# TP1 : Premiers pas avec git

## Mode d'emploi des TPs

Afin de s'adapter au rythme de progression de chacun, les TPs sont pensés pour être suivis de façon autonome. Ils sont construits de la façon suivante :

- Une **partie guidée**, où des concepts sont introduits à travers l'exécution commentée d'une série de commandes, avec un récapitulatif à la fin.
- Des **exercices** pour pratiquer l'utilisation de git.
- Une **antisèche** qui résume toutes les commandes qui ont été introduites.
- Des **informations complémentaires**, qui ne sont pas essentielles mais permettent d'approfondir certains aspects de l'outil.

Les commandes à exécuter dans un terminal sont représentées comme ceci :

```
$ whoami
```

La partie guidée du TP suppose que l'ensemble des commandes indiquées ont été exécutées, dans l'ordre indiqué, et à l'exclusion de toute autre commande. Si vous souhaitez faire des expériences pour vérifier que vous avez bien compris les explications, nous vous conseillons de créer un dépôt git à part pour cela afin de ne pas perturber le bon fonctionnement du TP.

L'idée de ce mode de formation est que chacun puisse avancer à son rythme, sans devoir se concentrer sur de longs discours magistraux. Mais le formateur est là pour répondre à toutes vos questions. N'hésitez surtout pas à le solliciter !

## Configuration du système

### Point de départ

Durant ces exercices, nous allons travailler dans la machine virtuelle. Il est possible que vous ayez besoin d'installer les outils de git par vous même : `git` et `gitg`. Vous pouvez rester sous Windows si vous savez installer les outils nécessaires.

### S'identifier auprès de git

Les gestionnaires de versions permettent à plusieurs personnes d'échanger des versions de fichiers entre eux. Il est alors utile de savoir qui a contribué quoi, et comment contacter cette personne.

git vous demandera donc de lui donner votre nom et votre adresse e-mail avant de vous laisser créer des versions, ce que vous pouvez faire comme ceci :

```
$ git config --global user.name 'Francine Dupont'
$ git config --global user.email fdupont@serveur-imaginaire.fr
```

### Choisir un éditeur de texte

git vous demandera parfois de saisir du texte avec un éditeur de texte. Cet éditeur est configurable, par défaut il s'agit de `vi`.

Le choix d'un éditeur de texte relevant de la guerre de religion, nous dirons juste qu'il est recommandé d'utilise un éditeur en mode console pour cet usage, et qu'il est conseillé à ceux qui n'ont jamais utilisé ce type d'éditeur de faire appel à `nano` pour les besoins de cette formation :

```
$ git config --global core.editor nano
```

## Premiers pas avec la gestion de version

### Créer un dépôt

N'importe quel dossier peut être mis sous gestion de version. Il devient alors ce que l'on appelle un "dépôt". On utilise pour cela la commande `git init` :

```
$ cd ~/Bureau
$ mkdir MonDepot && cd MonDepot
$ git init
```

Comme le message de statut l'indique, cette opération crée un sous-dossier "`.git`", où git stockera ses versions de fichiers.

### Statut et ajouts

On peut à tout moment demander quel est l'état actuel du dépôt avec `git status`. Pour l'instant, il ne s'y passe pas grand chose :

```
$ git status
```

Mais dès la création d'un fichier, le statut change :

```
$ touch MonFichier.txt
$ git status
```

Comme la plupart des gestionnaires de versions, git ne suit pas automatiquement les modifications de tous les fichiers du dossier dans lequel il est actif. Il faut indiquer quels fichiers on souhaite versionner avec `git add` :

```
$ git add MonFichier.txt
$ git status
```

Après cette opération, MonFichier.txt est suivi par git : lorsqu'on enregistrera une version du dépôt (un *commit*), ce fichier y sera présent.

En cas d'ajout non désiré, on peut bien sûr annuler cette opération. Avec une version moderne de git, il suffit d'utiliser `git reset <nom du fichier>`. Sur d'anciennes versions, il fallait utiliser pour cet usage la commande plus hideuse et spécifique `git rm --cached <fichier>`.

