Table des matières

Git

Installer Git (MacOs)

i. Installer Git depuis le site web

ii. Configurer Git via le terminal

```
git config --global user.name "Votre nom ou pseudo"

git config --global user.email "Votre@email.com"
```

Premier commit

i. `mkdir mon_repertoire`

ii. `git init`

iii. `touch README.md`

iv. `git add README.md` (`git add .` pour ajouter tous les fichiers du répertoire à l'index)

v. `git commit -m "Description de mon commit"` (l'option -m permet de spécifier un message, ici "Description de mon commit")

Astuce : pour mettre à jour un fichier déjà ajouté à l'index il est possible de condenser la commande d'ajout du fichier à l'index et du commit

```
git -a -m "Ma description pour ce nouveau commit"
```

ou même

```
git -am "Ma description pour ce nouveau commit"
```

L'option -a demande a Git de mettre à jour les fichiers déjà existants dans son index

Lire l'historique

- `git log`
- Donne le résultat suivant :

```
commit c11b9918d74948ff8760bb1b8577e35e41678a28 (HEAD -> master)

Author: Gilles Loriquer <gilles.loriquer@gmail.com>

Date: Thu Nov 1 18:34:20 2018 +0100

Commit de README.md
```

c11b9918d74948ff8760bb1b8577e35e41678a28 représente le SHA (cad l'identifiant unique du commit)

Se positionner sur un commit

Pour se positionner sur un commit il faut récupérer le SHA du commit en question (voir `git log`). Utiliser ensuite la commande suivante :

```
git checkout SHADuCommit
```

Exemple :

```
git checkoutc11b9918d74948ff8760bb1b8577e35e41678a28
```

Pour se positionner sur le commit le plus récent de la branche principale, utiliser la commande :

```
git checkout master
```

Annuler un commit

Il n'est pas vraiment possible d'annuler un commit mais il existe des solutions.

Pour faire un revert de commit (cad faire un commit de retour à un ancien commit), utiliser la commande suivante :

```
git revert SHADuCommit
```

Pour simplement modifier le message du dernier commit, utiliser la commande (attention, le message ne peut être modifié que si le commit n'a pas été pushé sur l'origine) :

```
git commit --amend -m "Votre nouveau message"
```

Enfin, si je n'ai pas encore fait mon nouveau commit mais que je veux annuler tous les changements que je n'ai pas encore commités, j'utilise la commande :

```
git reset --hard
```

Manipuler les branches

Lister les branches

```
git branch
```

Créer une branche

```
git branch ma-nouvelle-branche
```

Changer de branche

```
git checkout ma-nouvelle-branche
```

Cette commande permet de quitter la branche courante pour se rendre dans la branche 'ma-nouvelle-branche'

Créer une branche et s'y positionner

```
git branch -b checkout ma-nouvelle-branche
```

Cette commande est un condensé des deux précédentes. Elle permet de créer une branche et de s'y positionner

Merger une branche

Imaginons le scénario suivant : une branche 'master' avec une branche secondaire 'second'. Pour merger la branche second vers la branche 'master' (la branche principale), il suffit d'utiliser la commande :

`git checkout master` (je me positionne dans la branch 'master')

`git merge second` (je récupère le code développé de la branche 'second' vers ma branche 'master')

Bien sûr il est tout a fait possible d'effectué le merge dans l'autre sens

Résoudre un conflit

Pour résoudre un conflit lors d'un merge :

i. Modifier le fichier objet du conflit

ii. Réaliser un `git status` afin de constater que le fichier en question est affiché en tant que '**Unmerged paths**'. Cela signifie qu'il faut indiquer à Git que le conflit est maintenant résolu... Et pour cela :

iii. `git add monfichier`

iv. `git commit` (ne pas indiquer de message - cela ouvrira un fichier qu'il suffit d'enregistrer et de fermer)

v. A l'aide de la commande `git log`, il est possible de voir que les commits réalisés dans les deux branches sont affichés

