TD 3 — Utilisation distante et collaborative de Git

L'objectif de ce TP est de passer en revue des bases permettant d'utiliser Git *avec un dépôt distant* et de façon *collaborative*. Jusqu'à maintenant, les dépôts Git sont stockés localement sur une machine.

L'idée est d'utiliser Git en manipulant un dépôt présent sur un serveur distant. Ce TD est décomposé en trois parties. En premier lieu, nous allons configurer Git pour le faire fonctionner avec une plateforme de dépôt distan : nous allons utiliser un dépôt Git distant *de manière autonome*, avec une seule personne qui apporte des modifications sur le dépôt. Ensuite, nous nous placerons en binôme pour mettre en place un petit dépôt distant collaboratif afin de simuler un travail en équipe dans des cas simples.

1 Plateformes de gestion de projets

Git s'interface avec des plateformes en ligne, qui fournissent de nombreux outils facilitant la gestion de projets informatiqyes. Il en existe un certain nombre, les plus connues sont :

Github Il s'agit de la plateforme la plus connue. Elle appartient à Microsoft [1], et permet d'héberger gratuitement des projets.

BitBucket propose un accès gratuit, mais limité en nombre de dépôts créés.

GitLab C'est la plateforme qui héberge le logiciel libre éponyme. GitLab est avant tout un logiciel, installable sur n'importe quel serveur web, qui fournit les services permttant d'utiliser Git à distance.

GitLab est de très loin le logiciel le plus performant et le plus flexible pour travailler de façon collaborative avec Git. À ce jour, il reste moins utilisé (en environnement professionnel) que Github et BitBucket. Nous allons donc utiliser Github pour ce TD.

1.1 Configuration de la liaison SSH

1) Rendez-vous sur la page https://github.com/ et créez un compte sur la plateforme, avec votre adresse mail académique.

Adresse mail Github et configuration de Git

Il est important que Git soit configuré avec la même adresse mail que celle utilisée pour votre compte Github. Appliquez la commande `git config --get-all user.email` pour vérifier que votre adresse mail est correcte.

Si ce n'est pas le cas, voir le TD 1 pour modifier les configurations.

Nous allons maintenant établir une connexion entre votre machine locale et votre compte GitHub. La connexion se fait avec le protocole SSH, qui permet de chiffrer les communications de façon très robuste. Il est basé sur un système de clé publique/clé privée.

1. Ce qui pose toutes sortes de problèmes éthiques, économiques et e respect de la propriété intellectuelle

Gardez vos clés SSH en lieu sûr !

Les étapes ci-dessous expliquent comment créer un couple SSH de clé privé/clé publique. Ne partagez jamais la clé privée, et gardez-en une copie. Si la clé privée est perdue ou diffusée par erreur, la clé publique ne sert plus à rien (on peut alors la révoquer).

2) Pour générer une clé SSH, suivez les instructions de ce tutoriel : Generating a new SSH key an adding it to the ssh-agent.

3) Vérifiez que la génération de votre clé SSH a bien fonctionné avec la commande `ls ~/.ssh`.

4) Il faut maintenant associer la clé SSH avec votre compte GitHub, en suivant ce tutoriel : Adding a new SSH key to your GitHub account.

5) Enfin, GitHub fournit une procédure permettant de tester la liaison SSH entre votre machin et la plateforme : Testing yous SSH connection.

1.2 Votre premier dépôt distant

1) Depuis GitHub, créez un dépôt Git, nommé `r203_t3_<nomDeFamille>`.

2) Sur la page principale de votre dépôt, cliquez sur Code > Clone > SSH et copiez l'URL, qui doit être de la forme `git@github.com:<userName>/<depot>.git`

3) Depuis le terminal, placez-vous dans votre répertoire de travail ▸Documents▸R203_QDev▸TD3 et exécutez les commandes :

Terminal

```
$:~/Documents/R203_QDev/TD3/ git clone git@github.com:<userName>/<depot>.git
$:~/Documents/R203_QDev/TD3/ cd r203_t3_<nomDeFamille>/
```

4) Quel est le résultat de la commande `ls` ? Et la commande `git status` ?

1.3 Synchroniser un dépôt distant avec un dépôt local

On travaille (temporairement) depuis la branche principale `master`.

