Domaine Sciences et Technologies
Master Bio-informatique : DLAD

Aix*Marseille université
Socialement engagée

Année 2023-24

Programmation avancée : Gestion de versions
Code UE : SBICU05

Gestion de versions

Introduction

Lorsqu'on travaille sur le code un peu conséquent, il est utile d'utiliser un outil de gestion de version qui permet en autre de :

- conserver les différentes versions du code : conservation de l'historique des changements sous la forme d'un dépôt qui permet de revenir à n'importe quelle version (un programmeur peu confiant ne prend pas de risque, car il pourra toujours revenir en arrière) ;
- stocker le code à des endroits différents (machines personnelles des développeurs, serveur permettant gérer le code, ...) avec des outils de synchronisation entre les différents dépôts ;
- travailler en équipe en conservant l'origine de toutes les modifications (on sait donc qui blâmer lorsqu'il y a des erreurs) ;
- documenter toutes les modifications effectuées ;
- sauvegarder un travail sur un serveur distant, et ainsi de prévenir sa perte en cas de problème avec un ordinateur (vol ou panne).

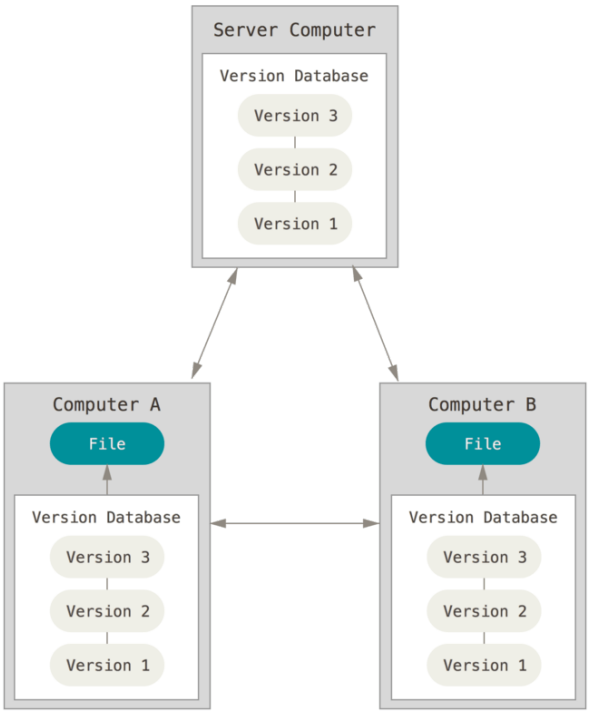
Dans ce cours, vous utiliserez git que vous pouvez télécharger au lien suivant : https://git-scm.com/download. Afin d'héberger votre code, vous avez accès à une instance *gitlab* à l'adresse suivante : https://etulab.univ-amu.fr/.

On peut exécuter git via l'IDE (IntelliJ IDEA intègre la gestion de version dans ces menus) ou bien directement en ligne de commande.

De manière simple, l'utilisation que vous allez faire de git va suivre le déroulement suivant :

- À la première utilisation, on crée une copie locale du dépôt git (`clone` ou `init`).
- À chaque commit :
  - on récupère la version courante du dépôt sur le serveur (`pull`) ;
  - on ajoute les fichiers à modifier (`add`) ;
  - on finalise le commit en donnant un message résumant les modifications (`commit`).
- Après un ou plusieurs commits, on met à jour la version distante avec nos modifications (`push`).

Git est un système de gestion de version distribué. Dans un tel système et contrairement aux systèmes de gestion de version non-distribués, les clients n'extraient pas seulement la dernière version d'un fichier, mais ils dupliquent complètement le dépôt. Ainsi, si le serveur disparaît et si les systèmes collaboraient via ce serveur, n'importe quel dépôt d'un des clients peut être copié sur le serveur pour le restaurer. Chaque extraction devient une sauvegarde complète de toutes les données.



Les explications qui suivent sont tirées du Pro Git book.

Utilisation basique de Git

Première utilisation de Git

La première chose à faire est d'installer Git ce que vous pouvez faire en suivant les instructions au lien suivant : https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Installation-de-Git.

