

# TD n° 1

## Fonctionnalités de base de Git

Le présent document vise à montrer en contexte des fonctionnalités de base de *Git*, en les intégrant dans la pratique — de manière un peu artificielle, certes...

Pour un premier panorama des commandes de git peut être trouvé ici <https://try.github.io/>.  
Beaucoup d'informations proviennent de <https://www.atlassian.com/git/tutorials/>.  
Merci à Arnaud Bey pour ses suggestions.

**Conventions :** Dans le document suivant, les lignes de commande shell seront indiqués de la *manière suivante* (avec les mots-clés notés *ainsi*). Des chevrons seront utilisés pour différencier les paramètres où vous pouvez mettre la valeur de votre choix de ceux qui ont une sémantique propre. Il n'est pas opportun de recopier les chevrons en question.

**Exemple :** Alain Connu, pour exécuter la commande `git config user.name "<nom>"`, saisira `git config user.name "Alain Connu"`.

### 1 Débuter avec git

Git est un logiciel open source et peut être obtenu ici : <http://git-scm.com/downloads>

Git permet de gérer les versions et évolutions de code source (et par extension de tout fichier texte). Pour ce faire, pour chaque modification significative du code (ajout de fonctionnalités, correction de bugs, etc.), on pourra faire un *commit*, c'est-à-dire sauvegarder une version du code.

Git stocke les versions successives d'un dossier. Une fois « surveillé » par Git, un dossier est un *repository* ou « *repo* ». En français on parlera de « dépôt ».

Git permet également de travailler à plusieurs. Afin de savoir qui fait quoi, git permet de stocker des informations à apposer à chaque *commit* de l'utilisateur.

- `git config user.name "<nom>"` : tous les *commit* effectués sur ce dépôt sont crédités à l'utilisateur appelé <nom>;
- `git config --global user.name "<nom>"` : tous les *commit* effectués à partir de ce compte utilisateur (quel que soit le dépôt) sont crédités à l'utilisateur appelé <nom>;

— `git config --global user.email "<email>"` : l'adresse e-mail de l'utilisateur est `<email>`.

**Exercice d'application :** Nous allons voir ici quelques uns des effets de ces commandes.

1. Installer Git
2. Définir son nom pour l'usage de git sur la machine utilisée
3. Dans le dossier utilisateur (`~` sous Linux / `C:\Users\NomDUtilisateur` sous Windows), ouvrir le fichier `.gitconfig`. Que remarquez-vous ?
4. Modifier l'adresse de courrier électronique associée à l'utilisateur
5. Que s'est-il passé dans le fichier de configuration ?

Beaucoup d'autres choses peuvent être configurées, du logiciel utilisé pour éditer les fichiers à celui pour traiter les cas de conflits entre les versions...

En effet, il peut arriver que deux versions entrent en conflit, dans ce cas l'utilisateur devra décider de la version à conserver. Plutôt que de manipuler ces différentes version à la main (git annote le code entrant en conflit) un logiciel de comparaison de fichier peut être utilisé. Voir par exemple :

- <https://alexjoz.gitbooks.io/code-life/content/chapter6.html> (Linux).
- <http://blog.nerdbank.net/2011/10/how-to-get-meld-working-with-git-on.html> (Windows);

## 2 Créer un dossier un dépôt *Git*

Pour créer un dépôt git vous pouvez :

- prendre n'importe quel dossier et l'initialiser en tapant `git init` si vous êtes dans le dossier ou `git init <dossier>`;
- cloner un dépôt git existant avec `git clone <dépôt> <directory>` (si `<directory>` n'est pas spécifié, c'est dans le répertoire courant que le *dépôt* sera cloné.)

**Exercice d'application :** Pour l'exercice suivant vous allez vous contenter de créer un dossier (en ligne de commande) dans un dossier existant. Vous ferez ensuite de ce dossier votre dépôt git.

6. Ouvrir la console.
7. Se déplacer jusqu'à votre dossier de travail en utilisant `ls` et `cd`<sup>1</sup>.
8. Créer un dossier au nom de votre projet.
9. Aller dans le dossier ainsi créé.