1) Commençons par récupérer des éventuels modifications présentes sur le dépôt distant (en principe, il n'y en a aucune à ce stade) : `git pull`.

2) Ajouter et modifier des fichiers de votre choix dans le dépôt Git local. Effectuer un commit avec les modifications (on utilisera librement toutes les commandes étudiées jusqu'à maintenant).

3) Nous allons maintenant pousser (*push*) les modifications locales vers le dépôt distant : `git push`.

4) Créer une branche `dev`, se placer sur cette branche et appliquer des modifications sur le code *depuis cette branche*. Appliquer la commande `git push origin dev`. Quel est son effet ?

2 Git en équipe

Cette partie est très inspiré de celui proposé par Antonin Callard.

Cet exercice est un exercice collaboratif par groupe de deux étudiant·es. Certaines questions de cet exercice sont à réaliser par l'un·e ou l'autre membre du groupe. Par souci de simplification, nous supposons que les deux membres du binôme se nomment Alice et Bob.

2.1 Initialiser le dépôt distant

Commencez par télécharger le fichier `random-jokes.py` et placez-le dans votre répertoire de travail `R203_QDev‣TD3`.

1) (Alice) Sur Github, créez un nouveau dépôt nommé `random-jokes` et invitez Bob en lui attribuant les bons droits d'accès.

2) (Tout le monde) Dans votre projet sur le portail GitHub, cliquez sur le bouton Clone et clonez le dépôt distant dans votre répertoire de travail local.

3) (Alice) Déplacez-vous dans le dossier `R203_QDev‣TD3‣random-jokes` et initialisez un nouveau dépôt Git. Ajoutez le fichier `random_joke.py` et faites un commit. Entrez ensuite les deux commandes suivantes (pour respectivement ajouter le nouveau dépôt distant au dépôt local, puis envoyer les modifications de la branche master) :

Terminal

```
git remote add origin <url-du-depot-git>
git push -u origin master
```

4) (Bob) Supprimez le dossier `R203_QDev‣TD3‣random-jokes`, et clonez plutôt le dépôt distant en utilisant `git clone <url-clone-with-https> random-jokes`.

2.2 Prendre connaissance du sujet

L'objectif de l'exercice est de compléter le développement du fichier `random_joke.py`, qui permet d'afficher une blague aléatoire. Un exemple typique d'utilisation sera :

Terminal

```
$ # Afficher une blague aléatoire
$ python3 random_joke.py random
I have got a ton of work done today.
 A skele-ton!
$ # Afficher deux blagues aléatoires
$ python3 random_joke.py random -n2
I have got a ton of work done today.
 A skele-ton!
Why are graveyards so noisy?
 -> Because of all the coffin!
$ # Afficher une blague contenant le mot "Papyrus"
$ python3 random_joke.py random "Papyrus"
Papyrus stood by the fire for too long.
 Now he's BONE-dry!
```

2.3 Pour bien commencer

1) (Alice) Complétez le code de la fonction `random_joke()`, qui devra afficher sur la sortie standard une blague choisie aléatoirement dans la liste `list_jokes`. Vous pourrez utiliser la fonction `random.choice(l)`, qui choisit aléatoirement un élément dans une liste donnée.

2) (Alice) En compagnie de votre camarade, ajoutez le fichier `random_joke.py` à l'index et créez un nouveau commit. Utilisez la commande `git log --oneline --all --graph` pour remarquer que votre copie locale du dépôt git est plus avancée que celle du serveur. Pour mettre à jour le serveur, utilisez la commande `git push`.

3) (Bob) En compagnie de votre camarade, utilisez la commande `git log --all --graph` pour comparer l'état de votre dépôt : une première fois maintenant ; puis après avoir utilisé la commande `git fetch` ; puis une dernière fois après avoir utilisé la commande `git pull`.

2.4 Travailler en parallèle

Dans cette section, le but est que les deux étudiant·es travaillent en parallèle afin d'implémenter des fonctionnalités différentes ! Ainsi, un·e étudiant·e réalisera les question marquées « Alice », en même temps que l'autre réalisera les questions marquées « Bob ».

Exercice (Manipulation réalisées par Alice). **1)** (Alice) Complétez la fonction `__filter_joke(regex)`, qui renvoie la liste des blagues contenues dans `list_jokes` qui matchent l'expression rationnelle `regex` donnée en paramètre. Pour cela, décommentez les deux lignes de commentaires et supprimez l'instruction `pass`. Ajoutez les modifications à l'index et créez un commit.

