2021

Utiliser git

Utilisation de git

Olivier D.

Table des matières

1 Débuter avec GIT 3
2 Petit schéma 4
3 Résoudre les conflits 5
4 Restaurer / récupérer son code 6
5 Créer et fusionner des branches 7
6 Ajouter un repo distant et travailler à plusieurs 8
7 Git dans VSCode 10
8 Déploiement continu 11
9 Créer une pull / merge request 12

1 Débuter avec GIT

Git est un **logiciel de gestion de versions décentralisé.** C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par plus de douze millions de personnes. (source : Wikipedia)

Installer git :

`choco install git` : installation avec chocolatey (sous Windows)

`apt install -y git` : installation avec Debian/Ubuntu

`git –version` : version du client git

Se situer au niveau du répertoire à synchroniser avec git puis

`git init` : initialiser le projet (crée un fichier .git qui contient les modif du projet)

`git config ––global user.name "odehecq"` : configurer username de git

`git config ––global user.email "odehecq@gmail.com"` : configurer email de git

`–– global` : configurer au niveau global (sinon cela configure juste pour le projet en cours)

`git add demo.txt` : prendre une « photo » du fichier demo.txt

`git add *` : prendre une « photo » de tous les fichiers du répertoire en cours

`git status` : **modification en cours et qui seront sauvegardées dans ma prochaine « photo »**

`git commit –m "[added] demo.txt"` : prend la photo et ajoute le message avec `–m`

`git log` : lister les commit du projet **pour la branche en cours** (indique aussi les identifiants des commit)

`git checkout <identifiant du commit>` : remet le projet à l'état du commit indiqué.

Attention : met dans l'état HEAD détaché (dans une autre branche)

`git checkout master` : remet le projet au dernier état de la branche master

`–f` : à ajouter pour forcer la branche master parce qu'on change de branche + il y avait des modif non commitées

Attention : supprime les modifications de la branche HEAD détachée

`git branch` : lister les branches existant dans le projet (par défaut, seulement master)

Une branche permet d'isoler les commit qui sont réalisées sur celle-ci. Ensuite un peut rapatrier la branche.

`git branch manouvellebranche` : ajouter une nouvelle branche manouvellebranche

`git checkout manouvellebranche` : passer sur la branche manouvellebranche

`git branch` : la branche en cours est en évidence (avec `*` devant)

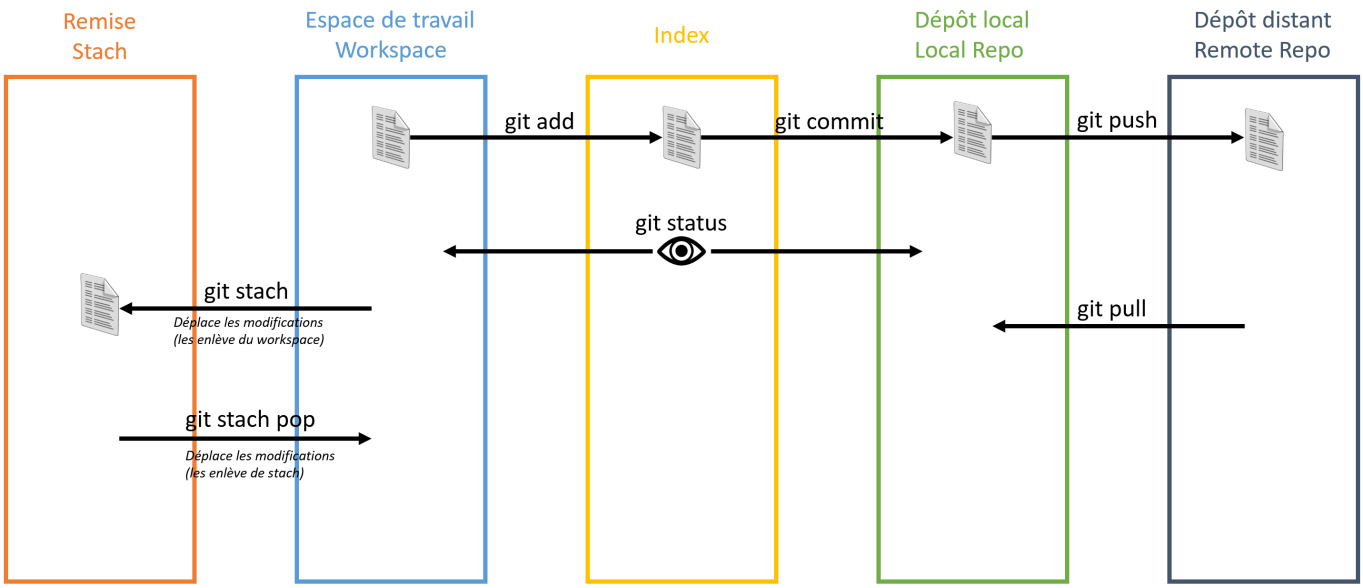
Fusionner une branche a dans une branche b (rapatrier les données de a vers b :

