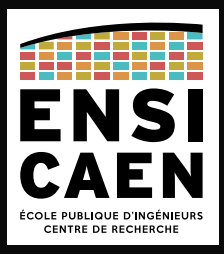


# Outils de développement logiciel

TP n° 3 : gestion des versions avec Git (commandes de base)

Alain Lebret

2023-2024

**Objectif**

S'initier à la gestion des versions avec Git.
**Pré-requis** : diaporama sur ***Git***
**Durée estimée** : 1 séance


**Figure 1:** Git Humor (source : http://geek-and-poke.com).

## Préliminaire

Comme dans le TP précédent, nous vous proposons d'utiliser l'application ***asciinema***. Cette application va vous permettre d'enregistrer les commandes que vous entrez dans le terminal et les résultats de ces commandes. Voici par exemple la commande à exécuter dans l'exercice n° 2 pour enregistrer le terminal dans le fichier "tp03-prenom-nom-ex2.cast" que vous remettrez en fin de séance.

```
$ asciinema rec tp03-prenom-nom-ex2.cast -i=1
```

Dans le cas où vous auriez stoppé l'enregistrement avant terme, relancez la commande précédente en ajoutant l'option "`--append`" de manière à continuer l'enregistrement du fichier.

Pour stopper l'enregistrement d'***asciinema***, cela se fait à l'aide de la commande "`exit`" ou de la séquence Ctrl + d.

## Introduction

Dans ce TP, vous allez vous familiariser avec la gestion des versions à l'aide de l'outil décentralisé ***Git***. Vous allez notamment apprendre à :

- initialiser un dépôt (`init` et `config`) ;
- modifier, indexer, valider et consulter l'état et l'historique d'un dépôt (`status`, `add`, `commit`, `log` et `diff`) ;
- utiliser un dépot distant (`remote`, `push` et `pull`).

Le site https://onlywei.github.io/explain-git-with-d3 peut vous aider à "visualiser" certaines des opérations que vous allez mettre en oeuvre par la suite.

## Initialiser un dépôt

### Exercice n° 1 : initialiser votre premier dépôt

1. Placez-vous dans votre dossier "odl", créez-y le dossier "tp03" ainsi que son sous-dossier "blog" qui contiendra les fichiers de votre blog personnel.
2. Depuis le sous-dossier "blog", lancez la commande `git init` afin de créer votre dépôt. Vérifiez que le dossier caché "**.git**" est apparu dans "blog".
3. Spécifiez votre nom ainsi que le courriel avec lequel vous allez soumettre vos révisions à partir de maintenant (que ce soit en local, sur Gitlab ou encore sur d'autres plateformes telles que GitHub) :

   ```
   git config --global user.name "Jean Saigne"
   git config --global user.email "jsaigne@ecole.ensicaen.fr"
   ```

4. Indiquez l'éditeur que vous souhaitez associer à ***Git*** pour les validations (l'éditeur par défaut sur Ubuntu est ***gedit***) :

   ```
   git config --global core.editor vim
   ```

### Exercice n° 2 : une première validation

***Git*** fonctionne avec trois espaces (arbres) :

- le **dossier de travail** (***working directory***) où vous effectuez les révisions – dans le cas présent, il s'agit du dossier "blog" ;

- l'**index** (***index*** ou ***stage area***) qui est un espace intermédiaire permettant d'indexer une révision avant sa validation ;
- le **dépôt** (***repository***) qui regroupe l'ensemble des révisions qui ont été validées.

Le passage du dossier de travail vers l'index est réalisé par la commande **`git add`** suivie du ou des noms de fichiers. Quant à la validation d'une révision indexée, elle est faite à l'aide de la commande **`git commit`**.

1. Dans le dossier "blog", créez le fichier "README.md" et complétez-le (un titre, votre nom, un début de description du contenu, etc.).
2. Relevez l'état de votre dépôt en entrant la commande `git status`. Vous devriez constater que votre fichier "README.md" n'est pas encore "suivi". La commande `git status` vous permet de savoir où vous en êtes, mais elle vous renseigne aussi sur ce que vous pourriez faire. Pensez à l'utiliser aussi souvent que possible afin de surveiller l'évolution du dépôt.
3. Ajoutez le fichier dans l'index à l'aide de la commande `git add README.md`, puis vérifiez avec `git status` que votre fichier est bien indexé.
4. Validez le fichier en entrant la commande : `git commit -m "Ajout du fichier README.md"` et vérifiez avec `git status` que la validation a bien été effectuée.

La commande `git commit` peut être utilisée sans argument, auquel cas l'éditeur par défaut est ouvert afin d'entrer le commentaire associé à la validation.

