TD PG106
Création d'un dépôt *git*

Un dépôt git

Introduction :

Dans ce TD, nous introduisons l'usage de *git* pour le suivi de versions des sources. Les dépôts créés ici pourront être supprimés par la suite, ils vont nous servir à tester des commandes aussi ne réflechissez pas trop aux messages de log (mais faites en sorte de pouvoir identifier facilement celui-ci) ou aux contenus des fichiers du dépôt.

▸**Exercice 1.** `init` :

Commençons par créer un dépôt git *vide sur notre machine locale. Pour cela on utilisera la commande* `git init pg106` *(à faire depuis un répertoire nommé* `td_git`*). Vérifier qu'un répertoire pg106 a bien été créé. Que contient ce répertoire ?*

▸**Exercice 2.** `config` :

La commande `git config` *permet de positionner un certain nombre de propriétés et comportement de* `git`*. Ces propriétés peuvent être au niveau système, utilisateur ou spécifiques au dépôt. Chaque niveau masque le précédent : si une propriété est présente au niveau dépôt, elle prend le dessus sur le niveau utilisateur et système. Les propriétés dépôts sont dans le fichier* `.git/config` *du dépôt, celles au niveau utilisateur dans le fichier* `$HOME/.gitconfig` *et au niveau système dans* `/etc/gitconfig`*.*

Utiliser la commande `git config` *pour positionner la valeur de* `user.name` *au niveau utilisateur et* `user.email` *au niveau du dépôt. Vérifier que les fichiers ont bien été modifiés.*

Remarque : Nous avons fait l'étape de config avant un ajout car si vos propriétés de *user* ne sont pas positionnées, *git* refusera de faire un *commit*.

▸**Exercice 3.** `status`, `add` *et* `commit`

Créer un fichier `LICENCE` *vide dans le dépôt (*`$> touch LICENCE`*) et vérifier que celui-ci est détecté par la commande* `git status`*. Ajouter le fichier au dépôt avec* `add`*, revérifier avec* `status` *et enregistrer cette modification au dépôt avec* `commit`*.*

Modifier le fichier `LICENCE` *et répéter l'opération d'enregistrement.*

Ajouter un fichier `AUTHORS` *au dépôt.*

Tester la commande `git log` *pour visualiser les modifiations passées : quel est l'auteur des modifications ?*

Comment modifier l'auteur du dernier `log` *? Faîtes cette modification et observer l'impact sur l'indentifiant du* commit *(les 40 caractères après le mot* commit*).*

▸**Exercice 4.** *config suite :*

Faites une modificiation (commit) et vérifier dans `git log` *que le changement de configuration ci-dessus est bien actif.*

Quelle propriété permet de choisir l'éditeur de texte pour saisir des messages (lors d'un `commit` *par exemple) ?*

▸**Exercice 5.** *le répertoire* `.git` :

En listant le contenu du répertoire `.git`*, donner la fonction de chaque élément trouvé.*

Modifier la description du dépôt.

Faites en sorte que les fichiers `~` *et* `.o` *soient ignorés par défaut lors d'une appel à* `add` *ou* `status`*.*

▸**Exercice 6.** *hook*

Le répertoire `.git/hooks` *contient des scripts qui peuvent être executés lors de différentes opérations. Dans un premier temps, faites en sorte que le script de* `pre-commit` *d'exemple soit actif : que fait ce script ? Mettez en avant une opération de* `commit` *qui est stoppée par ce script.*

En vous inspirant du contenu du script, faites en sorte que le script affiche l'ensemble des fichiers concernés par le commit. Faire un commit pour vérifier que votre modifiation marche bien.

Quelle option de `git commit` *permet d'éviter l'exécution des hooks ?*

En vous aidant du message affiché par `git status`*, annuler l'ajout de fichier puis supprimer celui-ci.*

▸**Exercice 7.** `gitk` *ou* `gitg`

`gitk` *(ou équivalent comme* `gitg`*) est un outil permettant de visualiser graphiquement les évolutions d'un dépôt git. Utiliser ce programme sur votre dépôt. Par la suite, utiliser ce programme pour avoir un visuel sur les modifications que vous faites sur les dépôts (penser à utiliser F5 pour mettre à jour gitk).*

▸**Exercice 8.** `git clone` *et* `remote`

Si vous utiliser `git remote` *sur le dépôt, vous observez qu'il n'y a aucun dépôt distant connecté.*

