesquelles ne le sont pas.

Exercice : Trouvez quel test du module https://github.com/St-Michel-IT/testing/blob/main/test_customer_database.py couvre quelle partie de ce code :

Pytest

Le secret de la réussite

Pytest est un framework complet voir complexe.
Mais quasi toutes les fonctionnalités auxquelles on peut penser sont déjà implémentées et décrites une documentation de qualité.
Comme avec toutes les grosses libraries, il faut avoir confiance en leur design et chercher dedans avant de réinventer la roue.
La liste des fonctionnalités est trop longue, nous venons juste d'en découvrir les principales.

Donc une fois encore, comme pour toutes les bonnes libraires à connaître, le secret est "**RTFM**" !

Qu'est-ce que Git ?

Caractéristiques principales de Git :

- C'est un protocole de gestion de versions décentralisé, A.K.A. Distributed Version Control System (DVCS). Décentralisé car chaque utilisateur a une copie complète du dépôt et de l'historique.
- Protocole au dessus de HTTP ou de SSH. Le plus souvent utilisé avec SSH, car plus sécurisé.
- Utilisé par les serveurs GitHub, GitLab, Bitbucket, Gitee, etc. Ne pas confondre Git et GitHub, ce sont deux choses distinctes, l'un est le protocole, l'autre sont implémentation dans un serveur à des fins commerciales.
- Il est open source et donc gratuit.
- L'intégrité de tout objet est vérifiée par SHA-1
- "Git is easy to learn"[24] 😇

[24] git, https://git-scm.com/

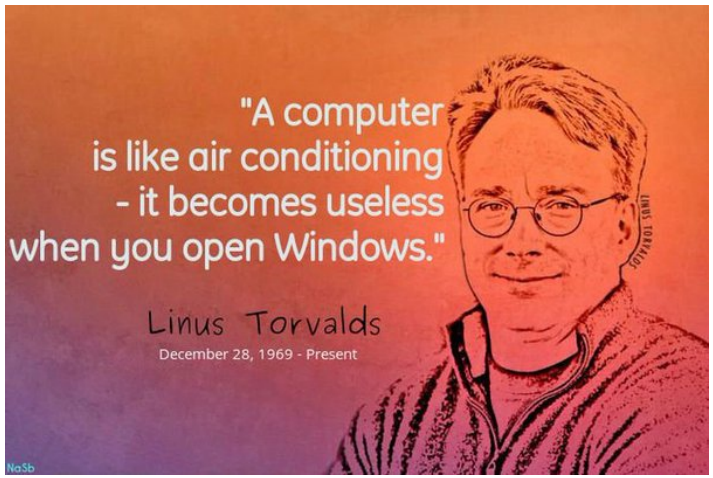
Qu'est-ce que Git ?

L'histoire de Git[25]

Linus Torvalds, le créateur de Linux, a créé Git en 2005 pour gérer le code source du noyau Linux. BitKeeper, un VCS propriétaire, était utilisé avant, mais la licence gratuite a été révoquée par l'éditeur.

Torvalds souhaite développer un VCS libre et gratuit, avec les qualités suivantes :

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

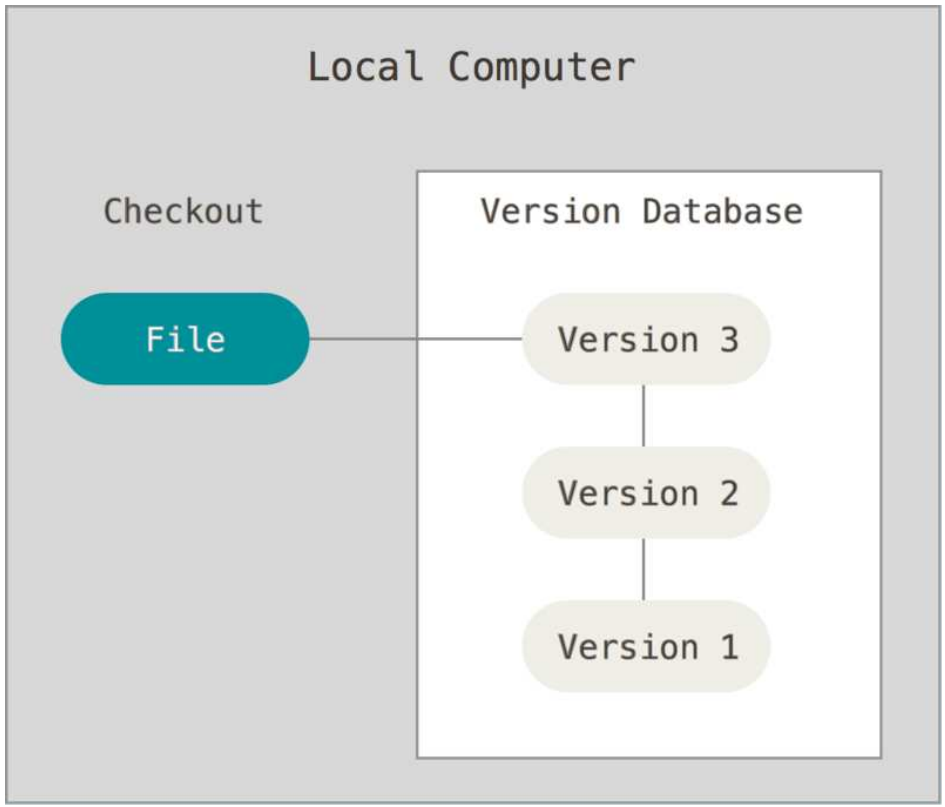


[25] A Short History of Git,

Qu'est-ce que Git ?

Extraits du manuel Git[26]]

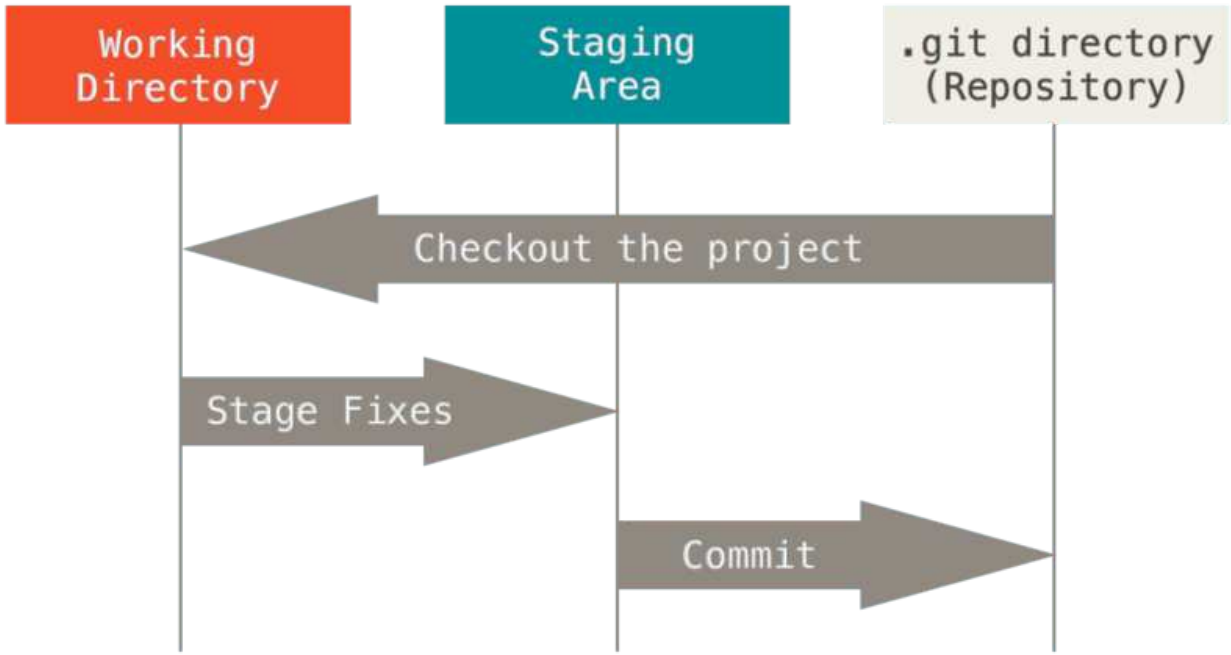
Le code source n'est qu'en un seul exemplaire sur le file system mais une base de données complète est présente sur chaque poste de travail. Elle est dans le `.git` à la racine du projet.



Qu'est-ce que Git ?

Extraits du manuel Git[26]

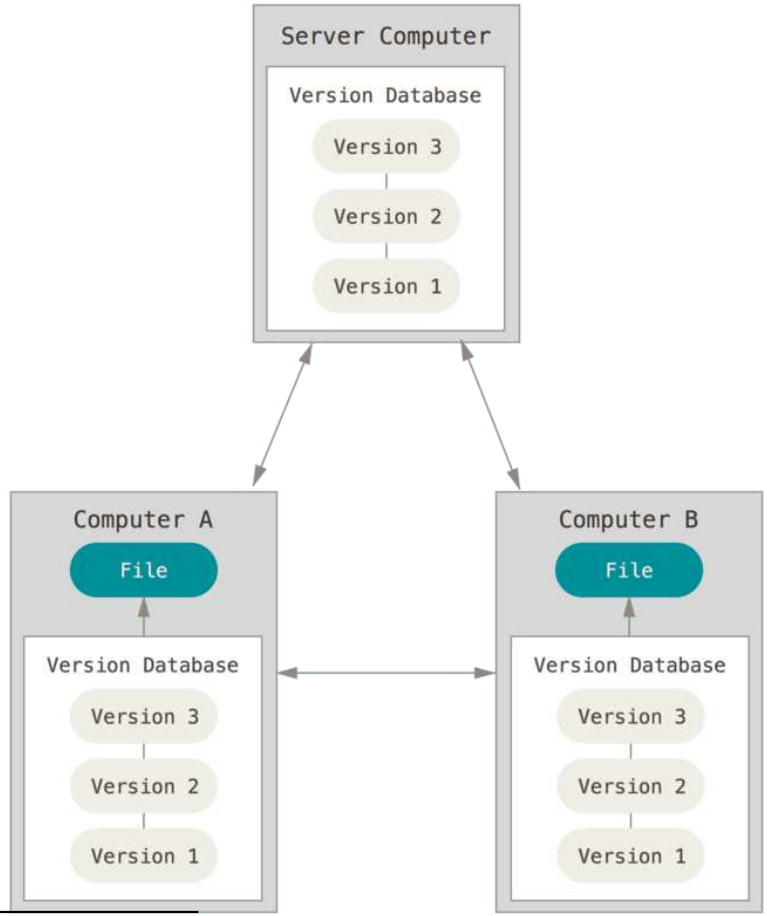
Le "checkout" est l'action de déployer en local une branche de code.
Le "staging" est l'action de désigner un fichier modifié comme faisant partie de la prochaine version, i.e., du prochain "commit".



Qu'est-ce que Git ?

