Versioning - Outillage et bonnes pratiques

Rappel

La versioning est un **méthode de gestion de version** qui consiste à historiser les modifications faites dans le code source d'un projet par les développeurs qui y contribuent successivement.

En cas de besoin, il est alors possible de revenir en arrière, comparer des versions antérieures du code, corriger des anomalies, toutes actions de maintenance qui sont liées à un état historique précis (version).

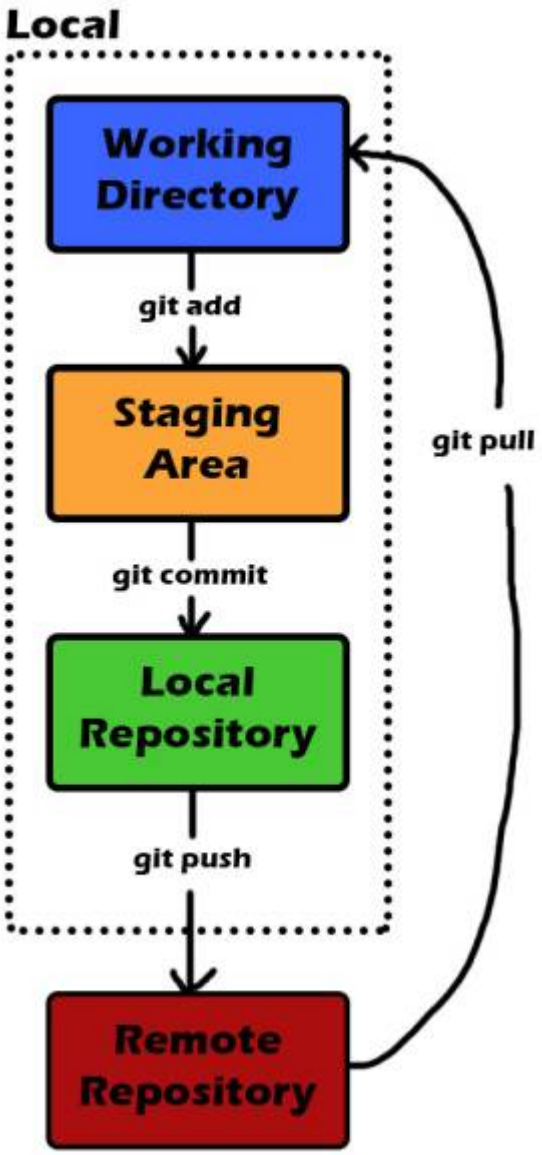
Il existe deux produits largement employées pour le versioning :

- Git : technologie moderne, décentralisée, imaginée par Linus Torvalds (auteur de Linux)
- SVN : implémente la technologie SubVersion, ancienne, rustique et centralisée mais éprouvée

Git

Principes

- **Chaque développeur** possède une copie complète du projet, ce qui permet de travailler hors ligne et de collaborer facilement sans modifier forcement le code du ou des autres développeurs.
- Chaque modification apportée aux différent fichiers est enregistrer lors de **l'envoie sur le serveur Git**, permettant de **revenir à des versions antérieures**, de comparer les changements ou de fusionner les modifications de différents développeurs.
- Une autre fonctionnalité de Git est de permettre de **créer des branches**, des copies indépendantes du projet, pour **développer des fonctionnalités** ou **corriger des bugs** sans affecter le code principal. Les branches peuvent ensuite être remonter dans la branche du projet principal.
- Il est possible de **contrôler l'accès** au différente branche du projet, en définissant des niveaux de permissions pour les différents utilisateurs.



Les principales opérations Git

- **git init** Initialise un nouveau Git dans un répertoire.
- **git add**: Ajoute des fichiers au “staging area”, qui est une zone intermédiaire avant de valider les modifications.
- **git commit**: Valide les modifications du “staging area” dans l'historique du référentiel.
- **git status**: Affiche l'état du référentiel, les fichiers modifiés et les changements non validés.
- **git log**: Affiche l'historique des commits du référentiel.
- **git branch**: Crée, liste ou supprime des branches.
- **git checkout**: Permet de passer d'une branche à une autre ou de revenir à une version antérieure.
- **git merge**: Fusionne une branche dans une autre.
- **git pull**: Télécharge les dernières modifications du référentiel distant et les fusionne dans la branche locale.
- **git push**: Envoie les modifications locales vers le dépôt de projet distant.

Usages

Git est utilisé dans de nombreux contextes, notamment pour :

- **développer des logiciels** collaborer avec d'autres développeurs sur un projet commun et suivre **les modifications apportées** tous au long du projet.
- des **collaboration entre scientifique**, notamment pour gérer les différente données de recherche, les analyses et les publications.

