Introduction à GIT

IPO, Alice Jacquot

Contrôle de version

Un logiciel de *contrôle de version* (ou *gestion de version*) permet de gérer un ou des fichiers qui évoluent avec le temps.

Il permet de retracer l'origine de chaque modification, de rétablir des versions précédentes et permet l'intégration de modifications effectuées en parallèle.

SVN et GIT sont des gestionnaires de versions, mais on en trouve également dans beaucoup de système de fichiers en ligne (type "drive" ou documents partagés)

Pourquoi ?

Sans logiciel de suivi de version plusieurs éléments peuvent être fastidieux :

- partager régulièrement du code en s'assurant de repartir de la bonne version, intégrer ses modifications à la bonne version
- comparer à une version antérieure, en particulier pour voir comment un bug est apparu
- avoir une version déployée sur laquelle des bugs sont corrigés quand une future version est encore en développement, et s'assurer d'intégrer ses corrections à la future version
- ne pas se mélanger entre divers versions d'un projet
- lourdeur mémoire et de gestion si on conserve de nombreuses versions
- ...

Concept

Le principe d’un gestionnaire de version est qu’il gère un document comme “une base” à laquelle est ajouté une suite de modifications.

Il y a un dépôt commun et les contributeurs travaillent sur des versions locales.

Lorsqu’un contributeur a réalisé une modification “qui fait sens” il l’envoie vers le dépôt, qui en garde la trace.

GIT

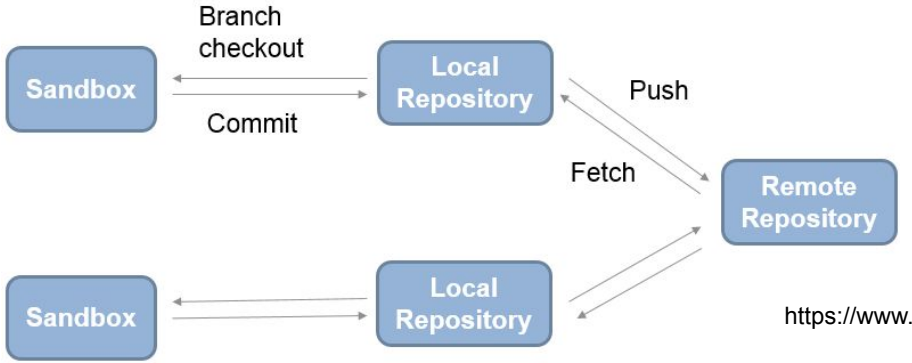
Git est un projet open source, initialement développé par Linus Torvald en 2005.

C’est aujourd’hui le logiciel de contrôle de version le plus répandu, principalement en développement informatique.

GIT : décentralisé

GIT est un gestionnaire de version décentralisé, c'est à dire qu'il fonctionne avec deux niveaux de dépôts : remote et local, en plus de la version de travail (où des modifications sont en cours).

L'idée est d'avoir son propre "brouillon" sur lequel on peut déjà faire du contrôle de versions, et de pouvoir remanier les modifications successives pour les regrouper en un tout plus cohérent avant de les soumettre.



https://www.mathworks.com/help/simulink/ug/push-and-fetch-files-with-git.html

Installer GIT

Suivez des instructions et de configuration pour votre système !

GIT est de base un outil en console, vous pouvez installer des outils graphiques.

IntelliJ intègre GIT, il est préférable de pouvoir le manipuler également en dehors (notamment pour le clone initial du dépôt)

Créer un dépôt initial

La commande git init permet la création d'un dépôt git local, en créant dans le répertoire un fichier .git, à lier à un serveur pour créer le remote…

Ou :

On crée un dépôt git sur github (https://github.com/) que l'on **clone** pour avoir son dépôt local

git clone

git clone <url du dépot> permet de copier un dépôt remote pour créer un local lié à ce remote

On vérifie que le dépôt a bien été créé, et on peut commencer à travailler dessus (sur la branche “master”)

Les branches

Le système de branche permet de gérer plusieurs versions existant en simultanée (développements en cours par exemple).