### Ignorer des fichiers

Tout contenu ne devrait pas être mis sous gestion de version. Il existe deux raisons classiques d'en exclure des fichiers :

- Ils contiennent des données que l'on ne souhaite pas partager avec autrui (ex : mots de passe, configurations d'outils relevant du goût personnel)
- Ils peuvent être régénérés à partir des fichiers restants (ex : résultats d'une compilation de code source, journaux d'exécution)

Supposons que nous ayons un tel fichier face à nous. Comment faire pour que `git status` cesse de nous inciter peu subtilement à le versionner ?

```
$ echo 'DEBUG : Un matin, Gregor Samsa se réveilla' > Poubelle.log
$ git status
```

Pour résoudre ce problème, on peut créer un fichier `.gitignore` à la racine du dépôt, qui indique que certains fichiers doivent être ignorés par git :

```
$ echo '*.log' > .gitignore
$ git status
```

En général, il est pertinent de mettre `.gitignore` sous gestion de version, son contenu étant sujet à évoluer au fil de l'évolution du projet.

```
$ git add .gitignore
```

`.gitignore` supporte les motifs usuels du shell Unix, comme l'étoile (*wildcard*), et sa configuration s'applique à chaque sous-dossier du dépôt. Pour gérer des situations plus complexes, des syntaxes plus sophistiquées sont disponibles, consultez `git help gitignore` pour plus de détails.

### Déplacement et suppression de fichier

A partir du moment où un fichier est sous gestion de version, il est préférable d'informer le gestionnaire de version quand on effectue un déplacement ou une suppression, en utilisant respectivement `git mv` et `git rm`.

```
$ git mv MonFichier.txt MonSuperFichier.txt
$ git status
```

En effet, le gestionnaire de versions n'est pas en position de détecter ces opérations, et les interprétera d'une façon inattendue et indésirable :

```
$ mv MonSuperFichier.txt MonIncroyableFichier.txt
$ git status
```

Comme vous le voyez, la commande "mv" standard a ainsi été interprétée comme la suppression d'un fichier sous gestion de version et la création d'un nouveau fichier non versionné. Ce n'est généralement pas ce qui était désiré.

Une commande pratique pour se tirer de ce genre de mauvais pas est `git add --all`, qui intègre à la gestion de version les ajouts de fichier et les suppressions. Ici, nous pouvons l'utiliser de la façon suivante :

```
$ git add --all .
$ git status
```

Git refusera par défaut de supprimer un fichier ayant des changements non enregistrés dans une version, car cette opération peut entraîner des pertes de données. Si l'on sait ce qu'on fait, l'option `--force` (qui s'abbrévie en `-f`) peut être utilisée pour confirmer l'opération.

```
$ git rm MonIncroyableFichier.txt
$ git rm -f MonIncroyableFichier.txt
```

### Gestion des dossiers

Contrairement au système de fichiers de votre système d'exploitation, git ne gère qu'une arborescence de *fichiers*. L'existence de dossiers n'est prise en compte que dans la mesure où il y a des fichiers à l'intérieur :

```
$ mkdir MonDossier
$ git status
$ ls -l
$ echo 'Un semblant de contenu' > MonDossier/Contenu.txt
$ git status
```

Cependant, la plupart des commandes git fonctionnent avec des dossiers, moyennant parfois l'utilisation de l'option `--recursive` (qui s'abbrévie en `-r`) lorsqu'il est risqué d'effectuer une opération sur un dossier sans connaître son contenu :

```
$ git add MonDossier/
$ git status
$ git rm MonDossier/
```

Notez que puisque le contenu de ce dossier n'a pas encore été sauvegardé, vous devriez même utiliser `-rf` dans ce cas précis pour convaincre git que vous savez ce que vous faites.

### Aide de git

Comme la plupart des outils Unix, git dispose d'une documentation de référence accessible depuis la ligne de commande, soit via la traditionnelle interface `man`, soit via la commande `git help`.

```
$ man git status
$ git help status
```

`git help` reproduit par défaut le comportement de la commande `man`, mais donne aussi accès à quelques documentations généralistes qui ne sont pas accessibles par le biais de `man`, comme par exemple `git help everyday`.

