

Table des matières

1	Préambule	2
1.1	Configuration	2
1.2	Git sur internet	2
1.3	Git localement	3
2	Commandes universelles git	3
2.1	Mettre à jour sa copie locale : pull	3
2.2	Lister les modifications locales : status	3
2.3	Ajouter des fichiers ou dossiers au projet : add	3
2.4	Voir les différences versions : diff	3
2.5	Enregistrer les modifications localement : commit	4
2.6	Propager les modifications locales sur le dépôts du serveur : push	4
2.7	Annuler un commit effectué par erreur	4
2.7.1	Modifier le message du dernier commit	4
2.7.2	Annuler un commit sans toucher aux fichiers (soft)	4
2.7.3	Annuler un commit et les modifications de fichier associées (hard)	4
2.7.4	Annuler les modifications d'un fichier avant un commit : checkout	4
2.7.5	Annuler un commit publié	5
2.8	Historique des modifications : log	5
3	Gérer un conflit	5
	Index	6

1 Préambule

Git permet de gérer un projet (de programmation généralement) et de garder en mémoire l'historique de toutes les versions d'un ensemble de fichiers. Il permet de gérer un projet à plusieurs, de programmer afin de pouvoir revenir en arrière, comparer avec d'anciennes versions et cie.

Le principe est d'avoir un serveur git (un seul possible) qui va garder en mémoire l'historique de toutes les versions et un client git (plusieurs possibles) qui vont se connecter au serveur pour mettre à jour la version des fichiers ou en récupérer les dernières versions.

Remarque : Il est possible que le serveur soit lui aussi client, dans le cas où il n'y aurait qu'un seul développeur et qu'on ne souhaite pas passer par internet.

1.1 Configuration

Afin d'avoir la coloration syntaxique, il faut faire :

```
git config --global color.diff auto
git config --global color.status auto
git config --global color.branch auto
```

De même, il faut configurer votre nom (ou pseudo) :

```
git config --global user.name "votre_pseudo"
```

Puis votre e-mail :

```
git config --global user.email moi@email.com
```

Vous pouvez aussi éditer votre fichier de configuration `.gitconfig` situé dans votre répertoire personnel pour y ajouter une section alias à la fin :

```
vim ~/.gitconfig

[color]
    diff = auto
    status = auto
    branch = auto
[user]
    name = votre_pseudo
    email = moi@email.com
[alias]
    ci = commit
    co = checkout
    st = status
    br = branch
```

1.2 Git sur internet

Je vais prendre l'exemple de google code, qui est celui que j'ai choisi et que je suis en train d'apprendre.

Une fois le projet créé (sur la page <http://code.google.com/hosting/createProject>), il faut faire la commande suivante pour récupérer le contenu du projet localement (et pouvoir envoyer les modifs par la suite) :

```
cd Formulaire
git clone https://autiwa@code.google.com/p/autiwa-tutorials/
```

Cette commande permet de récupérer le contenu du projet et de le copier dans un dossier **Formulaire** qui sera créé dans le dossier courant.

DÉFINITION 1 (CLONE)

Opération d'extraction d'une version d'un projet du repository vers un répertoire de travail local.

1.3 Git localement

Le serveur ET le client seront alors sur la même machine. Pour créer un projet, il faut créer un dossier, par exemple dans mon cas un dossier **mercury** dans mon **\$HOME**, puis je fais, dans mon répertoire utilisateur :

```
cd mercury
git init
```

pour un projet que j'appelle **mercury**.

2 Commandes universelles git

Ici, je note les commandes qui sont valables à la fois pour svn installé sur un serveur internet, ou sur une machine locale pour un usage personnel

2.1 Mettre à jour sa copie locale : pull

```
git pull
```

Pour celà, il faut que le dossier dans lequel on se trouve ait déjà été défini comme un dossier git via un *clone* (voir [§ 1.3])

2.2 Lister les modifications locales : status

La commande git status vous indique les fichiers que vous avez modifiés récemment :

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

2.3 Ajouter des fichiers ou dossiers au projet : add

Pour ajouter des fichiers il faut faire :

```
git add latex/ vim/
```

où **latex/** et **vim/** sont deux dossiers existant dans le dossier local de référence

Remarque : Au cas où ça serait pas clair. J'ai créé un dossier **/home/autiwa/Formulaires** grâce à [§ 1.3]. Dans ce dossier, j'ai créé et rempli à la main les sous-dossiers **latex/** et **vim/**. Maintenant, grâce à la commande ci-dessus, je définis ces sous-dossiers comme étant rattachés au projet. En faisant ainsi le contenu est rajouté récursivement.



Cette commande n'agit que sur le répertoire local (la *working copy*). Il faut ensuite appliquer ces changements au dépôt (voir [§ 2.5 on the following page]) pour les valider.

