TD numéro 4 - Git Pratique - Travail coopératif

Git #3

But

Le but est de vous faire manipuler git en situation de travail coopératif, c'est à dire d'utiliser git pour partager les fichiers d'un projets sur lequel vous travaillez à plusieurs, en même temps.
L'objectif est à la fois de vous apprendre les commandes et les mécanismes de git utiles dans cette situation, et aussi de vous entrainer au workflow que l'on souhaite vous voir utiliser (plusieurs développeurs, un intégrateur).

Afin de rendre les choses un peu intéressante, vous ferez ce TD non pas en binome mais en trinome ou quadrinome, suivant le nombre de personnes dans votre rangée. D'abord, cela pimentera un peu les choses (plus on est nombreux, plus c'est compliqué de se coordonner), ensuite, cela ressemblera plus aux situations réelles du reste de la licence (projet tutoré, projet php, ateliers, …).

Comptes

La première étape est de vous créer un compte sur [bitbucket] (https://bitbucket.org). Bitbucket, semblable à Github, vous permet de créer des projets et de partager des repository git. L'avantage par rapport à Github est la possibilité, avec un compte gratuit, de créer des projets privés.

Commencez donc par vous créer un compte personnel, si vous n'en avez pas déjà un. Vous pouvez utiliser comme adresse email l'adresse de l'université.

Clef ssh

La seconde étape, une fois votre compte créé, est de déposer votre clef ssh publique sur bitbucket.

Lorsque vous allez faire un push sur bitbucket, ce dernier va vous demander de vous authentifier, afin de vérifier vos droits. Cette authentification peut se faire de plusieurs manières, mais la manière la plus simple est d'utiliser votre clef ssh, car dans ce cas l'authentification est automatique, à condition d'avoir déposé préalablement votre clef publique.

Le concept est assez simple : une clef ssh est composée de 2 parties, une clef privée (qui reste chez vous) et une clef publique (à déposer sur bitbucket). Lorsque vous faite un push, l'information est codée avec votre clef privée, et comme le décodage ne peut se faire que par votre clef publique, c'est le décodage de l'information qui tient lieu d'authentification.

Le couple clef publique / clef privée se trouve dans le répertoire .ssh à la racine de votre répertoire home.

Commencez d'abord par créer votre clef ssh (composée en fait d'une clef privée et d'une clef publique) :

```
ssh-keygen
```

Acceptez le nom proposé pour le fichier qui va contenir la clef.
Entrez un mot de passe pour protéger votre clef. C'est un mot de passe que vous devrez retenir, mais il n'a pas à être identique à celui utilisé pour vous logger sur votre session linux.

Une fois ceci fait, vous pouvez vérifier que votre clef a bien été créée : vous devez trouver deux fichiers dans votre répertoire .ssh, un fichier **id_rsa** (la clef privée) et un fichier **id_rsa.pub** (la clef publique).

Sous linux, il y a une manipulation à faire pour enregistrer la clef :

`ssh-agent /bin/bash`

`ssh-add ~/.ssh/id_rsa` (entrez ici votre mot de passe de clef, comme demandé)

`ssh-add -l` (doit vous affichez la clef que vous venez d'enregistrer)

Sous MacOS X, l'enregistrement se fait automatiquement la première fois que vous utilisez la clef.

Il ne vous reste plus qu'à vous connecter sur votre page bitbucket, à aller dans *Manage account* > *SSH keys* et à coller dans le champs de texte “Key” votre clef publique, c'est à dire le contenu du fichier .ssh/id_rsa.pub

Une dernière remarque : pour que l'authentification se fasse de manière automatique à chaque fois, pensez bien, au moment de cloner un repository, de choisir le protocole **ssh** et non pas **https**.

Vous pourrez retrouver tous les points importants de la recette sur la page suivante :
[https://confluence.atlassian.com/bitbucket/set-up-ssh-for-git-and-mercurial-on-mac-osx-linux-270827678.html]
(https://confluence.atlassian.com/bitbucket/set-up-ssh-for-git-and-mercurial-on-mac-osx-linux-270827678.html)

Initialisation

Désignez dans votre équipe la personne qui initialisera le repository. Seule celle-ci suivra les instructions suivantes. Les autres peuvent passer directement au paragraphe suivant (**“Equipe”**)

Téléchargez la version web du livre **pro git** que vous trouverez ici :
[https://git-scm.com/book/fr/v2] (https://git-scm.com/book/fr/v2)
(la version web correspond à l'icone violette *“html 5”*)
Cela constituera le contenu de votre repository.