`git checkout b` : se placer sur la branche de destination

`git merge a` : récupérer les données de la branche a (**ne supprime pas la branche a**)

`git branch –d a` : supprime la branche a

2 Petit schéma



Exemple de fonctionnement

Olivier – de l'espace de travail vers le dépôt distant :

Olivier et Bob doivent travailler sur un projet « super_truc » ensemble. Olivier a déjà commencé à travailler dessus sur son pc, mais pas Bob.

Olivier prépare donc son **espace de travail** C:\super_truc\ pour git en faisant un `git init` (ça crée un fichier C:\super_truc\.git).

Olivier ajoute ensuite le contenu de c:\super_truc dans **l'index** en faisant un `git add *`, puis il ajoute les données à **son repo local** avec la commande `git commit –am '[added] first commit'`

En parallèle, il va sur gitlab (il aime bien les projets OpenSource) et crée un nouveau projet « super truc » vide en mettant le droit admin pour lui-même et maintainer pour Bob. C'est le **remote repo**.

Olivier fait ensuite un `git remote add origin https://gitlab.com/olivier/super_truc.git` pour câbler le **remote repo** à son **local repo**. Il crée ensuite une branch develop en faisant `git branch develop` puis un `git switch develop`, parce qu'il n'aime pas travailler dans la branche de production.

Enfin, il pousse les données de son **repo local** vers le **repo distant** en faisant un `git push origin develop`

A votre tour :

Quelles sont les commandes `git` que devra passer Bob pour rapatrier les données du **dépôt distant** vers son **workspace** ? Attention à récupérer les données de la branche develop.

Nota : si vous voulez tester ça « en vrai », vous pouvez créer un compte gitlab gratuit.

3 Résoudre les conflits

Si une branche de source et de destination contiennent des données différentes, il faut résoudre les conflits.

`git checkout b ; git merge a` : conflit de fusion

```
<<<<<<< HEAD (Modification actuelle)
Ligne présente dans la branche b
=======
Ligne présente dans la branche a
>>>>>>> a (Modification entrante)
```

Il faut donc modifier le fichier à la main puis `git add * ; git commit –m "resolution conflit"`

`git reset ––hard HEAD` : revenir au commit précédent

`git revert <identifiant du commit>` : revenir au commit <identifiant du commit>

`git diff <identifiant du commit>` : différences par rapport au commit <identifiant du commit>

`git diff <identifiant du commit> [fichier]` : différences dans le fichier par rapport au commit <identifiant>

`git diff <identifiant CommitA> <identifiant CommitB>` : différences entre 2 commit

Revenir à l'état antérieur / revenir à l'état actuel :

`git checkout <identifiant du commit>` : remet le projet à l'état du commit indiqué.

