

HOOK' IL EST BEAU NOTRE CODE !

GUIDER LA QUALITE DE CODE AVEC PRE-COMMIT

LA QUALITE PERD TOUJOURS SON CHEMIN . . .



Luc Sorel-Giffo – vendredi 30 juin 2023 - 16h amphi B – BreizhCamp

@lucsorelgiffo@floss.social – (Jérôme Marchand @Neken13)

QUALITE DE CODE, CONVENTIONS ET RELECTURES

Pourquoi nous relisons-nous ?

QUALITE DE CODE, CONVENTIONS ET RELECTURES

Pourquoi nous relisons-nous ?

- le code doit rendre le **service métier** attendu

QUALITE DE CODE, CONVENTIONS ET RELECTURES

Pourquoi nous relisons-nous ?

- le code doit rendre le **service métier** attendu
- perpétuer la **robustesse et la maintenabilité** du service numérique

QUALITE DE CODE, CONVENTIONS ET RELECTURES

Pourquoi nous relisons-nous ?

- le code doit rendre le **service métier** attendu
- perpétuer la **robustesse et la maintenabilité** du service numérique
- maintenir une cohérence de la base de code, une cohésion de l'équipe

QUELS OUTILS PEUVENT-ILS NOUS AIDER ?

QUELS OUTILS PEUVENT-ILS NOUS AIDER ?

- tests automatisés, en suivant :
 - la couverture de code testé
 - la durée
 - la consommation de ressources (RAM, CPU)

QUELS OUTILS PEUVENT-ILS NOUS AIDER ?

- tests automatisés, en suivant :
 - la couverture de code testé
 - la durée
 - la consommation de ressources (RAM, CPU)
- analyse statique de code
 - règles de formatage
 - détection d'antipatterns

ET SI L'ANALYSE STATIQUE COMMENÇAIT AU COMMIT ?

(juste avant, en fait...)

LES HOOKS GIT

Chaque action git est conditionnée par la bonne exécution de son hook.

```
super-projet/
  +-.git/
    +-- hooks/
      +-- commit-msg.sample
      +-- pre-commit.sample 00
      +-- pre-push.sample
      +-- pre-receive.sample
      +-- ...
```

- retirer `.sample` pour activer le hook
- 💡 le contexte d'exécution du hook est la racine du projet git, pas `.git/hooks`

EXAMPLE DE HOOK DE PRE-COMMIT

```
mkdir super-projet  
cd super-projet  
git init -b main  
git config ...
```

```
📄 .git/hooks/pre-commit  
#!/bin/sh  
echo "Pre-commit hook launched in $(pwd)"  
  
# simulates an error code at exit  
exit 1
```

```
touch test.txt  
git add test.txt  
git status  
> Changes to be committed [...]  
>       new file:   test.txt  
  
git commit -m "🎉"  
> Pre-commit hook launched in [...]/super-projet  
  
git status  
> Changes to be committed [...]  
>       new file:   test.txt
```

COMMENT VERSIONNER LES HOOKS DU PROJET ?

COMMENT VERSIONNER LES HOOKS DU PROJET ?

- .git / n'est pas versionnable

COMMENT VERSIONNER LES HOOKS DU PROJET ?

- .git/ n'est pas versionnable
-  créer un dossier à part (.hooks/) + configurer git pour les y chercher (git config core.hooksPath ./ .hooks)

COMMENT VERSIONNER LES HOOKS DU PROJET ?

- .git / n'est pas versionnable
-  créer un dossier à part (.hooks/) + configurer git pour les y chercher (git config core.hooksPath ./ .hooks)
-  utiliser un outil qui va gérer la *tambouille* entre les hooks git et les outils de vérification : pre-commit

