GIT pour H4213

Introduction

GIT est un logiciel de gestion de versions créé par le même développeur de Linux. Nous allons nous servir de GIT par l'intermédiaire du site Github, qui héberge notre projet, et du terminal, pour les commandes (pour ceux qui sont sous Linux), ou du logiciel Github pour Windows.

L'idée de ce tutoriel a été mis en place pour expliquer un peu comment ça marche GIT et montrer une méthode a suivre pour qu'on ait un répertoire propre et qu'on puisse bien suivre le travail de tous.

Je vous invite à lire la suite pour mieux comprendre le fonctionnement de cette méthode.

Glossaire

- **Répertoire Local:** Le répertoire local est où on garde notre projet (nos fichiers) sur notre ordinateur. Dans notre répertoire local on peut avoir des branches locales (pas forcement les mêmes que celles du *répertoire distant*.

- **Répertoire Distant:** Le répertoire distant est où notre projet est gardé et où on en a accès. Pour le récupérer le projet dans ton répertoire locale ça nous suffit de faire un clone du répertoire distant.

- **Branche:** Une branche de développement est une copie du répertoire originel où on peut travailler (ajouter des fonctionnalités, par exemple) sans modifier le code originel, donc sans le danger de créer des bugs sur le code originel.

- **Branche Master:** Branche principale où on ne fait *merger* que du code fonctionnel, une fois qu'on l'a déjà codé sur une autre branche et qu'on l'ai vérifié. C'est dans cette branche qu'on garde le code qui sera livré au client.

- **Branche Developpement:** Branche qui garde le code qui est en cours de développement, mais on ne code **jamais** sur cette branche, on *merge* les branches de développement de fonctionnalités une fois qu'elles soient vérifiées et, potentiellement, sans bug.

.

- **Commit:** On commit a chaque fois qu'on ait fini avec une partie importante d'une fonctionnalité (une classe, une architecture, une méthode compliquée). Les commits nous permettent de garder l'avancement du projet et de chaque fonctionnalité.

- **Merge:** La commande merge unit une branche désirée à la branche actuelle (normalement la branche de développement de la fonctionnalité à la branche “développement”).

- **Pull:** La commande pull nous permet de récupérer les fichier d’une certaine branche. Il sert aussi a faire la mise à jour des branches de ton répertoire local.

- **Push:** La commande push permet de mettre à jour le répertoire distant à jour par rapport à ton répertoire local. **/!\ Toujours faire un pull avant de faire un push /!**

Méthode Gitflow

La Méthode Gitflow est une méthode de développement collaboratif avec laquelle on arrive à ajouter des fonctionnalités, les tester et les debugger sans avoir un impact dans le code principal, en le laissant toujours stable. Pour cela il nous suffit de suivre la méthode décrite dans la suite.

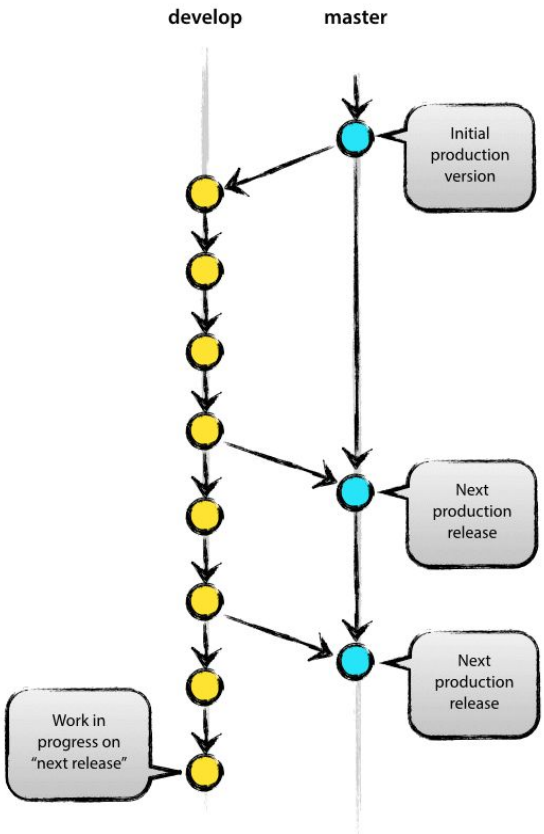
1) Création de branches master et développement (distante et locale)

En créant un répertoire git la branche master est automatiquement créée. Donc l’idée c’est de créer un branche développement pour ne pas avoir besoin de toucher la branche master (on va juste lui rajouter du code une fois qu’on a fini une “version” du projet). Comme indique le dessin à coté.

