TP

Gestionnaires de version: Git

L'objectif de ce TP est de se familiariser avec l'utilisation de Git.

1 Déroulement du TP

- Le TP est à faire par groupe d'**au plus deux personnes**.
- **Les solutions ne pourront pas être partagées entre les groupes**
- **Rendu** : A la fin de ce TP, vous devez rendre une archive du contenu de votre dépôt Git tel qu'à la fin de la partie 4.
- Les archives devront être déposées sur Moodle pour le **Vendredi 2 Février**.

2 Prise en main

De nombreux tutoriels sur Git sont disponibles sur Internet. Pour un premier contact avec Git, nous nous proposons de faire le tutoriel suivant (en anglais) :

`http://try.github.io`

Prenez le temps de bien comprendre ce qu'il se passe et de faire le lien avec ce qui a été présenté en cours.

Si vous êtes allergiques à l'anglais, le tutoriel `http://rogerdudler.github.io/git-guide/index.fr.html` peut être une alternative.

3 Configuration

Si ce n'est pas encore fait, commencez par mettre à jour la configuration de Git :

- Mettre à jour votre nom d'utilisateur
  `git config --global user.name <votre nom>`
- Mettre à jour l'adresse Email
  `git config --global user.email <votre adresse mail>`
- Définissez votre éditeur de texte favori
  `git config --global core.editor emacs`

Vous pouvez vérifier que vos changements ont bien été pris en compte en affichant le contenu du fichier `gitconfig` :

```
cat ~/.gitconfig
```

4 Appliquons ce que nous avons appris (et plus encore)

Dans cette partie vous allez maintenant dérouler un scénario. C'est à vous de décider quelles commandes utiliser pour mettre en œuvre ce scénario.

Dans notre scénario, nous allons avoir un dépôt *serveur* et un dépôt *local*. Les dépôts seront en fait hébergés sur la même machine (votre machine de travail) mais dans des répertoires différents. Notre dépôt va servir à stocker des fichiers textes, qui constitueront votre compte rendu [1].

1. Il n'est pas nécessaire de travailler la mise en page ou la présentation dans les différents fichiers que vous allez créer. Allez à l'essentiel !

Étape 1 Créez un nouveau dépôt serveur qui sera nommé `devopsTP1.git`

Étape 2 Créez un dépôt local qui vous associerez avec le dépôt distant (serveur) précédemment créé.

Étape 3 A la racine de votre dépôt, créez un fichier `AUTHORS` qui va contenir les noms et prénoms des membres de votre groupe, ainsi que leur adresse Email.

Ajoutez le fichier à votre dépôt, et commitez. Dans votre message de commit, faîtes référence à l'étape à laquelle correspond le commit.

Étape 4 Essayer de *Pusher* vos premières contributions vers le serveur. Vous devriez observer un message d'erreur mentionnant la notion d'`upstream`. A quoi correspond ce concept ? (ne pas hésiter à chercher des informations sur Internet). Corrigez le problème pour pouvoir *pusher* vos contributions.

Étape 5 Créez un répertoire `./CR`. Dans ce répertoire, créez un fichier `initial_steps.txt` dans lequel vous listerez les commandes Git que vous avez utilisées jusqu'ici, et expliquerez la notion d'`upstream`.

Commitez vos nouvelles modifications et pushez vers le serveur.

Étape 6 Créez une nouvelle branche appelée `fork1`. Dans cette nouvelle branche, créez un nouveau répertoire appelé `./branchtest`. Dans ce répertoire, créez un fichier `file1.txt` dont vous avez le choix du contenu.

Commitez vos nouvelles modifications.

A tout moment, vous pouvez utiliser l'utilitaire `gitk` pour visualiser l'historique de votre dépôt. Il est possible que vous ayez à modifier les options de visualisation pour que toutes les branches soient affichées en même temps.

Étape 7 Ajoutez un fichier `branches.txt` dans le répertoire `CR` de la branche `master`. Dans ce fichier, répondez aux questions suivantes :

1. Quelle commande avez vous utilisée pour créer une nouvelle branche ?
2. Quelle commande permet de connaître la liste des branches ?
3. Comment savoir dans quelle branche on travaille ?
4. Comment changer de branche ?
5. Quelles différences constatez vous dans votre répertoire de travail lorsque vous passez de la branche `fork1` à la branche `master`.

Étape 8 *Pushez* le contenu de vos deux branches vers le dépôt serveur.

Étape 9 *Mergez* les contributions de la branche `fork1` vers la branche `master`.

Ajouter un fichier `merges.txt` dans le répertoire CR de la branche `master`, dans lequel vous expliquez comment vous avez effectué le merge.

Étape 10 Depuis la branche `master`, créez une nouvelle branche `fork2`, et déplacez vous dans cette nouvelle branche pour y créer un répertoire `./statusTest`.

Nous allons maintenant étudier la commande `status` qui affiche l'état courant du dépôt et en particulier de l'index. La commande `status` range les fichiers dans 3 catégories principales :

- Changes to be committed
- Changes not staged for commit
- Untracked files

Faîtes les modifications nécessaires dans le répertoire `statusTest` pour que au moins un fichier apparaissent dans chacune des catégories.

Dans la branche `master`, créez un fichier `status.txt` dans le répertoire `CR` qui résume vos explications.

Est ce qu'un fichier peut apparaître dans au moins 2 catégories à la fois ? Expliquez.

