Git(Lab) : pour bien commencer

GitLab

GitLab est un gestionnaire de projets de développements collaboratifs. Il intègre notamment une panoplie d'outils visant à faciliter les différents aspects liés au développement d'une application que sont la gestion des versions du code, la collaboration entre plusieurs contributeurs, la documentation et le partage du projet. Le département d'Informatique de l'Université de Lille met à votre disposition un serveur GitLab que vous pouvez utiliser librement pour réaliser vos TPs et projets en collaborant efficacement avec vos camarades. Ce service est disponible à l'adresse suivante : `http://gitlab-etu.fil.univ-lille1.fr` et vous pouvez vous y connecter avec vos codes d'accès habituels. Vous pouvez également utiliser ce service pour rendre vos TPs et projets à votre chargé de TD/TP. Attention, les dépôts que vous avez créés sont automatiquement supprimés *en fin d'année*.
Cette fiche vous fournit donc un petit exemple d'utilisation de GitLab pour vous aider à débuter vos projets de développement sur de bonnes bases. Pour des explications sur le fonctionnement de GitLab ou des usages avancés, vous pouvez consulter la documentation en ligne de GitLab (`https://about.gitlab.com`)

Étape n°1 : Créer son dépôt distant

Connectez-vous sur l'interface en ligne de GitLab puis cliquez sur New project. Indiquez un Project name pour votre projet, par exemple `noms-des-binomes-COO-Projet`[1], cochez qu'il s'agit d'un projet Private, puis validez en cliquant sur Create project. Vous pouvez ensuite donner des droits à votre binôme sur ce dépôt pour qu'il puisse accéder et contribuer au code que vous allez développer collaborativement. Pour ce faire, cliquez sur le menu Members (en haut à droite, au niveau de la roue crantée), renseignez l'identifiant de votre binôme (son *login*), changez les droits d'accès pour Master, puis cliquez sur le bouton Add users to project. Renouvelez l'opération pour enregistrer votre chargé de TP/TP en tant que Developer afin qu'il puisse suivre et annoter vos développements.

Étape n°2 : Synchroniser son dépôt local

Votre projet est désormais créé sur le dépôt distant et vous allez pouvoir le synchroniser avec votre machine. Avant cela, afin de faciliter l'authentification et vous éviter de saisir votre mot de passe lors de chaque opération, vous pouvez enregistrer votre clé publique SSH qui sera utilisée par GitLab pour vous authentifier. Pour ce faire, cliquez sur le menu Profile Settings puis sur l'onglet SSH Keys. Dans le champ Key, copiez votre clé publique SSH en faisant un copier/coller du contenu retourné par la commande:

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAklOUpkDHrfHY17SbrmTIpNLTGK9Tjom/BWDSU
GPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7xlELEVf4h9lFX5QVkbPppSwg0cda3
Pbv7kOdJ/MTyBlWXFCR+HAo3FXRitBqxiX1nKhXpHAZsMciLq8V6RjsNAQwdsdMFvSlVK/7XA
t3FaoJoAsncM1Q9x5+3VOWw68/eIFmb1zuUFljQJKprrX88XypNDvjYNby6vw/Pb0rwert/En
mZ+AW4OZPnTPI89ZPmVMLuayrD2cE86Z/il8b+gw3r3+1nKatmIkjn2so1d01QraTlMqVSsbx
NrRFi9wrf+M7Q== toto@laptop.local
```

Si jamais le système d'exploitation vous indique que le fichier n'existe pas, alors vous devez créer une paire de clés publique/privée en utilisant la commande suivante :

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/toto/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/toto/.ssh/id_rsa.
Your public key has been saved in /Users/toto/.ssh/id_rsa.pub.
The key fingerprint is:
43:c5:5b:5f:b1:f1:50:43:ad:20:a6:92:6a:1f:9a:3a toto@laptop.local
```

Une fois votre paire de clés SSH créée, vous pouvez copier le contenu de la clé publique (le fichier `.pub`) dans GitLab pour qu'il vous reconnaisse ensuite. Attention, il est possible que GitLab prenne quelques secondes ou minutes pour reconnaître votre clé publique SSH. Vous pouvez créer et enregistrer dans GitLab plusieurs clés publiques SSH selon que vous travaillez sur une machine de l'Université ou chez vous, avec votre machine personnelle.
Vous pouvez maintenant synchroniser votre dépôt local avec le dépôt distant créé sur GitLab. Pour ce faire, dans un terminal, positionnez vous dans le répertoire du projet puis exécutez les commandes suivantes :

[1] Pour éviter des temps de réponse trop longs lors des opérations sur votre dépôt, il vous est **fortement déconseillé** de nommer votre dépôt *COO-quelque-chose*. Commencer par votre nom permet de garantir qu'il y aura peu de dépôts qui commenceront pas les mêmes caractères et donc des temps de réponse faibles lors des manipulations.

```
$ProjetCOO$ git init
Initialized empty Git repository in /Users/[MON_LOGIN]/COO/noms-des-binomes-COO-Projet/.git/
$ProjetCOO$ git remote add origin git@gitlab-etu.fil.univ-lille1.fr:[MON_LOGIN]/noms-des-binomes-COO-Projet.git
```

La première commande permet d'activer Git sur votre répertoire. Vous avez donc créé un dépôt Git local qui est considéré comme vide car aucun fichier n'a été enregistré pour l'instant. La deuxième commande configure l'adresse du dépôt distant (celui de GitLab) sur lequel vous pourrez partager votre code.

