Workflow de développement

Romain Tartière

26 novembre 2019

Table des matières

1 Mise en place initiale

1.1 Pré-requis

- Dépôt de code central ;
- Liste des bugs.

J'utilise habituellement GitLab qui remplis ces besoins et va un peu au delà. Voir l'article « The Joel Test : 12 Steps to Better Code » [1] de Joel Spolsky pour plus de détails.

1.2 Préparation du dépot

La branche par défaut est la branche `master`.

- cette branche est protégée : elle ne peut pas être supprimée, et il n'est pas possible de modifier des commits qui y ont été poussés ;
- cette branche est supposée toujours fonctionner : un développeur qui *casse* `master` est responsable de la réparer dans les plus brefs délais.

[1] https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/

1.3 Préparation du poste d'un développeur

Chaque développeur commence par clone le dépôt du projet dans son répertoire personnel. Le projet contient un fichier `scripts/bootstrap` qui installe automatise la mise en place des dépendances pour le développeur. Ce script est exécuté avec les permissions de l'utilisateur, il est idempotent et ne nécessite pas de privilèges particuliers [2] (notamment, les dépendances sont installées localement, par exemple un projet en Ruby utilise `bundler` avec l'option `--path vendor`).

```
% git clone git@hostname:project.git
% cd project
% ./scripts/bootstrap
```

2 Travail au quotidien

2.1 Début du développement d'une fonctionnalité

Chaque développement se fait dans une branche spécifique (*feature branch*) qui est faite au dessus de la branche `master`. Dès que le nouveau code « fait quelque chose » (entendre par là qu'on n'attends pas que le code soit fini, mais qu'au contraîre on essaye d'aller au plus vite), on le commit et on publie la branche sur le projet.

```
% git checkout -b my-feature-branch
% vim ...
% git add ...
% git commit
% git push -u origin my-feature-branch
```

Après avoir publié le code, on ouvre une demande de fusion pour avoir un espace de discussion avec les autres développeurs pour la revue de code et la discussion des problèmes éventuellements rencontrés.

2.2 Suite du développement d'une fonctionnalité

Les développement des autres développeurs sont intégrés dans `master` au fur et à mesure de leurs complétion. Il est plus facile de gérer d'éventuels conflits si on récupère régulièrement ces changements qui ont été poussés dans la branche principale.

```
% git checkout master
% git pull
% git checkout my-feature-branch
% git rebase master
% ./scripts/bootstrap
% git push -f
```

On peut reprendre le développement où on en était et ajouter des commits normalement.

À ce stade, il vaut mieux éviter d'ammender les commits qui ont été poussés : les pairs pourront ainsi voir les modifications apportées au code depuis leurs dernière revue sans devoir faire une revue intégrale à chaque fois. Une éventuelle réorganisation des commits pourra être faite lorsque le code a été validé et avant qu'il soit mergé (voir ci-dessous).

[2] Dans certains cas (par exemple les bases de données), on considère que le poste de travail a déjà le logiciel installé (système de gestion de base de données) et configuré (roles, bases de données, permissions, etc). Ces opérations sont automatisée par un système de gestion de configuration qui applique un profil adéquat aux machines des développeurs. On se contente donc de mettre en place un fichier de configuration pour que le système fonctionne.

2.3 Fin du développement d'une fonctionnalité

À ce stade, la branche `my-feature-brach` est généralement quelques commits au dessus de `master`. Avant de l'intégrer, il faut peut-être la nettoyer (fusionner des commits, réécrire les messages de commits, etc). Le rebasage interactif est un moyen assez simple de faire cela :

```
% git rebase -i master
[...]
% git push -f
% git checkout master
```

La branche est désormais prête à être mergée dans `master` par un pair, et on peut commencer à travailler sur une nouvelle fonctionnalité.

A Intégration d'un projet externe

Si on projet dépend d'un autre projet qui n'est pas une bibliothèque sur laquelle on peut si linker, il faut intégrer le code de ce dernier au dépôt. Les sous-modules git permettent de faire cela.

```
% git submodule add https://path/to/remote/project.git remote-project
```

Il faudra modifier le fichier de bootstrap (`scripts/bootstrap`) pour lancer les commandes suivantes :

```
git submodule init
git submodule update
```