

Travaux Dirigés

Java Avancé

—M1—

Git, JUnit


Git, EGit

NOTES IMPORTANTES :

- Dans ce TD vous allez créer votre propre dépôt Git sur Github. **Les dépôts Git sont individuels** et peuvent éventuellement contribuer à la note du CC. Si vous travaillez en binôme, vous pouvez réutiliser le code de votre binôme à **condition de noter son nom dans le fichier REPONSES.txt**. Encore une fois : vous êtes le seul responsable de votre dépôt Git.
- Dans l'exercice 1 et 2, nous allons manipuler différentes versions des fichiers de votre projet `td00`. Pour éviter les erreurs et les confusions, ne modifiez pas le fichier `REPONSES.txt` dans le répertoire `td00` pendant ces exercices. Faites en une copie dans votre répertoire *home* (`~`) et modifiez cette copie. À la fin de l'exercice 2, copiez la dernière version du fichier dans `td00`, et ajoutez la dans votre dépôt git.
- **Attention :** soyez très attentif avec les commandes Git, certaines erreurs peuvent être compliquées à résoudre. En cas de doute, demandez à votre chargé de TD.

► Exercice 1. Git en ligne de commande

Git est un outil de versionnage de code source. Il permet de garder un historique des différentes version de votre code source. Plus tard, nous l'utiliserons également comme outil de collaboration.

1. Rendez vous dans votre répertoire `javaavance` et transformez le en *dépôt Git* en utilisant la commande `git init` .
2. Avant chaque nouvelle version du projet, il faut indiquer à Git la liste des fichiers qui ont reçu des changements dignes d'intérêt par rapport à la version précédente. Pour cette première version, il s'agit de l'ensemble des fichiers source du projet. C'est-à-dire, les fichiers qui ne peuvent pas être re-générés à partir d'autres fichiers.

Faites la liste des fichiers qui doivent faire partie des sources du projet `td00`. Discutez en avec votre chargé de TD. Donnez des exemples de fichiers présents dans votre répertoire `td00` qui ne font pas partie des sources du projet.

3. Ajoutez les fichiers sources du projet avec la commande `git add fichier1`. Vous pouvez répéter cette commande autant de fois que nécessaire.
4. Effectuez votre premier *commit* en utilisant la commande `git commit`. Le commit va enregistrer les sources de votre projet telles qu'elles sont actuellement en tant que première version de votre projet.

Lorsque vous tapez la commande, Git ouvre un éditeur de fichier et vous demande de fournir une description des changements effectués depuis la dernière version. Comme il s'agit du premier *commit*, entrez le message « initial import »⁽¹⁾. Enregistrez et quittez l'éditeur pour finaliser votre commit.
5. Apportez une modification quelconque au fichier `PrimeCollection.java` (par exemple, modifiez la borne supérieure du générateur aléatoire et faites la passer de 1000 à 2000), sauvegardez le fichier, et rendez vous dans le terminal et tapez `git diff`. Qu'observez vous ?
6. Ajoutez la modification apportée en tapant de nouveau `git add` suivi du chemin vers le fichier `PrimeCollection.java`. Et effectuez votre deuxième commit en tapant `git commit`. Entrez un nouveau message pour décrire les changements apportés, et quittez l'éditeur pour finaliser le commit. **Note :** un bon message de *commit* pour la modification proposée serait : *Increase random generator upper bound*.
7. Consultez l'historique des modifications apportées à votre projet depuis la version initiale en utilisant la commande `git log`. Vous pouvez également utiliser l'interface graphique `gitk` pour visualiser l'historique de votre projet. **Note :** si vous tapez `gitk` dans le terminal, vous n'aurez plus la main sur le terminal jusqu'à la fin de l'exécution de la commande `gitk` (i.e. jusqu'à ce que vous fermiez l'interface graphique). Pour lancer `gitk` tout en gardant la main sur le terminal, vous devez taper la commande `gitk &`.
8. Chaque commit est identifié par un *hash* SHA-1⁽²⁾ calculé sur la dernière version de l'ensemble des fichiers de votre dépôt. Le résultat du hash est séquence de caractères alpha-numériques qui identifie (presque) uniquement votre version du code. À l'aide de la commande `git log`, trouvez et copiez le *hash* de votre premier commit (celui que vous avez normalement décrit par le message *initial import*).
9. En utilisant `git diff hash-commit-1 hash-commit-2`, vous pouvez déterminer la différence apportée entre deux commits. Testez cette commande pour visualiser la différence entre le premier et le deuxième commit.
10. Pour faciliter la tâche, le commit le plus récent est toujours référencé par le mot clé `HEAD`. Il est également possible de faire référence au commit précédent le dernier commit grâce au mot clé `HEAD^`. Reproduisez le résultat de la question précédente sans utiliser les *hash* des commits.