## Modifier, indexer, valider et consulter l'état et l'historique d'un dépôt

### Exercice n° 3 : indexer et valider

1. Copiez dans le dossier de travail le fichier "index.html" ainsi que le sous-dossier "images" que vous trouverez dans "ressources-odl-fisa/tp03/". Indexez-les, puis validez-les. Ouvrez alors "index.html" avec le navigateur (commande `firefox index.html`).
2. Vous vous rendez compte qu'il y a une faute d'orthographe dans votre code HTML. Modifiez le fichier en conséquence et retestez-le avec votre navigateur. Réalisez la commande **`git diff`** qui permet de constater les différences entre le dossier de travail et l'index.
3. Indexez la nouvelle version, mais sans la valider. Puis relevez les différences entre la version indexée et celle qui est validée en utilisant la commande **`git diff --staged`** (`git diff` seule ne donnerait rien). Exécutez la commande **`git status`** afin de savoir où vous en êtes.

4. Vous pensez que le style défini directement dans le fichier mériterait d'être placé dans le fichier "css/style.css". Modifiez "index.html" en conséquence et créez le fichier de style sans l'indexer, puis exécutez les commandes `git status`, puis `git diff` et enfin `git diff --staged`. Dans quels espaces se trouvent vos trois versions du fichier "style.css" ?
5. Indexez et validez alors "style.css", puis vérifiez l'état du dépôt.

Attention, avoir plusieurs versions dans les trois espaces peut conduire à une instabilité du dépôt si vous n'y prenez garde. Pensez donc à valider la version qui se trouve dans l'index, puis à indexer et valider celle qui se trouve dans votre dossier de travail. Un appel à `git status` vous aurait indiqué qu'il fallait prendre des précautions.

### Exercice n° 4 : consulter l'historique des validations

La commande **`git log`** permet d'afficher un historique des différentes validations réalisées.

1. Lancez la commande `git log` et vérifiez que toutes vos validations apparaissent. Quel est le numéro de la dernière validation effectuée ?
2. Réitérez avec la commande `git log --oneline` qui compacte le résultat.
3. Testez enfin la commande suivante qui permet un affichage compact et coloré :

```
git log --graph --pretty=format:'%Cred%h%Creset \
-%C(yellow)%d%Creset %s %Cgreen(%cr)%Cblue \
- %cn %Creset' --abbrev-commit --date=relative
```

## Annuler des modifications

Sous ***Git***, vous disposez de plusieurs moyens pour annuler des modifications malencontreuses, selon qu'elles ont été ou non indexées, puis validées.

### Exercice n° 5 : annuler une modification du dossier de travail

1. Supprimez le fichier "style.css".
2. Vous vous apercevez que cette suppression est une erreur. Vous pouvez rétablir la version préalablement validée de ce fichier en entrant la commande :

```
git checkout <id_commit> style.css
```

ou encore si la dernière validation concernait “style.css” :

```
git checkout HEAD style.css
```

(ne lancez pas les deux bien entendu !).

3. Que vous retournent les commandes `git diff` et `git diff --cached` ?
4. Vérifiez que le fichier est restauré dans sa version correcte.

Si vos modifications concernent plusieurs fichiers non validés (**ce qui n’est pas une bonne pratique !**), vous pouvez aussi employer la commande `git reset --hard HEAD` qui va effacer toutes les modifications que vous auriez indexées, ainsi que celles du dossier de travail qui ne l’auraient pas été.

La variable `HEAD` est une référence sur la dernière validation qui a été exécutée sur la branche courante. Il ne peut y avoir qu’une seule variable `HEAD` à tout instant. Le contenu de `HEAD` est stocké dans le fichier “.git/HEAD”.

### Exercice n° 6 : annuler une modification déjà validée

Si vous avez déjà validé une modification malencontreuse, il y a deux façons de régler le problème :

- effectuer, indexer et valider un changement qui annulera la modification précédente (**c’est la bonne méthode !**) ;
- revenir en arrière et modifier l’ancienne validation (ce qui est à éviter si vous travaillez à plusieurs).

1. Dans le sous-dossier “css”, faites une copie du fichier “style.css” que vous appellerez “style2.css”. Indexez, puis validez ce dernier. Vérifiez sa prise en compte à l’aide de la commande `git log`.
2. Vous êtes allé un peu vite en besogne puisque vous n’avez pas modifié son contenu. Annulez la dernière validation en entrant la commande `git revert <id_last_commit>` (en fait, `revert` crée une nouvelle validation). Vous auriez pu aussi exécuter `git revert HEAD`.
3. Vérifiez votre historique avec `git log`.

Pour annuler plus d'une validation, par exemple les ***3*** dernières, on utilisera la commande `git revert HEAD^^^` (ou encore `git revert HEAD~3`).

Depuis la version 2.23 de ***Git***, les commandes `git restore` et `git switch` sont venues simplifier l'utilisation des annulations de révisions et du changement de branche qui sont actuellement réalisés par la commande ***fourre-tout*** : `git checkout`. Cette commande ne respecte pas la philosophie d'Unix qui voudrait que toute commande ne fasse qu'une seule chose, mais bien.

