Introduction à Git et à ses plateformes

27 janvier 2020

1 Introduction

1.1 Un outil de gestion de version distribué

`Git` est un logiciel de gestion de version distribué dont le concurrent le plus célèbre est `Subversion`. `Git` est aujourd'hui l'outil de ce type le plus populaire avec plusieurs millions d'utilisateurs. L'application est notamment connue au travers de plateformes telles que `Github` ou encore `Gitlab`. Avant d'entrer plus en détails dans le fonctionnement de `Git`, il est important de comprendre ce qu'est un outil de gestion de version distribué. Il s'agit de maintenir un historique complet des fichiers et synchronisés sur l'ensemble des machines des développeurs participants au projet. Ainsi, il doit être possible de revenir en arrière en cas d'erreur ainsi que de collaborer à plusieurs. Ce genre d'outil permet la création de branches pour le développement de certaines fonctionnalités en parallèle et de *merging* afin de fusionner les contributions qu'auraient faits plusieurs développeurs sur le même fichier.

1.2 Architecture

La figure 1 décrit le fonctionnement de `Git`. L'ensemble des machines des développeurs se retrouvent ici sous l'appellation *Local*. Les lignes verticales défissent l'évolution temporelle d'un projet. Chaque développeur possède sur sa machine un *working directory* dans lequel il pourra travailler. Afin que `Git` surveille un nouveau fichier du dossier de travail, il convient d'exécuter la commande `git add`. Le fichier est maintenant surveillé par `git`. Après un certain nombre de modifications, le développeur doit marquer une "version". Il utilise alors la commande `git commit` qui, toujours en local, sauvegarde une version du code. Toute modification ultérieure ne changera pas cette version et il sera possible de revenir en arrière à cette version si besoin. Dans un projet à plusieurs, il est important de partager ses versions afin que chacun reste sur un état à jour du code. La commande `git push` permet de propager les versions locales sur le *remote repository*. La commande `git pull`, à l'inverse permet de récupérer les versions distantes.

Nous allons, au cours de cette séance, aborder les différentes étapes de gestion d'un outil de *versionning*.

2 Création d'un projet sur `Github` (Tâche individuelle)

Si `Git` n'est pas disponible sur votre machine, Google est votre ami...

2.1 Configurer `Git`

On commencera par modifier les préférences globales de `Git`.

```
git config --global user.name "John Smith"
```

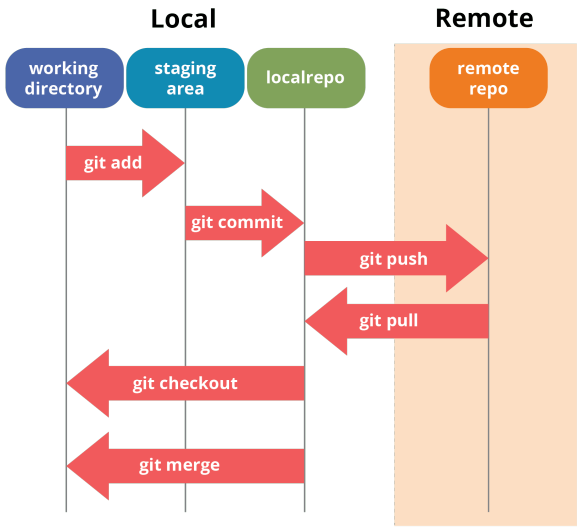

FIGURE 1 – Fonctionnement de Git

`git config --global user.email "john.smith@gmail.com"`

La commande suivante dit à `Git` de ne pas demander le mot de passe à chaque opération avec le serveur :

`git config --global credential.helper store`

2.2 Initialisation du projet

Afin de créer un projet, nous allons dans l'ordre (1) créer le dossier du projet, (2) créer le projet sur `Github` (créer un compte si cela n'a pas déjà été fait), (3) ajouter un premier fichier en local à notre projet git et (4) lier notre projet local à notre *remote repository* sur `Github`.

2.2.1 Création du projet localement

```
mkdir mon_projet
cd mon_projet
echo "# Projet L3" >> README.md
```

Cette dernière ligne crée un fichier README.md qui contiendra la phrase " # Projet L3 ". `md` signifie mardown, il s'agit d'un format permettant de formater du texte. Le fichier README.md sera le contenu affiché par défaut sur `Github`.

À ce stage, votre projet n'est toujours pas lié à `git`. Pour cela, nous allons exécuter la commande suivante :

`git init`

Puis, nous allons ajouter le fichier README.md à la liste des fichiers suivis :

`git add README.md`

Le fichier maintenant ajouté, nous allons créer une version de notre code avec la commande suivante :

`git commit -m "le message obligatoire à chaque commit"`

Enfin, nous allons relier notre projet local au projet remote sur `Github`.

Vous pouvez bien sûr ajouter plusieurs fichiers en une seule commande. Pour supprimer un fichier, utiliser la commande `git rm mon_fichier`.

2.2.2 Création du projet distant et lien avec le dossier local

Créez maintenant votre projet sur Github. Attention, n'ajoutez aucun fichier automatiquement pour l'instant. Une fois votre projet créé sur `Github`, exécutez la commande suivante en local afin de lier votre repository local au repository distant.

```
git remote add origin https://github.com/username/projet_name.git
```

Enfin, nous allons "pousser" nos données locales sur le projet distant :

```
git push -u origin master
```

le mot *origin* fait référence au repository distant et *master* à la branche principale sur laquelle vous êtes. Il n'est généralement pas utile de préciser les options `-u origin master` car cela est évident.

Vous pouvez maintenant rafraichir la page distante sur Github et observer le résultat obtenu. Comme vous avez pu le constater, Github proposait la possibilité de créer le projet directement avec un fichier README.md. Dans le cas où vous auriez choisi cette stratégie, la commande à utiliser est `clone` qui aurait cloné le repertoire distant dans un repertoire local à votre machine (ne pas exécuter si vous avez déjà réalisé les étapes précédentes) :

```
git clone https://github.com/username/projet_name.git
```

2.3 Quelques manipulations

Créez maintenant un fichier `toto.txt` dans votre projet et ajoutez un texte quelconque dans ce fichier.

Tout d'abord, ajoutons le fichier à la liste des fichiers surveillés :

```
git add toto.txt
```

Et créons une version avec ce fichier

```
git commit -m "ajout de toto.txt"
```

Allez sur `Github` et observez le résultat. Rien... En effet, la commande commit crée une version locale et non distante. Il faut maintenant utiliser la commande `push` afin de "pousser" vos versions locales sur le serveur distant. Il n'est pas nécessaire de pousser chaque version une par une et vous pouvez réaliser plusieurs commit avant un push. Allez sur `Github` à nouveau et observez cette fois-ci votre fichier. Cliquez dessus et modifiez le. Pour valider votre modification, vous devez appuyer sur le bouton "commit" du serveur distant.

La version distante du projet est maintenant en avance sur votre version locale.

Pour mettre votre version locale à jour, exécutez la commande suivante :

```
git pull
```

Observez votre fichier mis à jour sur votre machine.

Créez les fichiers toto1.txt, toto2.txt et toto3.txt. Ajoutez les, créez une version, poussez là sur le serveur. Puis supprimez les avec la commande `git rm`, créez une version et poussez là à nouveau sur le serveur. Rappelez-vous que toute opération création, modification ou suppression doit être versionné pour être conservée et partagée avec les autres.

2.4 Premières difficultés

Allez à nouveau sur `Github` et modifiez à nouveau le fichier toto.txt. Puis rajoutez une ligne à la fin du fichier toto.txt sur votre machine sans la synchroniser avec le serveur distant. Chaque repository contient donc une version différente du fichier toto.txt, aucune n'étant l'historique de l'autre. Exécutez la commande

```
git pull
```

Vous obtenez un message d'erreur indiquant que votre dossier de travail contient une information non versionnée. Avant de pouvoir compléter l'opération pull, vous devez faire un "commit".

```
git commit -m "mon message"
```

Vous avez peut-être face à vous une seconde erreur : “no changes added to commit”. Git permet de ne commit qu’une partie des fichiers. Ainsi, même si un fichier appartient au repository, il est parfois nécessaire de re-exécuter la commande `git add` afin d’indique que ce fichier doit être “commité”. Une autre stratégie est d’exécuter la commande suivante :

```
git commit -am "rajout de l'option -a"
```

Nous avons donc maintenant créé une version avec notre fichier modifié et nous pouvons à nouveau exécuter la commande `pull` :

```
git pull
```

On obtient à nouveau une erreur ! En effet, le fichier distant et le fichier local contiennent tout deux une information nouvelle. Il faut faire un “conflict merge”, à savoir, fusionner les deux contenus. Ouvrez la nouvelle version de votre fichier toto.txt. Celui-ci a la structure suivante :

```
<<<<<< HEAD
Contenu local
=======
Contenu distant
>>>>>>> ee4291c54c3d96f5736094a212c5592126a88d3d
```