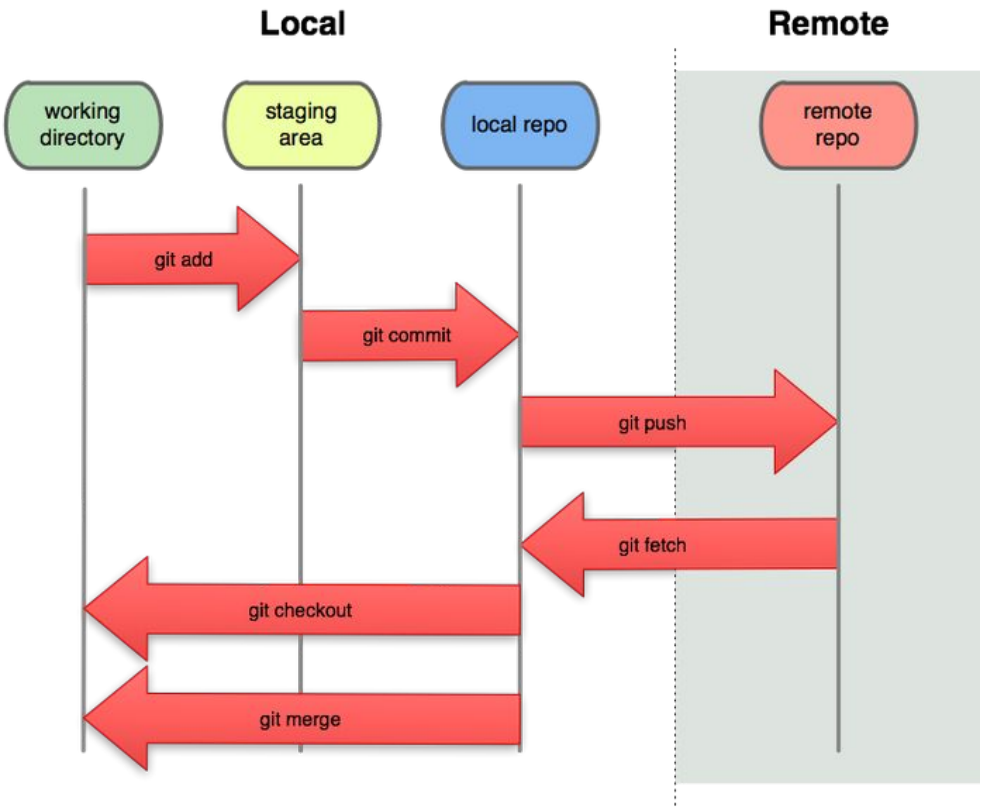
La branche “par défaut” est la branche master.

Sur des petits projets personnels, ce sera souvent la seule utilisée. Sur de gros projet massivement partagés, travailler directement dessus sera souvent vu comme une mauvaise pratique.

Nous n’en parlerons pas dans ce cours mais vous pouvez évidemment les utiliser !

Travailler sous git

Staging area :
l'ensemble des fichiers actuellement pris en charge par le dépôt



https://greenido.files.wordpress.com/2013/07/git-local-remote.png?w=696&h=570

git add

Lorsque vous créez un fichier dans votre version de travail, celui-ci n’est pas automatiquement intégré aux éléments gérés par le dépôt git, il faut pour cela utiliser la commande :

git add *nomDuFichier*

Pour ajouter tout le répertoire courant (par exemple pour intégrer toutes nos classes créés) on utilisera :

git add .

Il faut encore commit et push pour les mettre sur le répertoire distant !

Ignorer des fichiers

On peut créer/modifier le fichier .gitignore pour exclure des éléments sur git (par exemple les .class) en y indiquant des noms de fichiers

But : simplifier l’usage

git commit

La commande

git commit “Message du commit”

permet de déposer les modifications de la version de travail sur la version locale.

Les messages sont obligatoires, ils indiquent le contenu des modifications. Ils doivent être compréhensibles et informatifs, souvent dans des gros projets des normes sont données sur ce qu’ils doivent contenir

Exemple de messages : “lien entre la classe Arbre et la classe Pomme, ajout de méthodes correspondantes” “typo” “correction bug J347 affichage de espace en anglais” “merge dev v3.7 et preprod v3.6”

git push

la commande **git push** permet de pousser (la branche actuelle de) le local vers le remote :
git applique alors successivement tous les commit au remote.

C’est la commande qui “valide” nos changement, il ne faut pas l’oublier !

Il y aura un soucis si notre copie n’est pas à jour ! Il est possible **mais fortement déconseillé** de faire un force push pour écraser la copie distante par notre copie locale. Il faudra surtout commencer par faire un **git fetch.**

Par abus de langage, on entendra souvent “t’as commité ?” quand votre interlocuteur vous demande plutôt si vous avez fait un commit et un push.

git fetch, git checkout, git pull

La commande **git fetch** permet de mettre à jour notre dépôt local en récupérant l’ état courant qui remote. La commande **git checkout** permet de les appliquer à la copie de travail (et donc de les voir).

La commande **git pull** fait les deux à la fois.

Pensez à vous mettre à jour avant chaque session de travail !

C’est la première chose qu’un développeur fait chaque matin en se mettant u travail, dans beaucoup de cas.

git status

La commande **git status** vous permet de connaître l'état courant de vos copies locales (les modifications ont-elles été commitées, les fichiers ajoutés, les commit pushé).

Les interface graphiques (dont intelliJ) vous l'indique souvent par des couleurs et icones.

Le merge

Le ***merge*** est l'action de fusionner deux éléments : on ajoute les modifications de l'un à l'autre. L'opération est, on l'espère, dans la plupart des cas transparente mais des ***conflits*** peuvent apparaître : deux modifications sont vues comme incompatibles (usuellement : on a touché à la même ligne de code). Il faut alors résoudre le conflit, c'est à dire choisir/écrire à la main la version que l'on veut garder pour chaque conflit.

rebase → annule commit