Nous allons créer un nouveau dépôt local à partir du dépot précédent. Pour cela créer un nouveau répertoire `dev1` *en dehors du dépôt principal et dans ce répertoire utiliser la commande* `git clone chemin_vers_le_dépot_original`*. Vous avez un nouveau dépôt* `git`*. Dans ce dépôt, utiliser la commande* `git remote -v` *: il y a un lien entre ce nouveau dépot et le dépot d'origine.*

Regarder le contenu du répertoire `.git/hooks` *de ce nouveau dépôt : que voyez-vous ?*

Dans le dépôt d'origine, faites une modification (avec commit). Puis vérifier que vous arrivez bien a récupérer celle-ci depuis le dépôt de dev1 avec la commande `git pull`*. Utiliser* `git log` *et* `gitk` *pour voir l'impact du* `pull` *sur le dépôt.*

On va refaire la même opération mais avec `git fetch` *: Dans le dépôt d'origine, faites une nouvelle modification (avec commit). Dans le dépôt de dev1 avec la commande* `git fetch -v`*. Ici, il n'y a pas eu de fusion avec votre master locale. Vous pouvez basculer dans la branch* `origin/master` *avec* `git checkout origin/master`*. Revenez dans votre branch master locale (*`git checkout master`*). Visualiser avec* `git log --graph --branches` *: on peut voir* `origin/master` *qui comporte une modification non fusionnée. Fusionner les modifications avec* `git merge origin/master`*.*

Dans le dépot de `dev1`*, modifier le fichier AUTHORS et enregister cette modification (commit). Que vous indique maintenant la commande* `git status` *? Essayer de transmettre vos modifications au dépot original avec la commande* `git push`*.*

▸**Exercice 9.** *push*

La solution est de créer un dépôt bare *qui pourra accepter les push. Ce dépôt ne sera pas un dépôt de travail (pas de commit etc).*

Dans un répertoire `server`*, on va créer un dépôt bare vide :* `git init --bare pg106.git`*.*

Dans le dépôt d'origine, on va ajouter ce dépôt en remote *(*`git remote add origin ../server/pg106.git`*). Ceci étant fait, on peut envoyer avec* `git push` *l'état de notre master dans ce dépôt bare (suivez les indications données par* `git`*).*

Répétez cette opération avec le dépôt dev1.

▸**Exercice 10.** branch

Dans le dépôt de dev1, créer une branche appelée `bug1` *avec la commande* `git branch -c bug1` *et vérifier avec* `git branch -l`*. Pour basculer dans cette branche, utiliser la commande* `git checkout bug1` *(il est possible de combiner la création et le basculement avec* `git checkout -b bug1`*).*

Dans la branche bug1, enregister une modification (commit). Basculer dans master (`git checkout master`*) et ajouter/enregister un nouveau fichier* `foo`*. Repasser dans bug1, vérifier que* `foo` *n'est pas présent. Utiliser la commande* `git merge` *pour récuper le fichier.*

Basculer dans master, fusionner les modifications faites dans bug1 et envoyer votre nouvelle version de master au dépôt server.

▸**Exercice 11.** *dev2*

De même que pour dev1, créer un nouveau dépôt dev2 à partir du dépôt d'origine. Est-ce que la branche bug1 est visible dans ce nouveau dépôt ?

Aller dans dev1, basculer dans la branche `bug1` *et publier cette branche sur le dépôt server à l'aide de* `push`

Retourner dans `dev2` *et retrouver cette branche à l'aide de* `git branch -l -a`*. Vous pouvez basculer (*`checkout`*) dans* `bug1` *et participer à cette branche.*

▸**Exercice 12.** *entraînement en ligne...*

Le site internet `https://learngitbranching.js.org/` *permet de s'entrainer au système de branches et d'apprendre de nouvelles techniques (cherry picking et rebase).*

Nous avons vu les principales opérations de base pour utiliser `git`. Il reste encore certaines techniques que nous découvrirons lors des prochains tds.

▸**Exercice 13.** `init`

Créer maintenant un dépôt git pour les sources des tables de hashage que vous avez remaniées lors des tds précédents. C'est dans ce dépôt que nous travaillerons par la suite.

Ajouter les sources (`hash.h hash.c example.c clang-format.txt script_compile.sh`*) ainsi que votre fichier* `README.md`*. Ajuster la* `config` *du dépôt.*