Outils et bonnes pratiques de la reproductibilité

Violaine Louvet [1]

[1]CNRS, Laboratoire Jean Kuntzmann

Mini-symposium Reproductibilité, SMAI 2025

Reproductibilité et science ouverte

De quoi a-t-on besoin pour assurer la reproductibilité ?

- l'article, disponible sur une archive
- le code, disponible sur le web (forge, site web, ...)
- le jeu de données, diffusé via un entrepôt de données (Recherche Data Gouv, Zenodo ...)

- Les principes de la science ouverte offre un cadre idéal pour assurer la reproductibilité d'un travail.
- Mais est-ce suffisant ?

La question des licences : ai-je le droit d'utiliser le code ?

Licence : définition par les employeurs des auteurs du droit d'utilisation, de diffusion, de modification du logiciel. La licence fixe un cadre juridique à l'exploitation du logiciel.

Pourquoi utiliser une licence ?

- Pas de licence, pas de chocolat. Personne ne peut utiliser le code : il faut mettre une licence
- Garanties (protection) pour les utilisateurs et les auteurs (exemple : ne pas être responsable juridiquement d'une mauvaise utilisation du code ou d'un dysfonctionnement futur).
- Fixe clairement les règles d'utilisation du code
- Droit de modification du code et re-distribution,
- Obligation ou non de diffusion des sources,
- Citations des auteurs originaux ...

Attention aux briques incluses dans le code !

- Leurs licences peuvent contraindre le choix de la licence du code (licence diffusive)

Les licences libres

Dans le cadre de la reproductibilité et de la science ouverte, on va plutôt s'orienter vers les **licences libres** qui définissent 4 types de libertés pour l'utilisateur :

- **Utilisation sans restriction**, liberté d'exécuter le programme, pour tous les usages.
- **Modification**, liberté d'étudier le fonctionnement du programme, et de l'adapter à vos besoins. Accès au code source condition requise.
- **Copie**, liberté de redistribuer des copies.
- **Redistribution**, liberté d'améliorer le programme et de publier vos améliorations, pour en faire profiter toute la communauté. Accès au code source condition requise.

Il est essentiel d'**identifier clairement les auteurs** en amont du choix d'une licence.

Forge logicielle

La plupart du temps, l'endroit où on va trouver le code. L'outil de base pour :

- Assurer la traçabilité
- Disposer de l'historique complet du code, des modifications, des métadonnées (qui, quand, quoi)
- Permettre de revenir à une version précise pour reproduire des résultats
  - Chaque version de code est associée à un commit ou à une release identifiable
- Disposer d'outils intégrés de gestion des contributions (pull/merge requests)
- Attribuer clairement les modifications (via les commits, branches, reviewers)
- Permettre de discuter du code dans son contexte (issues)

→ Un outil essentiel à toutes les étapes du développement

Préférez les forges institutionnelles aux forges commerciales !

Le choix de la forge logicielle

- Forges de l'ESR
  - De nombreux établissements, organismes voir laboratoires ont déployé leur propre forge (en général gitlab mais il en existe quelques autres)
  - Liste non exhaustive : CNRS, INRIA, Huma-Num, INRAE, UGA, U Bordeaux, ...
  - La plupart de ces forges ont un **accès restreint** à leur communauté / personnels avec un degré d'ouverture plus ou moins important
- Forges commerciales (github, gitlab.com ...)
  - Très utilisées dans le cadre de collaborations internationales ou pour assurer plus de visibilité
  - Pas de contraintes d'accès comparé à la plupart des forges ESR
  - Mais attention aux conditions d'utilisation ! Et à la pérennité de ces plateformes qui sont souvent soumises à des juridictions extra-européennes.
  - Utilisation des sources sans réel contrôle pour l'apprentissage des outils d'aide à l'écriture de code (OpenAI Codex/GitHub copilot, GitLab suggestions, etc.).
  - Flou juridique sur la question du respect des licences pour ces outils.

