TP Git

1 Introduction

> Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par plus de douze millions de personnes.
>
> — https://fr.wikipedia.org/wiki/Git

Lors cette première séance du TPs, nous allons nous initier à l' utilisation du Git et puis à la maintenance d'un répertoire de travail de manière structurée et ordonnée.

2 Premiers pas avec Git

Git, un logiciel de contrôle de versions

Git [4] est un logiciel de contrôle de versions, il permet de sauvegarder l'historique du contenu d'un répertoire de travail. Pour ce faire l'utilisateur doit régulièrement enregistrer (en créant une révision ou *commit*) les modifications apportées au répertoire, il pourra ensuite accéder à l'historique de toute les modifications et inspecter l'état du dossier à chaque révision.
Git a la particularité de permettre de créer une copie d'un répertoire de travail, *working copy*, et de synchroniser entre eux plusieurs copies du même répertoire, permettant la décentralisation du travail.
De plus, Git permet d'utiliser une ou plusieurs *branches de développement* et de fusionner entre elles ces branches.

2.1 Création d'un nouveau dépôt

Nous allons d'abord nous intéresser à l'aspect gestionnaire de versions de Git : comment enregistrer l'historique des modifications apportées à un projet. Pour obtenir un dépôt Git sur lequel travailler, deux options sont possibles :

1. création d'un dépôt vide (typiquement utilisé pour commencer un nouveau projet de développement) ;
2. copie (*clone* dans le langage de Git) d'un dépôt existant pour travailler sur cette copie de travail (typiquement utilisé pour collaborer avec les développeurs d'un projet en cours).

Examinons la première option. Git a plusieurs interfaces utilisateur. La plus complète étant l'interface en ligne de commande (CLI), nous nous servirons de celle-ci.

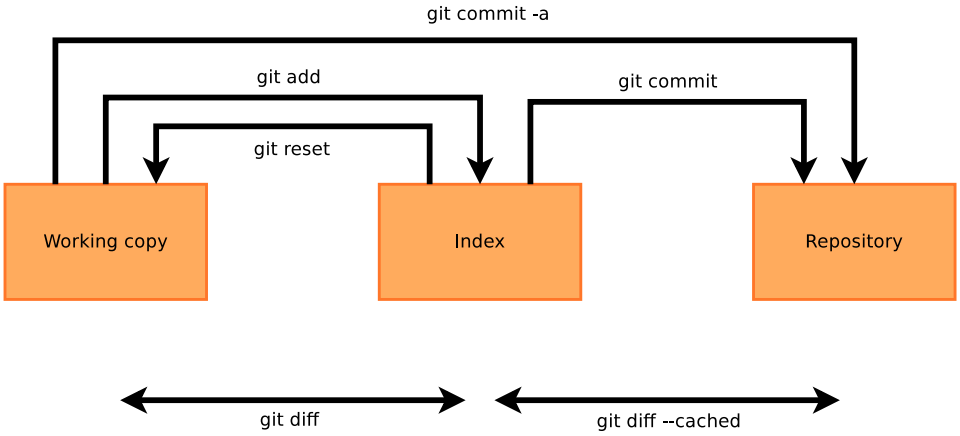
Pour créer un nouveau dépôt, on utilise la commande `git init monrepo`. Cette commande initialise un dépôt Git dans le répertoire `monrepo` (celui-ci est créé s'il n'existe pas). Ce repertoire contient alors à la fois une version de travail (dans `monrepo`) et un dépôt Git (dans `monrepo/.git`).

Question 2.1. *Initialiser un nouveau dépôt* Git *dans un répertoire* `sandwich`*, et créez le fichier* `burger.txt` *qui contient la liste des ingrédients d'un burger, un ingrédient par ligne.*

```
steak
salade
```

```
tomate
cornichon
fromage
```

2.2 Add et Commit



Source : http://thomas.enix.org/pub/conf/git2011/presentation.pdf

FIGURE 1 – `git add commit` *workflow*

Pour être intégrée dans l'historique des révisions du dépôt (pour être "commitée") , chaque modification doit suivre le *workflow* montré en Figure 1 :

1. la modification est d'abord effectuée sur la copie de travail ;
2. elle est ensuite mémorisée dans une aire temporaire nommée *index*, avec la commande `git add` ;
3. enfin, ce qui a été placé dans l'index peut être "commité" avec la commande `git commit`.

`git diff`, selon les paramètres d'appel peut être utilisé pour observer les différences entre les états en Figure 1 ; le format d'affichage est le même de la command `diff -u`.