PRE-COMMIT

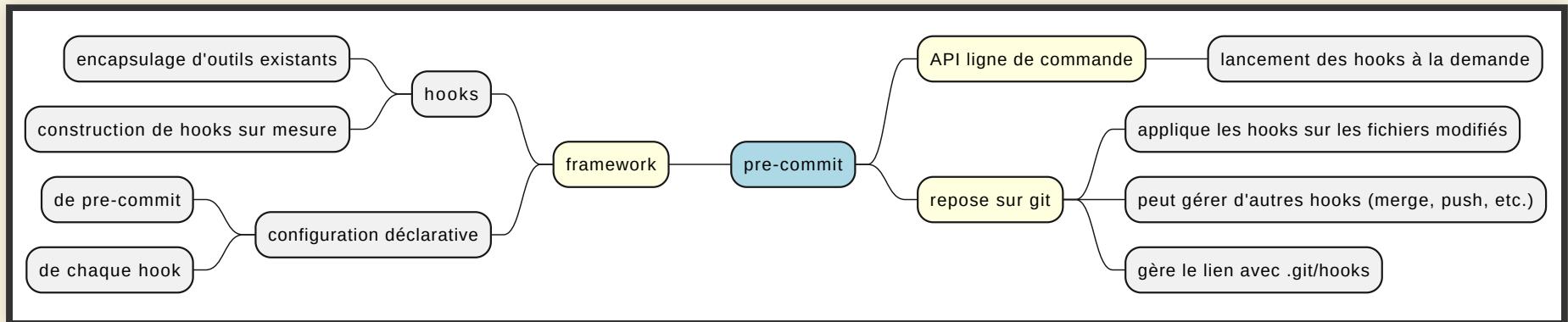


A framework for managing and maintaining multi-language pre-commit hooks.

- github.com/pre-commit/pre-commit
- 10.5k ⭐, 99+ releases (juin 2023)
- open-source (MIT license)

Figure 1. <https://pre-commit.com/>

TAMBOUILLE-AS-CODE



Bibliothèque écrite en Python ❤️💛

- dépendance de développement de votre projet Python
- ou exécutable via une commande docker
- configuration dans `.pre-commit-config.yaml`

LES HOOKS NATIFS

```
📄 .pre-commit-config.yaml
repos:
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: v3.3.3
  hooks:
    # vérifie la syntaxe des fichiers yaml (json, xml, toml, etc.)
    - id: check-yaml
    # supprime les caractères non imprimables de fin de ligne
    - id: trailing whitespace
    # s'assure que chaque fichier se termine par un retour à la ligne (un seul)
    - id: end-of-file-fixer

    # évite de versionner de gros fichiers (>100 ko)
    - id: check-added-large-files
      args: [--maxkb=100]

    # remplace les guillemets "doubles" par des 'simples' sauf si "ça n'est pas possible"
    - id: double-quote-string-fixer
```

Voir pre-commit.com/hooks.html.

COMMANDES

```
# câblage avec git (génère le script .git/hooks/pre-commit)
pre-commit install
pre-commit install --hook-type commit-msg

# lance tous les hooks sur tous les fichiers
pre-commit run --all-files

# lance un hook sur tous les fichiers
pre-commit run end-of-file-fixer --all-files

# met à jour les hooks natifs
pre-commit autoupdate
```

Voir pre-commit.com/#usage.

DEMO SUR UN PROJET PYTHON

Application progressive des hooks sur le projet d'exemple
github.com/lucsorel/hook-il-est-beau-notre-code/tree/main/expylliarmus.

TRI DES IMPORTS AVEC ISORT //

Ajout du hook dans .pre-commit-config.yaml :

```
- repo: https://github.com/PyCQA/isort
  rev: 5.12.0
  hooks:
    - id: isort
      # pour aller chercher la configuration dans pyproject.toml
      additional_dependencies: [toml]
```

TRI DES IMPORTS AVEC ISORT // //

Configuration de `isort` dans `pyproject.toml`:

```
[tool.isort]
# en cohérence avec les autres outils de formatage ou analyse de code
line_length = 120
# Mode 5 de groupage des imports : compatible avec yapf
# from third_party import (
#     lib1, lib2, lib3, lib4,
#     lib5, etc.
# )
multi_line_output = 5
balanced_wrapping = false
# TESTING->known_testing : crée une section spécifique d'imports concernant le contenu des tests
sections = ["FUTURE", "STDLIB", "THIRDPARTY", "FIRSTPARTY", "LOCALFOLDER", "TESTING"]
known_testing = ["tests"]
```

FORMATAGE DE CODE AVEC YAPF //

Ajout du hook dans .pre-commit-config.yaml :

```
- repo: https://github.com/google/yapf
  rev: v0.40.1
  hooks:
    - id: yapf
      name: Yapf
      # pour aller chercher la configuration dans pyproject.toml
      additional_dependencies: [toml]
```

FORMATAGE DE CODE AVEC YAPF //

Configuration de yapf pour une indentation "json" des ([{}]):

```
[tool.yapf]
based_on_style = "facebook"
# voir la section https://github.com/google/yapf#knobs
COALESCE_BRACKETS = false
# en cohérence avec les autres outils de formatage ou analyse de code
COLUMN_LIMIT = 120
DEDENT_CLOSING_BRACKETS = true
INDENT_DICTIONARY_VALUE = false
EACH_DICT_ENTRY_ON_SEPARATE_LINE = true
FORCE_MULTILINE_DICT = true
JOIN_MULTIPLE_LINES = false
SPACES_AROUND_DEFAULT_OR_NAMED_ASSIGN = false
SPLIT_BEFORE_CLOSING_BRACKET = true
SPLIT_BEFORE_DICT_SET_GENERATOR = true
SPLIT_COMPLEX_COMPREHENSION = true
SPLIT_BEFORE_EXPRESSION_AFTER_OPENING_PAREN = true
SPLIT_BEFORE_FIRST_ARGUMENT = true
```

Alternative : black (formateur volontairement sans configuration).

