Une intro à GIT dans un recto-verso !

Laurent FACQ - laurent.facq@math.u-bordeaux.fr - IMB/CNRS

Mars 2022 - v0.1

1 Introduction

GIT permet de garder l'historique intégral des versions de votre projet. L'objectif de ce recto-verso est de vous donner les éléments vitaux vous permettant d'utiliser GIT et de progresser sans remettre en question ce que vous aurez appris ici ! Nous ne verrons qu'une seule manière de faire, ce qui n'est déjà pas si mal !

Chaque version sauvegardée dans GIT est appelée un ***commit***. Un ***commit*** contient *l'ensemble des fichiers et répertoires* correspondants à cette version du projet[1]. Pour chaque commit, GIT calcul un ***hachage/hash*** prenant en compte l'ensemble des informations contenues dans ce ***commit***. Ce ***hash*** sert d'identifiant pour ce commit[2]. Les commits sont **chaînés** les uns avec les autres : chaque commit pointe sur le ou les commit(s) parent(s) dont il est issu.

Le plus simple pour créer un projet GIT est de commencer par créer le projet sur son gitlab/github préféré, puis de cloner ce projet dans une nouveau répertoire au nom du projet[3](i.e. recopier l'intégralité du projet avec toutes ses versions et extraire la "dernière version principale" pour initialiser la zone de travail et la zone de transit - voir ci-dessous) sur votre poste de travail avec la commande **git clone *URL-DU-DEPOT-DISTANT***. Privilégiez les dépôts avec accès par SSH (avec clef SSH) plutôt que HTTPS (avec mot de passe ou token).

Chaque dépôt (local ou distant) contient *l'ensemble* des versions/commits du projet.

2 Comment utiliser les 3 zones locales - zone de travail, zone de transit, dépôt local - et le(s) dépôt(s) distant(s) ?

- la zone de travail : la seule zone naturellement visible. C'est le répertoire dans lequel vous éditez vos fichiers, compilez, etc. Ce répertoire contient par défaut les fichiers et répertoires dans l'état correspondant à "la dernière version/commit" de votre projet. Les fichiers temporaires qui s'y trouvent, issus de compilation par exemple (.o .dvi), n'ont pas nécessité à être sauvegardés dans GIT. Ils seront simplement considérés comme "non suivi" par git[4].

- la zone de transit (*staging area ou index*) : c'est la zone de préparation de la prochaine version/commit, qui pourra - si on le souhaite - être enregistrée dans le dépôt local. Cette zone n'est pas visible directement, mais les différences avec la zone de travail sont affichables avec la commande **git status** (qui montre les différences en terme de fichiers - 3 états possibles : 1-modifiés et sera pris en compte dans la prochaine version, 2-modifiés mais ne sera pas pris en compte pour la prochaine version, 3-non suivi par GIT) ou **git diff** (qui montre les différences au niveau des contenus des fichiers). La zone de transit contient par défaut les fichiers et répertoires dans l'état correspondant à "la dernière version" de votre projet. Vous pouvez à tout moment y recopier, avec **git add *fichiers*** - depuis la zone de travail - de nouveaux fichiers ou de nouvelles versions de fichiers que vous avez modifiés et que vous prévoyez de prendre en compte dans le prochain commit enregistré.

- le dépôt local : c'est un dépôt GIT associé à votre copie local du projet. Comme tout dépôt GIT, il contient l'ensemble des commits, déjà créés, de votre projet. **git log** vous montre l'historique des commits contenus, avec pour chacun sont identifiant/hash.
  Pour enregistrer, dans ce dépôt local, un nouveau commit contenant *l'ensemble des fichiers et répertoires contenus dans la zone de transit*, on utilise la commande **git commit -m "*mon commentaire* "**

[1] en réalité bien sûr, dans notre dos, GIT minimise/optimise l'espace utilisé par le projet
[2] Il peut être abrévié, quand on veut faire référence à un commit, en ne gardant que les 4 premiers caractères (si pas d'ambiguïté)
[3] le nom du projet correspond au dernier morceau de l'URL en supprimant l'éventuel ".git" final
[4] GIT les signalera sauf si on utilise le mécanisme **git ignore *pattern***