La première chose à faire après l'installation de Git est de renseigner votre nom et votre adresse de courriel. C'est une information importante car toutes les validations dans Git utilisent cette information.

```
1 $ git config --global user.name "Prénom Nom"
2 $ git config --global user.email votre.adresse@etu.uni-amu.fr
```

Vous pouvez obtenir de l'aide sur les commandes git à l'aide de la commande *help* :

```
1 $ git help nom_de_la_commande
```

Démarrer un dépôt Git

Vous pouvez démarrer un dépôt Git de deux manières.

— Vous pouvez prendre un répertoire existant et le transformer en dépôt Git.
— Vous pouvez cloner un dépôt Git existant sur un autre serveur.

Pour créer un dépôt local, il suffit d'appeler la commande `git init` dans le répertoire dans lequel vous voulez

démarrer votre dépôt. Cela crée un nouveau sous-répertoire nommé `.git` qui contient tous les fichiers nécessaires au dépôt. Pour l'instant, aucun fichier n'est encore versionné.

Si vous souhaitez démarrer le contrôle de version sur des fichiers existants (par opposition à un répertoire vide), vous devrez probablement suivre ces fichiers et faire un commit initial. Vous pouvez le réaliser avec quelques commandes `add` qui spécifient les fichiers que vous souhaitez suivre, suivies par un `git commit` :

```
$ git add *.java
$ git commit -m 'initial project version'
```

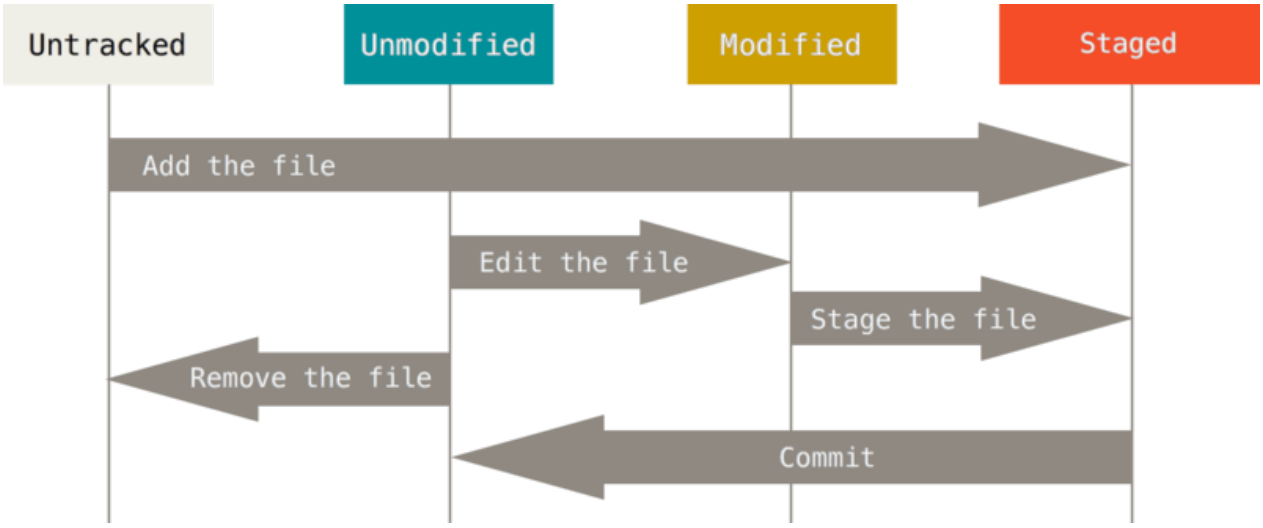
Pour obtenir une copie d'un dépôt Git existant, il faut utiliser la commande `git clone`. Vous clonez un dépôt avec `git clone url` avec url l'adresse serveur du dépôt.

Enregistrer des modifications dans le dépôt

Une fois que vous avez un dépôt Git valide et une extraction ou copie de travail du projet. Vous devez faire quelques modifications et valider des instantanés de ces modifications dans votre dépôt à chaque fois que votre projet atteint un état que vous souhaitez enregistrer.

Chaque fichier de votre copie de travail peut avoir deux états : sous suivi de version ou non suivi (*untracked*). Les fichiers suivis sont les fichiers qui appartenaient déjà au dernier instantané; ils peuvent être inchangés (*unmodified*), modifiés (*modified*) ou indexés (*staged*). En résumé, les fichiers suivis sont ceux que Git connaît. Tous les autres fichiers sont non suivis. Quand vous clonez un dépôt pour la première fois, tous les fichiers seront sous suivi de version et inchangés, car Git vient tout juste de les extraire et vous ne les avez pas encore édités.