---

1. `dir` et `cd` sous windows

10. Faire de ce dossier un dépôt git.

### 3 Effectuer un “commit”

Le flux de travail de base de *git* consiste en le fait d'effectuer des modifications puis de les grouper en ensemble cohérent avant de les stocker dans un état du code. On appelle l'action de créer une version du code un *commit*.

- La (sous-)commande *add* permet d'ajouter le ou les fichiers ainsi sélectionnés au prochain *commit* :
    - *git add <fichier>* : ajoute <fichier> au prochain *commit* ;
    - *git add <dossier>* : ajoute tout le contenu de <dossier> au prochain *commit* ;
    - *git add -p* : ajoute tous les changements au prochain *commit* ;
    - *git add --all <fichier>* : ajoute tous les changements à <fichier> (y compris la suppression) pour le prochain *commit* ;
    - *git reset <fichier>* : retirer les modifications à <fichier> de la sélection pour le prochain *commit*.
  - *git status* : permet d'avoir la liste des fichiers modifiés et leur situation par rapport au prochain *commit* ;
  - *git log* : permet d'avoir la liste des *commits* pour la branche courante ;
  - La (sous-)commande *commit* permet de créer une nouvelle version du code<sup>2</sup> :
    - *git commit* : ouvre un éditeur de texte pour spécifier les informations du commit, qui est effectué à la fermeture de l'éditeur de texte ;
    - *git commit -m "<message>"* : effectue le commit avec comme informations <message> ;
    - *git commit -a* : ajoute au commit suivant toutes les modifications effectuées sur des fichiers préalablement inclus dans un commit (les créations de fichiers ne sont pas ajoutées au commit) ;
    - *git commit --amend* : met à jour le dernier commit avec les changements sélectionnés.
- Attention** : cette commande est à utiliser avec parcimonie, surtout quand vous avez plusieurs dépôts pour le même projet. *git commit --amend* « réécrit l'historique », et ne doit être utilisé que si le *commit* précédent n'a pas été publié ou utilisé par ailleurs.

#### Exercice d'application :

11. Dans le dépôt, créer un fichier texte appelé *README.md*.
12. Dans ce fichier indiquer le nom de votre projet en titre. Vous pouvez utiliser pour cela la syntaxe *Markdown*<sup>3</sup>
13. Toujours dans le dépôt, créer un fichier texte appelé *LICENSE*.
14. Ajouter dans ce fichier, le contenu de la licence MIT<sup>4</sup>.

2. Pour toujours bien savoir ce qu'on fait, il est conseillé d'utiliser *git status* avant tout *commit*.

3. <https://guides.github.com/features/mastering-markdown/>

4. <http://opensource.org/licenses/MIT>



15. Exécuter la commande `git status`. Quelle est la situation ?  
\_\_\_\_\_
16. Préparer un *commit* pour stocker dans une nouvelle version l'ajout de la licence, puis regarder le statut.  
\_\_\_\_\_
17. Effectuer le *commit*.  
\_\_\_\_\_
18. Ajouter une section contributeurs dans votre `README.md`, y inclure votre nom.
19. Effectuer un *commit* pour cette modification  
\_\_\_\_\_
20. Vous avez oublié d'ajouter vos informations de contact : mettre à jour le `README` et amendez le *commit* précédent pour qu'il intègre votre modification.<sup>5</sup>  
\_\_\_\_\_

## 4 Gérer des branches

Git permet également d'expérimenter, de faire des tests sans affecter le code stable. Pour ce faire, on va avoir recours à ce que l'on appelle des « branches », qui permettront de scinder le projet en différente version. Dans la figure 1.1, 4 branches sont représentées simultanément (il est à noter qu'elles sont nommées).

- `git branch` : lister toutes les branches du dépôt ;
- `git branch <nomBranche>` : créer la branche `<nomBranche>` (pas d'espace dans les noms de branche) ;
- `git checkout <nomBranche>`<sup>6</sup> : passer sur la branche `<nomBranche>` (elle doit exister<sup>7</sup>) ;
- `git merge <nomDUneBrancheNonManipulée>` : Fusionne la branche courante avec le contenu de la branche `<nomDUneBrancheNonManipulée>` ;

**NB :** Il n'est pas possible de changer de branche tant que toutes les modifications n'ont pas été soit annulées soit « *committées* ». Pour traiter ce genre de problématique la commande `git stash`<sup>8</sup> existe, mais dépasse le cadre de cette formation. Il est bon de savoir qu'elle existe, malgré tout. De plus, fusionner des branches, peut créer des conflits, dans ce cas, il peut être bien pratique d'avoir un outil de fusion (*merge tool*) correctement configuré (cf. § 1 p. 2).