Extraits du manuel Git[26]

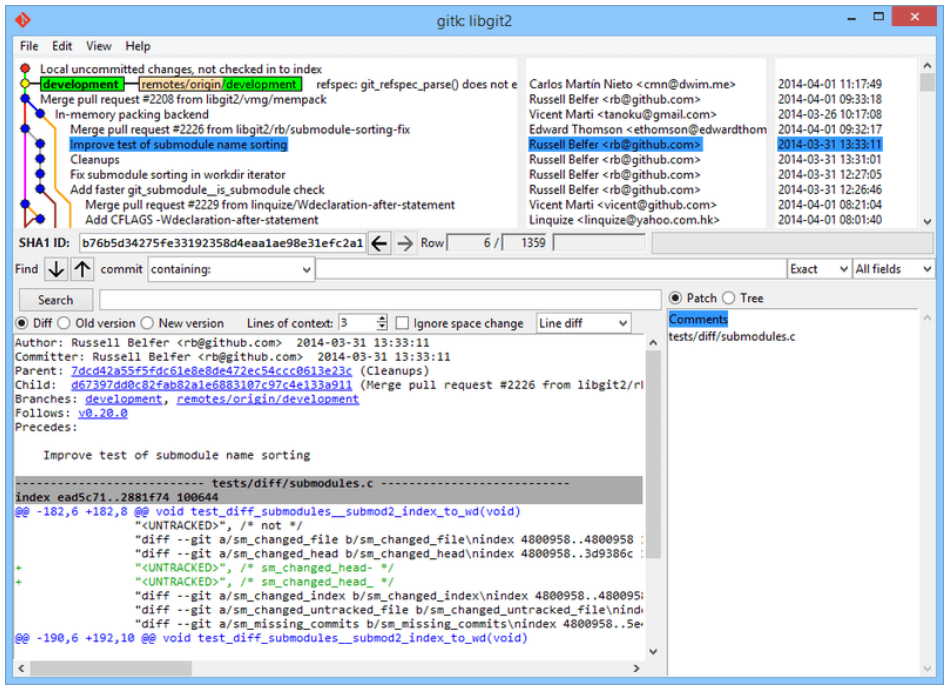
Les développeurs peuvent travailler sur les mêmes fichiers en parallèle. Chaque développeur peut même recréer un serveur en cas de disparition du serveur.



Les interfaces de Git

`gitk`, le GUI[27]

C'est principalement un navigateur de l'historique des commits du repository. Il se lance avec la commande `gitk`.

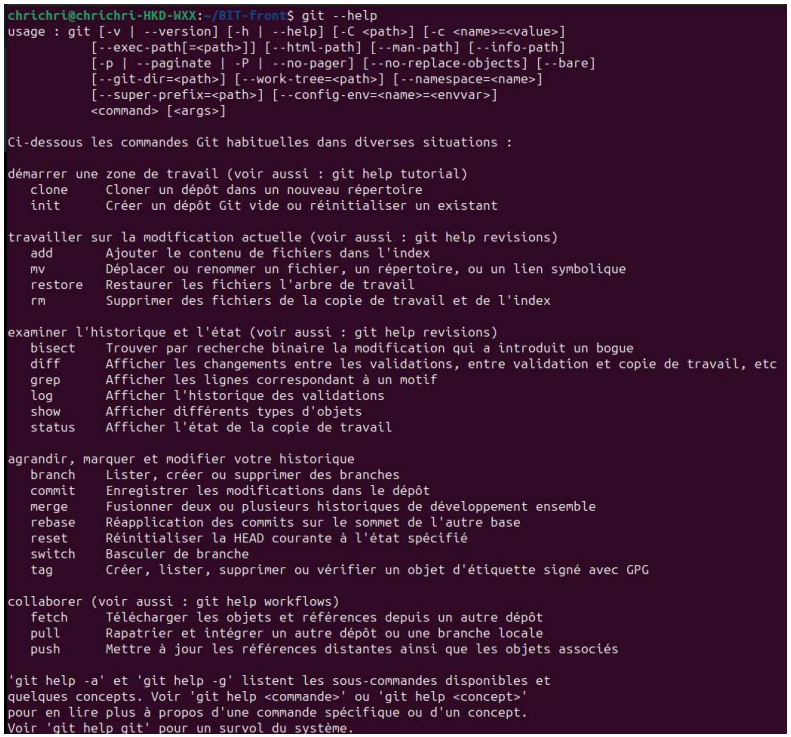


[27] Git in Other Environments - Graphical Interfaces, https://git-scm.com/book/en/v2/Appendix-A:-Git-in-Other-Environments-Graphical-Interfaces

Les interfaces de Git

`git`, la CLI[28]

C'est l'implémentation la plus orthodoxe. Elle porte absolument toutes les fonctionnalités, ce qui n'est pas le cas des autres.



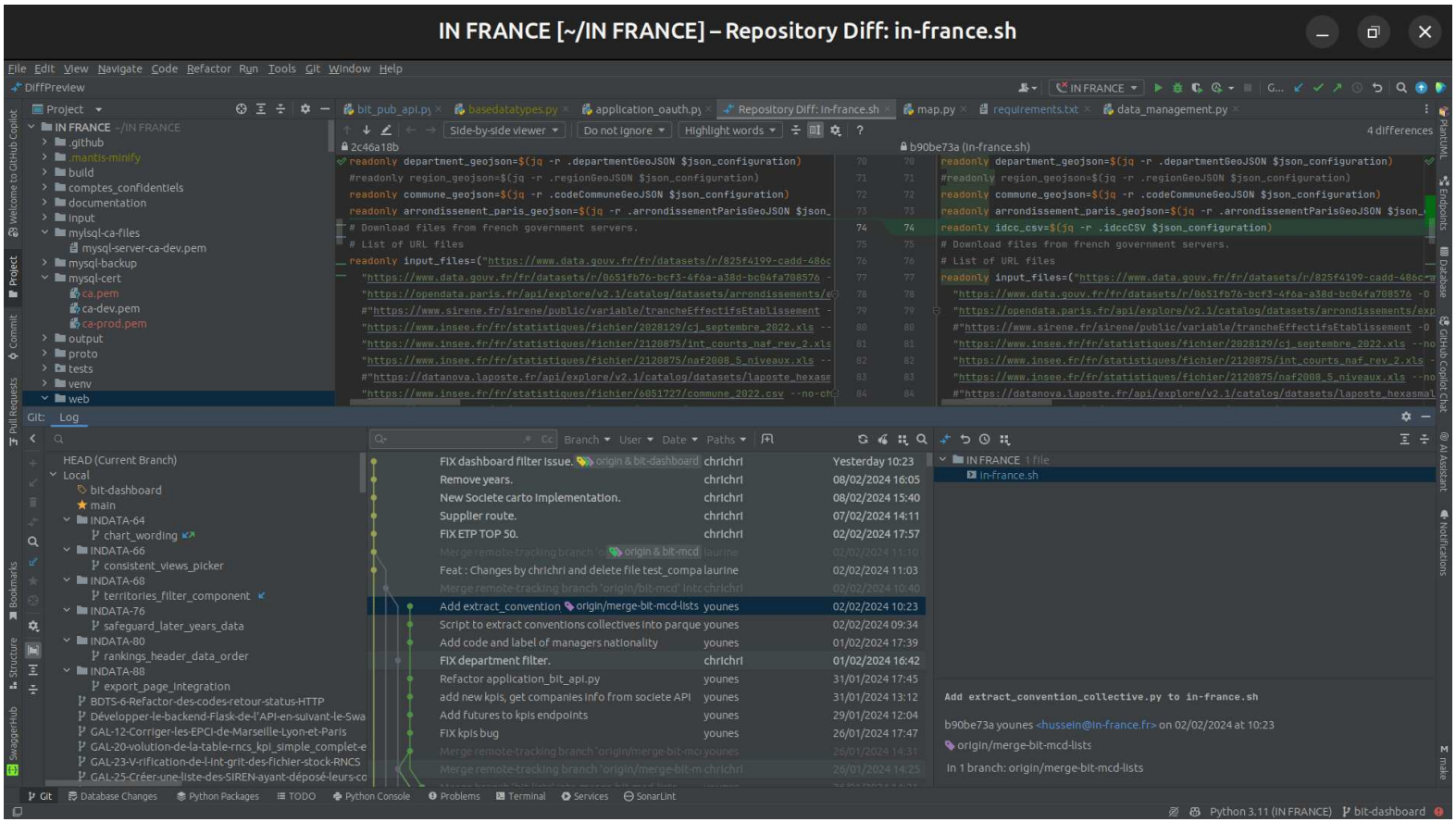
```
chrichri@chrichri-HKD-WXX:~/BIT-front$ git --help
usage : git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

Ci-dessous les commandes Git habituelles dans diverses situations :

démarrer une zone de travail (voir aussi : git help tutorial)
   clone     Cloner un dépôt dans un nouveau répertoire
   init      Créer un dépôt Git vide ou réinitialiser un existant

travailler sur la modification actuelle (voir aussi : git help revisions)
   add       Ajouter le contenu de fichiers dans l'index
   mv        Déplacer ou renommer un fichier, un répertoire, ou un lien symbolique
   restore   Restaurer les fichiers l'arbre de travail
   rm        Supprimer des fichiers de la copie de travail et de l'index

examiner l'historique et l'état (voir aussi : git help revisions)
   bisect    Trouver par recherche binaire la modification qui a introduit un bogue
   diff      Afficher les changements entre les validations, entre validation et copie de travail, etc
   grep      Afficher les lignes correspondant à un motif
   log       Afficher l'historique des validations
   show      Afficher différents types d'objets
   status    Afficher l'état de la copie de travail

agrandir, marquer et modifier votre historique
   branch    Lister, créer ou supprimer des branches
   commit    Enregistrer les modifications dans le dépôt
   merge     Fusionner deux ou plusieurs historiques de développement ensemble
   rebase    Réapplication des commits sur le sommet de l'autre base
   reset     Réinitialiser la HEAD courante à l'état spécifié
   switch    Basculer de branche
   tag       Créer, lister, supprimer ou vérifier un objet d'étiquette signé avec GPG

collaborer (voir aussi : git help workflows)
   fetch     Télécharger les objets et références depuis un autre dépôt
   pull      Rapatrier et intégrer un autre dépôt ou une branche locale
   push      Mettre à jour les références distantes ainsi que les objets associés

'git help -a' et 'git help -g' listent les sous-commandes disponibles et
quelques concepts. Voir 'git help <commande>' ou 'git help <concept>'
pour en lire plus à propos d'une commande spécifique ou d'un concept.
Voir 'git help git' pour un survol du système.
```

[28] Embedding Git in your Applications - Command-line Git, https://git-scm.com/book/en/v2/Appendix-B:-Embedding-Git-in-your-Applications-Command-line-Git

Les interfaces de Git

Git dans votre IDE préféré

Disponible dans les IDEs Jetbrains, VS Code et probablement la plupart des IDEs modernes. Par exemple, au plus près du code Python dans PyCharm :



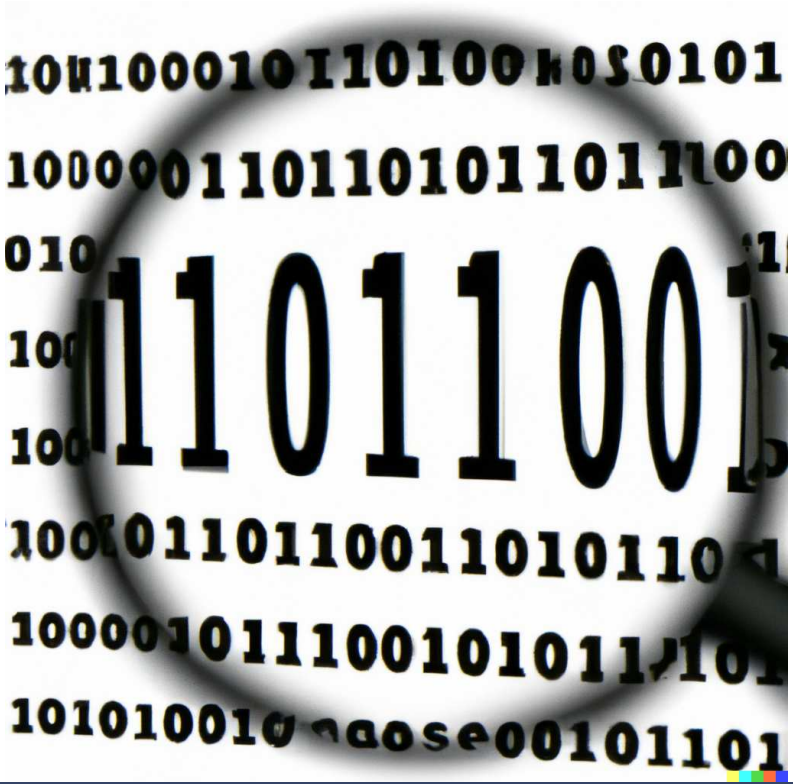
Les interfaces de Git

Les API git dans la pluspart des langages de programmation

Pour intégrer la version dans des rapports de tests, dans un souci de traçabilité par exemple. Le script de test peut appeler l'API pour avoir le hash du commit en cours de test.
Il peut même être afficher dans soft dans un souci de transparence, c'est le hash versioning.

