Gestion de projet informatique

Solal Nathan

April 2023

Pourquoi ?

Quand on fait de la recherche en informatique (et dans pleins d'autres domaines d'ailleurs) on produit souvent du **code**.

On peut s'inpirer des commaunutés open source qui maintienne des grandes code-base pour avoir des bonnes idées.

Visé de la présentation

- Pas un cours de transmission de connaissance ;
- Lancer des pistes de réflection ;
- Découvrir des choses, susciter la curiosité ;
- Partager des idées et des ressources entre nous.

Git

git est un logiciel de **gestion de version** decentralisé.

Pour la petite histoire créer par Linus Torvarld en 2005 (autour du noyau Linux).

```
$ git init
Initialized empty Git repository in /tmp/tmp.IMBYSY7R8Y/.git/
$ cat > README << 'EOF'
> Git is a distributed revision control system.
> EOF
$ git add README
$ git commit
[master (root-commit) e4dcc69] You can edit locally and push
to any remote.
 1 file changed, 1 insertion(+)
 crate mode 100644 README
$ git remote add origin git@github.com:cdown/thats.git
$ git push -u origin master
```

Git : Cheat Sheet

- `git config --global user.name "[name]"`
- `git config --global user.email "[email]"`
- `git init`
- `git clone [url]`
- `git branch [branch]`
- `git checkout [branch]`
- `git merge [branch]` (et stratégie de résolution de merge conflict, voir `rebase`)
- `git add -p` (hunk)

Exemple de Cheat Sheet

Git : Ressource

Oh Shit, Git!?!

Git is hard: screwing up is easy, and figuring out how to fix your mistakes is fucking impossible. Git documentation has this chicken and egg problem where you can't search for how to get yourself out of a mess, *unless you already know the name of the thing you need to know about* in order to fix your problem.

So here are some bad situations I've gotten myself into, and how I eventually got myself out of them *in plain english*.

Oh shit, I did something terribly wrong, please tell me git has a magic time machine!?!

```
git reflog
# you will see a list of every thing you've
# done in git, across all branches!
# each one has an index HEAD@{index}
# find the one before you broke everything
git reset HEAD@{index}
# magic time machine
```

You can use this to get back stuff you accidentally deleted, or just to remove some stuff you tried that broke the repo, or to recover after a bad merge, or just to go back to a time when things actually worked. I use `reflog` A LOT. Mega hat tip to the many many many many many people who suggested adding it!

Oh shit, I committed and immediately realized I need to make one small change!

```
# make your change
git add . # or add individual files
git commit --amend --no-edit
# now your last commit contains that change!
# WARNING: never amend public commits
```

This usually happens to me if I commit, then run tests/linters... and FML, I didn't put a space after an equals sign. You could also make the change as a new commit and then do `rebase -i` in order to squash them both together, but this is about a million times faster.

Warning: You should never amend commits that have been pushed up to a public/shared branch! Only amend commits that only exist in your local copy or you're gonna have a bad time.

Oh shit, I need to change the message on my last commit!

```
git commit --amend
# follow prompts to change the commit message
```

Stupid commit message formatting requirements.

ohshitgit.com

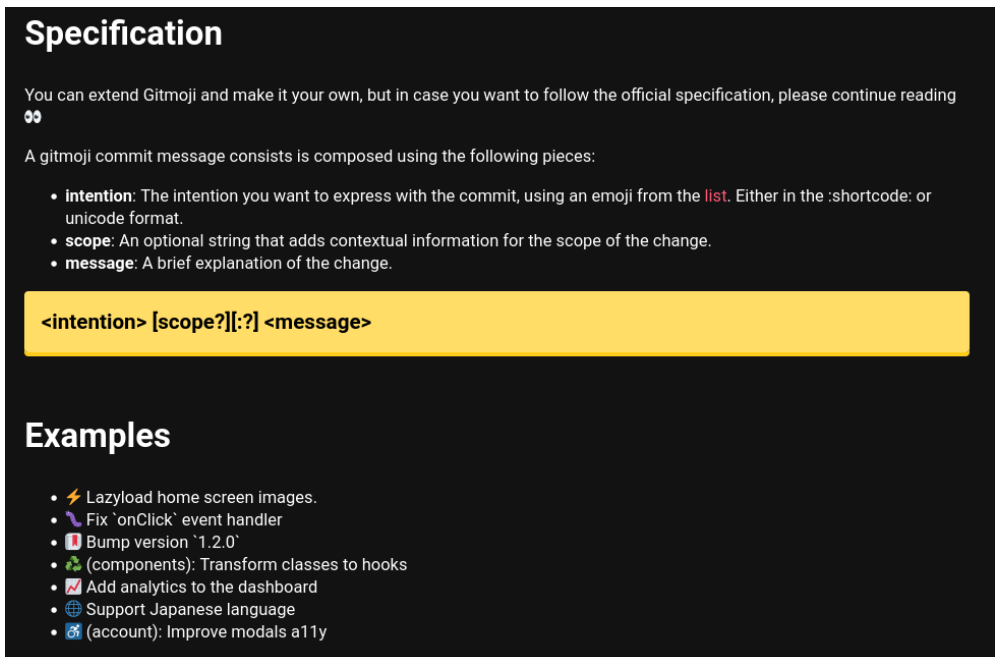
Commit message convention

```
<type>(<scope>): <subject>
```

"type" must be one of the following mentioned below!