Retrouver qui a fait une modification

Pour retrouver qui a fait une modification sur un fichier réaliser les opérations suivantes :

i. `git blame monfichier.extension`

ii. Cela donne le résultat suivant :

```
0e601679 (Martin Monnier 2018-11-03 13:02:40 +0100 1) Modification effectuée
```

iii. A l'aide du SHA 0e601679 réaliser la recherche du commit correspondant :

```
git show 0e601679
```

Eviter les commits superflus

Si je suis en train de faire des modifications sur un fichier dans une branche spécifique et qu'une urgence m'impose de traiter un bug dans la branche master par exemple, la solution que je connais jusqu'ici est de faire un commit des modifications en cours. Hors, cela signifie réaliser un commit inutile car ma fonctionnalité n'est pas encore complètement implémentée. Il existe une solution pour pallier à ce cas de figure :

- `git stash` permet de mettre de côté les modifications en cours sans avoir à réaliser de commit. Il est alors possible de quitter ma branche et d'aller travailler dans une autre branche
- `git stash` pop permet de rétablir les modifications qui ont été mises de côté (une fois de retour dans ma branche donc). **Attention** : cette commande vide le stash ! Si je quitte ma branche sans réaliser de commit ou si je ne réalise pas un nouvel appel à git stash mes modifications seront perdues
- `git stash apply` a la même fonction que git stash à la différence que si je quitte ma branche sans réaliser de commit, mes modifications mises en stash lors de l'appel à git stash seront conservées

Important ! Ignorer des fichiers

Pour ignorer des fichiers et des dossiers il suffit de créer un fichier .gitignore et d'ajouter à l'intérieur les fichiers ou dossiers que l'on souhaite ignorer.

i. `touch .gitignore` pour créer le fichier (à créer à la racine du repository)

ii. `vim .gitignore` pour éditer le fichier. Le contenu pourrait ressembler à ça :

```
# Ignorer le fichier test
test
# Ignorer le répertoire temp
temp/
# Ignorer tous les fichiers java
*.java
# Ignorer tous les fichiers txt du répertoire bin
bin/*.txt
```

Le fichier .gitignore doit être tracké par Git. Voici de manière générale les fichiers à exclure de l'index de Git :

- Tous les fichiers de configuration (config.xml, databases.yml, .env...)
- Les fichiers et dossiers temporaires (tmp, temp/...)
- Les fichiers inutiles comme ceux créés par votre IDE ou votre OS (.DS_Store, .project...)

Le plus crucial est de ne **JAMAIS versionner une variable de configuration**, que ce soit un mot de passe, une clé secrète ou quoi que ce soit de ce type. Versionner une telle variable conduirait à une large faille de sécurité, surtout si vous mettez votre code en ligne sur GitHub !

Si vous avez ce type de variables de configuration dans votre code, déplacez-les dans un fichier de configuration et ignorez ce fichier dans Git : nous allons voir comment faire cela dans la vidéo ci-dessous en utilisant le fichier **.gitignore**.

Quelques commandes utiles

git status

La commande `git status`affiche l'état des différents fichiers et dossiers. Plusieurs informations peuvent être affichées :

- '**Untracked files**' : le ou les fichiers ne sont pas ajoutés à l'index et ne sont donc pas suivis par Git
- '**Changes to be committed**' : le ou les fichiers ont été ajoutés à l'index via la commande `git add` mais n'ont pas encore été commités
- '**Changes not staged for commit**' : le ou les fichiers n'ont pas été ajoutés à l'index via la commande `git add` ou bien ils ont été ajoutés mais ont été modifiés avant ou après le dernier commit. Pour qu'ils soient a nouveau préparés pour être commités il faut faire à nouveau usage de la commande `git add`.

Note : il peut être intéressant de ne pas 'stager' un ou plusieurs fichiers si nous travaillons sur ceux-ci et qu'ils ne sont pas prêt à être livrés. Dans ce cas, lors du commit, ces fichiers ne seront pas poussés dans le repository Git.

git log (voir Lire l'historique)

La commande `git log` liste l'historique de tous les commit réalisés sur le repository.