2) (Alice) Modifiez la fonction `random_joke()` en une fonction `random_joke(regex)`, qui devra renvoyer une blague aléatoire qui matche l'expression régulière `regex`. Vous pourrez utiliser la fonction `random.choice(l)` sur le résultat renvoyé par `__filter_joke(regex)`.

3) (Alice) Modifiez votre fonction `main()` pour remplacer l'appel à `random_joke()` par `random_joke("")` (sur l'expression régulière vide), afin que votre programme soit toujours fonctionnel. Ajoutez les modifications à l'index et créez un commit.

4) (Alice) Modifiez la fonction `main()` pour que l'appel `random_joke.py random <regex>` affiche une blague aléatoire sur la sortie standard qui matche l'expression régulière `regex`. Si `<regex>` est fourni, vous pourrez utiliser `random_joke(regex)` ; si `<regex>` n'est pas fourni, vous continuerez à utiliser `random_joke("")`. Vous pourrez consulter si `<regex>` est fournie en paramètre en consultant le contenu de la liste `args`.

5) (Alice) Ajoutez les modifications à l'index et créez un nouveau commit.

6) (Alice) Envoyez tout votre travail sur le serveur distant avec `git push` (attention : si votre camarade a été plus rapide que vous, cela ne fonctionnera pas !)

Exercice (Manipulations réalisées par Bob). **1)** (Bob) Modifiez la fonction `random_joke()` en une fonction `random_joke(k)`, qui devra renvoyer `k` blagues aléatoires (potentiellement avec répétitions) parmi la liste `list_jokes`.

2) (Bob) Modifiez votre fonction `main()` pour remplacer l'appel à `random_joke()` par `random_joke(1)`, afin que votre programme soit toujours fonctionnel. Ajoutez les modifications à l'index et créez un commit.

3) (Bob) Modifiez la fonction `main()` pour que l'appel `random_joke.py random -nk` affiche `k` blagues aléatoires sur la sortie standard. Vous pourrez évidemment faire appel à la fonction `random_joke(k)`.

Indication : Si `-nk` est utilisé dans l'appel au script `random_joke.py`, cette option sera stockée dans la liste locale `options` de la fonction `main`. On rappelle qu'étant donné une chaîne de caractères `s`, il est possible d'en éliminer les deux premiers en utilisant `s[2:]`.

4) (Bob) Ajoutez les modifications à l'index et créez un commit.

5) (Bob) Envoyez tout votre travail sur le serveur distant avec `git push` (attention : si votre camarade a été plus rapide que vous, cela ne fonctionnera pas !)

2.4.1 Résoudre les conflits lors d'un travail collaboratif

Après la fin de la partie précédente, une des deux modifications (ou bien l'ajout du paramètre `-nk`, ou bien l'ajout d'une expression régulière) n'a pas pu être envoyée sur le serveur. Vous allez maintenant vous placer à deux sur l'ordinateur contenant cette modification qui n'a pas été synchronisée et travailler à résoudre le conflit.
Utilisez `git fetch` puis `git log --graph --oneline --all` pour visualiser le problème.
Fusionnez la branche distante `origin/main` dans votre branche principale `main`. Résolvez les conflits dans le fichier `random_joke.py`, puis terminez la fusion avec `git merge --continue`.
Visualisez le graphe git avec `git log --graph --oneline --all`, puis envoyez votre travail sur le serveur distant avec `git push`. Constatez que, cette fois, cela a bien fonctionné. Pour cela, visualisez à nouveau le graphe git.

Sur l'autre ordinateur, visualisez le graphe git puis récupérez le contenu du serveur distant avec `git pull`. Constatez qu'il n'y a pas de conflits, puis visualisez à nouveau le graphe git.

3 Pour approfondir (un peu)

Cet exercice est potentiellement *vraiment très formateur* pour les personnes qui souhaitent mieux visualiser les commandes Git en terme de manipulations du graphe associé ! Rendez-vous sur le site Learn Git Branching et travaillez sur les exercices proposés. Vous pouvez à la fois vous entraîner dans l'onglet « Main » sur les manipulations du graphe Git, et dans l'onglet « Remote » au travail collaboratif !