- `build`: Build related changes (eg: npm related/ adding external dependencies)
- `chore`: A code change that external user won't see (eg: change to .gitignore file or .prettierrc file)
- `feat`: A new feature
- `fix`: A bug fix
- `docs`: Documentation related changes
- `refactor`: A code that neither fix bug nor adds a feature. (eg: You can use this when there is semantic changes like renaming a variable/ function name)
- `perf`: A code that improves performance
- `style`: A code that is related to styling
- `test`: Adding new test or making changes to existing test

https://conventionalcommits.org



https://gitmoji.dev/

Commit message convention

- Les messages de commit sont fait pour être courts (même si c'est pas limité en pratique, à part à `size_t`);
- Recommendation
  - first line is short (50 characters max)
  - blank line
  - wrapped at 72 then
- And the footer (par exemple utilisé pour dire qu'on ferme une issue `Closes #76`)
- fun fact : `AUTHOR` et `COMMITER` c'est pas la même chose
- vous pouvez signer vos commits avec GPG

Commit message convention

```
fix: fix foo to enable bar

This fixes the broken behavior of the component by doing xyz.

BREAKING CHANGE
Before this fix foo wasn't enabled at all, behavior changes
from <old> to <new>

Closes #76
```

`.gitignore`

Exemple de `.gitignore` minimaliste pour Python.

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Jupyter Notebook
.ipynb_checkpoints
```

Note sur les fichiers volumineux

- binary files, datasets, etc
- `git LFS` peut être une option (à voir sur le serveur le supporte)
- sinon hébergement externe (Nextcloud du LISN avec un lien de partage en téléchargement seul par exemple)

Semantic Versionning

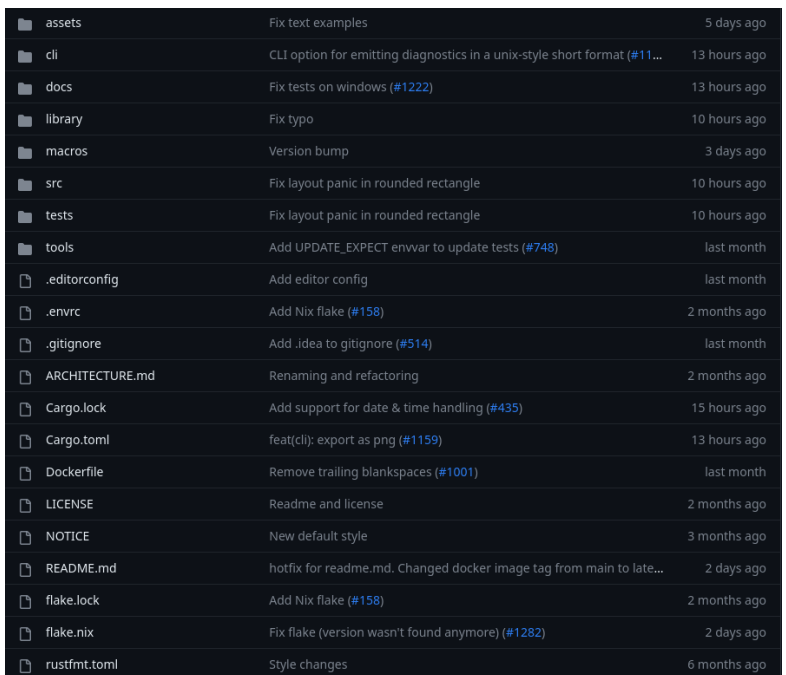
semver.org exemple: `2.0.1` ou `0.9.8-beta`

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes
2. MINOR version when you add functionality in a backward compatible manner
3. PATCH version when you make backward compatible bug fixes

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

Structure de fichier



assets	Fix text examples	5 days ago
cli	CLI option for emitting diagnostics in a unix-style short format (#11...	13 hours ago
docs	Fix tests on windows (#1222)	13 hours ago
library	Fix typo	10 hours ago
macros	Version bump	3 days ago
src	Fix layout panic in rounded rectangle	10 hours ago
tests	Fix layout panic in rounded rectangle	10 hours ago
tools	Add UPDATE_EXPECT envvar to update tests (#748)	last month
.editorconfig	Add editor config	last month
.envrc	Add Nix flake (#158)	2 months ago
.gitignore	Add .idea to gitignore (#514)	last month
ARCHITECTURE.md	Renaming and refactoring	2 months ago
Cargo.lock	Add support for date & time handling (#435)	15 hours ago
Cargo.toml	feat(cli): export as png (#1159)	13 hours ago
Dockerfile	Remove trailing blankspaces (#1001)	last month
LICENSE	Readme and license	2 months ago
NOTICE	New default style	3 months ago
README.md	hotfix for readme.md. Changed docker image tag from main to late...	2 days ago
flake.lock	Add Nix flake (#158)	2 months ago
flake.nix	Fix flake (version wasn't found anymore) (#1282)	2 days ago
rustfmt.toml	Style changes	6 months ago

Il y a quelques structrure de projets classiques

- `src`, `assets`, `static`, `docs` (concept de monorepo, *cf* monorepo.tools)
- `README.md`
- `LICENSE`
- `INSTALL` (souvent inclu dans le `README`)
- `CONTRIBUTING` guidlines for contributing
- `HACKING`, `BUG`, `SECURITY`, `CHANGELOG` (conventional-changelog), checklists, ...

Cookie Cutter Structure

- Cookie Cutter Data Science example