Attention : il n'est pas conseillé de faire des commit une fois revenu dans le temps

`git log` : lister les commit du projet **pour la branche en cours** (indique aussi les identifiants des commit)

`git log ––all` : lister tous les commit de cette branche. Le HEAD indique vers quoi le repo pointe

`git checkout master` : Revenir au dernier état de la branche (tout en haut dans le `git log ––all`)

4 Restaurer / récupérer son code

`git commit –am "message de commit"` : `–a` ajoute tous les fichiers modifiés

`git log ––oneline ––all` : pour voir tous les logs dont identifiants des commit sur une seule ligne et le HEAD

`git checkout <identifiant du commit>` : récupère l'état du commit dans le workspace & index (change HEAD)

on peut ainsi voir les fichiers tels qu'ils étaient au moment de ce commit

`git checkout master` : remet le workspace & index au dernier commit

Pour restaurer un fichier à un commit antérieur :

`git restore –source=<id du commit> fichier.txt` : pour restaurer le fichier.txt du commit <id du commit>

`git commit –am "remise du precedent fichier.txt"` : toujours commiter pour pousser les modifications

`–a` : ajoute les fichier trackés modifiés dans le commit. **Il faut les avoir ajoutés à l'index avant**

Pour restaurer tout un repos à un commit antérieur (pour supprimer un commit) :

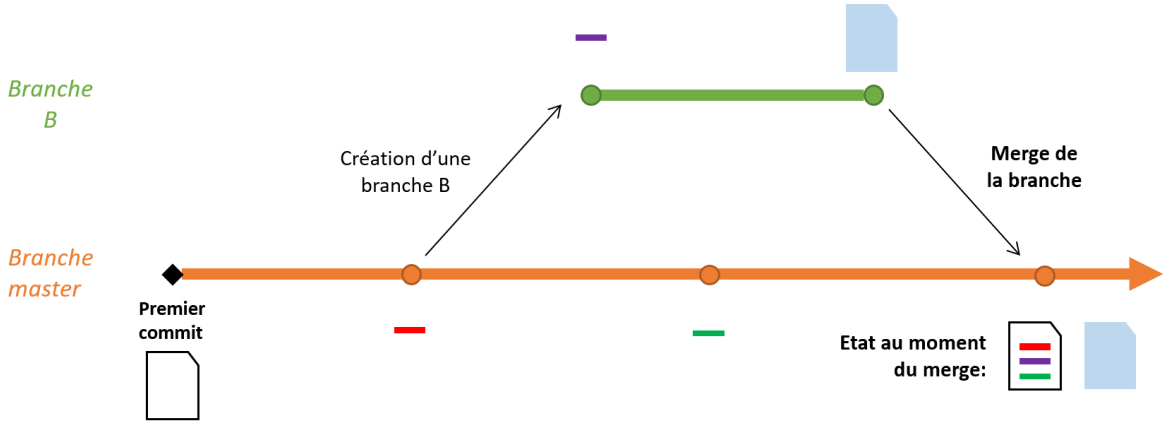
- Reset soft : supprime la sauvegarde mais garde les modifications entre les commit
- Reset hard : supprime la sauvegarde et supprime les modifications entre les commit

`git restore fichier.txt` : sans autre argument = restore fichier.txt au niveau de l'endroit où se trouve HEAD

`git reset ––hard <id du commit>` : restore au commit indiqué et supprime de la sauvegarde tout après

5 Créer et fusionner des branches

Merge



Merge d'une branche

`git init` : crée la branche master

`git add ; git commit –m "message commit"` : ajouter à l'index puis faire un commit

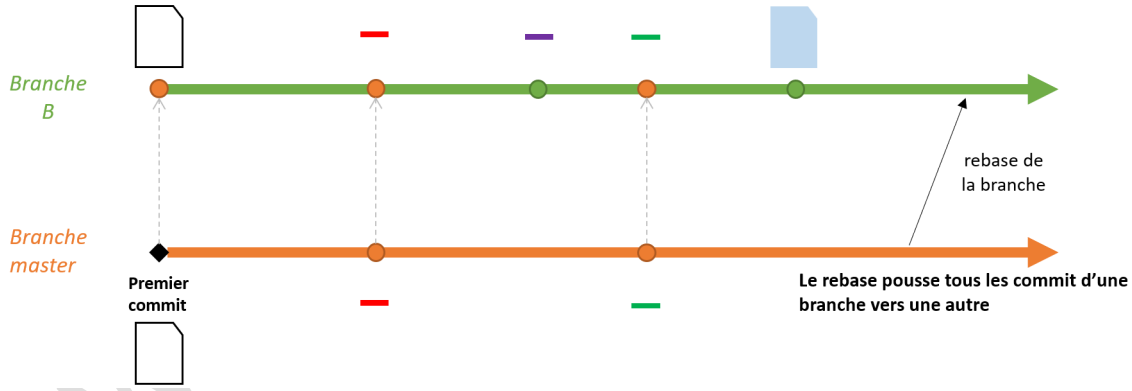
`git branch "brancheB" ; git switch brancheB` : créer une brancheB puis se déplacer dessus

