# Le système de gestion de versions : Git

Voici les commandes fréquemment utilisées lors de projets Git.

# Les commandes de démarrage

### Cloner un projet Git

Pour aller chercher un projet Git déjà existant, il faut faire la commande suivante :

git clone J:\\monProjet.git

### Prendre un projet et l’intégrer dans Git

Utiliser les commandes suivantes à partir du dossier où se trouve le projet :

git init

Ceci créera le répertoire .git, qui contiendra des fichiers utiles à Git. Si le projet possède déjà des fichiers, il faut les intégrer à Git comme suit :

git add \* (ou par exemple : git add \*.c)

Ces fichiers sont donc ajoutés à Git, mais ils ne sont pas encore « commiter ».

git commit -am "Premier commit du projet"

### Créer un projet Git « bare »/squelette

La commande suivante permet de faire un projet squelette Git. C’est un endroit où on ne travaille pas directement. On peut cependant y faire des « pull » et des « push ». Pour simuler une approche client/serveur avec Git, le serveur serait un projet « bare ».

git init --bare monProjet.git

Par standard, on ajoute l’extension .git aux projets « bare ».

## Faire des modifications sur les fichiers

Après avoir une copie locale du dépôt (ou « repository »), on peut y faire les opérations courantes :

### Ajouter un fichier

Lorsque vous ajouter un fichier, il n’est pas automatiquement pris en charge par Git, pour cela, il faut faire :

git add .

### Modifier un fichier

Après avoir modifié un fichier existant, vous pouvez constater que git est au courant de cette modification

git status

### Faire un « commit »

Lorsque l’on modifie ou ajoute un fichier, voir les commandes plus hautes, il faut également faire un « commit », permettant d’envoyer les modifications dans le dépôt local.

git commit –am "Added index.html file"

Après un commit, on peut recommencer le processus de modification/ajout de fichiers.

### Ignorer un fichier

Dans certain cas, on veut que git ignore un fichier, et qu’il ne fasse jamais parti du dépôt git.

Étape 1 : créer un fichier qui s’appelle « .gitignore »

Étape 2 : dans ce fichier ajouter le ou les fichiers à exclure.

Exemple :

.class

.out

.exe

Étape 3 : faire un git add du fichier .gitignore et faire un commit

Voilà, maintenant, même avec un git add \*, ces fichiers seront maintenant ignorés.

### Supprimer un fichier

Pour supprimer un fichier physiquement et pour mentionner a Git de ne plus suivre ce fichier, il faut utiliser la commande :

git rm index.html

Évidemment, il faut faire un « commit » pour valider la suppression.

### Pousser les modifications vers une destination

En ayant les droits d’accès, vous pouvez faire pousser vos modifications sur un ordinateur distant.

git push origin master

### Intégrer les modifications d’une source dans le projet local

git pull origin master

Si tout se passe bien, vous aurez la dernière version des fichiers. Autrement, il pourrait y avoir des conflits.

### Les différentes versions d’un fichier

Il arrive que l’on développe une fonctionnalité et que l’on veuille voir la version antérieure d’un fichier. « *Comment était le fichier avant que j’y touche ? »*

git diff nomfichier.java

### Annuler des modifications non « commité » d’un fichier

La ligne suivante permet de revenir à la dernière version « commité » du fichier nomFichier.php

git checkout HEAD monFichier.php

### Revenir à une version antérieure pour un fichier

(*revert to previous revision - terme sous perforce*)

Voici comment revenir à une version antérieure d’un fichier (un commit -1)

git checkout HEAD^ nomFichier.c # Revient au commit précédent

git checkout HEAD^^ nomFichier.c # Revient à deux commits précédents

…

# Les conflits

Les conflits arrivent lorsqu’un fichier modifié localement l’a également été sur la source (master) depuis le dernier pull et que Git n’est pas en mesure de converger les modifications du fichier lors du prochain pull.

Pour les résoudre, il faut :

* Ouvrir le fichier avec un éditeur de texte, modifier et résoudre les conflits. Ceux-ci sont délimités par des <<<<<<<<<<
  + Il faut donc nettoyer le fichier pour avoir la version voulue
* Après avoir enregistré le fichier, il faut obligatoirement faire les commandes :

git add

git commit –am "Fixed conflict" # Vous pouvez utiliser un autre message

Vous avez maintenant résolu le conflit et vous pouvez passer à autre chose.

# Les branches

Le mécanisme de branche avec Git est l’une des fonctionnalités les plus puissantes.

### Scénario

Imaginons la situation suivante :

On vient de terminer le release 1.0 de notre système et celui-ci est distribué aux clients. On continue ensuite à travailler sur la version 1.1, qui est assez différente, lorsque l’on reçoit un appel comme quoi la version 1.0 possède un bogue important. En travaillant avec des branches, il est facilement possible d’appliquer un correctif à la version 1.0 ainsi que dans la version actuelle.

### Concept général

L’idée est de créer une branche master, qui ne sera utilisée qu’uniquement lors des « release » (mise en ligne). Autrement, on travaille sur une branche « dev ». Lorsqu’il faut appliquer un correctif, on créé une branche « bugfix123 », corrige le problème, et l’applique aux branches « master » et « dev ».

On peut créer une branche par la commande suivante :

git checkout -b dev

Ceci va créer une branche nommée « dev ». À partir de ce moment, lorsque l’on fait des *commits*, on modifie cette branche.

Pour revenir à la branche master :

git checkout master

Tous les fichiers reviennent à l’état où on les avait laissés lors du dernier commit sur « master ». On peut ensuite se promener, de branche en branche avec git checkout *nomBranche*.

### Voir les branches existantes

Pour voir les branches existantes, on peut faire

git branch

### Supprimer une branche

Pour supprimer une branche, on fait l’opération suivante :

git branch -d nomBranche

### Faire un merge ou intégrer une branche à une autre

Si on veut prendre une branche et l’intégrer dans une autre, voici comment faire :

Si on est dans la branche « master », on peut faire :

git merge nomBranche

Ceci prendra les modifications de la branche et l’appliquera dans la branche master (où nous sommes actuellement)

## Les tags

Les tags permettent de mettre un descriptif à un commit. Les cas les plus fréquents est pour identifier une version, par exemple : v1.0.0

### Création d’un tag

Après avoir fait un commit, on peut ajouter un tag de la façon suivante :

git tag –a v1.0.0 –m "Première version distribuée du logiciel"

Note : Lors d’un git push, les tags ne sont pas envoyés au serveur automatiquement. Pour y parvenir, il faut exécuter la commande suivante :

git push origin --tags

### Liste des tags disponibles

Voici la commande qui permet de voir les tags créés :

git tag