Les bases de Git

Les commandes de base de Git

I. Démarrer un dépôt Git

Cloner un dépôt existant

```
$ git clone ssh://bob@serveur/git/projet.git
$ git clone https://idul@projets.fsg.ulaval.ca/git/scm/cpp/projet.git
```

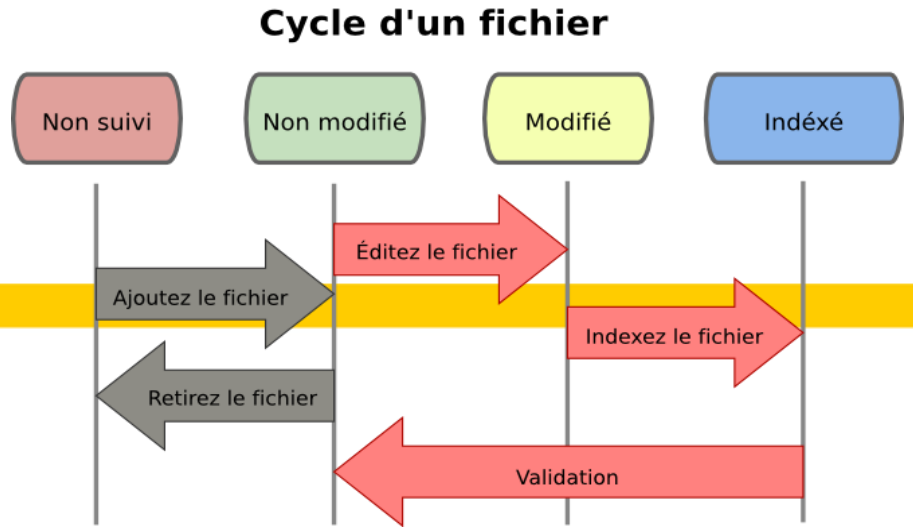
La commande **«clone»** crée une copie d'un dépôt Git existant. Vous clonez un dépôt avec git clone [url]. On a à la fois le clone par l'url *ssh* qui demande une clé public et par *https* qui demande à chaque fois l'authentification.

Les commandes de base de Git

II. Enregistrer des modifications dans un dépôt

Quatre états d'un projet Git:

- Non suivi: fichier n'étant (n'appartenant) pas ou plus géré par Git;
- Non modifié: fichier sauvegardé de manière sûre dans sa version courante dans la base de données du dépôt;
- Modifié: fichier ayant subi des modifications depuis la dernière fois qu'il a été soumis;
- Indexé: idem pour modifié, sauf qu'il sera pris instantané dans sa version courante de la prochaine soumission (commit).



Les commandes de base de Git

II. Enregistrer des modifications dans un dépôt

a. Vérifier l'état des fichiers

```
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#    LISEZMOI
nothing added to commit but untracked files present (use "git add" to track)
```

Untracked files : fichiers non suivis car il n'est pas indexé.

b. Indexer l'ajout ou les changements d'un fichier avant de soumettre (commit) les modifications

```
$ git add -A
```

Les commandes de base de Git

II. Enregistrer des modifications dans un dépôt

c. Valider les modifications

```
$ git commit –m « Mon premier commit»
```

La commande **«commit»** est faite pour valider ceux qui a été indexés avec «**git add**». Pad d'indexe pas de validation.

Après l'option **–m** est suivi d'un commentaire de l'utilisateur décrivant ce qui a été accompli et le fichier est ajouté au répertoire Git/dépôt (local) mais pas encore sur le dépôt distant.

d. Visualiser l'historique des validations

```
$ git log
```

Par défaut, git log énumère en ordre chronologique inversé les commits réalisés. Cela signifie que les commits les plus récents apparaissent en premier.

Les commandes de base de Git

II. Enregistrer des modifications dans un dépôt

e. Pousser son travail sur un dépôt distant

```
$ git push origin master
```

La commande **«push»** sert à envoyer tout les **«commits»** effectués se trouvant dans le répertoire Git/dépôt (HEAD) de la copie du dépôt local vers le dépôt distant.

f. Récupérer et tirer depuis des dépôts distants

```
$ git pull
```

La commande **«pull»** permet de mettre à jour votre dépôt local des dernières validations (modifications des fichiers). La commande est faite avant l'indexation des modifications.

Les branches avec Git

Faire une **branche** signifie diverger de la ligne principale de développement et continuer à travailler sans se préoccuper de cette ligne principale.

La branche par défaut dans Git quand vous créez un dépôt s'appelle ***master*** et elle pointe vers le dernier des commits réaliser.

Pourquoi des branches ?

- ✓ Pouvoir se lancer dans des évolutions ambitieuses en ayant toujours la capacité de revenir à une version stable que l'on peut continuer à maintenir indépendamment.
- ✓ Pouvoir tester différentes implémentations d'une même fonctionnalité de manière indépendante.

Les branches avec Git

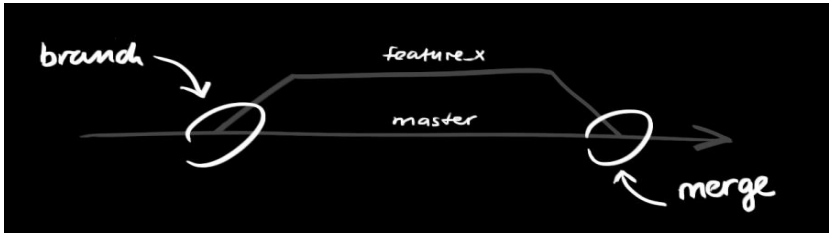
1. **Créer une nouvelle branche nommée « feature_x »**

   ```
   $ git branch feature_x
   ```

2. **Basculer vers une branche existante**

   ```
   $ git checkout feature_x
   ```

   Cela déplace (HEAD) le pointeur vers la branche feature_x. Tous les commits à ce moment sont fait sur la branche courante.



3. **Retourner sur la branche principale**

```
$ git checkout master
```

Les branches avec Git

4. **Supprimer la branche**

```
$ git branch -d feature_x
```

5. **Incorporation des modifications d'une branche dans la branche courante (HEAD) par fusionner (merge)**

```
$ git merge <branche>
```

Fusion d'une autre branche avec la branche active (par exemple master). Il est possible qu'il y ait des conflits à résoudre lors d'un merge.

Les conflits de fusion

Lorsque vous modifiez différemment la même partie du même fichier dans les deux branches que vous souhaitez fusionner, Git ne sera pas capable de réaliser proprement la fusion :

- Aucun "commit" de fusion n'est créé et le processus est mis en pause.
- Vous devez alors régler ces conflits manuellement en éditant les fichiers indiqués par git:
  - Faire *git status* qui donne les fichiers n'ayant pas pu être fusionnés (listés en tant que "**unmerged**").
  - Marquer les conflits comme résolus en faisant la commande *git add <fichier> ou git commit -a*
  - On peut, après résolution de tous les conflits, soumettre les modifications sous forme d'objet **"commit"** de fusion avec *git commit –m « Mon premier commit» ou git push* et terminer ainsi le processus de fusion.

Liens et ressources

Guides

Livre « Pro Git » : http://git-scm.com/book/fr/v1

Apprendre Git: https://www.atlassian.com/git/?utm_source=stash&utm_medium=in-app-help&utm_campaign=learn-git?locale=fr_FR,fr

Les bases de Git : http://djibril.developpez.com/tutoriels/conception/pro-git/?page=page_2