`git add ; git commit –m "message commit brancheB"` : ajouter à l'index puis faire un commit

`git switch master` : se déplacer sur la destination avant de merger

`git merge navigation –m "fusion avec brancheB"` : merge

Rebase :



Rebase d'une branche

Pour revenir avant le merge :

`git switch master` : se déplacer sur la branche sur laquelle on a fait le merge

`git reset –hard <id du commit avant merge>` : Pour revenir avant le merge

Pour faire le rebase :

`git switch brancheB` : avant de faire le rebase de brancheB

`git rebase master` : pousser les commit de master sur la brancheB

Pour récupérer les commit de brancheB dans master en gardant l'ordre des commit :

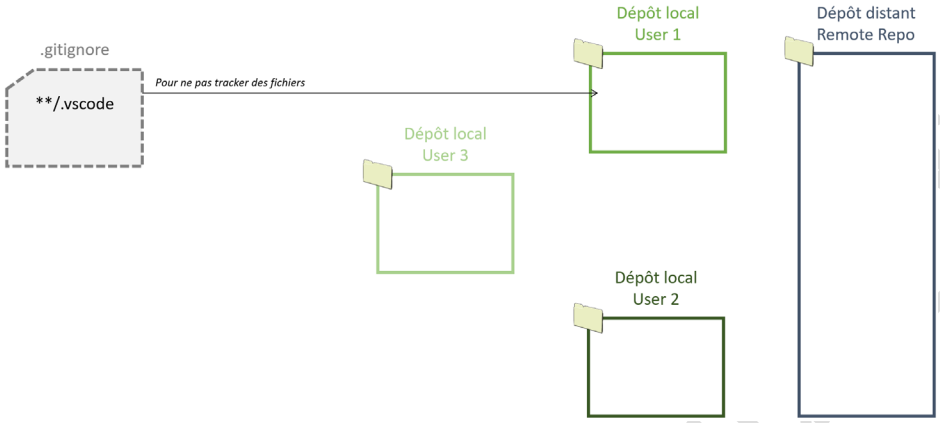
`git switch master` : pour se placer sur master

`git merge brancheB` : pour récupérer les commit de brancheB

6 Ajouter un repo distant et travailler à plusieurs

Un dépôt distant peut être :

- Un dossier
- Une clé usb
- Un serveur
- Un github



Fonctionnement de ce tuto

User1 : crée un remote repo et fais un commit dedans

Se place dans son dossier de repo local

```
git init
git add fichier.txt .gitignore
git commit -m "premier commit"
```

se placer dans le repote-repository

`git init --bare` : pour créer un remote repos en local

se placer dans le repo local

`git remote` : liste les repo distants rattachés à ce repo local (si vide, alors il n'y en a pas)

`git remote add origin /path/of/remote/repo/` : ajoute un remote repo et le nomme **origin**

`git remote` : renvoie origin car le remote repo **origin** est rattaché au repo local

`git push origin master` : pousse (envoie) la branche **master** vers le remote repo **origin**

User2 : crée une branche et la pousse vers le remote

Se place dans son dossier de repo local

`git clone /path/of/remote/repo/ .` : clone le contenu du remote repo dans le dossier actuel (**.**)

git clone crée le repo local et le repo distant avec son nom

`git branch branche2` : crée une nouvelle branche2 (`git branch` pour voir)