Exemple de libraires :

- Libgit2 pour le C/C++
- JGit pour le Java
- go-git, pour le Go
- Dulwich pour Python



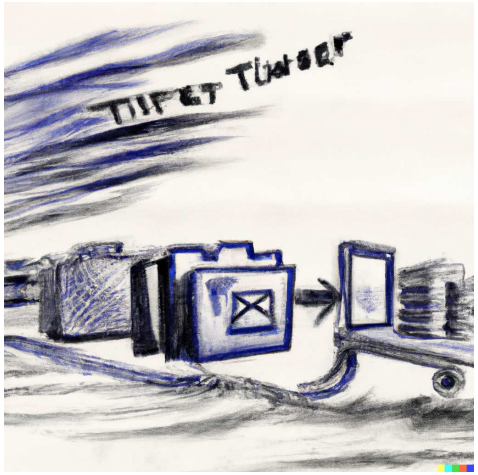
Git, les actions et leurs commandes

Etape 1, récupérer les sources et l'historique d'un repository existant[29]

Pour récupérer le code d'un sur Github par exemple, il suffit de cliquer sur le bouton "Code" et de copier l'URL du protocole HTTPS ou SSH si authentifié avec des clés SSH .
La commande suivante va cloner le repository dans le dossier courant en ajoutant le `.git` et les sources :

```
$ git clone <URL>
```

Par exemple les 7 Go du noyau Linux !



[29] Git Basics - Getting a Git Repository,
https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Git, les actions et leurs commandes

Etape 1, si le repo n'existe pas, qu'on le créé[29][30]

Si le repo n'existe pas, que le projet démarre de zéro, il faut initialiser le dépôt avec les commandes suivantes :

```
$ mkdir mon-nouveau-projet
$ cd mon-nouveau-projet
$ git init
$ git add *.c
$ git add LICENSE
$ git commit -m 'Initial project version'
```

Pour ajouter un dépôt distant (sur le un serveur comme Github par exemple), on peut utiliser la commande :

```
$ git remote add origin <URL>
```

[30] Git remote, https://git-scm.com/docs/git-remote

Git, les actions et leurs commandes

Le local et le remote sont bien différents[32]

Le dépôt local est donc lié à un server distant (GitHub, Gitlab, Bitbucket, etc), le "remote". Il faut bien comprendre que ces deux pools de code peuvent être dans des états bien différents.
Sans action sur le local, les changements (évolution du code, création de branche, etc) du "remote" ne sont pas connus en local. Il faut donc les récupérer avec la commande suivante :

```
$ git fetch
```

La commande `fetch` ne "merge" pas, i.e., ne fusionne pas les modifications du serveur distant.
Plus de détails dans Git Pro book[31].

---

[31] Git Basics - Recording Changes to the Repository, https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

[32] Fetching changes from a remote repository, https://docs.github.com/en/get-started/using-git/getting-changes-from-a-remote-repository#fetching-changes-from-a-remote-repository

Git, les actions et leurs commandes

Créer une version à partir des changements locaux[32]

Pour ajouter un nouveau fichier source ou “stage” un fichier modifié, on peut utiliser la commande :

```
$ git add <fichier source>
```

Les modifications qui ne sont pas stage ne feront pas partie de la prochaine version et ne pourront donc être poussées sur le serveur
Pour créer une version à partir des changements locaux avec un message, on peut utiliser la commande :

```
$ git commit -m "<Message de commit>"
```

Il est aussi possible de stage automatiquement toutes les modifications lors du `commit` avec l’option `-a` comme all, pour gagner du temps.

```
$ git commit -a -m "<Message de commit>"
```

Git, les actions et leurs commandes

Créer une version à partir des changements locaux[32]

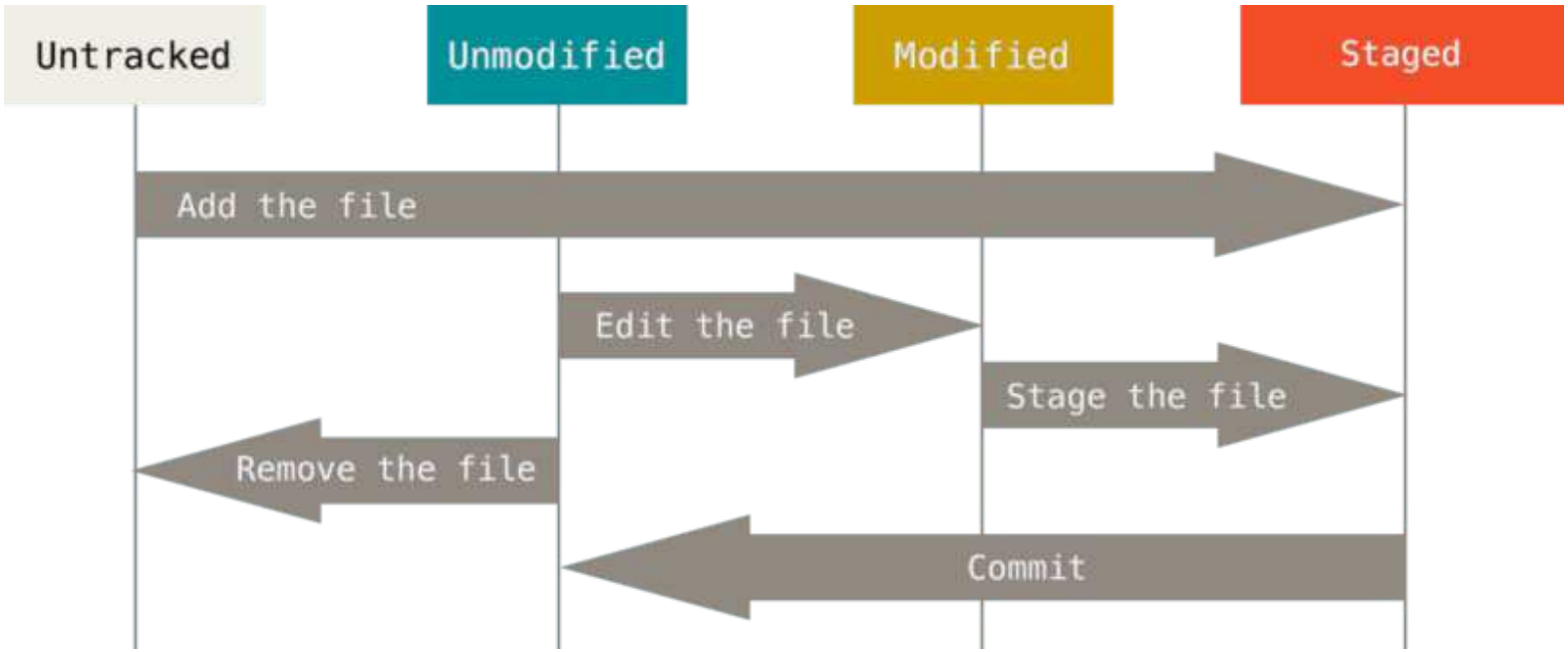
Avant un commit, pour suivre les modifications, lesquelles sont “stage” et lesquelles ne le sont pas, on peut utiliser la commande :

```
$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.

Modifications qui seront validées :
  (utilisez "git restore --staged <fichier>..." pour désindexer)
        modifié : "doc/Qualit\303\251-Code-Source-L3-CSI.tex"

Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git restore <fichier>..." pour annuler les modifications dans le
       répertoire de travail)
        modifié : "doc/build/pdf/Qualit\303\251-Code-Source-L3-CSI.pdf"

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

Git, les actions et leurs commandes

Créer une version à partir des changements locaux[32]

Les différents status d'une source sont donc :



Les bonnes pratiques veulent qu'on ne suit pas, i.e., on ne "stage" pas les fichiers de configuration pour des raisons de sécurité, de données hors jeux de test, de fichier généré (build, log, cache, fichier temporaire, documentation générée, etc).

Git, les actions et leurs commandes

Créer une version à partir des changements locaux[33]

La version est identifiée par un hash, le “commit hash”. Calculé par la fonction de hashage SHA-1, il est unique pour chaque version.
Des commandes permettent de lire l’historique des versions locales, comme par exemple :

```
$ git log
commit 43c417023f1b6277a996cb32843bd6fe955e5aea (HEAD -> master, origin/master)
Author: chrichri <christophe.brun@papit.fr>
Date: Sun Feb 11 01:19:44 2024 +0100

    PDF re-built.

commit 5d939ba92563d9110e5dcf3207d87e383743b33c
Author: chrichri <christophe.brun@papit.fr>
Date: Sun Feb 11 01:18:16 2024 +0100

    Add missing images.
```

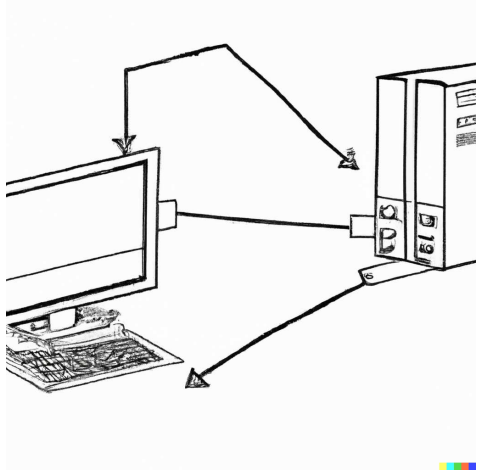
[33] Git Basics - Viewing the Commit History, https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History

Git, les actions et leurs commandes

Pousser la version locale sur le serveur[34]

Pour pousser la version locale sur le serveur, sur la même branche que la locale, on peut utiliser la commande :

```
$ git push
```



Pour pousser une branche locale spécifique, on peut utiliser la commande :

```
$ git push origin <nom-de-branche>
```

Pour pousser une branche locale sur une branche distante différente, on peut utiliser la commande :

```
$ git push origin <nom-de-branche-locale>:<nom-de-branche-distante>
```

Un merge donne lieu à une fusion en mode “fast-forward”.

[34] Git Basics - Working with Remotes, https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes#_pushing_remotes

Git, les actions et leurs commandes

Tirer les modifications du serveur sur le dépôt local[34][36]

Pour tirer du code distant dans le dépôt local, une commande possible est :