Supposons que vous veniez d'ajouter un fichier à Git avec git add et que vous vous apprêtiez à le « commiter ». Cependant, vous vous rendez compte que ce fichier est une mauvaise idée et vous voudriez annuler votre git add.

Il est possible de retirer un fichier qui avait été ajouté pour être « commité » en procédant comme suit :

```
git reset HEAD -- fichier_a_supprimer
```

2.4 Voir les différences versions : diff

```
git diff
```

Remarque : Il est possible de regarder les différences sur un fichier en particulier.

2.5 Enregistrer les modifications localement : commit

Pour mettre à jour les versions sur serveur à partir des modifications effectuées localement, il faut :

```
git commit -a -m "initialisation"
```

où **"initialisation"** est le commentaire qui décrit la mise à jour et les modifications effectuées.



Un commit avec git est local : à moins d'envoyer ce commit sur le serveur comme on apprendra à le faire plus loin, personne ne sait que vous avez fait ce commit pour le moment. Cela a un avantage : si vous vous rendez compte que vous avez fait une erreur dans votre dernier commit, vous avez la possibilité de l'annuler (ce qui n'est pas le cas avec SVN!).

2.6 Propager les modifications locales sur le dépôts du serveur : push

On peut modifier ces commits là, tant qu'ils sont en local. Puis une fois vérifié. Il ne reste plus qu'à les envoyer sur le serveur. Pour cela on vérifie tout d'abord ce qu'on s'appête à envoyer, et on envoie sur le serveur :

```
git log -p
git push
```

2.7 Annuler un commit effectué par erreur

2.7.1 Modifier le message du dernier commit

```
git commit --amend
```

2.7.2 Annuler un commit sans toucher aux fichiers (soft)

```
git reset HEAD~
```

annule le dernier commit et revient à l'avant dernier. Pour autant les fichiers ne sont pas modifiés, seul le commit en lui même est annulé. Pour annuler les modifications liées au commit, il faut faire un hard reset (section suivante).

2.7.3 Annuler un commit et les modifications de fichier associées (hard)

```
git reset --hard HEAD~
```

Annule le dernier commit et toutes les modifications qui s'y rapportent.

Pour indiquer à quel commit on souhaite revenir, il existe plusieurs notations :

- HEAD : dernier commit ;
- HEAD~ : avant-dernier commit ;
- HEAD^^ : avant-avant-dernier commit ;
- HEAD~2 : avant-avant-dernier commit (notation équivalente) ;
- d6d98923868578a7f38dea79833b56d0326fcba1 : indique un numéro de commit précis ;
- d6d9892 : indique un numéro de commit précis (notation équivalente à la précédente, bien souvent écrire les premiers chiffres est suffisant tant qu'aucun autre commit ne commence par les mêmes chiffres).

2.7.4 Annuler les modifications d'un fichier avant un commit : checkout

Pour annuler les modifications locales et revenir à la dernière révision :

```
git checkout *.f90
```

va remettre en l'état tous les fichiers du dépôt qui sont présents dans le dossier courant (on peut aussi ne préciser qu'un seul fichier).

2.7.5 Annuler un commit publié

Si vous publiez un commit sur le serveur, mais que vous souhaitez l'annuler, c'est quand même possible, mais c'est un peu plus barbare qu'un commit non encore propagé (à l'aide de push).

Pour cela il faut créer un nouveau commit qui annule les modifications du commit que vous souhaitez annuler. Il faut alors connaître l'ID du commit visé et faire :

```
git revert 6261cc2
```

Il faut préciser l'ID du commit à « revert ». Il n'est pas obligatoire d'indiquer l'ID en entier (qui est un peu long), il suffit de mettre les premiers chiffres tant qu'ils sont uniques (les 4-5 premiers chiffres devraient suffire). On vous invite à entrer un message de commit. Un message par défaut est indiqué dans l'éditeur.

Une fois que c'est enregistré, le commit d'annulation est créé. Il ne vous reste plus qu'à vérifier que tout est bon et à le publier (avec un git push).

2.8 Historique des modifications : log

La commande log permet de voir un historique des commentaires de révision, soit de l'ensemble des fichiers, soit d'un fichier en particulier.

```
git log
```

affiche l'historique de toutes les révisions.



Chaque commit est numéroté grâce à un long numéro hexadécimal comme 12328a1bcbf231da-8eaf942f8d68c7dc0c7c4f38. Cela permet de les identifier.

3 Travailler avec des branches

4 Gérer un conflit

Index

Ajouter des fichiers, **3**

commande git

add, **3**

clone, **2, 4**

commit, **3**

pull, **2**

git

status, **3**

Mise à jour du dépôt, **2**

mise à jour local -> serveur, **3**