Rapport du collège logiciels et codes sources du COSO sur les forges

Le README

- **LE point d'entrée du code**, c'est la première chose qu'on voit
- Il doit permettre de comprendre ce que fait le code, comment on l'installe et comment on l'utilise
- Il doit être bien structuré, facile à lire, efficace
- On le lit avant de lire la documentation
- Un bon README renforce la confiance des utilisateurs et attire les futurs contributeurs.

Quelques outils pour aider à faire un bon README :

- readme.so
- makeareadme

L'installation du code, la galère des dépendances

La difficulté d'installation est un frein majeur à l'utilisation.

Ce qu'on veut :

- installer facilement le code
- y compris ses dépendances
- dans leur bonne version
- Maintenant
- Et dans 10 ans
- Sur différents systèmes d'exploitation

L'installation du code, la galère des dépendances

La difficulté d'installation est un frein majeur à l'utilisation.

Ce qu'on veut :

- installer facilement le code
- y compris ses dépendances
- dans leur bonne version
- Maintenant
- Et dans 10 ans
- Sur différents systèmes d'exploitation

Pour cela :

- **Documenter les dépendances et leurs versions** : *requirements.txt*, *environment.yml*, *pyproject.toml*
- Etre précis et clair dans le README
- Utiliser des **environnements** adaptés et portables

L'installation du code, environnements logiciels

Environnements logiciels

- Conteneurs (Docker, Podman)
- Systèmes de paquets (Guix, Nix)
- Paquets python (Conda, ...)

→ L'intégration continue (souvent incluse dans les forges logicielles) permet de tester que l'installation se déroule bien

→ Guix et Nix pousse la reproductibilité à un niveau supérieur, en décrivant non seulement les dépendances logicielles, mais aussi leur construction, leur environnement système, et leurs interactions.

Comment je l'utilise ?

- Le fichier README est la première documentation.
- Il peut / doit être complété par plusieurs modalités de documentation
  - Une documentation utilisateur plus détaillée
  - Des exemples et tutoriaux
  - Une documentation développeur pour faciliter les contributions
  - Les commentaires dans le code permettant de générer automatiquement la documentation technique de la structure du code
  - Certains fichiers comme CHANGELOG, CONTRIBUTING, LICENSE, ... donnent des informations sur les versions, les contributions, les droits ...

Quelques outils

- **Documentation générale** : Markdown, MkDocs, autres outils de génération statique de site
- **Génération automatique** : Sphinx, Doxygen ...
- **Documentation interactive** : Jupyter Notebooks

Quelle version pour cette publi ?

- En général la publication contient le lien sur le **dépôt du code**
- Mais quelle release, quel commit a été utilisé pour cette publication en particulier ?
- Il faut pouvoir citer précisément la version
- Et qu'elle soit accessible sur le long terme

→ Utiliser une archive pérenne plutôt que la forge

Quelle version pour cette publi ?

- En général la publication contient le lien sur le **dépôt du code**
- Mais quelle release, quel commit a été utilisé pour cette publication en particulier ?
- Il faut pouvoir citer précisément la version
- Et qu'elle soit accessible sur le long terme

→ Utiliser une archive pérenne plutôt que la forge

Software Heritage

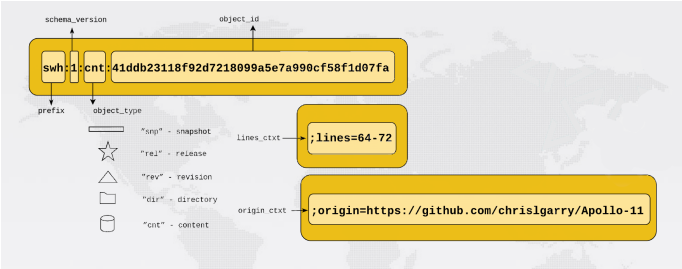
- Initiative dont l'objectif est de construire une archive universelle des codes sources
- En les collectant, les préservant et les partageant sur le long terme
- Lancée en 2016 par INRIA et soutenue par l'UNESCO
- Collecte de l'intégralité des logiciels disponibles publiquement sous forme de code source.
- Depuis des plateformes d'hébergement de codes, comme GitHub, GitLab.com ou Bitbucket, et des archives de paquets, comme Npm ou Pypi ...