```
$ git pull
```

Suite à cette commande, il y a trois cas de figure :

- S'il y a des différences entre les codes, elles sont mergés avec ou sans "fast-forward". Le "fast-forward" est juste une façon de construire l'historique des modifications sans commit de merge. Par défaut il y a "fast-forward", pour avoir le commit de merge il faut ajouter l'option `--no-ff`.
- Si le dépôt local n'a pas été modifié depuis le dernier `fetch`, le `pull` est un simple `fast-forward`.
- S'il y a conflict, c'est à dire que des mêmes lignes ont été modifiés en parallèle, il faut les résoudre avant de pouvoir `commit`[35].

[35] Conflits de merge Git, urlhttps://www.atlassian.com/fr/git/tutorials/using-branches/merge-conflicts

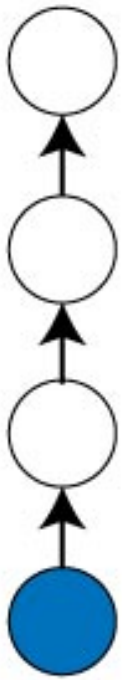
[36] Merge, Fast-Forward et rebase: un peu de culture git, https://www.dynamic-mess.com/developpement/git-merge-fast-forward-rebase-culture-git/

Git, les actions et leurs commandes

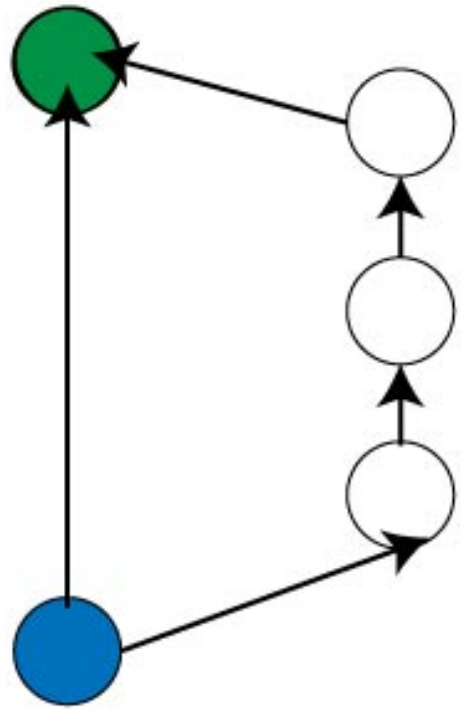
Tirer les modifications du serveur sur le dépôt local[37]

Différence, `fast-forward` et commit de merge, seul l'historique change pas le code :

git fetch suivi d'un
git merge



git fetch suivi d'un
git merge --no-ff



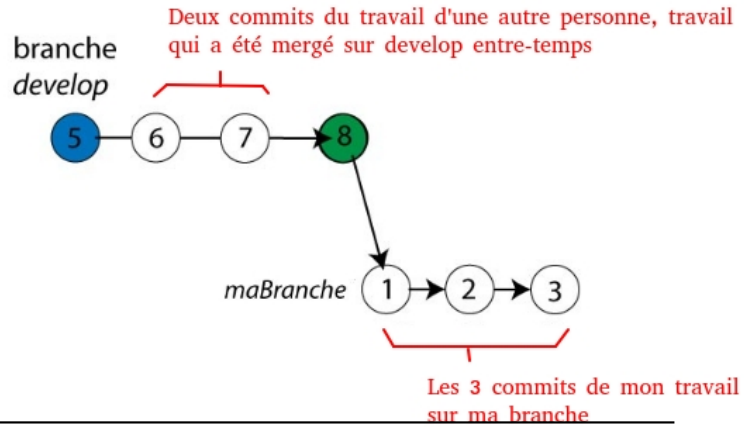
Git, les actions et leurs commandes

Tirer les modifications du serveur sur le dépôt local[39]

Pour tirer du code distant dans le dépôt local, une commande est possible est :

```
$ git rebase
```

Le `rebase` est une autre façon de construire l'historique des modifications sans commit de merge. Les modifications sont placées à la fin de l'historique, ce qui donne un historique plus lisible.



Tout comme avec le `pull`, il peut y avoir des conflits à résoudre. Une fois cela resolu il faut `commit` les modifications. Puis terminer le `rebase` avec la commande avec `git rebase --continue`[38].

[38] Resolving merge conflicts after a Git rebase, https://docs.github.com/en/get-started/using-git/resolving-merge-conflicts-after-a-git-rebase

[39] Merge, Fast-Forward et rebase: un peu de culture git, https://www.dynamic-mess.com/developpement/git-merge-fast-forward-rebase-culture-git/

Pour passer d'une branche à l'autre dans le dépôt local, la commande est :

```
$ git checkout <nom-de-branche>
```

Pour créer une nouvelle branche à partir de la branche courante et passer dessus, rajouter l'option `-b` :

```
$ git checkout -b <nom-de-branche-à-créer>
```

Pour supprimer une branche, rajouter l'option`-d` :

```
$ git branch -d <nom-de-branche-à-supprimer>
```

Pour fusionner une autre branche locale dans la branche courante, la commande est :

```
$ git merge <nom-de-branche-àmerger>
```

[40] Git Branching - Basic Branching and Merging, https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Git, les actions et leurs commandes

Le branching[40]

La fusion de code, appelée “merge”, peut être effectuée automatiquement par `git`.

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
```

Des lignes modifiées dans les deux branches sont en conflit, `git` donne les instructions de résolution :

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)
Changes to be committed:
   modified: index.html
```

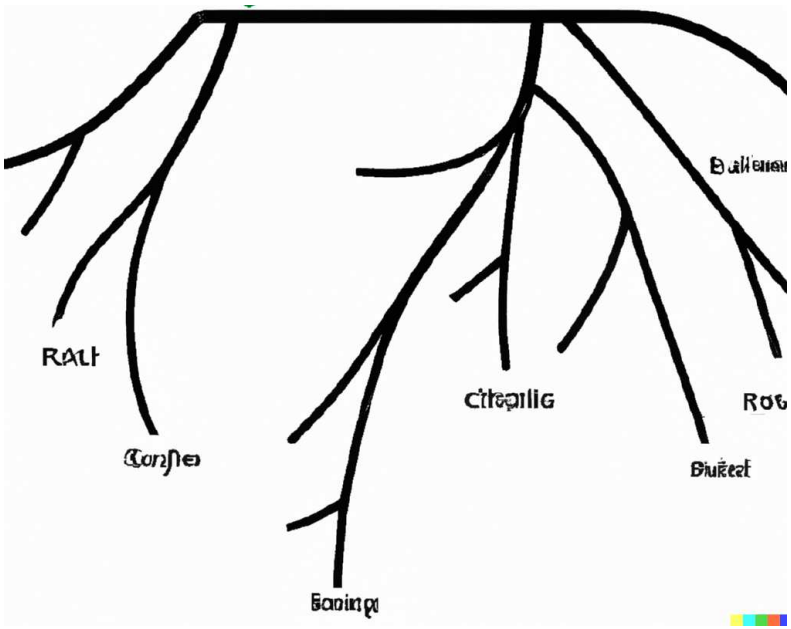
En cas de “conflict”, `git` ajoute des marqueurs de résolution de conflict. Ils permettent de voir les différences entre les deux branches et de choisir la version à conserver ou de créer un mélange des deux.

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

A noter, qu’évidement, ils cassent toutes les syntaxes de tous les langages de programmation. On ne peut donc pas les oublier.

Git, les actions et leurs commandes

Pourquoi créer une branche ?



- Pour développer une nouvelle fonctionnalité sans perturber le code en production.
- Pour développer une correction de bug sans perturber le code en production.

Les bonnes pratiques sont qu'une branche a un nom qui explicite le but du développement.
Son périmètre est restreint pour éviter les conflits de fusion et les codes review trop complexes lors de la pull request.

Git, les actions et leurs commandes

Patchning avec `git rebase`[41]

Avec `git rebase`, on peut patcher n'importe quel commit et modifier l'historique des commits.
Par exemple pour les 5 derniers commits :

```
$ git rebase -i HEAD~5
pick 6a1c95c A bit more Git slides.
pick 5d939ba Add missing images.
pick 43c4170 PDF re-built.
pick 88d6af7 More Git.
pick f820b3e Almost the end of Git