Étape n°3 : Ajouter des fichiers

Un dépôt Git est destiné à héberger du code source et non pas des artefacts compilés. Pour éviter de stocker de tels fichiers par erreur, nous allons enregistrer un fichier `.gitignore` qui va indiquer à Git les fichiers qu'il doit ignorer. La façon la plus simple de générer ce fichier est de vous rendre sur le site `https://www.gitignore.io`, puis d'indiquer que vous réalisez un projet `Java` en utilisant `Eclipse`, de cliquer sur le bouton `generate` et de copier le contenu qui vous est retourné dans un fichier `.gitignore` stocké à la racine de votre projet :

```
$ProjetCOO$ touch .gitignore
$ProjetCOO$ vi .gitignore
```

Pour enregistrer dans Git les fichiers sources et répertoires que vous créerez lors du développement de votre projet, il vous faut d'abord les ajouter au dépôt en utilisant la commande :

```
$ProjetCOO$ git add .gitignore src/
```

Cette commande enregistre le répertoire `/src` (et le fichier .gitignore) et tout son contenu dans Git. Il faudra en faire de même avec l'ensemble des répertoires à enregistrer, comme `/test` par exemple. Ensuite, nous allons lui donner un numéro de version en exécutant la commande :

```
$ProjetCOO$ git commit -m "feat: Bootstrapping the project."
[master (root-commit) 7336815] feat: Bootstrapping the project
 2 files changed, 90 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 src/main/java/fil/coo/App.java
```

Le message de *commit* passé en paramètre n'est pas à négliger car il doit synthétiser la nature de la modification qui a été apportée sur le code du projet. Il existe des conventions que nous vous encourageons fortement de suivre : `https://github.com/angular/angular.js/blob/master/CONTRIBUTING.md#commit`

À cet instant, si vous vous connectez sur l'interface de GitLab ou que votre binôme essaie de cloner votre dépôt (ou celui de votre binôme) en utilisant la commande :

```
git clone git@gitlab-etu.fil.univ-lille1.fr:[MON_LOGIN]/noms-des-binomes-COO-Projet.git
```

Il récupérera un projet vide. En effet, les fichiers sont versionnés par Git sur votre machine mais pour autant les fichiers ne sont toujours pas visibles dans l'interface de GitLab. Pour ce faire, il faut donc synchroniser votre dépôt local avec le dépôt distant en exécutant la commande :

```
$ProjetCOO$ git push -u origin master
Counting objects: 2, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (15/15), 1.58 KiB | 0 bytes/s, done.
Total 15 (delta 0), reused 0 (delta 0)
To git@gitlab-etu.fil.univ-lille1.fr:[MON_LOGIN]/noms-des-binomes-COO-Projet.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

Étape n°4 : Récupérer des fichiers

Les fichiers sont désormais disponibles en ligne et votre binôme peut les récupérer en exécutant la commande :

```
$ProjetCOO$ git pull
remote: Counting objects: 2, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 15 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (15/15), done.
From gitlab-etu.fil.univ-lille1.fr:[MON_LOGIN]/noms-des-binomes-COO-Projet
 * [new branch]      master     -> origin/master

$ProjetCOO$ ls
src
```

Pour résumer

À chaque fois que vous souhaiterez partager un fichier avec votre binôme, il suffira donc de l'éditer localement puis d'exécuter la séquence de commandes :

```
$ProjetCOO$ git add mon_fichier.ext mon_repertoire/
$ProjetCOO$ git commit -m "Mon message de commit"
$ProjetCOO$ git push
```

Et quand vous voudrez récupérer les modifications qui ont été enregistrées sur le dépôt distant, il vous suffira d'exécuter la commande :

```
$ProjetCOO$ git pull
```

Bien entendu, Git ne se limite pas à ces quelques commandes. N'hésitez pas à consulter la documentation de Git pour découvrir son fonctionnement et les différentes commandes utilisables : `https://git-scm.com/documentation`.

Projets

Les rendus de vos projets se feront obligatoirement et uniquement via votre dépôt Gitlab.
Vous ajouterez systématiquement à votre projet un fichier `README.md` (syntaxe markdown possible et recommandée) sur la branche master. Celui-ci aura pour objectif de présenter votre projet de manière exhaustive. En particulier, ce document doit être utilisé pour présenter et justifier vos choix de conception, les fonctionnalités que vous avez mises en œuvre, ainsi que les limitations que vous avez identifiées. Il doit également fournir les explications nécessaires à la bonne compréhension et exécution de votre programme (doc, tests et jar exécutable).
Un `java -jar target/xxx.jar` doit ensuite suffire à l'exécuter. Pour l'évaluation de votre projet, votre enseignant exécutera en premier lieu ces commandes (javadoc, exécution des tests, exécution du jar). Vous serez fortement pénalisé si elles ne fonctionnement pas correctement.

Foire Aux Questions

1. Doit-on obligatoirement utiliser GitLab pour rendre nos projets de COO ?

Oui.