Manuel du responsable de version

La gestion de version avec le workflow GitHub

1. Introduction

Un workflow, flux opérationnel, définit un plan d'organisation de la collaboration de plusieurs développeurs pour la production de logiciels. La version présentée ici est le *GitHub Flow* qui est une version simplifiée du *Git-Flow*. Ce flux opérationnel est adapté au développement de projets sans réelle gestion de distributions, tels que les projets étudiants. La collaboration se fait par l'intermédiaire d'un dépôt centralisé que partagent tous les développeurs et de dépôts locaux propres à chaque développeur.

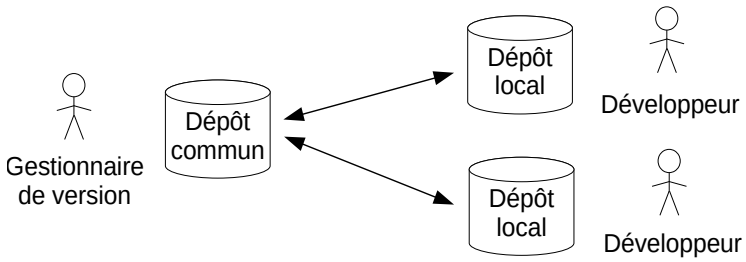

Figure 1: Architecture de collaboration Git.

Dans la description qui suit, le dépôt central du projet est supposé localisé à l'adresse https://gitlab.ecole.ensicaen.fr. L'IDE est supposé être IntelliJ IDEA.

2. Organisation du dépôt central

Le développement d'un projet s'organise typiquement autour d'une branche permanente de production nommée `master`, commune à tous les développeurs, et de plusieurs branches temporaires de travail, personnelles à chaque développeur. La Figure 2 donne un exemple d'une telle organisation avec deux branches temporaires.

2.1 Une branche permanente de production : *master*

La branche `master` contient la version courante du logiciel. C'est la branche de fusion des contributions des développeurs. Seul le responsable du dépôt peut modifier cette branche.

La version du logiciel dans `master` doit toujours être fonctionnelle. La branche utilise l'intégration continue pour assurer que tout ce qui est ajouté à cette branche ne remet pas en cause le caractère fonctionnel du logiciel.