En principe, git help donne également accès à des formats de documentation alternatifs, tels que des pages HTML. Mais ces derniers ne sont malheureusement que rarement inclus dans les distributions de git. Si vous souhaitez consulter la documentation au format HTML, il est donc plus sûr de le faire en ligne à l'adresse https://www.git-scm.com/docs.

Si vous ne cherchez pas une explication détaillée du fonctionnement d'une commande, mais seulement un bref rappel de la syntaxe et des options disponibles, vous pouvez aussi utiliser l'option `-h` :

```
$ git mv -h
```

## Conclusion

Dans ce premier TP, nous avons vu comment...

- Configurer son système pour l'utilisation de git
- Mettre un répertoire sous gestion de version
- Connaître l'état actuel de git
- Désigner quels fichiers doivent être versionnés ou non
- Effectuer des opérations de base sur ces fichiers
- Utiliser l'aide de git

Mais des zones d'ombres demeurent. Comment créer des versions ? Et quelle est donc cette "branche master" dont `git status` ne cesse de parler ? Nous répondrons à ces questions dans les prochaines séquences.

---

## Exercices

1. Créez un nouveau dépôt git.
2. Configurez le dépôt pour ignorer les fichiers portant l'extension ".o".
3. Créez un fichier et préparez git à le versionner.

4. Déplacez ce fichier vers un autre emplacement de telle façon que `git status` ne parle que de deux fichiers (dont le "gitignore").
5. Affichez l'état final du dépôt. En ignorant pour l'instant la partie "Sur la branche master", comprenez-vous les autres messages affichés ?

## Antisèche

Afficher la configuration actuelle de git :

```
$ git config --list
```

Saisir les identifiants utilisateur qui seront insérés dans les commits :

```
$ git config --global user.name 'Francine Dupont'
$ git config --global user.email fdupont@serveur-imaginaire.fr
```

Définir l'éditeur de texte utilisé par git :

```
$ git config --global core.editor nano
```

Initialiser un dépôt git dans le répertoire courant :

```
$ git init
```

Afficher l'état du dépôt git actif :

```
$ git status
```

Mettre un fichier (ou un dossier) sous gestion de version :

```
$ git add <fichier>
```

Mettre tous les fichiers du répertoire courant sous gestion de version :

```
$ git add .
```

Intégrer à la gestion de version tous les changements du répertoire courant, y compris les suppressions, les déplacements...

```
$ git add --all .
```

Supprimer un fichier sous gestion de version :

```
$ git rm <fichier>
```

Retirer un fichier de la gestion de version sans le supprimer :

```
$ git rm --cached <fichier>
```

Déplacer un fichier sous gestion de version :

```
$ git mv <source> <destination>
```

Ignorer les fichiers d'extension ".log" (à exécuter à la racine du dépôt) :

```
$ echo "*.log" >> .gitignore
```

Afficher le manuel d'une commande git (ou l'explication d'un concept git) :

```
$ git help <commande>
```

Afficher un bref rappel de la syntaxe d'une commande git :

```
$ git <commande> -h
```

---

## Informations complémentaires

### Interface de configuration

git peut être configuré selon trois périmètres :

1. Ensemble des utilisateurs du système hôte ("system")
2. Compte utilisateur actif ("global")
3. Dépôt git actif ("local", choix par défaut)

Quand deux périmètres sont en désaccord, c'est la configuration du périmètre le plus étroit qui est utilisée. Par exemple, si la configuration système spécifie que l'éditeur de texte à utiliser est `vi` et la configuration utilisateur spécifie que c'est `emacs`, git utilisera `emacs` comme éditeur de texte.

L'emplacement de la configuration varie d'un système d'exploitation à l'autre. Mais vous n'avez heureusement pas besoin de l'apprendre par coeur, car vous pouvez manipuler cette dernière avec l'utilitaire `git config`, dont voici quelques utilisations fictives :

```
$ sudo git config --system reglage 'Valeur pour tout le système'
$ git config --global reglage 'Valeur spécifique à un utilisateur'
$ git config reglage 'Valeur spécifique à un dépôt git'
```

Bien entendu, la manipulation de réglages système nécessite de posséder des droits administrateur sur la machine.

La configuration actuelle peut être affichée avec `git config --list`.