ANALYSE DE PRATIQUES AVEC RUFF //

Ajout du hook dans .pre-commit-config.yaml :

```
- repo: https://github.com/astral-sh/ruff-pre-commit
  rev: v0.0.275
  hooks:
    - id: ruff
```

ANALYSE DE PRATIQUES AVEC RUFF // //

Configuration du linter ruff dans pyproject.toml :

```
[tool.ruff]
# en cohérence avec les autres outils de formatage ou analyse de code
line-length = 120
# familles de vérifications activées (https://beta.ruff.rs/docs/rules/)
select = ["A", "B", "E", "F", "W", "N", "SIM", "C4"]
# ne pas vérifier la longueur des lignes, isort et yapf le font déjà
extend-ignore = ["E501"]

# mettre ce hook en 1er si on veut activer l'autofix
# (pour que le formatage de code passe après)
fix = false

[tool.ruff.per-file-ignores]
# autoriser les imports inutilisés dans les fichiers __init__.py (ils exposent des fonct
"__init__.py" = ["E402"]
```

DEMO !



EXPYLLIARMUS !

UN OUTIL POLYGLOTTE - PRE-COMMIT.COM/HOOKS.HTML

-  jshint
- eslint
- tslint
- prettier
- validate-html
- sass-lint
- csslint
- nglint
-  flutter-analyze
- swiftlint
-  maven
- ktlint
- dotnet-format
- gradle-check
-  commitlint
- gitlint
- gitlab-ci-linter
- circle-ci-validator
- terraform_tflint
- ansible-lint
- puppet-lint
- clang-format
- cpplint
- php-lint
- rubocop
- golangci-lint
- go-lint
- shell-lint
- cargo-check
- perltidy
- etc.



METHODOLOGIE



METHODOLOGIE

- discuter les règles de formatage en équipe (communautés de pratiques)



METHODOLOGIE

- discuter les règles de formatage en équipe (communautés de pratiques)
- nuancer les règles *indispensables* et celles *de goût*



METHODOLOGIE

- discuter les règles de formatage en équipe (communautés de pratiques)
- nuancer les règles *indispensables* et celles *de goût*
- adapter les règles au besoin métier (guillemets, indentation de dict / hashmap)



METHODOLOGIE

- discuter les règles de formatage en équipe (communautés de pratiques)
- nuancer les règles *indispensables* et celles *de goût*
- adapter les règles au besoin métier (guillemets, indentation de dict / hashmap)
- tester les règles sur la base de code puis discuter les différences obtenues en équipe



METHODOLOGIE

- discuter les règles de formatage en équipe (communautés de pratiques)
- nuancer les règles *indispensables* et celles *de goût*
- adapter les règles au besoin métier (guillemets, indentation de dict / hashmap)
- tester les règles sur la base de code puis discuter les différences obtenues en équipe
- accepter les compromis faits par le formateur de code : il y aura des cas où le résultat ne sera pas foufou, le but est de tendre vers une homogénéité de la base de code

PRE-COMMIT EN INTEGRATION CONTINUE

Pourquoi des bretelles quand on a la ceinture ?

CONTOURNEMENT DU HOOK

```
git commit --no-verify -m "😎 ahahaha ! 🤘"
```

→ lancer les hooks dans l'intégration continue aussi

ETAPÉ DE LINT DANS L'INTEGRATION CONTINUE //

Pre-commit repose sur git :

- installer git sur le serveur qui lance le job de lint
- initialisation minimale git du projet

```
apt-get install --no-install-recommends -y git
cd my-project
git init .
git add -A
poetry run pre-commit run --all-files
```

ETAPÉ DE L'INTÉGRATION CONTINUE // //

Ou avec Docker :

```
docker run --rm -v $(pwd):/data fxinnovation/pre-commit run -a
```

PRE-COMMIT SUR UN PROJET D'INFRASTRUCTURE-AS-CODE



(voyez avec Jérôme...)