`git log` donne le résultat suivant :

```
commit c11b9918d74948ff8760bb1b8577e35e41678a28 (HEAD -> master)

Author: Gilles Loriquer <gilles.loriquer@gmail.com>

Date: Thu Nov 1 18:34:20 2018 +0100

Commit de README.md
```

c11b9918d74948ff8760bb1b8577e35e41678a28 représente le SHA (cad l'identifiant unique du commit).

git remote -v

La commande `git remote -v` permet de lister les remotes associés au repository Git.

GitHub

Cloner un repository depuis GitHub vers ma machine locale

i. Ouvrir un terminal

ii. Créer un dossier

```
mkdir mondossier
```

iii. Se rendre dans ce dossier

```
cd mondossier
```

iv. Via le navigateur et sur le repository GitHub à récupérer, cliquer sur '**Clone or Download**' et copier le lien HTTP

v. De retour sur le terminal, cloner le repository avec la commande suivante :

```
git clone LienHTTPSDuRepo
```

Si un login/mdp est demandé lors du `git clone` il s'agit du login/mdp de GitHub

Créer un repository

Il existe deux solutions détaillées ci-dessous.

Solution 1 : création du repository et clone vers la machine locale

La première solution consiste à créer un repository sur le site GitHub en suivant les étapes suivantes :

i. Dans la partie repository du site, cliquer sur 'New'

ii. Donner une description si nécessaire

iii. Laisser le repository en 'Public' (un compte premium est nécessaire pour rendre privé un repo)

iv. Cocher '**Initialize this repository with a README**'. Cela permet d'initialiser un premier fichier README.md dans notre repo (laisser les autres options par défaut)

v. Cliquer sur 'Create repository'

vi. Pour récupérer ce repository en local, suivre la section concernant le clone de repository

Solution 2 : création du repository et push depuis la machine locale

i. Dans la partie repository du site, cliquer sur 'New'

ii. Donner une description si nécessaire

iii. Laisser le repository en 'Public' (un compte premium est nécessaire pour rendre privé un repo)

iv. Ne **PAS** cocher 'Initialize this repository with a README' et laisser les autres options par défaut. Le fait de ne pas créer de fichier README.md signifie que le repository est créé vide

v. Cliquer sur 'Create repository'

vi. Dans la page suivante, copier le lien HTTPS généré par la création du repository

vii. Ouvrir un terminal

viii. Créer un répertoire : `mkdir mondossier`

ix. Se rendre dans ce répertoire : `cd mondossier`

x. Créer un fichier README.md et y ajouter le nom du repository utilisé lors de la création du repository GitHub :

```
echo "# nomdemonrepo" >> README.md
```

xi. Initialiser le répertoire : `git init`

xii. Ajouter le fichier README.md à l'index Git : `git add README.md`

xiii. Premier commit : `git commit -m "Mon premier commit"`

xiv. Lier ce répertoire Git au remote repository GitHub :

```
git remote add origin LienHTTPSDuRepo
```

1. Pousser les fichiers et répertoires indexés du projet Git sur GitHub :

```
git push origin master
```

Solution 2 bis : pousser un repository Git existant

L'option 'Initialize this repository with a README' est à cocher uniquement dans le cas où vous n'avez pas encore créé le repository en question sur votre machine (solution 1)

La solution 2 concerne le plus souvent le cas où un projet Git existant (sur la machine locale) doit être poussé sur un repository GitHub fraichement créé. Dans ce cas, il suffit de :

i. Lier le repository Git local au repository GitHub via la commande :

```
git remote add origin LienHTTPSDuRepo
```

ii. Pousser les fichiers et répertoires indexés du projet Git sur GitHub :

```
git push origin master
```

Lors du `git push origin master` il est possible que le login/mdp GitHub soit demandé

Récupérer des modifications

Pour récupérer les dernières modifications d'un repository il suffit (à l'image d'un push) d'exécuter la commande suivante :

```
git pull origin master
```