Notes sur GIT

Récupérer un repository distant

git clone <url>

Voir les branches

git branch

Se positionner dans une branche

git checkout <nomBranche>

Voir les modifications en cours

git status

Valider (committer) localement des modifications

Valider toutes les modifications :

git commit -a -m '<mon message de commits>'

Valider certaines modifications :

git commit -a -m '<mon message de commits>' <fichier1> <fichier2> ...

Envoyer les modifications sur la branche distante

git push

Si on vient juste de créer la branche locale (ici, généralement, doit être remplacé par le nom de la branche locale pour avoir des noms identiques entre branche locale et branche distante) :

git push -u origin <nomBranche>

ou, de manière identique mais plus claire :

git push --set-upstream origin <nomBranche>

Envoyer toutes les modifications de toutes les branches locales vers les branches distantes correspondantes :

git push --all

Récupérer les modifications provenant de la branche distante

git pull

Créer une nouvelle branche

Le processus est le suivant :

d'abord se positionner sur la branche depuis laquelle on souhaite diverger

ensuite, créer la nouvelle branche

faire nos modifications

valider nos modifications locales

envoyer sur le serveur distant les modifications locales

git checkout <nomBrancheSource>

git checkout -b <nomNouvelleBranche>

faire les modifications... puis les valider localement :

git commit -a -m '<mon message de commits>'

git push --set-upstream origin <nomNouvelleBranche>

Annuler des commits locaux

Là, on se trouve dans le cas où l'on a fait un ou plusieurs commits sur la branche locale mais que ces commits n'ont pas encore été poussés vers le repo distant.

Deux approches possibles :

je veux supprimer toute information, toute modification de ce commit

je veux supprimer l'information de commit mais garder les modifications faites à mon code.

Le premier cas est "non mais là, j'ai merdé en fait. Tout ce que j'ai fait est à jeter. Heureusement, je n'ai rien poussé sur le repository principal, les collègues se moqueraient de moi". L'idée est donc de supprimer toutes les modifications et revenir au code tel qu'il était avant ma modification.

git reset --hard HEAD^1

Le deuxième cas est "ha zut, je me suis trompé, j'aurais du ajouter/retirer tel fichier à mon

commit” ou “ha zut, j’ai oublié de modifier tel fichier”.

git reset --mixed HEAD^1

Là, vous pouvez ensuite continuer vos modifications et validez votre commit.

Vous pouvez annuler plusieurs commits consécutifs mais toujours si vous ne les avez pas poussés sur le repo distant. Pour cela, identifier le numéro de commit précédent vers lequel revenir à l’aide de la commande git log puis l’indiquer. Exemple :

cedric@portable:~/GIT/blog.cedrictemple.net-test-git$ git log

commit 696625ea73bbb2e39400d9ed5ccb19b04b306da6 (HEAD -> master)

Author: Cedric Temple <email>

Date: Mon Dec 30 17:10:28 2019 +0100

  test 2

commit 4ca8357b99af667cc8c31ec517e4ae01e5d405b0

Author: Cedric Temple <email>

Date: Mon Dec 30 17:10:09 2019 +0100

  test 1

commit d248572d571d1746b808bf762909c623b77a6c74 (origin/master)

Author: Cedric Temple <email>

Date: Wed Dec 25 12:27:22 2019 +0100

  Ajout de /etc/resolv.conf

commit 0a16f9172c038be6f8867b02257a581342058c9c

Author: Cedric Temple <email>

Date: Mon Dec 23 18:01:46 2019 +0100

...

...

...

cedric@portable:~/GIT/blog.cedrictemple.net-test-git$ **git reset --mixed d248572d571d1746b808bf762909c623b77a6c74**

Neutraliser un commit distant

Il n'est pas possible de supprimer un commit distant : en effet, rien ne dit que d'autres ne l'ont pas déjà récupéré, si on le supprime, les autres seront très perturbés. De ce fait, on va neutraliser un commit : on va créer un commit qui inverse un autre commit.

