Git de survie*

Jean-Baptiste Louvet ♡

Version 1.2 – 3 septembre 2018

Résumé

Ce document a pour but de donner les commandes nécessaires pour être utiles avec git. Il n'est pas exhaustif et le recours au manuel de git par la commande `$ man git` est fortement recommandé en plus de la lecture de ce git de survie.

FIGURE 1 – XKCD sur git (`https://xkcd.com/1597/`).

1 Le principe de git

Git [1] est un logiciel gestion de version. Il permet de travailler à plusieurs sur un même code source, en même temps, sans avoir à s'échanger de code par mail, clé usb ou pigeon voyageur. Son utilisation permet de gagner du temps de développement, réaliser un projet clair, documenté et versionné. Git est utilisé dans toutes les bonnes entreprises et tout particulièrement par les pompiers de Paris pour sauver des chatons. Le premier ordre a perdu les plans de l'étoile noire car il n'utilise pas git.

À l'INSA, vous pouvez vous créer un serveur git sur l'instance gitlab de l'INSA à `https://gitlab.insa-rouen.fr`.

Ne pas utiliser git dans un projet d'informatique à l'INSA est puni d'un an d'emprisonnement et de 16 000 euros d'amende. Nicolas Malandestre viendra personnellement vous flageller dans le cachot de l'école en vous obligeant à lire le manuel de git.

2 Les commandes essentielles

Remarque : les instructions commençant par « `$` » correspondent à des lignes de commande entrées sur une console.

- `$ git init` crée un dépôt en local (sur votre machine, dans le répertoire courant).

*Adapté de la page wikipedia sur git: `https://fr.wikipedia.org/wiki/Git`

1. `https://git-scm.com/`

• `$ git clone` clone un dépôt distant. En utilisation avec la monprojet :
`$ git clone https://LOGIN@monprojet.insa-rouen.fr/git/PROJET` .

• `$ git add` permet de charger un fichier pour le prochain commit s'il n'est pas déjà sur le dépôt, ou charge les modifications faites sur un fichier par rapport à la version du dépôt pour le prochain commit.

• `$ git commit` enregistre les ajouts réalisés avec `$ git add` , en y associant un message de commit. Quand vous faites un `$ git commit` , un éditeur de texte en ligne de commande s'ouvre en vous invitant à écrire le message de commit.

Il est important de faire des commit réguliers d'éviter les commits énormes en fin de journée. L'idéal est de faire un commit pour chaque modification indépendante.

Le message de commit parfait est composé de deux parties, séparées par une ligne vide :

— un message clair et concis résumant la modification réalisée sur le dépôt dans ce commit ;
— un message plus long donnant des détails techniques sur les modifications. Ce second message n'est pas systématiquement nécessaire, il n'est utile que pour les modifications nécessitant une explication détaillée.

Exemple de message de commit :

```
Congolexicomatisation des lois du marché.

Imposer la force veeers, le valium, ça veut dire l'estime du savoir, les gens qui
connaissent beaucoup de choses et cristalliser, imposer, iiiiinnnnntentionner dans
toute la République Démocratique du Congo pour que nous puissions avoir la
congolexicomatisation des lois du marché propre aux congolais. C'est à dire mettre
un accent sur... Les revenus aussi à voir hein ! C'est un problème de TGO ! la Théorie
Générale des Organisations, comment nous pouvons parvenir à des... Relever aussi des
défis, par exemple le Brésil, à part les végétariens là, le végétalisme, nous avons
cette même climatologie.
```

Autre exemple (sérieux, cette fois) :

```
Correction d'un dépassement de capacité dans factorielle.c.

Utilisation de unsigned long long à la place de char pour permettre de calculer
des factorielles jusqu'à 18 446 744 073 709 551 615.
Résolution du bug #18.
```

Comme le montre ce commit :
`https://github.com/torvalds/linux/commit/690b0543a813b0ecfc51b0374c0ce6c8275435f0`, un enfant de quatre ans est capable d'écrire un bon message de commit, je vous invite à vous en inspirer.