- trailing-whitespace
- end-of-file-fixer
- check-yaml
- check-json
- pretty-format-json

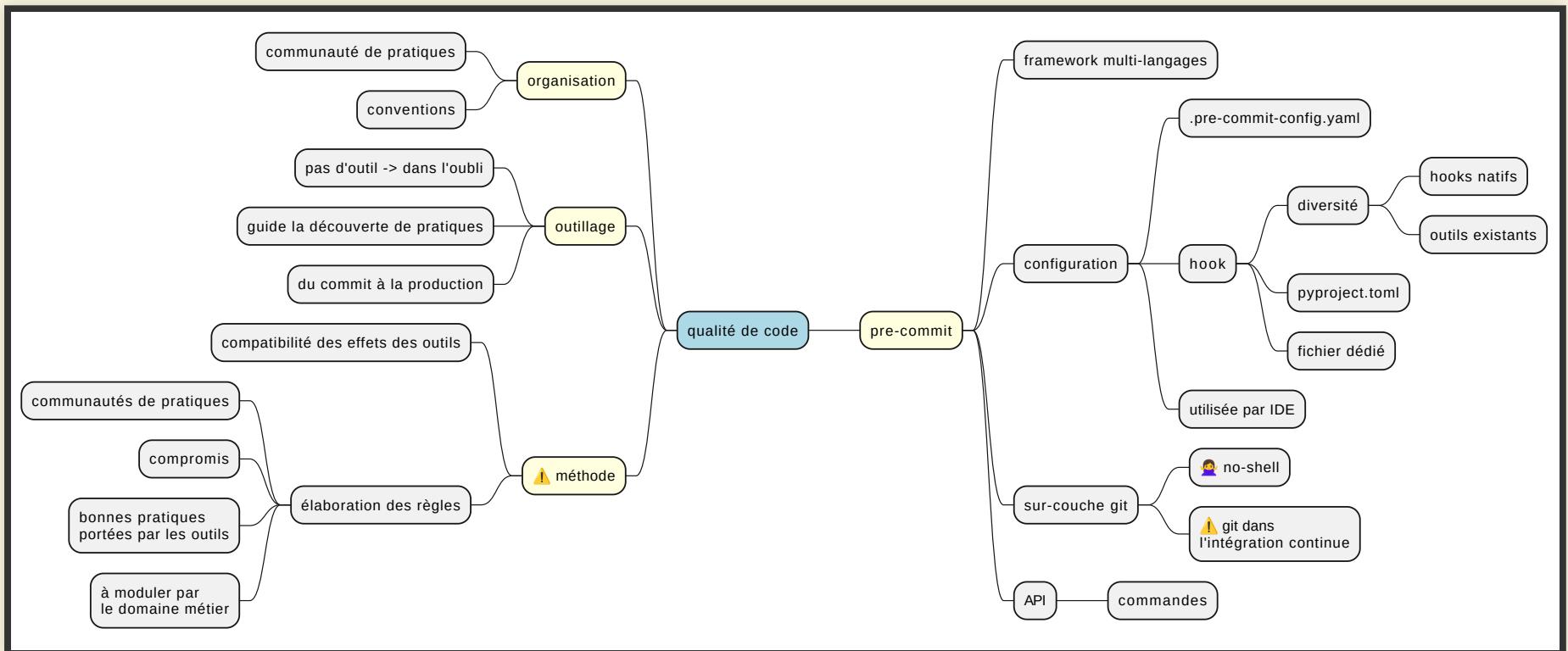
- check-executables-have-shebangs
- check-case-conflict
- detect-aws-credentials
- detect-private-key
- mixed-line-ending
- ...

HOOK POUR TERRAFORM

pre-commit-terraform

```
repos:
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: v4.3.0
  hooks:
    .....
- repo: https://github.com/antonbabenko/pre-commit-terraform
  rev: v1.81.0
  hooks:
    - id: terraform_fmt
    - id: terraform_docs
      args:
        - --args=--config=.terraform-docs.yaml
    - id: terraform_tflint
    - id: terraform_tfsec
      args:
        - >
          --args=--format json
          -e google-iam-no-project-level-service-account-impersonation,google-storage-en
```

EN SYNTHESE



MERCI !

DES QUESTIONS ?

Présentation à retrouver sur github.com/lucsorel/hook-il-est-beau-notre-code.

Rediffusions de sessions précédentes :

- <https://www.youtube.com/watch?v=QoMOZuegwUY> (Python Rennes)
- <https://www.youtube.com/watch?v=V-5ZxVF8ACw> (Rennes DevOps)