Commitez tous vos changements. Attention la branche `fork2` ne doit pas être pushée vers le dépôt.

Étape 11 Nous allons maintenant fusionner les modifications de la branche `fork2` dans la branche `master` mais cette fois ci en utilisant l'opération `rebase`. Une fois le `rebase` effectué, il vous faut faire avancer (*fast-forward*) la branche `master` jusqu'à la dernière version créée. Qu'est ce qu'un *fast-forward* ?

Créez un fichier `rebase.txt` dans le répertoire `CR` de la branche master pour décrire les commandes utilisées ici. Quelles différences observez-vous par rapport à l'utilisation de `merge` précédemment ? Incluez une description de la notion de *fast-forward* dans ce fichier

Étape 12 La branche `fork2` devient inutile, vous pouvez donc la supprimer.

Ajoutez l'explication sur la suppression de branches dans le fichier `rebase.txt` et pushez toutes vos modifications vers le serveur.

Étape 13 Dans le futur, nous aimerions que pour notre dépôt git ignore tous les fichiers avec une extension `.pdf`, ainsi que tout le contenu d'un répertoire `tmp` qui serait présent à la racine de votre dépôt.

Effectuez les actions nécessaires pour obtenir ce comportement. Détaillez ces actions dans un fichier `ignore.txt` que vous ajouterez au répertoire `CR`. Pushez tous vos changements vers le serveur.

Pensez à vérifier que vous obtenez bien le comportement attendu.

Étape 14 Nous allons maintenant nous placer dans une situation dans laquelle nous aurons un conflit à gérer.

Vous devez ajouter de nouvelles modifications à la branche `master` ainsi qu'à la branche `fork1` de telle manière que lorsque vous allez merger les modifications de `fork1` dans `master`, il y ait un conflit à résoudre.

Résolvez le conflit pour terminer le merge.

Décrivez les modifications que vous avez faites pour générer le conflit dans un fichier `conflit.txt` ajouté au répertoire `CR` de la branche master. Pushez toutes vos modifications vers le serveur.

Étape 15 Dans la branche `master`, modifiez le fichier `conflit.txt` pour y ajouter la phrase suivante : *"Je ne sais pas comment résoudre les conflits"* et pushez vos modifications vers le serveur.

En réalité, vous savez résoudre les conflits. Il vous faut donc maintenant annuler vos derniers changements. Faîtes le et expliquez dans un fichier `./CR/cancel.txt`. Commitez vos changements.

Étape 16 Modifiez à nouveau le fichier `conflit.txt` pour y ajouter le contenu de votre choix.

... Ah mais non ! Il ne fallait pas modifier ce fichier. Restaurer le contenu dans l'état précédent votre modification. Ajoutez l'explication de cette étape au fichier `./CR/cancel.txt`.

Étape 17 Il me semble que le contenu du fichier `rebase.txt` est incorrect. Vérifiez qui a modifié ce fichier avec la commande `blame`. Vous pouvez utiliser `blame` avec l'option `-e`. Décrivez le résultat de cette commande et l'utilité de l'option mentionée dans un fichier `./CR/blame.txt`.

Finalement, le contenu du fichier `rebase.txt`semble correct, pas la peine de vous facher avec son auteur.

Étape 18 La commande `git stash` est utilisée dans le cas où vous voulez changer de branche alors que vous avez des modifications non committées dans la branche courante.

Dans le fichier `conflit.txt` de la branche `master`, ajoutez un court paragraphe décrivant un cas d'utilisation dans lequel vous auriez besoin de mettre de coté (*to stash*) vos modifications.

Changez ensuite de branche vers `fork1` :

1. Que se passe-t-il ?
2. Comment passez vers `fork1` sans committer vos modifications ?
3. Quel est le contenu du fichier `conflit.txt` lorsque vous revenez dans `master` ?
4. Comment voir la liste des `stash` ?
5. Comment restorer vos modifications ?

Ajoutez un fichier `CR/stash.txt` dans la branche `master` dans lequel vous répondrez aux questions précédentes. Committez et pushez toutes vos modifications, y compris les modifications du fichier `conflit.txt`.

Étape 19 (bonus) La commande `reset` a plusieurs variantes (brièvement introduites en cours). Ecrivez un fichier `CR/reset.txt` dans lequel vous proposerez des scenarios pour tester les variantes suivantes :

- `git reset monfichier`
- `git reset commitID`
- `git reset --soft commitID`
- `git reset --hard commitID`

Pour chaque cas, expliquez le scenario et le résultat attendu.

Étape 20 Avant de terminer le TP, prenez le temps d'ajouter un fichier `feedback.txt` au répertoire `CR` dans lequel vous ajouterez vos suggestions pour améliorer le TP. Toutes les suggestions sont bienvenues !

Étape 21 Il est maintenant temps de créer l'archive pour votre rendu de TP. Pour cela :

1. Pushez toutes vos dernières modifications vers le serveur
2. Créez l'archive à partir du contenu du répertoire `devopsTP1.git`.
   ```
   tar zcvf devopsTP1_nombinome1-nombinome2.tar.gz devopsTP1.git
   ```
3. Pensez à vérifier qu'en extrayant l'archive et en clonant le dépôt obtenu, on obtient bien le contenu de votre TP.
4. Déposez votre compte-rendu sur Moodle à l'aide du lien prévu à cet effet.