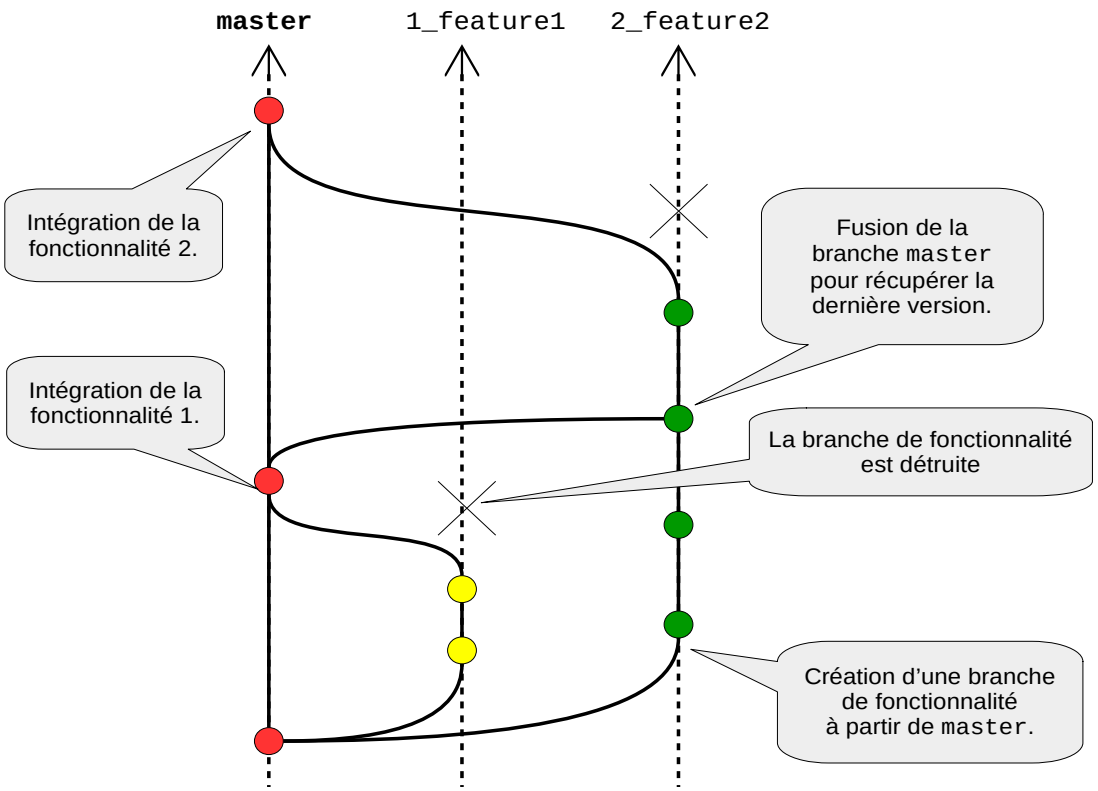

Figure 2 : Organisation des branches sur le dépôt central.

2.2 Des branches de développement temporaires

À chaque fois qu'un développeur veut modifier la version courante du logiciel dans `master` pour ajouter une fonctionnalité ou corriger un bug, il doit créer une branche de travail spécifique qui émane de `master`.

Quand la fonctionnalité est complète, la branche est poussée sur le dépôt central et le responsable du dépôt la fusionne à la branche `master` courante, puis la branche de fonctionnalité est détruite. Cette branche réside essentiellement sur le dépôt local du développeur jusqu'à la fin du développement où elle est alors poussée sur le dépôt commun pour être fusionnée à la branche `master` puis détruite. Elle peut toutefois être poussée régulièrement sur le dépôt central pour assurer des sauvegardes intermédiaires ou échanger avec les autres développeurs.

3. Le workflow

Examinons plus précisément comment une petite équipe formée de trois développeurs, Alpha, Bravo et Charlie collaborent selon le GitHub Flow via un dépôt centralisé sur GitLab.

Alpha est responsable du dépôt centralisé.

3.1 Initialisation du dépôt

3.1.1 Alpha initialise le dépôt central sur GitLab

Alpha crée le dépôt central sur *gitlab.ecole.ensicaen.fr*. Alpha réalise ces opérations par l'intermédiaire de l'interface Web de GitLab. Elle ajoute Charlie et Bravo comme développeurs.

3.1.2 Tous les développeurs clonent le dépôt central

Les trois développeurs créent leur dépôt local par copie du dépôt central en utilisant l'interface d'IntelliJ. Chacun est prêt à collaborer au développement du logiciel.

3.2 Développement d'une fonctionnalité

Charlie commence son travail de développement d'une fonctionnalité. Le diagramme d'activité en Figure 3 résume les activités de développement d'une fonctionnalité.