# Rebasage de 49cc10e..f820b3e sur 49cc10e (5 commandes)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
...
```

[41] Git Commands - Patching, https://git-scm.com/book/en/v2/Appendix-C:-Git-Commands-Patching

Git, les actions et leurs commandes

Patchning avec `git rebase`[41]

- “pick” pour garder le commit tel quel.
- “reword” pour modifier le message du commit, si le message n’est pas explicite.
- “edit” pour modifier le code du commit, en cas de bug par exemple.
- “squash” pour fusionner le commit avec le précédent. Souvent utilisé pour amener de la clarté dans l’historique en diminuant le nombre de message.
- “drop” Si une modification, i.e., un commit, n’a plus lieu d’être.

Git, le workflow

Les pull requests, A.K.A PR[42]

Deux pools de code venant de branche ou d'un fork différent du même dépôt peuvent être fusionnées "merge" sur le serveur.

Si on est contributeur dans le même dépôt, c'est une pull request d'une branche dans une autre. Sinon, si on est pas contributeur on peut proposer une pull request d'un fork dans le dépôt original.

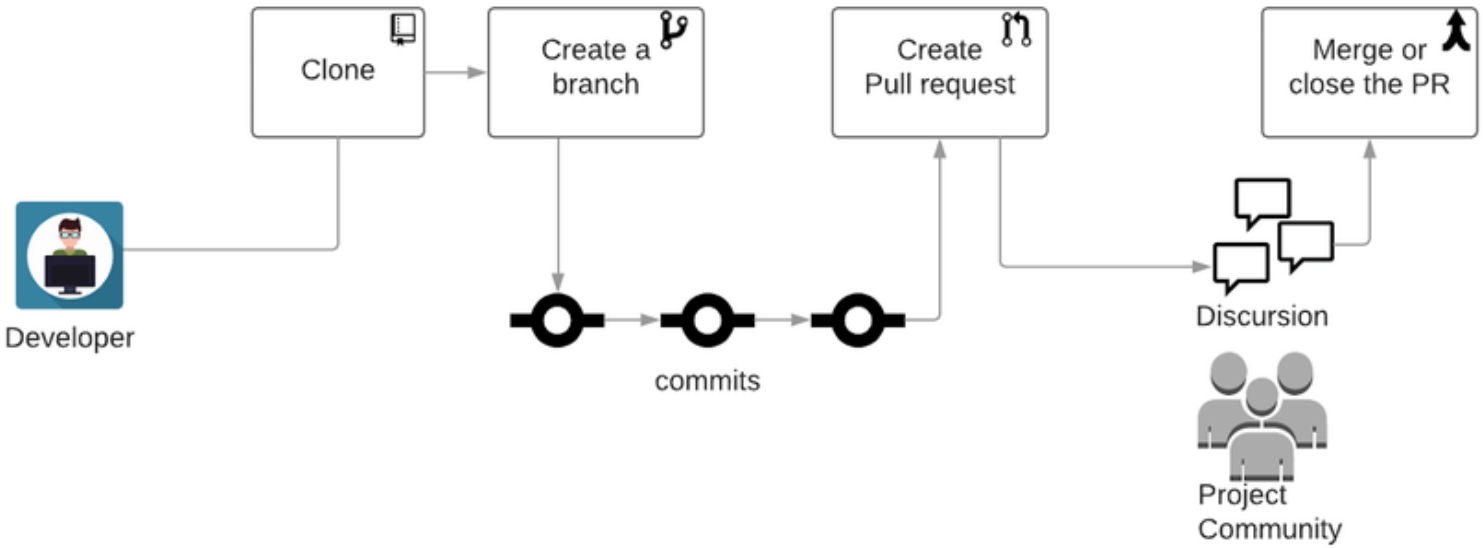
Une page synthétise les différences et permet de discuter des modifications, avant de valider ou non la PR .

En cas de souci lors de la revue, les autres développeurs expliquent les modifications à apporter pour que la PR soit acceptée et la refuse. ❌
Une fois la PR revue et validée par une cohorte minimale de développeurs, le code est fusionné dans la branche cible. ✔️

[42] GitHub - Contributing to a Projecthttps://git-scm.com/book/en/v2/GitHub-Contributing-to-a-Project

Git, le workflow

Les pull requests, schéma récapitulatif[43]



[43] GitHub Pull request flow. https://www.researchgate.net/figure/GitHub-Pull-request-flow_fig1_326295010

Git, travaux dirigés

Utilisé le `clone`, le `branch`, le `push` et faire une PR

Chaque étudiant d'un groupe doit tirer le code du dépôt, écrire un nouveau test dans sa branche, le pousser sur le serveur et faire une PR .

La PR se fait de la branche de l'étudiant vers la branche "main" du dépôt.

C'est le critère d'évaluation de la compréhension de la gestion de version avec Git.

CI/CD est l'acronyme de "Continuous Integration/Continuous Delivery" ou "Continuous Deployment".

"le but ultime du processus CI/CD qui consiste à supprimer l'intervention humaine pour gagner en temps et en efficacité. Le code est compilé et testé sur l'ordinateur grâce à ce que l'on appelle un build automatisé. S'il n'y a pas de bug le tout est mis en production automatiquement. Cette approche permet aux développeurs de se concentrer sur la conception et le développement et sur les fonctionnalités de l'application."[44]

[44] Qu'est-ce que l'approche CI/CD - Qu'est-ce que le processus CI/CD ?, https://www.oracle.com/fr/cloud/definition-approche-ci-cd/

Plusieurs remarques :

- Une des avancées est l'automatisation de multiples tâches.
- Le mot intégration fait penser qu'on a plusieurs systèmes, logiciels, développeurs, équipes, etc.
- L'"ordinateur" n'est pas défini, on verra qu'il peut être celui du développeur ou un serveur distant voir un ordinateur inconnu.
- "S'il n'y a pas de bug le tout est mis en production automatiquement." vous y croyez ?
- Le but est toujours le même, éliminer un maximum de bug.

Histoire du CI/CD

Une histoire récente d'une vingtaine d'années

- 2005, première version de Hudson, le précurseur de Jenkins par Kohsuke Kawaguchi.
- 2010, le terme "Continuous Integration" est utilisé pour la première fois dans un livre par Jez Humble[45].
- 2011, Jenkins est libéré.
- 2024, plus de 38 plateformes mainstream[46].

[45] Continuous Delivery, https://www.continuousdelivery.com/

[46] 38 Best CI/CD Tools For 2024, https://www.lambdatest.com/blog/best-ci-cd-tools/

Les avantages du CI/CD

Si tout peut être réalisé en local, pourquoi utiliser un serveur ?

- Les plateformes de CI/CD sont avant tout des outils collaboratifs. Elles permettent le partage des développements dans l'équipe et dans l'entreprise. On peut suivre en direct l'état d'avancement et la qualité des développements de ses collègues.
- Elles s'intègrent à de nombreux outils comme les SSO, Git, les outils de gestion de projet et de qualité comme ceux d'Atlassian (Jira, Confluence, Bitbucket, etc).
- Une UX/UI plus simple que des données un terminal. La prise en main peut-être faite par des métiers autre que celui de développeur (Testeur, Release Manager, Hardware).
- Permet de partager les ressources hardware requises pour la compilation, les tests, entre des machines plus ou moins disponibles. E.g., dans l'embarqué, cela permet de ne pas multiplier le hardware et de limiter le nombre de bancs de test. En générale la compilation qui peut prendre beaucoup de ressources CPU et peut être délocalisée sur un serveur dédié.

Les avantages du CI/CD