Au fur et à mesure que vous éditez des fichiers, Git les considère comme modifiés, car vous les avez modifiés depuis le dernier instantané. Vous indexez ces fichiers modifiés et vous enregistrez toutes les modifications indexées, puis ce cycle se répète.



L'outil principal pour déterminer quels fichiers sont dans quel état est la commande `git status`. Supposons que vous souhaitez ajouter un nouveau fichier `README.md` que vous venez de créer. Ce fichier n'est pas en suivi

de version. Pour commencer à suivre ce nouveau fichier, il faut utiliser la commande `git add` suivi du nom de fichier. Vous pouvez entrer ceci :

```
$ git add README.md
```

Si vous lancez à nouveau la commande git status, vous pouvez constater que votre fichier `README.md` est maintenant suivi et indexé. La commande `git add` accepte en paramètre un chemin qui correspond à un fichier ou un répertoire ; dans le cas d'un répertoire, la commande ajoute récursivement tous les fichiers de ce répertoire.

Il est aussi possible d'indexer (*stage*) des fichiers déjà suivis afin d'enregistrer par la suite dans le dépôt la modification du fichier. La commande `git add` est multi-usage : elle peut être utilisée pour placer un fichier sous suivi de version, pour indexer un fichier ou pour d'autres actions telles que marquer comme résolus des conflits de fusion de fichiers. Sa signification s'approche plus d'ajouter ce contenu pour la prochaine validation que d'ajouter ce contenu au projet.

Valider vos modifications

Maintenant que vous avez choisis les fichiers indexés, c'est-à-dire les fichiers dont les ajouts ou modifications seront stocké dans le dépôt, vous pouvez valider votre mise-à-jour. Souvenez-vous que tout ce qui est encore non indexé — tous les fichiers qui ont été créés ou modifiés, mais n'ont pas subi de `git add` depuis que vous les avez modifiés — ne feront pas partie de la prochaine validation. Ils resteront en tant que fichiers modifiés sur votre disque. Dans notre cas, la dernière fois que vous avez lancé `git status`, vous avez vérifié que tout était indexé, et vous êtes donc prêt à valider vos modifications. La manière la plus simple de valider est de taper `git commit`.

Vous constatez que le message de validation par défaut contient une ligne vide suivie en commentaire par le résultat de la commande `git status`. Vous pouvez effacer ces lignes de commentaire et saisir votre propre message de validation, ou vous pouvez les laisser en place pour vous aider à vous rappeler ce que vous êtes en train de valider.

Autrement, vous pouvez spécifier votre message de validation en ligne avec la commande git commit en le saisissant après l'option -m, comme ceci :

```
$ git commit -m "Story 182: Fix benchmarks for speed"
```

Souvenez-vous que la validation enregistre l'instantané que vous avez préparé dans la zone d'index. À chaque validation, vous enregistrez un instantané du projet en forme de jalon auquel vous pourrez revenir ou avec lequel comparer votre travail ultérieur.

Travailler avec des dépôts distants

Il y a plusieurs manières de travailler avec un dépôt distant. Si vous avez cloné un dépôt avec la commande `git clone`, votre dépôt est automatiquement lié au dépôt distant que vous avez cloné. Si vous avez besoin d'ajouter un nouveau dépôt distant Git, il faut exécuter la commande `git remote add url` avec `url` l'adresse du dépôt.

Lorsque votre dépôt vous semble prêt à être partagé, il faut le pousser en amont, c'est-à-dire envoyer vos commits dans le dépôt distant. La commande pour le faire est simple : `git push`.

Cette commande ne fonctionne que si vous avez cloné depuis un serveur sur lequel vous avez des droits d'accès en écriture et si personne n'a poussé dans l'intervalle. Si vous et quelqu'un d'autre clonez un dépôt au même moment et que cette autre personne pousse ses modifications et qu'après vous tentez de pousser les vôtres, votre poussée sera rejetée. Vous devrez tout d'abord tirer (commande `git pull`) les modifications de l'autre personne et les fusionner avec les vôtres avant de pouvoir pousser.