Une fois le fichier téléchargé et décompressé, initialisez git dessus :

```
git init
git add .
git commit
```

Ensuite, connectez-vous à votre page bitbucket, et créez un nouveau repository **privé**.
Vous pouvez suivre les consignes sous la rubrique *“I have an existing project”*, mais **attention**, comme il faut utiliser le protocole **ssh** et non pas **https**, n'oubliez pas de changer l'url du remote en conséquence.

Normalement, cela doit ressembler à :

`git remote add origin git@bitbucket.org:<login>/<repo_name>.git` (ajout du repository distant)

`git push -u origin --all` (push toutes les branches locales – il n'y a que *master* pour l'instant – sur le repo distant, en gardant le tracking)

Pour tester si tout a bien fonctionné, allez dans un autre répertoire et essayez de cloner le repository que vous venez de créer.

Equipe

Afin de permettre à tous les membres de votre tri/quadrinome de cloner le repository qui a été créé précédemment (et qui est sensé être privé), il vous faut créer une équipe.

Ceci est la responsabilité de la personne qui a créé le repository sur bitbucket. Il faut donc que cette personne crée une nouvelle équipe pour ce projet, et ajoute les différentes personnes du groupe à cette équipe. Tout se passe à travers le site web de bitbucket, bien entendu.

Une fois l'équipe créé, chaque membre de l'équipe doit cloner le repository sur son poste.
Pour cela, allez sur la page bitbucket du projet, choisissez l'url correspondante dans le champs prévu à cet effet (pensez bien à prendre le protocole **ssh**), et faites sur votre poste local :

```
git clone <url>
```

Afin de vérifier que tout fonctionne correctement, **CHACUN A VOTRE TOUR** :

- faites un `git pull` pour être sur d'être à jour
- modifiez un fichier quelconque
- faites un `git commit -a`
- faites un `git push` pour envoyer la modif sur le repo bitbucket

Respectez bien le tour de chacun, afin d'éviter des conflits intempestifs (pas le sujet ici).

Normalement, chacun devrait avoir le droit de pusher des modifs sur le repo bitbucket. Si tel n'est pas le cas, cherchez d'ou vient l'erreur et corrigez-là.

Branche locale

Il est temps maintenant de mettre en place le workflow expliqué en cours.
Vous allez chacun à votre tour prendre le rôle de l'intégrateur.

Avant cela, comme dit en cours, vous allez chacun travailler dans votre branche locale. Chacun doit donc créer une branche locale (`git checkout -b`) ayant comme nom son nom d'utilisateur sur bitbucket (plus falice à repérer).

Comme cette branche locale doit pouvoir être mergée avec *master* par l'intégrateur, il faut bien sur que l'intégrateur ait accès à cette branche, donc qu'elle soit aussi pushée sur le repo bitbucket.

Cela se fait avec la commande suivante :

```
git push -u origin <nom_de_la_branche>
```

Une fois que **chaque** membre de l'équipe aura fait cela, n'oubliez pas **tous** de faire un `git pull` pour récupérer ces branches.

Vous pouvez aussi vérifier sur l'interface web de bitbucket que les branches ont bien été créées et pushées.

Intégrateur

Chacun à votre tour, vous allez prendre le role d'intégrateur :

- chacun fait des modifs dans sa branche (modifier un fichier, ajouter les modifs à l'index, commiter).
- chacun push ses commits quand il le souhaite (puisque cela reste dans sa branche)
- lorsqu'un développeur considère qu'il a poussé une ou plusieurs modifs intéressante, il fait un “pull request” vers l'intégrateur (utilisez l'interface web de bitbucket)
- l'intégrateur a en charge le merge de la branche du développeur avec la branche *master*

N'oubliez pas, en cas de conflit, une fois le conflit résolu, de faire un `git add` et un `git commit` pour indiquer à git que le conflit a été résolu.

Une fois le merge fait, n'oubliez pas de pusher le commit de merge.

Vous pouvez aussi tester la revue de code lors d'un pull request, ainsi que le refus d'un pull request : dans ce cas le développeur doit modifier sa branche, et resoummettre un pull request

Mettez bien en pratique le workflow, n'hésitez pas à tester des cas compliqués (grosses modifications simultanées sur les mêmes fichiers, suppressions de fichiers d'un coté et ajouts de l'autres, etc).
Il vaut mieux tester ces cas maintenant, en TD, là ou l'on peut encore vous aider, plutot que lorsque vous serez en plein développement de projet, la tête dans le guidon…