Quelles différences avec le DevOps ?

Le CI/CD est un des outils du DevOps.

C'est l'outil dédié au développement et à la livraison de logiciels de qualité. On confond parfois les deux car le terme DevOps étant plus hype que CI/CD, est plus souvent est utilisé.

Le fondamental du CI/CD

Livre et site web de Jez Humble[45]

Jez Humble fait un travail academic depuis les années 2000 pour mesurer l'impact du CI/CD sur les professionnels de l'IT .

En plus de plusieurs paramètres liés à la qualité logiciel, il mesure même un impact positif sur l'ambiance de travail !

Son livre sur le sujet, "Continuous Delivery", est une référence dans le domaine. Son site web `https://www.continuousdelivery.com/` contient déjà beaucoup de ressources.

Le business du CI/CD

Ce n'est pas une simple technologie mais un marché, attention aux coûts

Git, Python, que nous avons vu, sont des technologies open source devenues des standards. Toutes leurs fonctionnalités sont disponibles gratuitement. Contrairement à elles, certaines plateformes de CI/CD sont payantes.

Parmi les business model, on trouve :

- Le freemium, une version gratuite limitée en fonctionnalités.
  - Github est gratuit jusqu'à une certaine consommation de ressources (temps CPU, disque, etc) et est limité en fonctionnalités.
  - Gradle est toujours "full featured" en local mais Devlocity, le SAAS qui va avec et rajoute des fonctionnalités est payant.
  - Gitlab Community Edition VS Entreprise edition.
- L'auto-hébergement d'une solution complète open source et le on-premise.
  - Jenkins.
- Les solutions propriétaires.
  - TeamCity est payant en auto-hébergement et on-premise.
  - Travis CI .

Les différences entre plateformes

À lire absolument, un comparatif des différentes plateformes par Atlassian https://www.atlassian.com/fr/continuous-delivery/continuous-integration/tools.

Les différences sont nombreuses, on peut citer :

- Certains permettent de créer des clusters sur n'importes quelles machines. Par exemple Jenkins avec son cluster formé de "slave(s)" et d'un "master". Tout ce qui se passe sur les "slaves" est visible sur l'IHM du "master".
- AWS CodePipeline permet d'avoir de multiples nodes mais sur des machines AWS uniquement 🤑.
- Devlocity n'a qu'une instance centrale du serveur
- Leur communauté, une communauté open source peut développer des plugins, des scripts, des outils, etc, pour virtuellement toutes les applications qui ne sont pas prévues dans la version de base. De ce point de vue, Jenkins est dans les premiers.
- L'organisation en jobs ou en pipelines de jobs.

Les limites du CI/CD

Quelles sont les limites du CI/CD ?

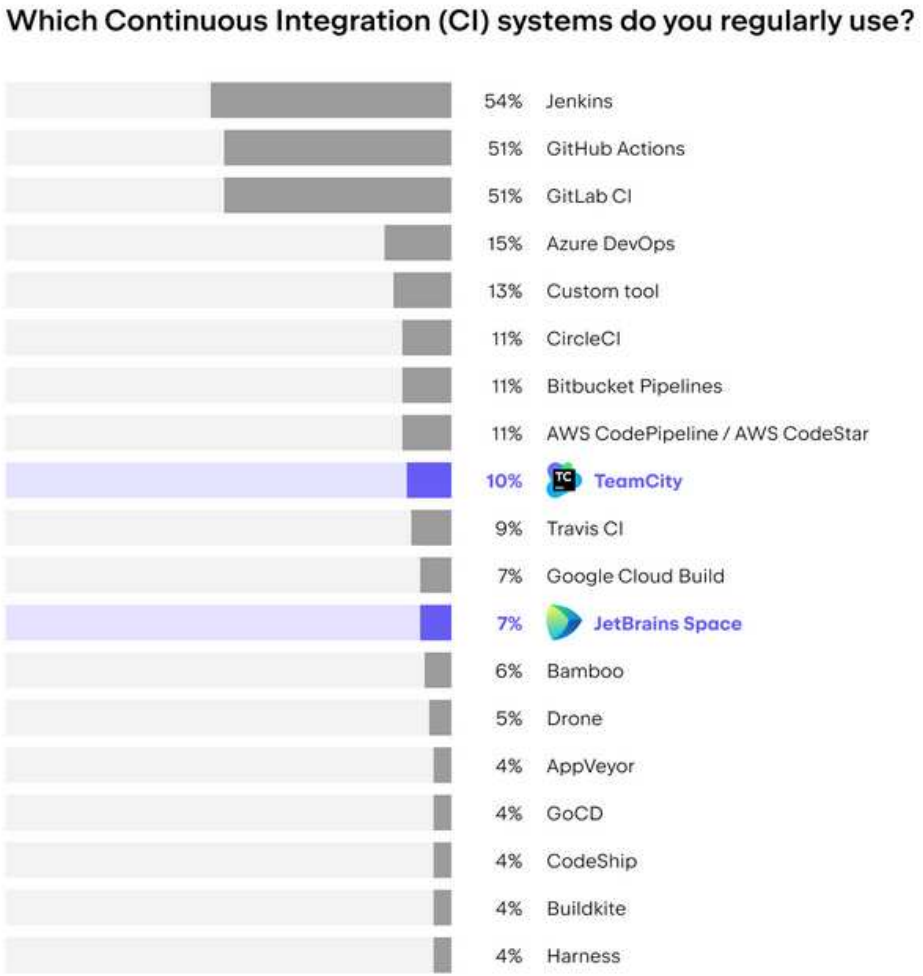
Les limites du CI/CD

Quelles sont les limites du CI/CD ?

- le coût des licences ou des machines.
- Pas de limites sur le type de logiciel, on peut y connecter un banc de test pour l'embarqué et l'IOT, dans le web, la data, etc...
- Les ressources des machines disponibles.
- La complexité des tests, plus ils sont nombreux et plus ils sont longs à faire tourner et analyser.

Les plateformes les plus utilisées CI/CD

Résultat de l'enquête JetBrains 2023 sur le CI/CD[47]

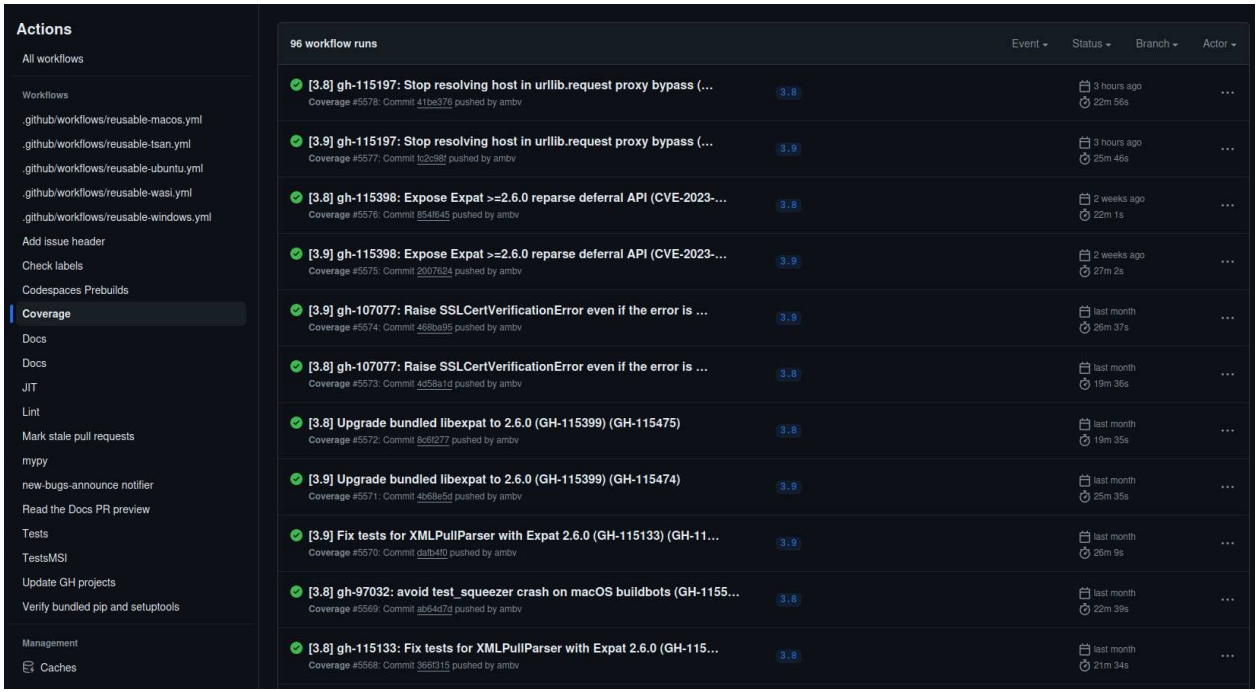


[47] Best Continuous Integration Tools for 2024 – Survey Results, https://blog.jetbrains.com/teamcity/2023/07/best-ci-tools/

Exemple de CI/CD

Exemple de CPython sur GitHub[48] Actions

Une partie seulement de CPython, l'implémentation classique officielle de Python est sur GitHub, il y en a aussi sur Azure DevOps. On y trouve 21 workflows différents !

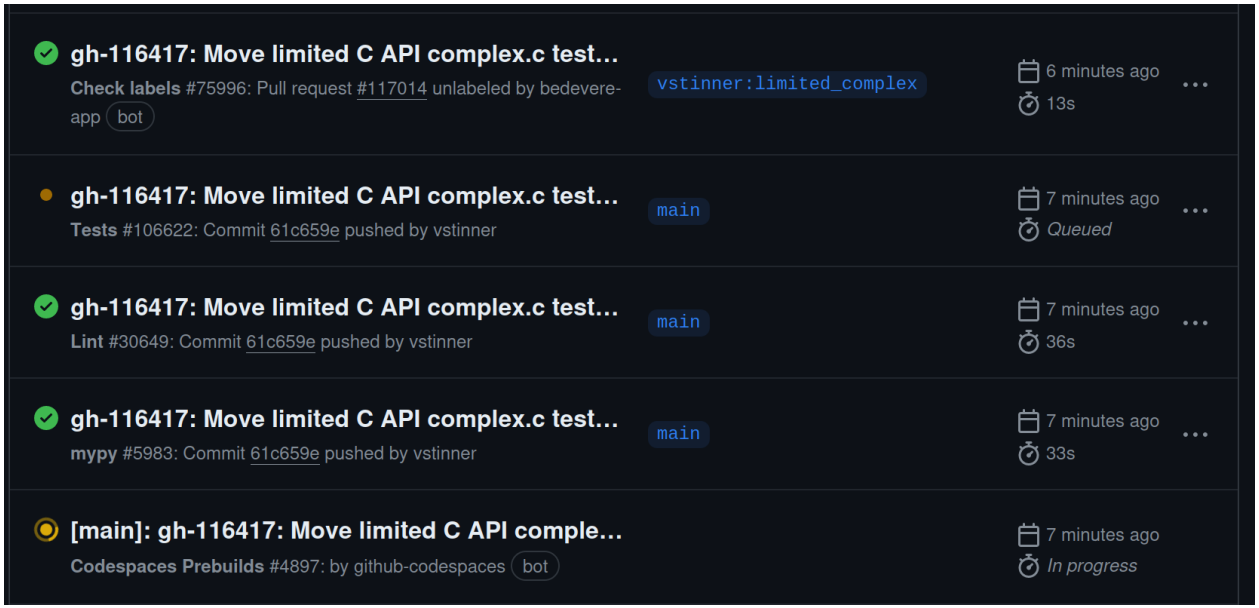


[48] CPython implémentation, https://github.com/python/cpython?tab=readme-ov-file

Exemple de CI/CD

Exemple de CPython sur GitHub[49] Actions

5 workflows “triggered” par un push et une PR !



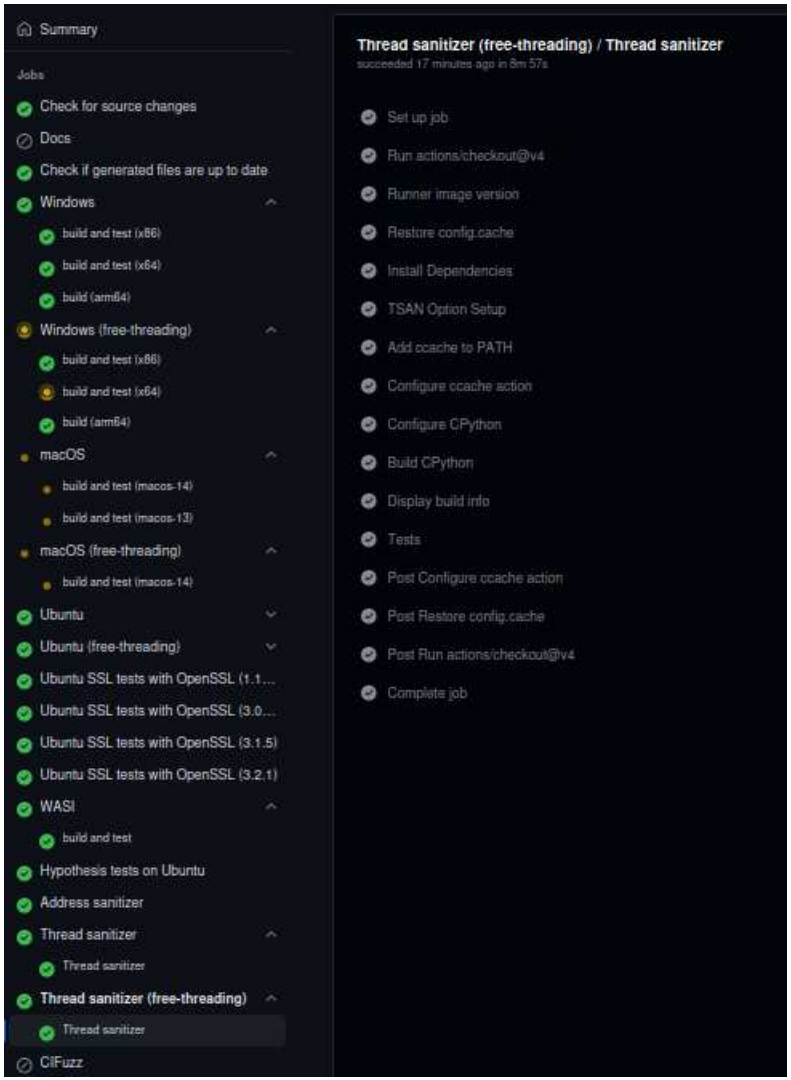
[49] CPython implémentation, https://github.com/python/cpython?tab=readme-ov-file

Exemple de CI/CD

Exemple de CPython sur GitHub[50] Actions

1 seul workflow compte des dizaines de jobs !

1 seul job compte plusieurs steps !



[50] CPython implémentation, https://github.com/python/cpython?tab=readme-ov-file

Exemple de CI/CD

Exemple de worflow GitHub[51]

Definition :

- Un workflow est une suite de jobs.
- Un job est une tâche à réaliser.
- Un step est une étape de la tâche.
- Un artefact est un fichier généré par un step.

[51] Automate your workflow from idea to production, https://github.com/features/actions

Exemple de CI/CD

Exemple de worflow GitHub de ce repository, exercice de 15 minutes

Que peut-on faire comme workflow, GitHub Actions, etc, sur les dépôts de ce cours ?

- https://github.com/St-Michel-IT/qualite-code-source/
- https://github.com/St-Michel-IT/testing/

Exemple de CI/CD

Exemple de worflow GitHub de ce repository[52]

Les workflows sont dans les fichiers suivants :