```
├── LICENSE
├── Makefile           <- Makefile with commands like `make data` or `make train`
├── README.md          <- The top-level README for developers using this project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default Sphinx project; see sphinx-doc.org for details
│
├── models             <- Trained and serialized models, model predictions, or model summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number (for ordering),
│                         the creator's initials, and a short `-` delimited description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── references         <- Data dictionaries, manuals, and all other explanatory materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in reporting
│
├── requirements.txt   <- The requirements file for reproducing the analysis environment, e.g.
│                         generated with `pip freeze > requirements.txt`
│
├── setup.py           <- Make this project pip installable with `pip install -e`
├── src                <- Source code for use in this project.
│   ├── __init__.py    <- Makes src a Python module
│   │
│   ├── data           <- Scripts to download or generate data
│   │   └── make_dataset.py
│   │
│   ├── features       <- Scripts to turn raw data into features for modeling
│   │   └── build_features.py
│   │
│   ├── models         <- Scripts to train models and then use trained models to make
│   │   │                 predictions
│   │   ├── predict_model.py
│   │   └── train_model.py
│   │
│   └── visualization  <- Scripts to create exploratory and results oriented visualizations
│       └── visualize.py
│
└── tox.ini            <- tox file with settings for running tox; see tox.readthedocs.io
```

Licensing

choosealicense.com

- GNU AGPLv3
- GNU GPLv3
- GNU LGPLv3
- Apache 2.0
- MIT
- CC BY-SA-NC-ND

Les arcanes des forge logicielles

- issues
- merge requests
- fork
- protected branches

Commentaire et documentation

- écrivez les pour les autres, mais surtout pour vous
- utilisez des outils et formats standard (docstring)
- un site web pour une doc est overkill pour des petits projet mais a du sens pour une bibliothèque

Bonnes pratiques

Il existe pleins de « bonnes pratiques » disponible sur internet.

Exemple en ML : Tips for Publishing Research Code (qui est devenu les guidelines officiels pour NeurIPS)

- Awesome Guidelines

Linting

- tabs vs spaces ? et combien ?

Linting

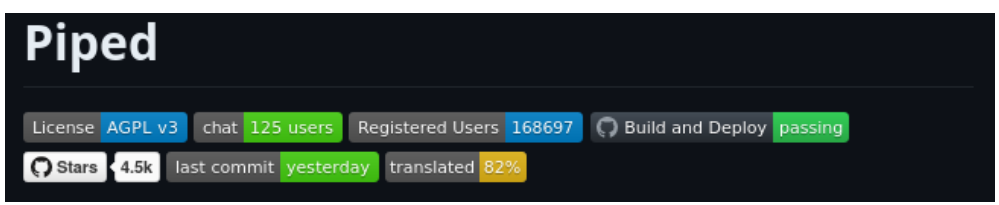
- tabs vs spaces ? et combien ?
- il y a des options de liting pour tout
  - python (pylint, flake8, PYFlakes, Bandit, black, **ruff**, isort, ...)
  - tous les languages : C, caml, JS, Rust, ...
  - bash (shellchecker)
  - html/markdown (prettier, ...)
  - la prose (proselint)
  - les liens morts (linkchecker)
  - ...

Coding styles & docstring

- un style unifié rends le projet plus lisible et facile à maintenir
- peut être inforce par une CI (de façon plus ou moins stricte)
- pre-commit hook
- **PEP8** en Python
- docstring style (Google, Numpy, etc)
  - peut être utilisé par un site static de doc (Sphinx, readthedocs, ...)

Testing

- code quality
- coverage
- shields (*cf* shields.io)



Refactoring

- des règles générales comme : éviter copier-coller du code (boiler plate)
- code smells
- refactoring.guru

CI/CD

- Intégration Continue (CI)
- Livraison Continue (CD)
- CI/CD automatise vos constructions, vos tests et vos déploiements afin que vous puissiez livrer les changements de code plus rapidement et de manière plus fiable.
- DevOps, Docker, automatic testing, automatic artefact generation, ...
- Gitlab-CI

Note sur la sécurité

- utilisez des clé SSH (ed25519 de préférence, sur un token physique en option)
- 2FA c'est bien

Dépendences

- *Standing on the shoulder of giants*
- En avoir trop c'est pas toujours bien (log4j)
- attention au *dependency rot* (pensez à mettre à jour, en plus les nouvelle features sont cools vous allez voir)
- `requirements.txt`
- `virtualenv`
- `docker`

Merci de votre attention