Pour cela, on utilise la commande git revert en passant en argument le numéro de commit que l'on veut neutraliser. Dès lors, GIT va automatiquement créer un nouveau commit inversant le commit précédent.

git revert ID_COMMIT

exemple :

git revert 4ca8357b99af667cc8c31ec517e4ae01e5d405b0

Dès lors, une fois le commit poussé, l'historique GIT reflétera exactement l'histoire :

- un premier commit X a été fait

- d'autres commits ont (peut-être) été faits

- un autre commit X' a été fait pour annuler le premier commit X

Ce n'est pas (totalement) magique : si d'autres commits ont modifié les lignes du commit que l'on veut neutraliser, un conflit va se créer. Il faut alors le résoudre manuellement.

Supprimer une branche locale uniquement

Si l'on veut effacer une branche locale uniquement, le processus est la suivant :

d’abord, se positionner sur une autre branche

ensuite, effacer la branche locale

git checkout master

git branch -D todelete

Supprimer une branche

Ici, on veut supprimer la branche nommée “todelete”. Le processus est le suivant :

d’abord, se positionner sur une autre branche

ensuite, effacer la branche locale

enfin, pousser la référence vide (indiquée ici par le caractère ‘:’) sur le serveur.

git checkout master

git branch -D todelete

git push origin :todelete

Nettoyer un repository

Il est possible de nettoyer son repository pour virer toutes les données devenues inutiles. Exemple ici, avec un tout petit repository :

git gc

Décompte des objets: 28512, fait.

Delta compression using up to 8 threads.

Compression des objets: 100% (8548/8548), fait.

Écriture des objets: 100% (28512/28512), fait.

Total 28512 (delta 20391), reused 27529 (delta 19639)

Créer un tag

Un tag est une étiquette pour marque un commit. Généralement, on taggue lorsqu’on crée

une version de son logiciel, de sa documentation, …

git tag <nomDuTag> -am 'message pour les humains'

 # exemple :

git tag v1.1 -am 'ma toute nouvelle version v1.1 qui contient ...'

 # puis, on doit pousser le tag sur le serveur :

git push --tags

 # puis plus tard

git tag v1.2 -am 'ma toute nouvelle version v1.2 qui contient ...'

git push --tags

Lister les tags

git tag --list

Se positionner sur un tag

Pour retrouver le code que l'on a taggué, c'est simple :

 # pour identifier le tag

git tag --list

 # pour se positionner sur le tag

git checkout <nomTag>

 # exemple :

git checkout v1.0

Récupération de commits d'une autre branche

Lorsqu'on développe, on développe parfois en parallèle plusieurs versions. Donc on développe sur plusieurs branches en même temps. Cependant, on aimerait bien qu'une modification donnée soit reportée d'une branche à l'autre. Avec GIT, c'est faisable simplement.

Imaginons le cas suivant pour l'exemple : j'ai deux branches 2.3.8-develop et 2.4.0-develop. J'ai fait un correctif sur la branche 2.3.8-develop que j'aimerai bien appliqué sur 2.4.0-

develop. Le processus est le suivant :

se positionner sur la branche 2.3.8-develop

identifier la modification que l'on souhaite pousser sur la branche 2.4.0-develop grâce à git log

se positionner sur la branche 2.4.0-develop

récupérer le commit avec **git cherry-pick**

git checkout 2.3.8-develop

git log

j'identifie mon commit en utilisant le hash SHA, par exemple : 84e2a37b46e604189a8bc18eb942775cb5a52987

git checkout 2.4.0-develop

git cherry-pick 84e2a37b46e604189a8bc18eb942775cb5a52987

j'ai récupéré le commit, je peux le pousser sur le serveur distant :

git push

Bien entendu, cela fonctionne automagiquement lorsqu'il n'y a pas de conflit. S'il y a des conflits, il faut les résoudre manuellement.