(1). Traditionnellement, les message de commit sont écrit en anglais, vous pouvez écrire en Français si vous préférez mais restez cohérents au cours du développement de votre projet.

(2). <https://en.wikipedia.org/wiki/SHA-1>

Vous connaissez les bases de l'utilisation de git en local. Vous êtes très fortement encouragés à suivre le tutoriel proposé par les créateurs de git disponible ici <https://git-scm.com/docs/gittutorial>. (À la maison.)

► Exercice 2. Synchronization de dépôts Git

Dans cet exercice, vous allez utiliser le service web Github qui permet d'héberger des dépôts git, et de les rendre accessibles partout sur internet.

1. Rendez vous sur github.com et créez un compte si vous n'en n'avez pas déjà un. Vous pouvez utiliser n'importe quel nom d'utilisateur, cela n'a pas d'importance. Github vous demandera de confirmer votre adresse email, il faut donc entrer une adresse email valide.
2. Une fois que vous avez confirmé votre adresse, rendez vous sur la page Github du cours en cliquant sur le lien suivant.

`https://classroom.github.com/a/TwBn9745`

Acceptez l'autorisation, trouvez votre nom dans la liste des étudiants des étudiants et cliquez dessus. (En cas de problème, demandez à votre chargé de TD). Une fois que l'opération est terminée, cliquez sur le lien pour vous rendre sur la page de votre projet (*assignment*) et copiez l'url de votre dépôt git. Elle doit avoir la forme suivante `https://github.com/Dauphine-Java-M1/tds-nom.git`

3. Depuis le terminal, rendez vous dans votre repertoire `javaavance/`, et tapez la commande `git remote add origin <url du depot>`. Cette commande permet de connecter deux dépôt git, celui qui est sur votre disque dur, et celui de git hub.

Pour envoyer les derniers changements de votre projet dans le dépôt Github tapez `git push origin master`. Cela synchronisera les deux dépôts. Relancez `gitk` (ou actualisez le avec `f5`), comment `gitk` représente le fait que les deux dépôts sont synchronisés ?

4. Dans un nouveau repertoire à la racine de votre home, récupérez une copie de votre dépôt Github avec la commande suivante.

```
1 git clone https://github.com/Dauphine-Java-M1/tds-nom.git clone-de-javaavance
```

Faites une modification **au début** du fichier `REPONSES.txt` du repertoire `clone-de-javaavance` et committez ce changement. Envoyez vos changements sur le dépôt git avec `git push origin master`.

5. Faites une modification **à la fin** du fichier `REPONSES.txt` du repertoire original `javaavance`. Tentez d'envoyez vos changements sur le dépôt git avec `git push origin master`. Que se passe t'il ?

6. Toujours depuis le repertoire `javaavance`, effectuez une opération `git pull origin master`. Que contient le fichier `REPONSES.txt`. Observez et interprétez l'historiques des modifications avec `gitk`.
7. À votre avis, que se serait-il passé si vous aviez apporté les changements sur la même ligne du fichier `REPONSES.txt`? Vous pouvez en lire plus à cette adresse <https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/>.
8. Supprimez votre repertoire `clone-de-javaavance` pour éviter toute erreur possible.

► Exercice 3. JUnit

1. Dans Eclipse, ouvrez la classe `AppTest.java`, cette classe a été créée automatiquement par Maven lorsque vous avez créé le projet. Elle est destinée à contenir les tests de la classe correspondente `App.java`. Notez la présence de la `import junit.framework.Test`; en début de fichier, qui permet d'accéder à la classe de la librairie JUnit, qui permet de faire des tests unitaires en Java.
2. Comme vous aviez renommé `App` en `PrimeCollection`, renommez la classe `AppTest` en `PrimeCollectionTest`. (Utilisez la fonctionnalité *refactor* pour vous faciliter la tâche.) Exécutez les tests tel quel en faisant un clic droit, puis `Run as` et `JUnit test`. Qu'observez vous ? Modifiez la méthode `testPrimeCollection()` (anciennement `testApp()`) et remplacez `assertTrue(true)` en `assertTrue(false)` qu'observez vous à présent ? Que fait la méthode `assertTrue` ;
3. Restaurez la méthode et ajoutez dès à présent la classe `PrimeCollection` à votre dépôt git, et committez votre nouvelle version. (Si vous le souhaitez, vous pouvez gérer votre dépôt git directement dans Eclipse, mais soyez attentif à ce que vous faites, et vérifiez votre historique régulièrement).
4. En utilisant la méthode `testPrimeCollection()`, écrivez trois nouvelles méthode de test `test0IsPrime()`, `testTwoIsPrime()` et `test9IsNotPrime()`, qui vérifient respectivement que la fonction `isPrime` de `PrimeCollection` fonctionne correctement lorsqu'on l'exécute avec l'argument 0, 2 et 9.
5. Relancez vos test et vérifiez qu'ils fonctionnent comme convenu. Puis committez la nouvelle version de votre projet