Références

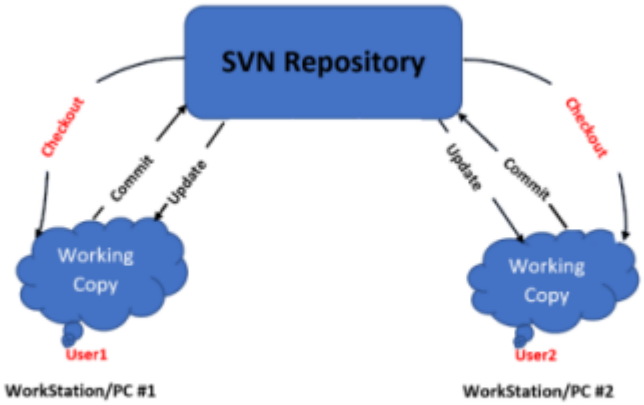
- https://git-scm.com/docs/git/fr
- https://www.jesuisundev.com/comprendre-git-en-7-minutes/

Y compris liens vers des ressources Internet synthétiques

SVN

Les principales opérations SVN

- **CHECKOUT :** Importation sur un client d'un projet mis en partage sur un serveur SVN
- **UPDATE :** Synchronisation du projet, du serveur vers le client (download des mises à jour)
- **COMMIT :** Synchronisation du projet, du client vers le serveur (upload des mises à jour)
- **DISCONNECT :** Déconnexion du partage (avec ou sans suppression des fichiers SVN locaux)



Organisation SVN

Sur un serveur SVN, le projet, les Branches et les Tags sont stockés dans des dossiers spécifiques nommés res-pectivement « trunk », « branches », et « tags ». Ce n'est qu'une convention de nommage que vous êtes libres de respecter ou d'adapter, mais les bonnes pratiques vous conduiront naturellement à adopter ce nommage. L'intérêt des étiquettes sous Subversion est d'utiliser des

noms symboliques plutôt que des numéros de révisions pour se référer à un état précis, comme par exemple ’release-1.1’, plutôt que ’488’. Un nom symbolique permet de revenir facilement à une version identifiée. Pour créer un tag, il suffit de copier l'état actuel d'une version de développement dans un sous-répertoire du dépôt. Les règles suivantes sont communément admises :

- Liste à puceLes différents tags correspondent chacun à un sous-répertoire, lui-même contenu dans un sous-répertoire nommé tags du projet ;
- On ne « commit » pas dans un tag ;

En fonction de l’ampleur et du nombre de projets contenus dans votre dépôt, l’emplacement de ces trois dos-siers peut varier. Il existe en fait deux formes recommandées en fonction de vos besoins :


1. Commun à tous les projets

```
trunk/
  projetA/
  projetB/
tags/
  projetA/
    version1/
  projetB/
branches/
```

2. Propre à chaque projet

```
projetA/
  trunk/
  tags/
  branches/
projetB/
  trunk/
  tags/
  branches/
```

Quelle que soit l’architecture choisie pour les dossiers de base, l’utilisateur crée tous les dossiers intermédiaires librement. Par exemple, la branche « germanVersion » pourra être stockée dans le dossier branches/germanVersion.
À consulter : http://www.lacl.fr/gava/cours/M2/IngLog/annexe3.pdf

Bonnes pratiques

Gestion macroscopique du développement

La gestion de versions n’est pas un mécanisme de sauvegarde, mais un mécanisme de travail collaboratif. Il n’est donc pas question de remonter les modifications « microscopiques » apportées aux fichiers aussi souvent qu’on les enregistre localement, au cours de la mise au point.

On ne remontera que des modifications dont on sait qu’elles sont opérantes au regard d’un besoin global exprimé. Par exemples : « validation W3C des interfaces », « Améliorations fonctionnelles dans la gestion du panier », etc.

Synchronisation descendante

Lorsque l’on fait du versioning « collaboratif », la copie locale du projet sur laquelle on travaille a de grandes chances de ne pas être à jour puisqu’on n’est pas seul à travailler dessus. Pour limiter les risques de conflits et ne pas atteindre les limites offertes par le mécanisme du versioning, il faut systématiquement :

- en SVN, faire précéder un COMMIT par un Update to HEAD ;

- en Git, faire précéder un PUSH par un PULL/Merge ;

Commenter précisément

La gestion de version permet de retracer l'historique des modifications apportées au code. Sans un commentaire précis et significatif, l'historique est inexploitable et la gestion de versions ne présente que peu d'intérêt. **Un effort conséquent est donc à produire dans la rédaction des commentaires de COMMIT**.