Archiver, signaler et citer son code

- Software Heritage est une **archive de codes sources**
- Il ne permet pas de disposer de **métadonnées descriptives du logiciel** : auteurs, affiliation, liens avec des publications, ...
- **HAL** dispose d'un type de dépôt logiciel associé à Software Heritage, ainsi qu'un processus de **modération** qui assure une certaine qualité des métadonnées
- Et propose différents formats d'export pour faciliter la **citation**

Citer son code avec un SWHID

- Le SWHID est un identifiant unique basé sur un hash du code concerné
- Il permet d'identifier précisément **une version, un commit, un fichier et même quelques lignes de code**
- C'est une **norme ISO** formalisée en avril 2025



Example of a Software Heritage identifier (SWHID)

Checklist pour plus de reproductibilité

Checklist pour plus de reproductibilité

- Contrôle de version
  - ✓ Utiliser un système de contrôle de version et une forge logicielle depuis le premier jour
  - ✓ Travailler avec des branches, faire des commits atomiques fréquents avec des messages explicites
  - ✓ Utiliser l'intégration continue (vérification des tests, de l'installation, déploiement de la documentation ...)

Checklist pour plus de reproductibilité

- Contrôle de version
- Documentation
  - ✓ README bien construit et complet
  - ✓ Commentaires dans le code et génération automatique de doc
  - ✓ Si besoin, documentation utilisateur /développeur via site statique

Checklist pour plus de reproductibilité

- Contrôle de version
- Documentation
- Gestion des dépendances
  - ✓ Fichier de dépendances versionnées
  - ✓ Utilisation d'environnement logiciel / conteneur

Checklist pour plus de reproductibilité

- Contrôle de version
- Documentation
- Gestion des dépendances
- Organisation du code
  - ✓ Arborescence claire (*src*, *tests*, *docs* ...)
  - ✓ Règles de programmation cohérentes

Checklist pour plus de reproductibilité

- Contrôle de version
- Documentation
- Gestion des dépendances
- Organisation du code
- Tests
  - ✓ Tests au minimum de non régression
  - ✓ A valider à chaque commit ou changement majeur via l'intégration continue

Checklist pour plus de reproductibilité

- Contrôle de version
- Documentation
- Gestion des dépendances
- Organisation du code
- Tests
- Partage et archivage
  - ✓ Projet public sur une forge (si possible académique)
  - ✓ Présence de la licence
  - ✓ Présence du fichier AUTHORS et des détenteurs des droits patrimoniaux
  - ✓ Archivage sur Software Heritage
  - ✓ Notice sur HAL et citation associée dans le README

Importance des contributions aux logiciels libres

Importance des contributions aux logiciels libres

La science cumulative est un des enjeux majeurs de la science ouverte.
Comme pour les publications qui s'enrichissent mutuellement et continuellement de nouveaux résultats scientifiques, les logiciels libres se développent grâce aux contributions de leurs communautés, qui peuvent être de différents types :

- indiquer un bug, corriger un bug,
- suggérer une nouvelle fonctionnalité, ajouter une nouvelle fonctionnalité,
- poser une question, faire de la documentation, ...

- Il s'agit de faciliter les contributions :
  - Pour soi et pour les autres
  - Pour capitaliser sur le travail réalisé
  - Pour profiter de l'existant (ne pas réinventer la roue)

La checklist précédente est un excellent début pour ça !

Aller plus loin : le Réseau Français de la Recherche Reproductible

- Initiative nationale informelle rassemblant des scientifiques engagés dans l'étude des facteurs favorisant la reproductibilité de la recherche
- Issues de toutes les disciplines et avec des fonctions variées.
- Site web : https://www.recherche-reproductible.fr/
- Liste de diffusion : https://groupes.renater.fr/sympa/info/recherche-reproductible