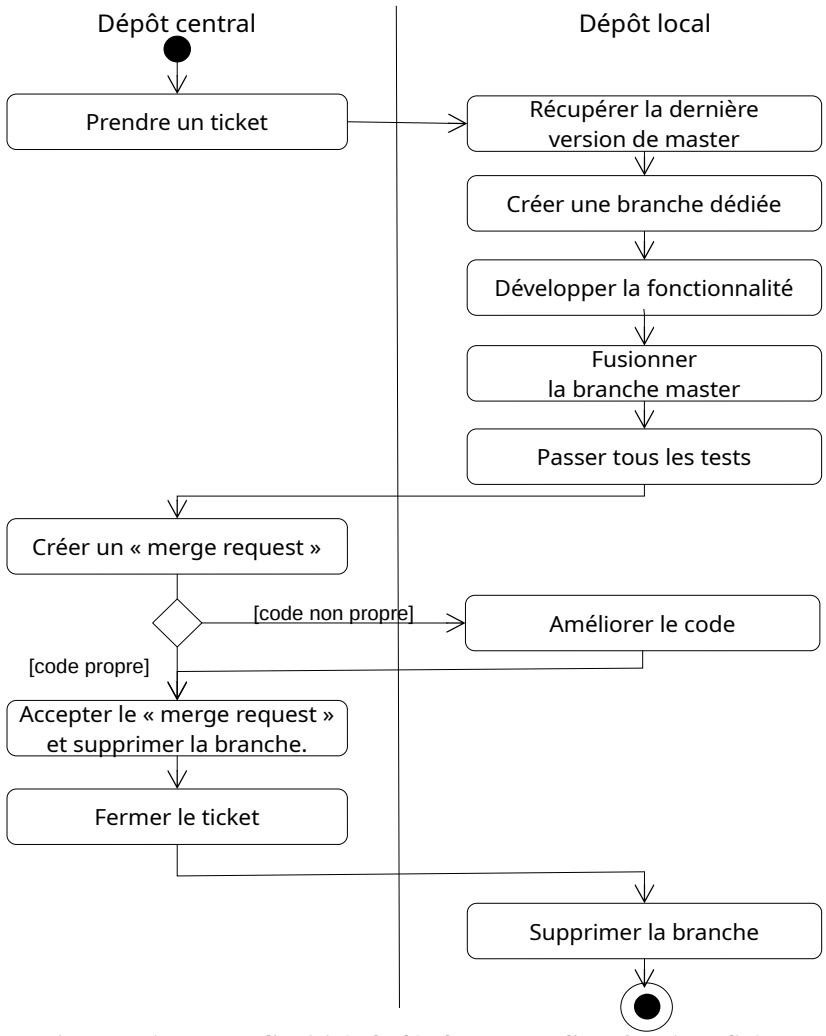

Figure 3. Diagramme d'activités de développement d'une fonctionnalité.

3.2.1 Charlie prend un ticket dans GitLab

Il faut toujours qu'il y ait un ticket (*issue* en anglais) correspondant à tout

changement dans le code. La plupart des tickets sont créés au début d'un sprint à partir du backlog. D'autres tickets sont créés à la volée par remontée de bug ou identification de sous-tâches apparues au cours du développement.

Le ticket doit porter un nom très explicite, par exemple : « *Query database* » pour une fonctionnalité ou « *Bug on signal processing* » pour un bug.

Pour commencer son travail, Charlie s'assigne le ticket en utilisant l'interface de GitLab qu'il place dans la travée IN PROGRESS.

3.2.2 Charlie crée sa branche de travail

Charlie crée une branche pour ce ticket à partir de la branche `master`. La branche est d'abord une copie de la version courante de `master` du dépôt. Le nom de la branche devrait être construit avec le numéro et le nom du ticket, par exemple « *15_query_database* ».

3.2.3 Charlie développe sa fonctionnalité

Sur la branche `15_query_database`, Charlie édite, crée, modifie et supprime des fichiers du projet avec son IDE pour ajouter sa fonctionnalité. En suivant le principe des petits pas, il fait autant de *commits* que nécessaire pour garder trace de ses différentes modifications et pouvoir revenir en arrière en cas de nécessité. Cela peut représenter plusieurs *commits* par heure, en fait à chaque fois qu'une étape est franchie dans le développement de la fonctionnalité.

3.3 Publication d'une fonctionnalité

Charlie termine sa fonctionnalité et souhaite l'intégrer au dépôt central.

3.3.1 Charlie synchronise sa branche avec la branche *master*

Avant d'envoyer son travail pour intégration au projet, Charlie doit synchroniser sa branche avec la branche `master` du dépôt central qui a pu évoluer avec les contributions de Bravo depuis qu'il en a dérivé. Cela nécessite de résoudre tous les conflits entre les deux versions. L'objectif est de ne pas laisser ce travail à Alpha qui n'a pas la connaissance sur les modifications apportées par Charlie.

3.3.2 Charlie repasse tous les tests

Il est nécessaire de repasser tous les tests pour s'assurer que la fusion n'a pas introduit de régression.

3.3.3 Charlie fait une demande d'intégration de son travail

Une fois la résolution des conflits effectuée et les tests passés, Charlie pousse sa branche sur le dépôt central. Puis, il remplit un « *merge request* » dans GitLab pour demander à Alpha de fusionner sa branche à `master`.

3.3.4 Alpha reçoit le « merge request » de Charlie et fait une revue de code

Quand Alpha reçoit la demande d'intégration, elle contrôle la qualité de la contribution, notamment le respect des conventions de codage et la présence des tests. Elle peut décider de modifier ou de faire modifier la contribution par Charlie avant l'intégration dans le projet officiel.

3.3.5 Charlie termine le travail de révision de sa fonctionnalité

Une fois que la contribution de Charlie est jugée acceptable par Alpha, Alpha fusionne la contribution sur la branche `master` en utilisant simplement le bouton "Merge" de l'interface de GitLab.

De son côté, Charlie déplace son ticket (issue #15) sur le tableau Kanban, le faisant passer de la travée IN PROGRESS à la travée CLOSED. Une autre solution est d'ajouter dans le message associé au « *merge request* », le mot « *close #15* » pour signifier que cela conclut sa branche et ferme automatiquement le ticket correspondant.