Donc dans la mesure du possible et si ce n'est pas déjà le cas, passez à la version 2.23 afin de profiter de ces nouvelles fonctionnalités.

## Utiliser des dépôts distants

Un dépôt qui serait placé sur la machine d'un des collaborateurs à un projet pourrait bien sûr être accessible aux autres à l'aide de SSH sur un réseau interne. Toutefois, cela nécessiterait que la machine reste allumée en permanence. Il est en général plus simple de créer des dépôts facilement accessibles à plusieurs collaborateurs.

Gitlab est un service d'hébergement de dépôts distants qui utilise ***Git*** et peut être installé sur un serveur Web. L'école en propose un afin de vous permettre de développer vos projets, seuls ou en groupe. Le serveur est accessible à l'adresse : https://gitlab.ecole.ensicaen.fr/.

1. Entrez la commande suivante et répondez oui, puis appuyez ensuite deux fois sur Enter afin de ne pas choisir de mot de passe :

   ```
   ssh-keygen -t rsa -b 2048 -C "votre_courriel@ecole.ensicaen.fr"
   ```

   Cette commande crée deux fichiers situés dans le dossier "**.ssh**" à la racine de votre dossier personnel : le fichier "`~/.ssh/id_rsa`" qui contient la clé privée utilisée par le protocole SSH, et "`~/.ssh/id\_rsa.pub`" qui contient la clé publique. **Ne confiez jamais la première à qui que ce soit !** Par contre, la seconde facilitera votre connexion à Gitlab depuis votre compte. Visualisez la clé publique en entrant la commande :

   ```
   cat $HOME/.ssh/id_rsa.pub
   ```

   Vous devriez voir quelque chose de ce type :

   ```
   ssh-rsa AAAAB3NzaC1.....................................
   ............................. votre_courriel@ecole.ensicaen.fr
   ```

   Copiez l'intégralité du texte avec votre souris.

2. Depuis votre navigateur, connectez-vous sur Gitlab et dans l'onglet "LDAP" de la page d'accueil, renseignez votre nom d'utilisateur et votre mot de passe (les mêmes que pour vous connecter à votre compte).
3. Cliquez en haut à droite de la barre de menu de Gitlab et sélectionnez "Paramètres". Dans la zone qui apparaît à gauche, choisissez "Clés SSH". Vous allez pouvoir ajouter votre clé publique en collant ce que vous avez préalablement copié dans la zone de texte prévue à cet effet. Renseignez un titre pour votre clé et une date d'expiration (par exemple, 15 septembre 2025). Cliquez alors sur "Ajouter une clé". Celle-ci devrait apparaître en dessous. Vous pourrez réitérer cette opération chez vous, depuis votre machine personnelle, de manière à générer une autre clé.

### Exercice n° 7 : pousser votre dépôt sur Gitlab

Vous allez à présent créer localement un lien vers l'adresse du dépôt de Gitlab où sera stocké "blog". Ce lien aura pour nom "**origin**" qui est le nom utilisé par défaut. Il sera créé à l'aide de la commande `git remote add` de la manière suivante :

```
$ git remote add origin https://gitlab.ecole.ensicaen.fr/<id_utilisateur>/blog
```

Une fois ce lien défini, il est possible de synchroniser le dépôt local et le dépôt distant à l'aide de la commande :

```
$ git push –u origin master
```

Le serveur devrait vous demander de vous identifier pour ensuite créer et synchroniser le dépôt distant. Vérifiez sur l'interface Gitlab que le dépôt privé "blog" a été créé et que tous vos fichiers y sont présents.

"master" est le nom de la branche sur laquelle la synchronisation sera réalisée. Nous verrons plus tard l'utilité et la gestion des branches.

Si des modifications ont été faites par un collaborateur qui a modifié un des fichiers sur la branche principale du dépôt sur le serveur, vous pouvez les récupérer localement à l'aide de la commande :

```
$ git pull
```

Vérifiez que votre projet se retrouve bien sur le serveur Gitlab.

## Livrable

Vérifiez que les fichiers d'animation ("tp03-prenom-nom-ex*.cast") sont valides en initiant leur lecture à l'aide de la commande `asciinema play ...` (pas la peine de les lire en entier !)

Déplacez alors l'ensemble des fichiers "*.cast" vers le sous-dossier "anims" que vous aurez préalablement créé, puis compressez ce dossier à l'aide de la commande `zip`. Déposez l'archive compressée sur Moodle.

## Résumé

Dans ce TP nous n'avons fait qu'effleurer la gestion des versions sous ***Git***. Il nous reste à voir les notions de branches et de collaboration entre développeurs. ***Git*** est un outil plutôt difficile à appréhender, pour lequel il vous faudra user de patientes, et parfois pénibles expérimentations, avant d'en apprécier la puissance.