Si vous voulez faire votre commit et écrire le message en une seule commande, vous pouvez utiliser l'option `-m` :
`$ git commit -m 'correction d'une erreur typographique: Maloul --> Malou'`

Soignez bien vos messages de commit, c'est souvent très irritant quand, lors d'une session de débogage en urgence, on tombe sur des messages de commit comme :

— `Ahhhhhhhhhhhhhhhhhh, Yolooooooooooooooo !`
— `Marche`
— `Marche pas`
— `Re-marche`
— `J'écris un message de commit`
— `PUTAIN D'INDICE DE BOUCLE`
— `J'ai tout cassé, je vais me coucher.`
— `Plop`
— `Lol`
— `azertyuiop`

Voir aussi `https://xkcd.com/1296/`.

Si vous êtes en panne d'inspiration au moment d'écrire un commit, imaginez-vous que vous vous envoyez un courriel dans le futur pour expliquer la raison de ce commit. Le titre du commit est l'objet du courriel et le corps du commit le contenu du message.

• `$ git pull` récupère la version du dépôt présente sur le serveur et la synchronise avec la version locale. Si vous avez des modifications en local, git va les fusionner avec celles du dépôt. Si cela se passe sans conflit, vous aurez simplement à valider un commit de fusion (merge). Pour éviter d'avoir à faire un commit de fusion (cela fonctionne mais ce n'est pas très beau, et le commit en question est inutile), vous pouvez passer l'option `-r` à git : `$ git pull -r` . De cette manière, les commit seront fusionnés sans avoir besoin d'un commit spécialement dédié à la fusion. Cette technique est plus propre et donne une meilleur lisibilité à la liste de vos messages de commit.

Dans le cas où vous avez un conflit, il vous faut aller le résoudre à la main dans les fichiers concernés puis faire un ajout et un commit pour valider la résolution du conflit. Git vous guide dans la résolution de ces problèmes.

• `$ git push` permet d'envoyer ses modifications sur le serveur. Pour envoyer ses modifications, il faut que la version locale du dépôt soit à jour avec celle du serveur (un `$ git pull -r` vous renvoie que le dépôt est à jour).

Le tout premier push à réaliser sur un serveur vierge nécessite une option supplémentaire :
`$ git push -u origin master`

Veillez à ne jamais envoyer une version du programme qui ne compile pas ou qui ne fonctionne pas (si elle fonctionnait au préalable). Si vous voulez éviter que les gens avec qui vous codez ne vous détestent, prenez l'habitude de compiler, exécuter et lancer les tests (s'il y en a) d'un projet avant de l'envoyer, autrement vous risquez d'empêcher vos camarades d'avancer. Les contrevenants à cette règle seront déclarés hérétiques et se feront écarteler en place publique.

• `$ git status` donne l'état du dépôt local :
— le nombre de commits réalisés depuis le dernier envoi ;
— les fichiers ajoutés mais non commités ;
— les fichiers modifiés par rapport à la version du dépôt ;
— les fichiers non présents sur le dépôt.

• `$ git diff` montre les modifications réalisées et non ajoutés sur les fichiers par rapport à leur version du dépôt. On peut voir les modifications sur un fichier en particulier avec `$ git diff <fichier>.` L'option `--cached` montre les différences entre les fichiers ajoutés avec un `$ git add` et les fichiers présents sur le dépôt.

• `$ git reset` annule un `$ git add` : `$ git reset [fichier]` .

• `$ git rm` supprime un fichier : `$ git rm <fichier>` .

• `$ git mv` renomme un fichier : `$ git mv <fichier> <destination>` .

• `$ git log` montre la liste des commits.

• `$ git stash` met temporairement de côté tous les fichiers modifiés pour revenir à la version du

dépôt. Particulièrement utile, l'air de rien, à savoir utiliser.

- `$ git stash list` liste les `$ git stash` réalisés.

- `$ git stash pop` rétablis les modifications mises de côté par le `$ git stash` le plus récent.

- `$ git stash drop` supprime les modifications mises de côté par le `$ git stash` le plus récent.

- `$ git checkout` annule les modifications dans votre copie du fichier et revient à la version du dépôt : `$ git checkout <fichier>` .