```
chrichri@chrichri-HKD-WXX:~/Documents/Campus-St-Michel-IT/qualite-code-source$
    tree .github/
.github/
--- workflows
   --- build-document.yml
   --- checkmytex.yml
   --- shell-check.yml
```

- `build-document.yml` pour générer le document PDF que vous lisez.
- `shell-check.yml` pour vérifier la syntaxe des scripts Shell.
- `checkmytex.yml` pour lister les erreurs dans le fichier LaTex.

[52] Worflow de génération des documents à partir de Latex, https://github.com/St-Michel-IT/qualite-code-source/blob/master/.github/workflows/

Exemple de CI/CD

Exemple de worflow GitHub du repository testing[53]

Les workflows sont dans les fichiers suivants :

```
chrichri@chrichri-HKD-WXX:~/Documents/Campus-St-Michel-IT/testing$ tree .github
.github
--- workflows
    --- build-document.yml
    --- checkmytex.yml
    --- pylint.yml
    --- shell-check.yml
    --- black-formatter.yml
    --- unit-tests.yml
```

- `unit-tests.yml` lance les tests avec pytest.
- `pylint.yml` un linter qui génère un rapport des erreurs et des warnings.
- `black-formatter.yml` Black formatter qui fait un check sans rien changer.

[53] Worflow de génération des documents à partir de Latex, https://github.com/St-Michel-IT/testing/blob/master/.github/workflows/

Exemple de CI/CD

Pas de runner officiel des worfkflows GitHub en local

Tous les workflows doivent tourner sur des machines GitHub. Avec les coûts et la latence que cela implique.

Quelles sont donc les bonnes pratiques pour éviter les pièges des plateformes comme GitHub, Circle CI, Jenkins, etc ?

[54] Accelerate developer productivity, https://gradle.org/

Exemple de CI/CD

Pas de runner officiel des worfkflows GitHub en local

Tous les workflows doivent tourner sur des machines GitHub. Avec les coûts et la latence que cela implique.

Quelles sont donc les bonnes pratiques pour éviter les pièges des plateformes comme GitHub, Circle CI, Jenkins, etc ?

- Éviter des fonctionnalités propres à ces plateformes. C'est à dire agnostiques aux plateformes.
- Écrire des scripts qui peuvent être exécutés en local. Par exemple (scripts BASH, scripts PowerShell, scripts Gradle[54], Dockerfiles, scripts Ansible)
- Ces scripts peuvent-être appelés dans les jobs du CI/CD .
- Nettoyez vos environnements après le build pour économiser les ressources.

[54] Accelerate developer productivity, https://gradle.org/

Gradle

Gradle comme une passerelle entre le local et la plateforme de CI/CD

Gradle[54] est un outil de build open source. On script en Groovy[55] ou Kotlin[56], en local des tâches de build, équivalente aux jobs des plateformes de CI/CD .

On débug en local, on teste en local, on commit en local. Cela fait gagner un temps considérable (pas de commit/push...) dans le développement du CI/CD . Et donc, cela fait également économiser des ressources de CI/CD .

> Toutes les plateformes de CI/CD ont un plugin/workflow Gradle. La portabilité est donc assurée.

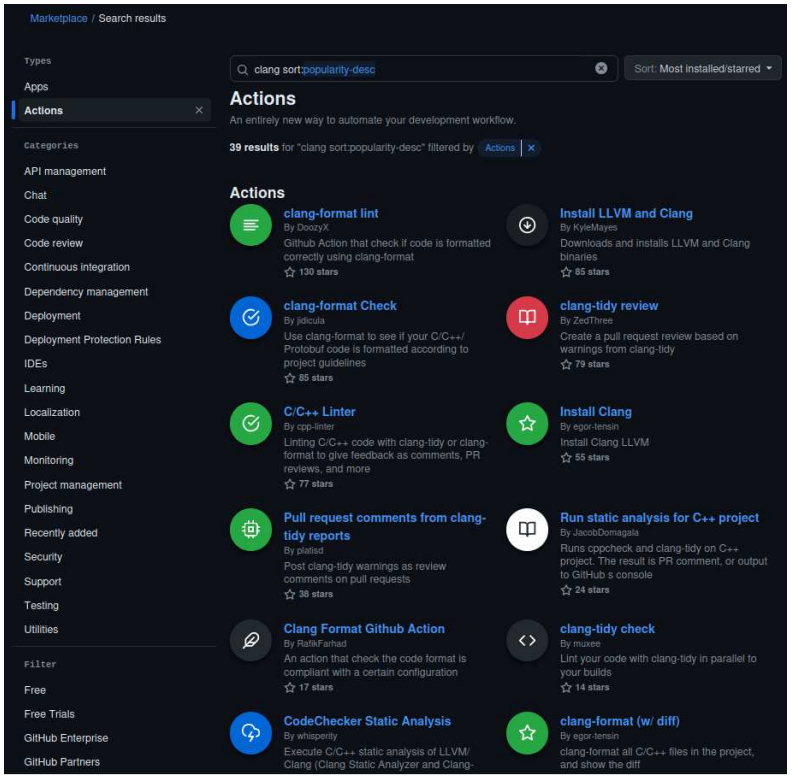
[55] Groovy, A multi-faceted language for the Java platform, https://groovy-lang.org/

[56] Kotlin, https://kotlinlang.org/

Les marketplaces des plateformes de CI/CD

Un “market” gratuit ou payant

Ne pas avoir peur du nom marketplace, beaucoup de plugins Jenkins ou de workflows GitHub sont gratuits.



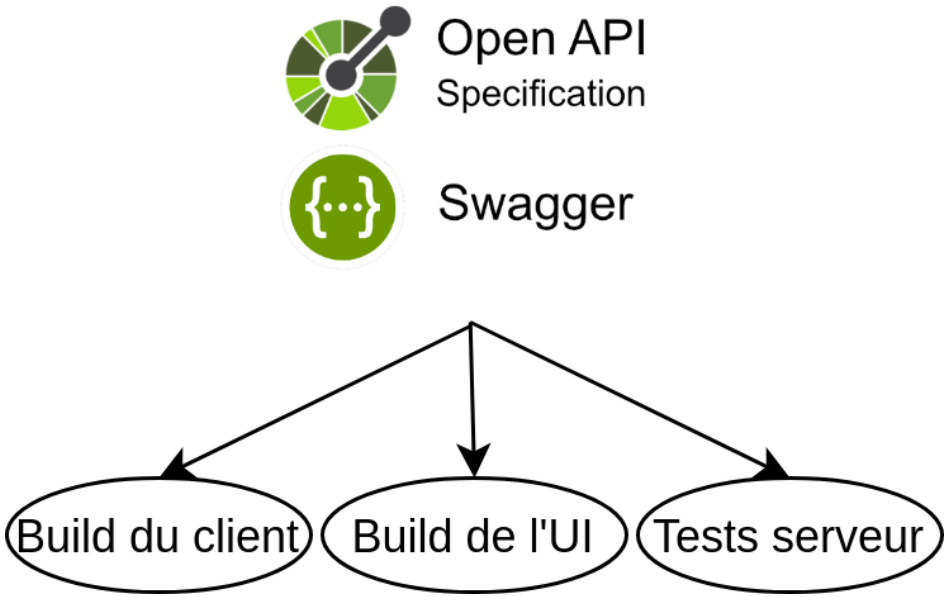
- Plus de 1 900 plugin Jenkins[57].
- Plus de 22 000 actions GitHub[58].

Ils sont essentiels pour développer plus rapidement les jobs (reuse, comme dans n’importe quel code).

[57] Jenkins Plugins, https://plugins.jenkins.io/

[58] GitHub Actions, https://github.com/marketplace?type=actions

Swagger est un fichier de définition d'une API . Il est écrit en YAML ou en JSON. Et les données pour décrire l'API sont dans la OpenAPI Specification[59].



[59] API Development for Everyone, https://swagger.io/

Les exemples de CI/CD dans le développement d'API

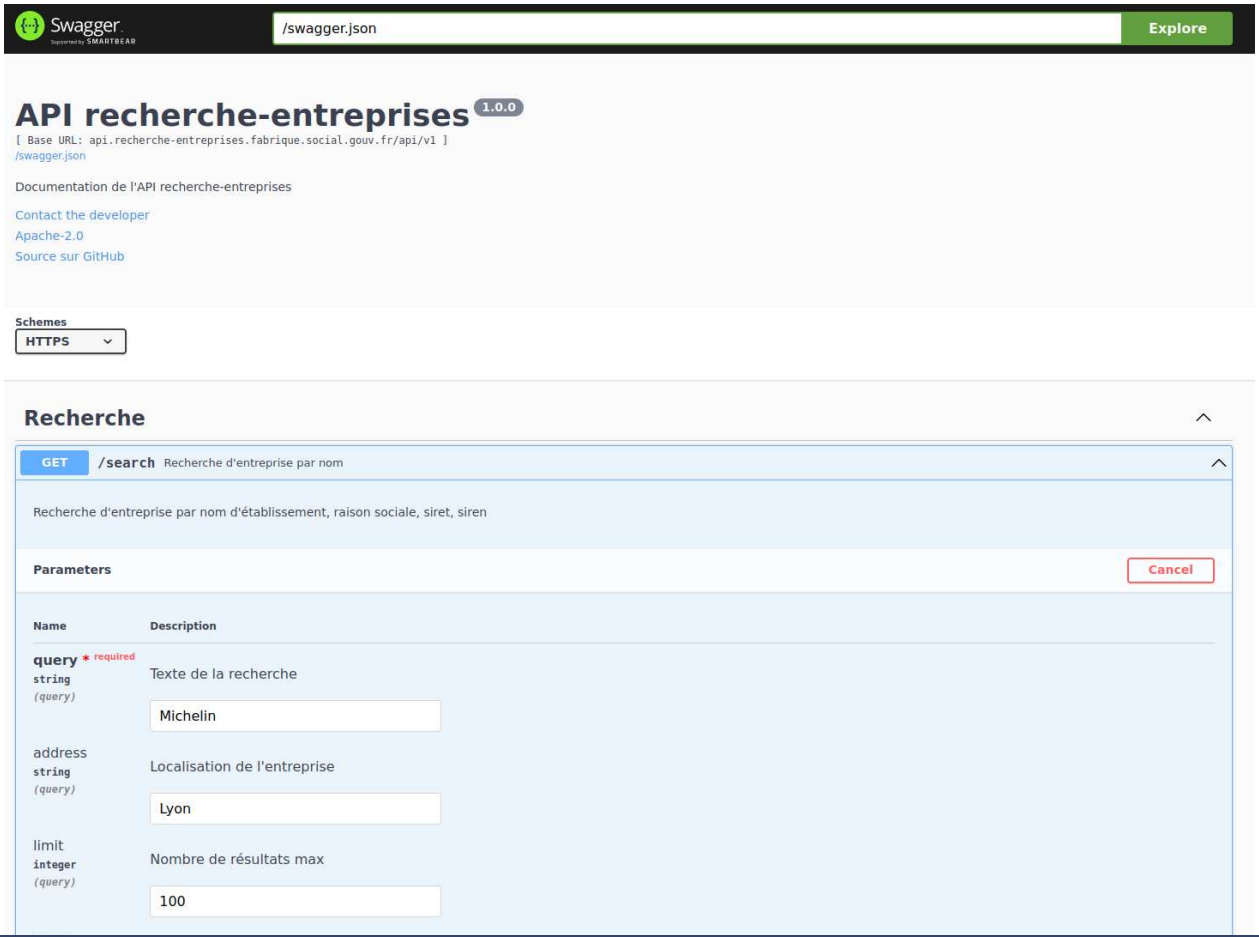
Grâce à Swagger et OpenAPI

```
/lists:
   get:
     summary: Gets all available data lists.
     description: Returns all custom data lists available to the current user.
     tags:
       - Lists
     responses:
       200:
         description: 'OK'
         content:
           application/json:
             schema:
               type: array
               items:
                 # Reuse du schema de DataList !
                 $ref: '#/components/schemas/DataList'
       401:
         # Reuse des responses !
         $ref: '#/components/responses/Unauthorized'
       5XX:
         $ref: '#/components/responses/ServerError'
```

Les exemples de CI/CD dans le développement d'API

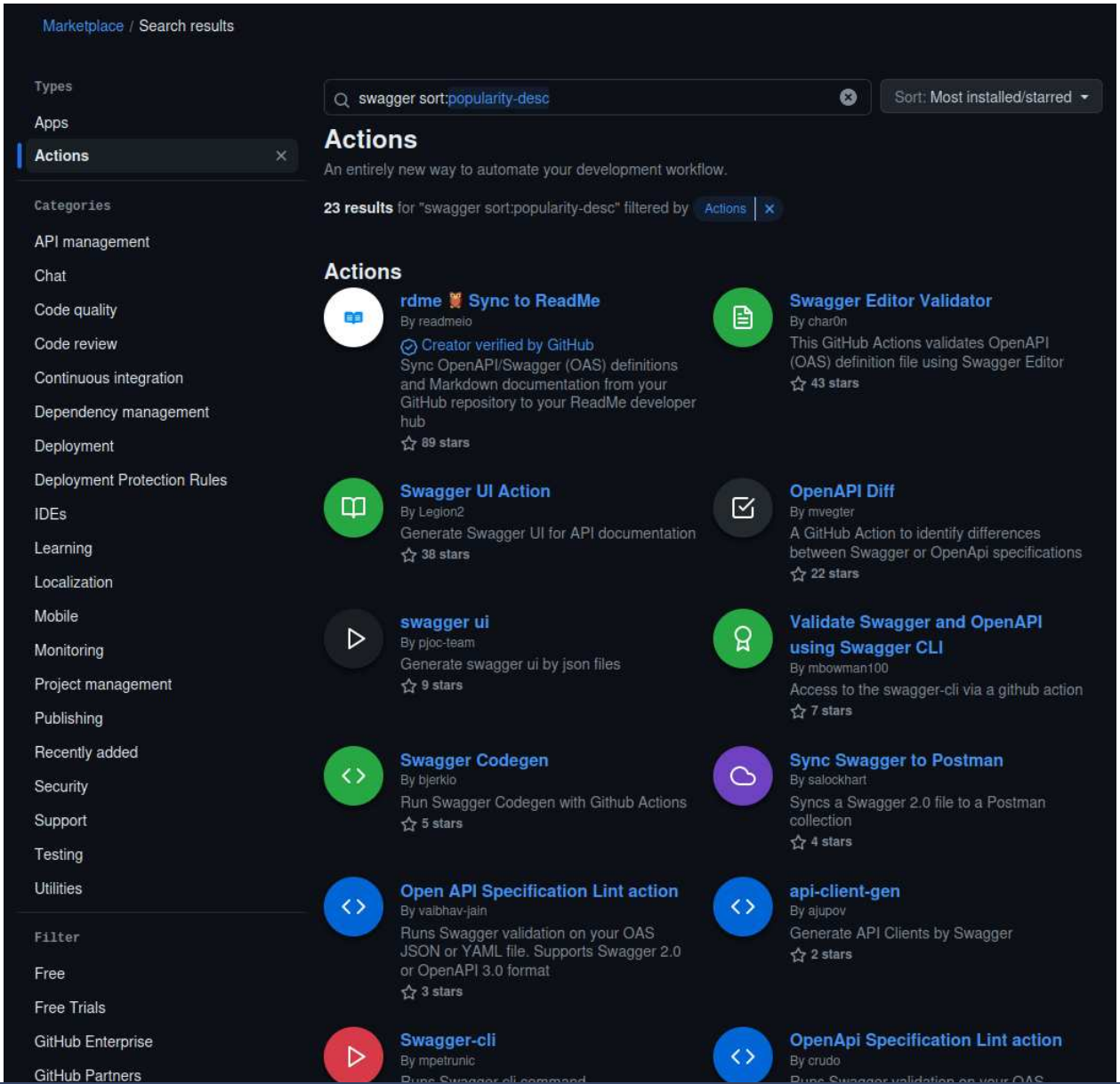
Une UI pour découvrir l'API

L'UI est un espace convivial pour découvrir l'API . Il fait aussi office de documentation. Par exemple https://api.recherche-entreprises.fabrique.social.gouv.fr/#/Recherche/search.



Les exemples de CI/CD dans le développement d'API

Toutes les fonctionnalités sont déjà dans la marketplace (ou les plugins pour Jenkins)



Les exemples de CI/CD avec le déploiement d'une image Docker

GitHub ou Docker Hub peuvent héberger des images Docker[60]