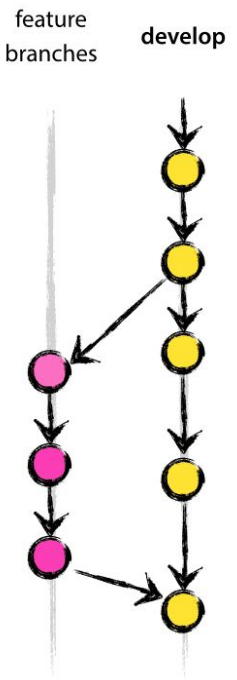
- (a) git branch developpement
- (b) git push -u origin developpement

*Cette phase est faite juste par celui que crée le répertoire et une seule fois.

(a) - création de la branche “developpement” dans le répertoire local.
(b) - création de la branche “developpement” dans le répertoire distant (“-u” c’est pour lier la branche développement locale a celle qu’on est en train de créer).



2) Création de branche de fonctionnalité (distante et locale)



Cette partie concerne à tous les participants de l'équipe qui sont impliqués dans le développement de nouvelles fonctionnalités. On va créer une "developpement" locale et une branche par fonctionnalité, comme ça on pourra la développer sans toucher a la branche principale "developpement".

- (a) git clone http://github.URLduRepertoireDistant
- (b) git checkout -b developpement origin/developpement
- (c) git checkout -b dev-fonctionnalite developpement

(a) On clone le repertoire distant localement
(b) On crée une branche "developpement" locale en se basant sur la branche "developpement" distante et on se place à cette branche.
(c) On crée une branche "dev-fonctionnalite" qui se base sur la branche "developpement" locale et on se place à cette branche.

* On ne codera que sur la branche "dev-fonctionnalite"
** On répetira la création d'une branche fonctionnalité a chaque nouvelle fonctionnalité.

3) Commits

Passons au moment d'enregistrer une avance dans notre fonctionnalité (une classe, quelques méthodes, un bug corrigé, etc…). On fait un commit a chaque phase réussite dans le processus de coder la fonctionnalité. Les commits ils servent à enregistrer l'avancement mais aussi à avoir un "historique" de notre avancement. Donc hésitez pas à commiter dès que vous pouvez.

- (a) git status
- (b) git add un_fichier (ou . , ou *, ou --all .)
- (c) git commit -m "message"

(a) On vérifie l'état actuel de ta branche, sur quel branche on est et son avancement par rapport à la branche distante correspondante.
(b) On ajoute/supprime tous les fichiers qui ont été créé, supprimé ou modifié.
(c) On fait un commit avec la message "message"

4) **Merge vers la branche développement**

Une fois qu'on a fini notre fonctionnalité et qu'elle ait été vérifié, on va la "ajouter" à la branche developpement.

- (a) git pull origin developpement
- (b) git checkout developpement
- (c) git merge "dev-fonctionnalite"
- (d) git push origin developpement
- (e) git branch -d dev-fonctionnalité

(a) On met a jour notre branche developpement locale avec la branche developpement distante

(b) On se déplace vers notre branche developpement locale

(c) On merge la branche "dev-fonctionnalite" locale dans la branche developpement locale

(d) On met la branche developpement distante à jour avec notre branche locale actuelle (developpement).

(e) On supprime notre branche locale "dev-fonctionnalité". En faisant cela il faudra penser à récréer une nouvelle branche lors du developpement d'une nouvelle fonctionnalité.

Quelques Commandes utiles

- git pull
- git push
- git checkout nom-de-la-branche
  - changement de branche
  - -b : création d'une nouvelle branche et changement de branche
- git branche
  - liste toutes les branches locales
  - -r : liste toutes les branches distantes
  - -a (ou --all) : liste toutes les branches locales et distantes
  - nom-de-la-branche: création d'une nouvelle branche sans déplacement
  - -d nom-de-la-branche: suppression de branche
- git merge nom-de-la-branche
- git add
- git status
- git commit
- git clone
- git mergetool

***Deux sites qui ont bien aidé:**

- http://rogerdudler.github.io/git-guide/
- https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow