TP - Git.

Brendan Guillouet

31 Janvier 2020

Contents

Ce TP présente un aperçu des manipulations de bases utiles à la gestion d'un projet informatique avec le logiciel de gestion de version **Git** et le service d'hébergement **Github**.

1 Créer un dépôt en local

Les commandes de cette section vont permettre de recréer les exemples des slides 15, 19 et 21 du cours.

1.1 Index et dépôt

- Créez un dossier et placez-vous dedans à l'aide de la commande suivante :

```
mkdir exemple1 && cd exemple1
```

- Vérifiez que ce dossier est vide (`ls -a`). Initiez votre `dépôt` (`git init .`) et vérifiez que le dossier caché *.git* est bien présent.
- Créez un fichier *test.txt* contenant le texte *version 1*, ajoutez-le à l'`index` puis à votre `dépôt`. (On pourra s'aider des slides du cours).
- La commande suivante permet d'afficher les objets référencés dans l'`index`:

```
git ls-files --stage
```

  Constatez que le *hash* généré pour le `blob` est bien le même que celui du cours et affichez son contenu à l'aide de la commande suivante:

```
git cat-file -p 83baae61804e65cc73a7201a7252750c76066a30
```

  affichez également le contenu du `HEAD` qui pointe sur votre premier `commit` :

```
git cat-file -p HEAD
```

  et constatez enfin que le *hash* généré pour le `tree` est également le même que sur les slides.

- **Exercice:** Appliquez les commandes successives de la slide 15 jusqu'à arriver à l'état final. Pour validez cet état final appliquez la commande `git status` et vérifiez que la sortie de cette commande est identique à la sortie affichée slide 16 du cours.

  Tout au long de ces étapes, vous pouvez appliquer les commandes `git ls-files` et `git cat-file` (def slide 18) afin d'observer les changements effectués.

1.2 git reset

1.2.1 V1

- A l'aide des commandes `git ls-files` et `git cat-file` , affichez le contenu du `blob` vers lequel pointe *test.txt* dans l'`index`. Qu'affiche-t-il?
- Quel sera le résultat de la commande suivante?

```
git reset HEAD -- test.txt
```

  Appliquez cette commande, puis appliquez les commande du point précédent afin d'afficher à nouveau le contenu du `blob` vers lequel pointe *test.txt* dans l'`index`. Qu'affiche-t-il? Que pouvez dire du fichier *test.txt* dans le `working directory`?

- A l'aide de la commande `git log` (def slide 19) retrouvez le *hash* de votre premier commit. Puis appliquez la commande suivante, en remplaçant *HasPremierCommit* par la bonne valeur.

```
git reset HashPremierCommit -- foo.txt
```

  Quel est le résultat de cette commande? Comment pouvez vous le constatez?

- Appliquez les changements nécessaire avant de revenir à l'état final de la slide 15. Comme précédemment validez cet état à l'aide de la commande `git status` en vérifiant que la sortie de cette commande est identique à la sortie affichée slide 16 du cours.

1.2.2 V2

- Exécutez la commande suivante :

```
git reset --soft HashPremierCommit
```

  Cette commande n'a entraîné aucun changement, ni dans l'`index`, ni dans le `working directory`. Vérifiez-le.

  Appliquez ensuite la commande `git status`. Quel changement observez-vous? Comment l'expliquez vous?

- Exécutez la commande suivante :

```
git reset --mixed HashPremierCommit
```

  Cette commande n'a entraîné aucun changement dans le `working directory`. En revanche, l'état de l'`index` est maintenant identique à celui du premier `commit`. Vérifiez-le.

  Appliquez ensuite la commande `git status`. Quel changement observez-vous? Comment l'expliquez vous?

- Exécutez la commande suivante :

```
git reset --hard HashPremierCommit
```

  L'état de l'`index` est maintenant identique à celui du premier `commit`, ainsi que le `working directory`! Cette commande est à manier avec précaution.

2 Créer un dépôt en remote.

Github est un service d'hébergement de dépôt et de gestion de développement. Il va vous permettre de partager nos `dépôt` en ligne.

2.1 Créer un nouveau compte sur github.

- Rendez-vous sur le site web **https://github.com/**.
- Créez un compte (SignUp).
- Choisissez un nom d'utilisateur, email et mot de passe.
- Choisissez l'option individuel et gratuite.
- Passez l'étape des questions.

- Rendez-vous sur votre boite email et confirmez votre inscription.
- Vous êtes maintenant sur une page vous permettant de créer votre premier dépot. Passez à la section suivante.

2.2 Créer le dépôt.

- Donnez un nom à votre dépot. (exemple: **GestionDeProjet4A**)
- Choisissez l'option privé.
- Choisissez l'option permettant de créer votre répot avec un *README* puis sur cliquez sur créer.

Vous venez de créer votre premier dépôt **Git** en remote. La procédure que vous venez de suivre est propre à l'outil **Github**, mais il existe de nombreuses autres façons de créer un `dépôt`.
Votre `dépôt` est pour l'instant constitué d'un seul fichier: le fichier *README.md*.

3 Cloner votre dépôt en local.

Votre dépôt **GestionDeProjet4A** n'existe pour l'instant que sur le serveur distant de **Github**. Pour pouvoir récupérer ce `dépôt` sur votre ordinateur, en local, vous allez effectuer un *clone* de ce `dépôt`.

3.1 Connecter votre Git local avec votre compte Github.

Avant de cloner votre `dépôt`, vous allez devoir configurer votre **Git** en local afin que le serveur distant vous reconnaisse lorsque vous travaillez depuis votre poste. Pour cela vous allez executer les deux commandes suivantes dans votre terminal en remplaçant USERNAME par le nom d'utilisateur et USERMAIL par l'e-mail que vous avez choisi pour créer votre compte **Github**.

```
git config --global user.name USERNAME
git config --global user.email USERMAIL
```

Assurez vous que vous avez bien configuré votre **Git** en local, en constatant qu'un fichier caché *.gitconfig* a bien été créé dans votre *home*.
Vous pouvez également vérifier vos différentes modifications à l'aide de la commande :

```
git config --list
```

3.2 Cloner le dépôt.

Maintenant que votre **Git** est configuré. Vous pouvez ramener le `dépôt` en local en utilisant la commande `git clone`. Veillez à ne pas appliquer ce clone dans le dépôt créer en début de TP, mais dans un autre dossier.

L'adresse $<$ *repo url* $>$ se trouve sur la page d'accueil de votre `dépôt` **Github**. Cliquez sur le bouton vert, **Clone or Download**. Une fenêtre intitulé *Clone with HTTPS* s'ouvre avec une adresse URL. Copiez la et insérez la dans la commande suivante:

```
git clone <repo url>
```

Le `dépôt` est maintenant présent sur votre machine dans le dossier **GestionDeProjet4A**. Au sein de ce dossier vous pouvez constater que le fichier *README.md* est bien présent.

4 Récupérer des modifications.

Nous allons voir maintenant comment récupérer des modifications effectuées sur un dépôt. Cette manipulation est très utile puisqu'elle permet de mettre à jour un projet, sans avoir à le re-télécharger entièrement.

4.1 Effectuer une modification sur Github.

Retournez sur la page d'accueil de votre dépôt **Github** et cliquez sur le crayon en haut à gauche du REAMDE.

Apportez les modifications que vous souhaitez pour décrire à votre manière le dépôt **GestionDeProjet4A**. Cliquez sur *Commit Change*.

4.2 Récupérer les modifications en local

La commande `git pull` est une commande qui permet de récupérer les derniers fichiers mis à jour dans un dépôt.

Depuis votre terminal, déplacez vous dans votre dossier **GestionDeProjet4A** et tapez la commande suivante :

```
git pull origin master
```

NB Paramètres de la commande :

- *origin* : *origin* est un alias désignant le nom de votre dépôt distant. C'est le nom choisi par défaut. Dans ce tuto nous n'utiliserons pas d'autres dépôts.
- *master* : *master* désigne le nom de la branche principale de notre dépôt sur laquelle nous travaillons actuellement. Par définition celle-ci s'appelle toujours *master*. Nous verrons plus tard dans ce TP comment gérer plusieurs branches.

Ouvrez le fichier *README.md* et constatez que les modifications effectuées dans le fichier *README.md* ont bien été prises en compte.

Cette commande est suffisante si vous voulez simplement suivre le développement d'un projet en le mettant à jour régulièrement sans y participez vous même.

4.3 Exercice

Cloner le `dépôt` *https://github.com/wikistat/Exploration.git* dans un dossier en local.

Maintenant que vous avez cloner ce `dépôt`, vous allez pouvoir régulièrement récupérer les mises à jour de ce `dépôt` à l'aide de la commande `git pull` sans avoir à re-télécharger entièrement le `dépôt`.

Cette exercice vous sera également très utile pour suivre les modifications des `dépôts` **Apprentissage** et **AI-Frameworks** l'année prochaine.

5 Effectuer et pousser des modifications.

Vous souhaitez maintenant vous même participer au projet. Pour cela vous souhaitez modifier les fichiers existants, créer de nouveaux fichiers et les envoyer vers le `dépôt` distant.

Effectuez les actions suivantes dans votre dossier **GestionDeProjet4A** :

- Ajoutez une ligne au fichier *README.md*
- Créez un fichier *mon_fichier.txt*

5.1 Observer les modifications effectuées.

A l'aide des commandes `git status` (slide 16) et `git diff` (slide 40). observez les modifications effectuées.

5.2 Ajouter les modifications effectuées au dépôt local.

- Ajoutez les modifications à votre `dépôt` local à l'aide de `git add` (slide 7)

```
git add README.md
git add mon_fichier.txt
```

  Appliquer la commande `git status`. Qu'affiche laé commande?

- Appliquez un commit à l'aide de la commande `git commit` (slide 7)

```
git commit -m "mon premier commit en ligne"
```

  L'option `-m` vous permet d'entrer un message qui va décrire votre commit et l'état de votre projet à cet instant *t*.

Effectuez de nouveau un `git status`. Qu'affiche maintenant cette commande?

5.3 Ajouter les modifications effectuées au dépôt distant.

Rendez-vous à présent sur la page de votre `dépôt` **Github** et cliquez sur l'onglet *commit*.

Vous pouvez observer que le commit que vous venez de réaliser n'apparaît pas. C'est normal, celui-ci à été effectué localement, sur votre poste. Pour pousser ces modifications sur le serveur distant, nous allons utiliser la commande `git push`.

```
git push origin master
```

Actualisez à présent la page commit de votre `dépôt` **Github**. Qu'observez vous?

Vous pouvez alors naviguer entre les différentes version de votre projet en cliquant sur l'identifiant du commit que vous souhaitez retrouver.

5.4 Fichier *.ignore*.

Il peut arriver dans un `dépôt` d'ajouter des fichiers qui vous seront utile dans votre travail mais qui ne seront d'aucune utilité aux autres utilisateurs de votre `dépôt`. Dans ce cas de figure, on utilise le fichier *.ignore*. Dans ce fichier, vous allez écrire les noms des fichiers que vous ne souhaitez pas envoyer dans le `dépôt` distant.

- Créez un fichier *mon_fichier_local.txt*
- Effectuez la commande suivante :

```
git status
```

  Qu'observez vous?

- Créez un fichier *.gitignore* (le point est important car il s'agit d'un fichier caché!).
- Ajoutez le nom du fichier *mon_fichier_local.txt* dans le fichier *.gitignore*
- Effectuez la commande suivante :

```
git status
```

Qu'observez vous?

5.5 Exercice.

Votre `git status` vous informe que des modifications ont été effectuées.

Suivez la procédure décrite aux points 5.2 et 5.3 pour ajouter, commiter et pousser vos modifications.

Attention : A ce stade, demandez confirmation à l'intervenant avant de passer à la suite.

6 Gérer un projet à plusieurs.

A ce stade passez à un binôme par poste.

6.1 Créer un nouveau compte sur Github

Créez un second compte **Github** sur le poste que vous venez d'allumer comme dans la section 2.1.

Attention ne créez pas de nouveau `dépôt`!

Pour la suite de ce TP nous nous référerons à ce nouvel utilisateur comme *utilisateur bis* et au premier utilisateur comme *utilisateur original*.

Les opérations qui vont suivre seront à effectuer à chaque fois sur un seul des deux postes. Assurez vous cependant de continuer à suivre le TP ensemble.

6.2 Partager le dépôt.

- Depuis le compte **Github** de l'*utilisateur original*, rendez-vous sur le `dépôt` **GestionDeProjet4A** et cliquez sur *Send invitation*.
- Invitez l'*utilisateur bis* en utilisant son adresse mail et invitez-le avec les droits d'écriture.
- L'*utilisateur bis* doit alors accepter l'invitation via sa boîte mail. Le `dépôt` est alors accessible à l'*utilisateur bis*.

6.3 Exercice

- Depuis le poste de l'*utilisateur bis*, configurez votre git local (Section 3.1) et clonez le `dépôt` **GestionDeProjet4A** (Section 3.2) .
- Constatez que le dossier **GestionDeProjet4A** est maintenant bien présent sur le poste de l'*utilisateur bis*. Constatez également que les fichiers *README.md* et *mon_fichier.txt* sont bien présents et conformes aux fichiers présents sur le poste de l'*utilisateur original*
- Toujours depuis le poste de l'*utilisateur bis*, ajoutez un nouveaux fichier dans *mon_fichier_bis.txt* dans votre dossier, ajoutez également une ligne dans le fichier *mon_fichier.txt*.
- Ajoutez ces modifications au `dépôt` local et distant.
- Récupérez ces modifications depuis le poste de l'*utilisateur original*.
- Rendez-vous sur l'onglet commit de votre `dépôt` sur **Github**. Qu'observez-vous?

7 Gestion de conflits

Jusqu'à présent vous avez effectué des modifications et poussé celles-ci sur le **dépôt** distant à tour de rôle. Il peut arriver que ces modifications soient effectuées en parallèle et qu'un même fichier se retrouve avec deux versions différentes. Les deux fichiers sont alors en conflit. Nous allons voir dans cette partie comment gérer un conflit.

- Depuis le poste de l'*utilisateur original*, ajoutez une ligne au fichier *mon_fichier.txt* et poussez le sur le **dépôt** distant à l'aide des commandes suivantes :

```
git add mon_fichier.txt
git commit -m "Version original du conflit"
git pull origin master
git push origin master
```

Attention Vous remarquerez qu'une commande `git pull` a été effectuée entre un `git commit` et un `git push`. Il est très important d'effectuer cette commande à ce moment (entre chaque commit et push) afin de ne pas pousser des modifications sans prendre en compte la version présente sur le **dépôt** distant et éviter d'écraser des modifications.

- Depuis le poste de l'*utilisateur bis*, ajoutez également une ligne au fichier *mon_fichier.txt* et effectuez les commandes suivantes :

```
git add mon_fichier.txt
git commit -m "Version bis du conflit"
git pull origin master
```

Comme prévu un conflit à été détecté. Vous observez alors le message suivant sur le terminal :

```
Auto-merging mon_fichier.txt
CONFLICT (content): Merge conflict in mon_fichier.txt
Automatic merge failed; fix conflicts and then commit the result.
```

En effet, une différence a été constatée entre la version de votre projet que vous venez de commiter, et la version présente sur le **dépôt** distant, envoyée par l'*l'utilisateur original*. C'est pour cette raison qu'il est très important d'effectuer la commande `git pull` à ce moment. Ainsi avant d'effectuer votre push, vous devez régler ce conflit.

- Ouvrez le fichier *mon_fichier_.txt*. Celui-ci doit contenir des lignes de cette forme :

```
<<<<<<< HEAD
Phrase ecrite par l'utilisateur bis
=======
Phrase ecrite par l'utilisateur original
>>>>>>> fe40f3ff3e1ead2110815e86c8b8784f3a73cec3
```

Les lignes pour lesquelles les deux versions du fichier différent sont encadrées par les termes `<<<<<<<HEAD` et `>>>>>>> fe40f3ff3e1ead211...`. Elles sont séparées l'une de l'autre par le terme `=======`.

- Prenez une décision : Décidez que finalement la ligne écrite par l'*utilisateur original* est bien plus pertinente. Supprimez le reste et sauvegardez le fichier.

- Validez ce changement et poussez-les sur le `dépôt` distant à l'aide des commandes suivantes :

```
git add mon_fichier.txt
git commit -m "Conflit résolu"
git push origin master
```

- Rendez-vous maintenant sur l'onglet *commit* de votre `dépôt` sur **Github**. Qu'observez vous?

  La gestion de ce conflit est assez facile, car vous n'avez pas vraiment fait évoluer le projet. Mais il peut être beaucoup plus compliqué, notamment quand il s'agit de gérer des versions de différentes branches entre elles.

8 Les Branches

Comme expliqué plus haut, (Section 4.2), nous sommes actuellement sur la branche *master*, branche principale, créée par défaut lors de la création du `dépôt`. Vérifiez ceci grâce à la commande suivante :

```
git branch --list
```

Imaginez maintenant que vous souhaitez implémenter une nouvelle fonctionnalité dans votre projet. Cependant vous ne voulez pas impacter le projet lors du développement de cette fonctionnalité. La solution est de créer une nouvelle branche, qui sera un copie exacte du projet à un moment précis et sur laquelle vous pourrez développer votre fonctionnalité.

8.1 Création d'une branche

Depuis le poste de l'*utilisateur bis*, créez une branche *new-branch-feature* à l'aide de la commande suivante :

```
git branch new-branch-feature
```

Observez l'état de vos branches grâce a la commande :

```
git branch --list
```

Vous observez que la nouvelle branche a bien été créée. Par contre l'asterix * nous signale que nous sommes toujours sur la branche *master*. Pour changer de branche, nous utiliserons la commande `git checkout`. Appliquez les deux commandes suivantes afin de changer de branche:

```
git checkout new-branch-feature
git branch --list
```

Puis revenez sur la branche principale.

```
git checkout master
```

A ce stade, la nouvelle branche a été créée en local, mais celle-ci n'existe pas sur le `dépôt` distant. Pour cela on pousse la nouvelle branche sur le `dépôt` distant.

```
git push origin new-branch-feature
```

Constatez que la nouvelle branche a bien été créée sur le `dépôt` distant.

8.2 Récupérez une branche sur le dépôt distant

Depuis le poste de l'*utilisateur principal* effectuez un `git pull` afin de récupérer les dernières modifications effectuées.

```
git pull origin
```

Un message vous informe qu'une nouvelle branche est présente. Cependant, la commande suivante

```
git branch --list
```

vous informe que la branche n'est pas présente en local. En effet git vous informe qu'un utilisateur a créée une nouvelle branche mais il ne ramène pas automatiquement cette branche en local. En effet, si vous ne voyez pas l'utilité de travailler vous même sur cette branche vous n'êtes pas obligé de la ramener en local. Si en revanche vous souhaitez le faire, il vous faut créer une nouvelle branche à partir de la branche distante que vous souhaitez copier avec la commande suivante :

```
git checkout -b new-branch-feature origin/new-branch-feature
```

Vous recevez alors un message de confirmation de création de cette nouvelle branche a partir de la branche distante.

8.3 Merge

Le merge est la dernière commande de **Git** que nous allons voir dans ce TP. Cette commande permet de combiner le travail effectué sur deux branches différentes.

Ici nous allons nous mettre dans la situation où *l'utilisateur bis* a développé de nouvelles features sur la branche *new-branch-features* tandis que l' *utilisateur bis* a continué de travailler sur la branche principale, *master*.

- Depuis le poste *utilisateur original*, assurez-vous d'être sur la branche *master*. Puis ajoutez une nouvelle ligne au fichier *mon_fichier.txt*. Enfin, poussez vos modifications sur le `dépôt` distant.

  ```
  git checkout master
  git add mon_fichier.txt
  git commit -m "Avancé du projet sur la branche master"
  git pull origin master
  git push origin master
  ```

- Depuis le poste *utilisateur bis*, assurez-vous d'être sur la branche *new-branch-features*. Puis créez un nouveau fichier *ma_nouvelle_features.txt*. Enfin, ajoutez une nouvelle ligne au fichier *mon_fichier.txt*. Puis poussez vos modifications sur le `dépôt` distant.

  ```
  git checkout new-branch-feature
  git add mon_fichier.txt
  git add ma_nouvelle_feature.txt
  git commit -m "La nouvelle feature est prête"
  git pull origin new-branch-feature
  git push origin new-branch-feature
  ```

- Rendez-vous sur la page *commit* de votre `dépôt` **Github**. Qu'observez vous?

- Maintenant que la feature est prête, l'*utilisateur bis* souhaite ajouter sa nouvelle feature sur la branche principale, *master*. Pour cela il va se déplacer sur la branche *master*, récupérer les dernières modifications puis, merger la branche *new-branch-feature*

```
git checkout master
git pull origin master
git merge new-branch-feature
```

  A ce niveau, vous pouvez constater que le fichier *ma_nouvelle_feature.txt* a bien été transféré sur la branche *master*. Par contre, un conflit a été rencontré sur le fichier *mon_fichier.txt*.

- Réglez le conflit comme vu section 7. Cette fois conservez la ligne ajoutée par *l'utilisateur principal* mais également la ligne ajoutée par *l'utilisateur bis*.

- Une fois le conflit géré, confirmez les modifications et poussez le résultats sur le `dépôt` distant avec les commandes suivantes:

```
git add ma_nouvelle_feature.txt
git add mon_fichier.txt
git commit -m "Merge terminé/Conflit résolu"
git pull origin master
git push origin master
```

- Rendez-vous sur la page *commit* de votre `dépôt` **Github**. Qu'observez vous?

- Maintenant que le travail sur cette branche est terminé, vous souhaitez la supprimer.

  – Dans un premier temps on supprime la branche en local :

  ```
  git branch -d new-branch-features
  ```

  – puis sur le serveur distant :

  ```
  git push origin --delete new-branch-feature
  ```