Le merge n’indiquera que les lignes différentes entre le fichier local et distant et non les lignes qui n’ont pas bougé. La partie entre HEAD et ===== contient votre contenu local et la partie entre ==== et la séquence hexadécimale contient le contenu distant. Modifiez le fichier localement afin de corriger ce conflit (gardez le contenu qui vous intéresse). Exécutez la commande suivante afin d’indiquer que le fichier est maintenant prêt à être à nouveau versionné :

```
git add toto.txt
```

Puis réalisez votre commit :

```
git commit -m "mon fichier corrigé"
```

Vous auriez pu exécuter directement le commit avec les options “-am”. Enfin, poussez vos modifications sur le serveur distant : `git push`. Allez voir sur `Github` le résultat de vos actions.

Pensez à partager très fréquemment votre code afin de ne pas vous retrouver dans cette situation trop souvent.

2.5 Gestion des branches

Imaginons un projet complexe auquel plusieurs développeurs participent. Vous souhaitez proposer une nouvelle fonctionnalité. Malheureusement, le développement de celle-ci risque de perturber le fonctionnement de l’application. Pour cela, vous pouvez créer une “branche” qui sera en quelque sorte une seconde version de votre projet. Vous ferez vos commit et vos push dans cette branche. Une fois la fonctionnalité développée, vous pourrez fusionner la branche avec le “master” afin de partager votre fonctionnalité avec tous. Évidemment, cette opération peut entraîner un certain nombre de conflits qu’il faudra résoudre.

Créons une nouvelle branche avec la commande suivante :

```
git checkout -b ma_branche
```

Vous avez créé votre branche en locale, mais celle-ci n’est pas encore visible sur le serveur distant :

```
git push -u origin ma_branche
```

Allez sur `Github` et observez que votre branche y est bien présente.

Nous allons nous attaquer à notre nouvelle fonctionnalité révolutionnaire : le fichier bob.txt. En répétant les opérations précédentes, créez votre fichier, ajoutez le, créez une version et poussez la sur le serveur distant.

Mince ! Les autres ont besoin de vous sur l’ancienne branche (i.e. “master”) alors que vous n’avez pas terminé votre fonctionnalité ! Cela n’est pas gênant, on peut sauter d’une branche à l’autre.

```
git checkout master
```

Vous revoilà sur la branche master. Attention, il est possible que les autres développeurs aient continuer à avancer le projet. Pour mettre à jour votre branche locale, exécutez la commande `git pull`. Créez un fichier tata.txt, ajoutez-le, créez une version avec et poussez là sur le serveur distant. Enfin, revenez sur votre branche :

```
git checkout ma_branche
```

Vérifiez que le fichier tata.txt n'est bien plus présent. Modifiez le fichier bob.txt, créez une version avec cette modification et poussez là sur le serveur.

La fonctionnalité est maintenant développée et nous souhaitons l'ajouter sur la branche principale "master". Pour cela, nous devons retourner sur la branche "master" :

```
git checkout master
```

Nous allons maintenant exécuter la fusion :

```
git merge ma_branche
```

On vous demande probablement d'ajouter un message de fusion.. Si vous êtes sur `vi`, tapez " :!q", puis appuyez sur "entrer".

Si besoin (si demandé), commitez le merge :

```
git commit -m ''je viens de merge''
```

Poussez votre commit :

```
git push
```

Supprimez la version locale de votre branche, maintenant inutile :

```
git branch -D ma_branche
```

Et supprimez la version distante de la branche :

```
git push origin --delete ma_branche
```

Si vous retournez sur `Github`, tous les fichiers sont là et la branche n'existe plus !

2.6 Autres fonctionnalités de `Github`

Vous pouvez observer que `Github` offre un certain nombre de fonctionnalités de gestion de projet supplémentaires. Ainsi, l'onglet "project" permet de simuler l'effet gestion de projet/autocollant sur un tableau propre aux méthodes agiles.

2.7 Comment ajouter un collaborateur sur `Github`

Pour ajouter un collaborateur à votre projet, rendez vous sur l'onglet "Settings" puis "Collaborators" et cherchez votre collègue par *username*.

3 Création de votre projet (Tâche collective)

Avec vos connaissances fraichement acquises, rejoignez les membres de votre groupe et créez le repository Git de votre projet. Celui-ci sera évalué le jour de la soutenance.