- `$ git blame` donne, pour chaque ligne d'un fichier, le code et l'auteur du dernier commit ayant modifié cette ligne : `$ git blame <fichier>` . L'intérêt est de trouver le responsable d'un code peu rigoureux pour le dénoncer à l'administration de votre département. Les étudiants dénoncés à plus de trois reprises feront l'objet de sanctions pouvant aller jusqu'à la lapidation et le retrait de deux points sur leur permis de conduire.

- `$ git show` permet de montrer les modifications contenues dans un commit :
`$ git show <code du commit>` . Le code du commit (ou révision ou hash) se trouve grâce au `$ git blame` ou `$ git log` .

- `$ git tag -a <annotation>` permet d'étiqueter la version courante du dépôt avec une annotation (d'où le `-a`). Très utile pour indiquer clairement l'état d'avancement du projet lié au dépôt sans polluer les messages de commit. Peut s'agrémenter d'un message explicatif à l'aide de l'option `-m`. S'utilise notamment pour les numéros de version du projet :
`$ git tag -a V2.1 -m 'Ajout du chiffrement de la BDD'` . L'idée ici est de se simplifier le travail à venir. En effet l'exemple de tag donné ci-dessus permet de répondre facilement à la question « À quel commit correspond la version 2.1 du projet et à quoi correspond cette version ? ». Les tag permettent de rapidement revenir à le dernière version stable ou fonctionnelle d'un projet par exemple.

- `$ git tag` liste les tags faits sur le dépôt.

Les plus motivés peuvent aussi essayer de voir l'utilisation de `$ git bisect` , mais c'est hors programme et cela ne vous sera pas demandé lors du TOEIC.

3 Le fichier .gitignore

Il y a certains fichiers que l'on ne veut pas mettre sur le dépôt. Ce qu'on appelle les résidus de compilation (fichiers binaires, .o, .a en C par exemple) ou qui peuvent être générés à partir du contenu du dépôt (comme la documentation). Pour indiquer à git de ne pas tenir compte de ces fichiers, on peut créer un fichier `.gitignore` dans lequel on spécifie les noms des documents que l'on ne veut pas voir sur le dépôt. La wildcard (`*`) fonctionne pour généraliser les noms de fichiers. Le .gitignore peut lui-même être ajouté ou non au dépôt.

Exemple de `.gitignore` pour un document LaTeX :

```
*.log
```

```
*.aux
*.out
*.pdf
```

À vous de l'adapter à votre projet.

4 Pour aller plus loin

Un peu plus loin, avec la cheat sheet duckduckgo : `https://duckduckgo.com/?q=git+cheat+sheet&t=ffsb&ia=cheatsheet&iax=1`.

Beaucoup plus loin, avec la documentation officielle : `https://git-scm.com/doc`. Les illustrations valent le détour pour comprendre plus en profondeur la structure de données en graphe orienté acyclique.

Une autre cheat sheet : `http://ndpsoftware.com/git-cheatsheet.html`.

Un tutoriel interactif est disponible à cette adresse : `https://try.github.io/`.

C'est la panique ? Vous avez (encore) tout cassé ? `http://justinhileman.info/article/git-pretty/`

Deux outils sont intéressants à utiliser dans un environnement git : tig (`https://jonas.github.io/tig/`) pour remplacer `$ git log` et git-imerge (`https://github.com/mhagger/git-imerge`) pour gérer les conflits plus efficacement.

5 Derniers conseils

Git est assez explicite dans les messages qu'il renvoie après une commande. Prenez le temps de les lire, cela aide souvent à résoudre les problèmes.

Prenez le temps d'apprendre à utiliser git, c'est un outil incontournable si l'on veut faire de l'informatique. Remarque : effacer son dépôt local et faire un `$ git clone` à chaque fois que l'on a un conflit lors d'un `$ git pull` n'est pas une bonne pratique et est souvent contre-productif. Savoir résoudre des conflits sans tout péter est une compétence nécessaire, avec git comme dans la vie.

Bonne continuation mes petits farfadets.

Des suggestions sur ce git de survie ? Contactez-moi à jeanbaptiste.louvet@insa-rouen.fr.