```
    - name: Log in to Docker Hub
      uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
      with:
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}
    - name: Extract metadata (tags, labels) for Docker
      id: meta
      uses: docker/metadata-action@9ec57ed1fcdbf14dcef7dfbe97b2010124a938b7
      with:
        images: my-docker-hub-namespace/my-docker-hub-repository
    - name: Build and push Docker image
      uses: docker/build-push-action@3b5e8027fcad23fda98b2e3ac259d8d67585f671
      with:
        context: .
        file: ./Dockerfile
        push: true
        tags: ${{ steps.meta.outputs.tags }}
        labels: ${{ steps.meta.outputs.labels }}
```

[60] Publishing Docker images, https://docs.github.com/en/actions/publishing-packages/publishing-docker-images

Les exemples de CI/CD avec la publication de builds

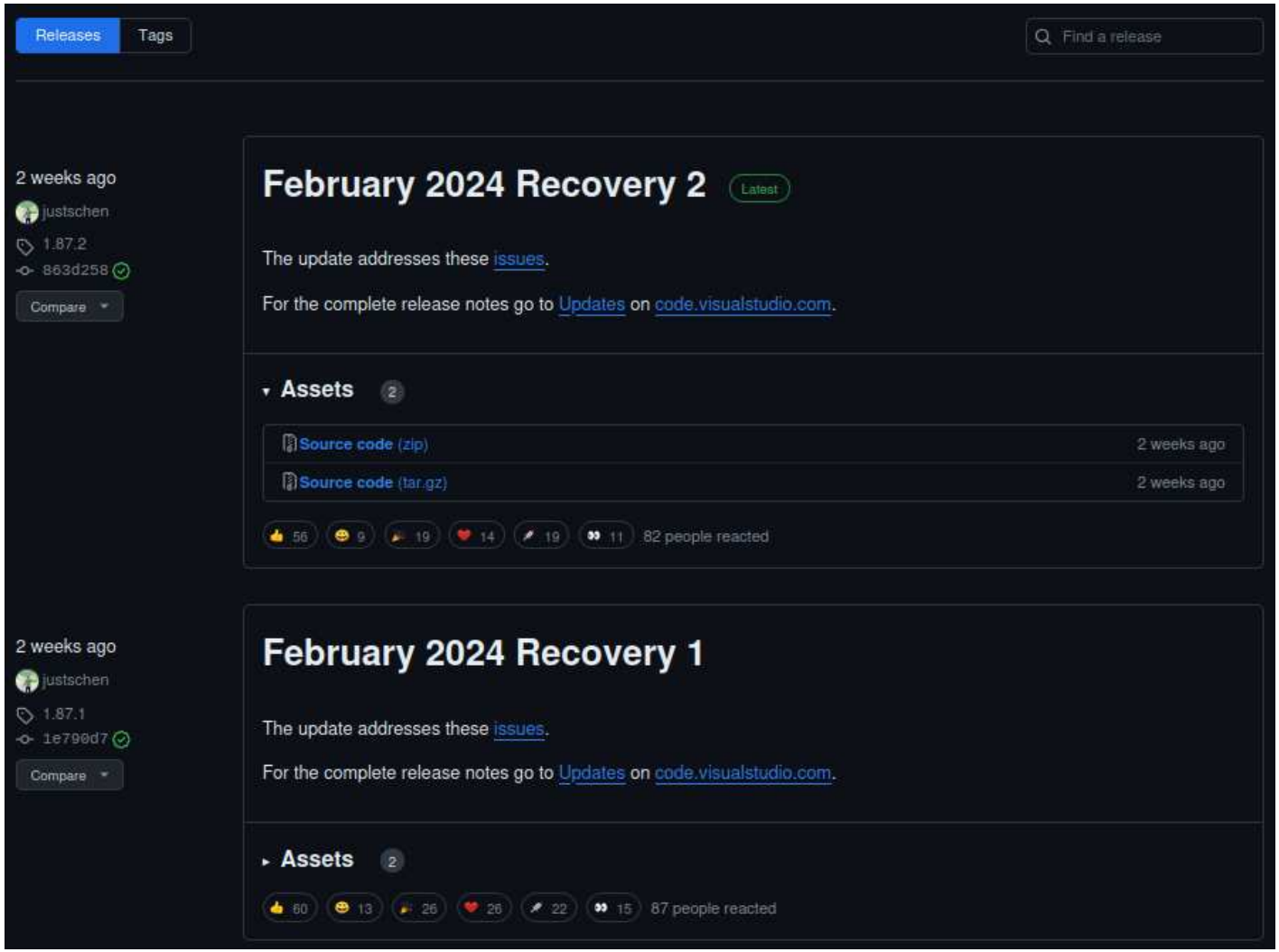
Pour publier un soft compilé ou packagé sur sa page du dépôt GitHub[61]

```
name: Releases

on:
  push:
    tags:
    - '*'

jobs:

  build:
    runs-on: ubuntu-latest
    permissions:
      contents: write
    steps:
    - uses: actions/checkout@v3
    - uses: ncipollo/release-action@v1
      with:
        artifacts: "release.tar.gz,foo/*.txt"
        bodyFile: "body.md"
```

[61] Create Release, https://github.com/marketplace/actions/create-release

Les exemples de CI/CD avec la publication de builds

Publication de VS Code packagé[62]



[62] microsoft/vscode, https://github.com/microsoft/vscode/releases

Quelques exemples d'étapes de compilation du CI/CD

Quelles types de jobs pour passer d'un code source à un exécutable ?

Quelques exemples d'étapes de compilation du CI/CD

Quelles types de jobs pour passer d'un code source à un exécutable ?

- Compilation de C/C++ en binaire avec GCC et Vagrant pour cibler plusieurs les architectures.
- Compilation de Python en bytecode.
- Compilation de Java en Jar.
- Packaging de NodeJS en NPM .
- Packaging de Python en Wheel.
- Compiler LaTex en PDF .
- etc...

La machine distante peut-être un cloud public, un serveur dédié, un serveur de l'entreprise, etc.

On peut entre autre utiliser :

- Pousser une image Docker sur un registre et trigger un redémarrage de l'exécution du conteneur.
- Pousser un exécutable ou du code sur une machine distante à travers SSH, donc `sftp` ou `rsync` par exemple.

Les jobs sont des tâches unitaires.
Un job est plus simple, c'est principalement un ensemble de commandes de "build".
Ce sont des commandes shell, console ou PowerShell en fonction de l'environnement.

Les pipelines sont des enchainements de jobs. Le pipeline est écrit en Groovy[55] et mais le principe est similaire au fichier Gradle ou workflow GitHub[63].

[63] Pipeline Examples, https://www.jenkins.io/doc/pipeline/examples/

Pour être agnostique à une plateforme de CI/CD, on peut exécuter dans la plateforme de CI/CD un build Gradle.
Jenkins, comme les autres plateformes, permet évidement de faire cela dans les builds ou les pipelines[64].

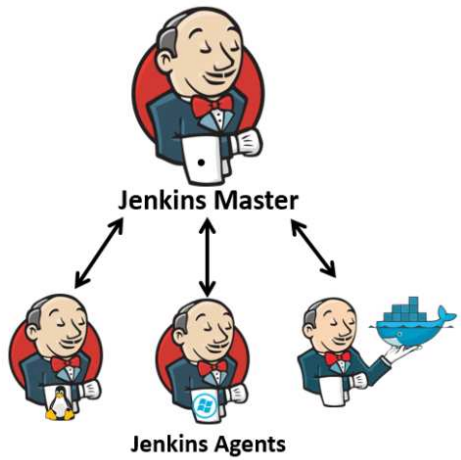
Si un maximum de tâches, d'intelligence, de scripting sont dans le build Gradle, il sera plus simple de changer de plateforme de CI/CD .

[64] Jenkins Gradle plugin, https://plugins.jenkins.io/gradle/

Quelques notions de base de Jenkins

Les clusters de slaves et master[65]

Différentes machines esclaves peuvent être reliées à une machine maître. Soit pour répartir la charge, soit pour avoir des machines avec des environnements ou des configurations différentes. Ce sont des Jenkins slave agents aussi appelés node.



[65] Jenkins master-slave setup, https://www.linkedin.com/pulse/jenkins-slave-node-soma-sekhar-k-gusdc/

Risques cyber liés au CI/CD

Quels sont les risques cyber liés au CI/CD ?

[66] Echoes of SolarWinds, https://www.scmagazine.com/news/echoes-of-solarwinds-jetbrains-teamcity-servers-under-attack-by-russia-bac

[67] The CircleCI secrets hack is a red flag for security teams on software supply chain risk, https://www.reversinglabs.com/blog/circleci-hack-is-a-red-flag-for-security-teams-on-the-software-supply-cha

Risques cyber liés au CI/CD

Quels sont les risques cyber liés au CI/CD ?

Une mauvaise configuration des droits, une faille non patchée, un secret exposé peuvent permettre d'accéder aux données du CI/CD . Donc probablement le code source.
Exemples de hacks majeurs liés au CI/CD :

- TeamCity non patché dans l'attaque de SolwarWinds[66]. Un fournisseur de services et logiciels de supervision de réseau. Les hackers ont ensuite exploité les failles de Solar Winds et accédé aux réseaux des clients comme MS qui a vu un source de Windows fuiter.
- Circle CI a vu des secrets exposés[67]. Le hack n'a pas touché ceux qui avait mis en place Open ID Connect, la configuration est donc importante.

[66] Echoes of SolarWinds, https://www.scmagazine.com/news/echoes-of-solarwinds-jetbrains-teamcity-servers-under-attack-by-russia-bac

[67] The CircleCI secrets hack is a red flag for security teams on software supply chain risk, https://www.reversinglabs.com/blog/circleci-hack-is-a-red-flag-for-security-teams-on-the-software-supply-cha

Conclusions sur le CI/CD

- C'est un outil de plus pour améliorer la qualité des livrables (documentation, rapport de tests, librairies, exécutables, etc).
- Pour éliminer du bug.
- Un outil collaboratif pour partager les résultats dans l'équipe et l'entreprise.
- Faire d'abord en local puis exécuter dans la plateforme CI/CD .
- S'inspirer des outils qu'on a vu dans ou l'IDE ou en testing pour écrire des scripts.
- Attention aux ressources sur les machines locales.
- Attention aux coûts sur le cloud.

Travaux dirigés sur le CI/CD

L'évaluation de ce chapitre porte sur la création de workflows GitHub Actions (ou une autre plateforme si vous avez la possibilité) pour les dépôts de ce cours.

Ils doivent contenir un **maximum** de tâches automatisées pour monitorer la qualité du code source.

Les chapitres 1 sur les bonnes pratiques, et 2 sur les tests unitaires doivent être intégrés à un CI/CD .

Chaque étudiant doit au moins développer un workflow, ceci sera évalué grâce aux users Git.

Conclusion sur le cours Qualité du Code Source

- Savoir programmer, connaître une syntaxe n'est pas suffisant pour livrer du code selon les standards modernes.
- De nombreux outils permettent d'éliminer des bugs et faciliter les évolutions et la maintenance.
- La lisibilité du code est un facteur important pour la qualité du code.
- La documentation est clé pour permettre aux autres développeurs de comprendre le code.
- Les tests unitaires sont un outil pour évaluer la qualité du code.
- Tous ces outils peuvent être automatisés et les résultats partagés grâce au CI/CD .
- "Stay tuned" car les choses changent vite, DYOR, RTFM !