- le dépôt distant : généralement hébergé sur un gitlab/github. Avec **git push** vous envoyez (synchronisez) le contenu de votre dépôt local vers le dépôt distant. Avec **git pull** vous récupérez (synchronisez) les modifications du dépôt distant vers votre dépôt local.
  Si le dépôt distant a changé depuis votre dernier **git pull**, vous devez refaire un **git pull** avant de pouvoir faire un **git push**.
  Si quelqu'un a fait des modifications dans les mêmes zones que vous, **git pull** va signaler un conflit que vous devez résoudre en 1-éditant les fichiers en conflit[5] et 2-en faisant un **git add** sur chacun de ces fichiers, puis **git commit -m "mon commentaire"**.

3 Et les branches ?

Les branches sont une notion un peu plus avancée, on peut commencer sans dans une première approche, qui permet de créer/faire cohabiter (et travailler sur) plusieurs "variantes" de votre projet. Cela permet de tester des variantes ou de travailler à plusieurs, chacun dans sa branche, sachant que l'on peut ensuite facilement reporter les modifications d'une branche sur une autre.
La branche par défaut s'appelle généralement **master** ou **main**.
git branch -a vous affiche toutes les branches existantes[6], dans votre dépôt, avec une "*" signalant **la branche courante**.
Une branche est juste un pointeur sur un commit[7] - rien de plus ! - et ce "pointeur de branche" avance automatiquement sur le commit suivant quand vous *commitez* une nouvelle version, à la différence des tags - créés avec (**git tag *nom-du-tag-a-creer*** - qui sont des pointeurs sur des commits mais qui ne bougent pas tous seuls.

À part **git clone**, qui prend en compte toutes les branches, toute les commandes GIT vues jusqu'ici ne travaillent que sur la branche courante. **git checkout -b *NOM-NOUVELLE-BRANCHE*** permet de créer une nouvelle branche et d'en faire la branche courante.
git checkout NOM-DE-BRANCHE permet de se placer sur une branche (en faire la branche courante) ce qui s'accompagne de la restauration, dans les zones de travail et de transit, de tous les fichiers et répertoires suivis par git dans leur état correspondant au (dernier) commit pointé par cette branche.

On peut "fusionner"[8] une branche dans une autre, afin de fusionner les modifications issues de deux branches. Généralement on créé ainsi un nouveau commit "de fusion"[9]. Pour cela on se place sur la branche receveuse/destination (la seule qui sera modifiée) avec un **git checkout BRANCHE-DESTINATION**. Puis on "fusionne" dedans la branche source avec un **git merge BRANCHE-SOURCE**. Ici aussi il peut y avoir des conflits... que l'on résout comme vu précédemment !

4 Pour finir

Toujours garder présent à l'esprit que :

- Apprendre à utiliser GIT prend du temps...
- Apprendre à bien utiliser GIT prend du temps...
- A tout moment, notamment lorsque vous êtes perdus : la commande git status vous indique dans quel état vous êtes et vous suggère des actions pertinentes.
  item Il ne faut pas hésiter à demander de l'aide à un collègue de proximité (ou à l'auteur) qui a de forte chance d'être compatissant car tout le monde sait que git n'est pas simple quand on débute !

Bon GIT :-) !

[5] GIT vous signale les zones de conflit avec les marqueurs "<<<<<<" "------" ">>>>>>" et vous montre les deux versions à "réconcilier". A vous de jouer ! Généralement on édite le fichier pour au moins faire disparaître ces marqueurs et garder les bonnes versions, ou mixer les versions de manière cohérente

[6] locales et distantes

[7] on pourrait directement dire : qui pointe sur une chaîne de commits, puisqu'ils sont chaînés entre eux)

[8] le terme fusion est impropre car a l'issu d'une fusion on a toujours deux branches et la branche source - ou donneuse - reste inchangée.

[9] particularité : ce commit a deux commits parents