5. Pour bien voir ce qu'il se passe, vous pouvez utiliser `git status` et `git log`.

6. On peut également s'en servir pour voir un *commit* passé. Chaque *commit* a un identifiant unique (accessible avec `git log`), qui peut être passé en paramètre de `git checkout`.

7. `git checkout -b <nomBranche>` permet de basculer sur la branche `<nomBranche>`, après l'avoir créée si elle n'existait pas

8. <http://git-scm.com/docs/git-stash>

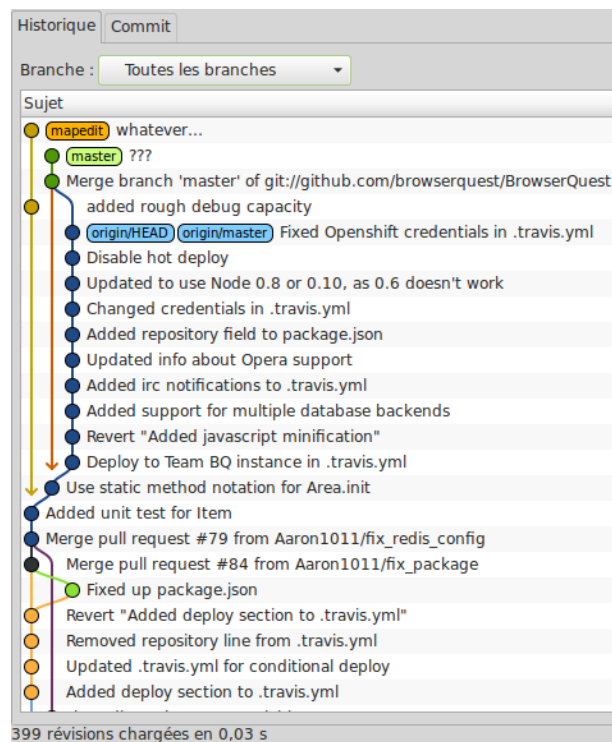


FIGURE 1.1 – Représentation graphique (gitg) de différentes branches d'un projet

### Exercice d'application : <sup>9</sup>

21. Comment s'appelle la branche principale par défaut ? Comment l'avez-vous trouvée ?  
\_\_\_\_\_
22. Créer une branche « testInstall » et l'afficher (vérifier que vous êtes bien sur la bonne branche avec `git branch`)  
\_\_\_\_\_  
\_\_\_\_\_
23. Une fois dans la branche « testInstall », créer un fichier `INSTALL.md`, qui contiendra quelques informations (du genre « nécessite un serveur Apache avec php 5 »). Vous en profiterez pour ajouter à `README.md` un paragraphe « install » indiquant de se référer au document que vous venez de créer.
24. « Commiter » ces changements sur la branche « testInstall ».
25. Retourner effectivement sur la branche `master`. Que remarquez-vous ?
26. Vérifier que vous n'avez rien perdu en retournant sur la branche `testInstall`.

9. Sur une formation aussi courte, c'est difficile de montrer à quel point c'est pratique. On se contentera ici de montrer comment ça marche...

27. Dans la branche `master`, modifier le fichier « README.md », pour y inclure une section « Install ». Et faire un `commit` des changements.
  28. Passer d'une branche à l'autre pour vérifier que « README.md » est effectivement différent selon la branche.
  29. Finalement, cela vous semble préférable d'avoir un fichier séparé pour l'installation, nous allons récupérer les modifications faites dans la branche `testInstall` et les insérer dans la branche `master`. Se mettre dans la branche `master` et appeler la commande `merge` avec la branche `testInstall`. Que se passe-t-il ?
- 
- 

30. Pour corriger ce problème, il va falloir résoudre le conflit manuellement en modifiant « README.md », avant de l'ajouter aux modifications à venir, puis faire le `commit`.