`git switch branche2` : switcher vers la branche2

`git commit -am "modification du fichier.txt"` : après modif du fichier.txt pour commiter vers la branche2

`git push origin branche2` : pousser la branche2 vers le remote repo nommé origin

User1 : fusionne la branche vers master

/!\ `git pull origin branche2` pousserait les données de branche2 vers la branche master (on ne veut pas ça)

`git switch –c branche2` : switch vers la branche2 (-c : crée cette branche si elle n'existe pas)

`git pull origin branche2` : récupère la branche2 depuis le remote repo origin

`git switch master` : se placer sur master

`git merge branche2 –m "fusion branche2 vers master"` : fusionner branche2 vers master

`git push origin master` : pour pousser master vers origin

User2 : met à jour master

`git switch master` : se place sur master

`git pull origin master` : récupère master depuis origin

User3 : récupère le remote en local

`git init` : pour créer son repo local

`git remote add origin /path/of/remote/repo/` : ajoute un remote repo et le nomme **origin**

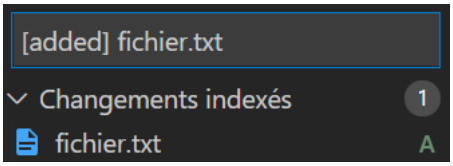
`git pull origin master` : récupère le master de origin vers son master local

7 Git dans VSCode

VSCode est un éditeur de texte spécialisé dans la création de code en mode texte. C'est un outil indispensable quand on fait du code, d'autant plus qu'il est gratuit et dispose de nombreuses extensions (parfois même fournies par les éditeurs comme Puppet ou RedHat).

Microsoft propose une formation gratuite à l'intégration de git dans VSCode : [Utiliser les outils de gestion de versions Git dans Visual Studio Code - Learn | Microsoft Docs]

Pour commiter : écrire un message puis [ctrl]+[enter]



Le fichier.txt est ainsi ajouté dans le dépôt local

8 Déploiement continu

```
docker run -it -p 1313:1313 registry.gitlab.com/pages/hugo:latest sh
# hugo new site blog
# cd blog/
# hugo new posts/article.md
# apk add vim
# vim content/posts/article.md
draft: false
---
Bienvenue sur mon nouveau blog.
# apk add git
# git init
# git submodule add https://github.com/panr/hugo-theme-hello-friend.git themes/hello-friend
# vi config.tml
```

```
languageCode = "FR-fr"
title = "Mon Blog"
theme = "hello-friend"
[params.logo]
  logoText = "Mon Blog"
```

```
# hugo server --bind 0.0.0.0
# git remote add origin https://gitlab.com/olivier.dehecq/blog.git
# git add -A
# git status
# git commit -am "[added] first commit"
# git push origin master
```

A la racine du repository, créer un fichier .gitlab-ci.yml

```
image: registry.gitlab.com/pages/hugo:latest

variables:
  GIT_SUBMODULE_STRATEGY: recursive

pages:
  script:
  - hugo
  artifacts:
    paths:
    - public
  only:
  - master
```

Dans CI/CD > Pipelines, le pipeline est en cours puis réussit. Il est ensuite déployé

Dans Setting > Pages, la page d'accès est indiquée et l'url est renseignée : https://olivier.dehecq.gitlab.io/blog

L'accès échoue car le fichier config.toml est mal renseigné. Il faut le renseigner :

Config.toml :

```
baseURL = https://olivier.dehecq.gitlab.io/blog
(…)
```

Puis commit. Le pipeline s'exécute puis la page est accessible

9 Créer une pull / merge request

- Github : pull request
- Gitlab : merge request

Même fonctionnement : pousser des modifications d'une branche dans une autre.

https://www.youtube.com/watch?v=Zyf5gRJjXzY

Suivre le contributing.md quand il est présent

Reporter le problème : dans issues. Faire une recherche avant dans les filtres puis « new issue »

Faire un fork : copie du repos dans notre propre compte. Pour faire des modifications sans gêner le propriétaire du repo original

Commiter et commenter en anglais

Créer la pull request