Question 2.2. *Vérifiez avec* `git status` *l'état dans lequel se trouve votre dépôt. Vos modifications (l'ajout du fichier* `burger.txt`*) devraient être présentes seulement dans la copie de travail.*

Question 2.3. *Préparez* `burger.txt` *pour le commit avec* `git add burger.txt`*. Utilisez* `git status` *à nouveau pour vérifier que les modifications ont bien été placées dans l'index. Puis, utilisez* `git diff --cached` *pour observer les différences entre l'index est la dernière version présent dans l'historique de révision (qui est vide).*

Question 2.4. *Commitez votre modification avec* `git commit -m "<votre_message_de_commit>"`*. Le message entre guillemets doubles décrira la nature de votre modification (généralement* ≤ 65 *characteurs).*

Question 2.5. *Exécutez à nouveau* `git status`*, pour vérifier que vos modifications ont bien été commités.*

Question 2.6. *Essayez à présent la commande* `git log` *pour afficher la liste des changements effectués dans ce dépôt ; combien y en a-t-il ? Quel est le numéro (un hash cryptographique en format SHA1) du dernier commit effectué ?*

Question 2.7. *Créez quelques autres sandwiches* `hot_dog.txt`, `jambon_beurre.txt`*...et/ou modifiez les compositions de sandwiches déjà créés, en commitant chaque modification* séparément*. Chaque commit doit contenir une et une seule création ou modification de fichier. Effectuez au moins 5 modifications différentes (et donc 5 commits différents). À chaque étape essayez les commandes suivantes :*

— `git diff` avant `git add` *pour observer ce que vous allez ajouter à l'index ;*
— `git diff --cached` après `git add` *pour observer ce que vous allez committer.*

Note : la commande `git commit <file>` a le même effet que `git add <file>` suivie de `git commit`.

Question 2.8. *Regardez à nouveau l'historique des modifications avec* `git log` *et vérifiez avec* `git status` *que vous avez tout commité.* Git *offre plusieurs interfaces, graphiques ou non, pour afficher l'historique. Essayez les commandes suivantes (*`gitg` *et* `gitk` *ne sont pas forcément installés) :*

— `git log`
— `git log --graph --pretty=short`
— `gitg`
— `gitk`

2.3 Voyage dans le temps

Question 2.9. *Vous voulez changer d'avis entre les différents états de la Figure 1 ? Faites une modification d'un ou plusieurs sandwiches, ajoutez-la à l'index avec* `git add` *(vérifiez cet ajout avec* `git status`*), mais ne la commitez pas. Exécutez* `git reset` *sur le nom de fichier (ou les noms de fichiers) que vous avez préparés pour le commit ; vérifiez avec* `git status` *le résultat.*

Question 2.10. *Votre modification a été « retirée » de l'index. Vous pouvez maintenant la jeter à la poubelle avec la commande* `git checkout` *sur le ou les noms des fichiers modifiés, qui récupère dans l'historique leurs versions correspondant au tout dernier commit. Essayez cette commande, et vérifiez avec* `git status` *qu'il n'y a maintenant plus aucune modification à commiter.*

`git checkout` est une commande très puissante. Elle vous permet de voyager entre différentes branches (voir plus loin) et aussi de revenir *temporairement* à une version précédente de votre copie de travail.

Question 2.11. *Regardez l'historique de votre dépôt avec* `git log` *; choisissez dans la liste un commit (autre que le dernier). Exécutez* `git checkout COMMITID` *où* `COMMITID` *est le numéro de commit que vous avez choisi. Vérifiez que l'état de vos sandwiches est maintenant revenu en arrière, au moment du commit choisi. Que dit maintenant* `git status` *?*

`git log` n'affiche plus les commits postérieurs à l'état actuel, sauf si vous ajoutez l'option `--all`.

Attention, avec `git checkout` les fichiers de votre copie de travail sont modifiés directement par Git pour les remettre dans l'état que vous avez demandé. Si les fichiers modifiés sont ouverts par d'autres programmes (e.g. un éditeur de texte comme Emacs), il faudra les réouvrir pour observer les modifications.

Question 2.12. *Vous pouvez retourner à la version plus récente de votre dépôt avec* `git checkout master`*. Vérifiez que cela est bien le cas. Que dit maintenant* `git status` *?*

4 Git- **Cheat sheet [1], [2]**

Creation
Cloner localement un dépôt distant `..... git clone https://github.com/<remote>`
Créer un nouveau dépôt local `..... git init`
Changements locaux
Fichiers modifiés dans le répertoire de travail. À utiliser fréquemment ! `..... git status`
Intégrer les changements d'un fichier au prochain commit `..... git add <fichier>`
Retirer un fichier du prochain commit `..... git rm <fichier>`
Envoyer les changements vers le dépôt `..... git commit`

Historique des commits
Afficher tous les commits commençant par les plus récents `..... git log`
Afficher les changements dans le temps pour un fichier spécifique `..... git log -p <fichier>`
Responsable du dernier changement d'un fichier `..... git blame <fichier>`
Mettre à jour et publier
Télécharger toutes les modifications depuis REMOTE et les fusionner / les intégrer à HEAD `..... git pull <remote> <branch>`
Télécharger toutes les modifications depuis REMOTE, mais ne pas les intégrer à HEAD `..... git fetch <remote>`
Fusionner et Versionnage
Fusionner une branche dans le HEAD actuelle `..... git merge <branch>`

Références

[1] git cheat sheet. https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf.

[2] git cheat sheet interactif. http://ndpsoftware.com/git-cheatsheet.html.

[3] git livre. https://git-scm.com/book/en/v2.

[4] git page d'accueil. https://git-scm.com/.

[5] git tutoriel. https://git-scm.com/docs/gittutorial.