

Tutoriel Git et GitHub

Lucien Cartier-Tilet

October 5, 2018

Contents

| | | |
|----------|--|----------|
| 1 | Git ? Qu'est-ce donc ? | 3 |
| 2 | Ça a l'air cool, comment ça s'utilise ? | 3 |
| 2.1 | Et surtout, comment ça s'installe ? | 3 |
| 2.2 | Ok c'est bon, et il y a une configuration à faire ? | 3 |
| 3 | Ok très bien, mais je veux utiliser git moi ! | 4 |
| 3.1 | Je commence comment ? | 4 |
| 3.2 | Et pour ajouter des fichiers ? | 4 |
| 3.3 | Cool, mais j'ai accidentellement mis un fichier en staging | 5 |
| 3.4 | En fait, j'ai juste oublié un truc dans mon commit précédent | 5 |
| 3.5 | Euh, j'ai oublié ce que j'ai changé lors du dernier commit | 6 |
| 3.6 | Il y a des fichiers dont je me fiche dans mon dépôt | 7 |
| 4 | J'ai entendu parler de GitHub... | 7 |
| 4.1 | J'ai téléchargé un projet en zip | 7 |
| 4.2 | Et si je veux créer mon propre dépôt sur GitHub | 8 |
| 4.3 | Et du coup, comment je met tout ça en ligne ? | 8 |
| 4.4 | Quelqu'un a fait des modifications depuis mon dernier commit, je récupère ça comment ? | 8 |
| 4.5 | Je suis en train de travailler sur le même fichier que Ginette | 9 |

1 Git ? Qu'est-ce donc ?

Git est un logiciel de version de fichiers permettant de garder une trace de toutes les modifications apportées aux fichiers suivis dans un répertoire (un dépôt) et ses sous-répertoires –sous couvert qu'ils n'aient pas été ignorés explicitement. Il permet également de conserver plusieurs versions parallèles du projet, comme par exemple une version stable et une version de développement, et permet l'ajout de modifications d'une de ces versions parallèles à une autre via des fusions partielles ou totales de branches, avec une automatisation des fusions de fichiers lorsqu'il n'y a pas de conflit entre ces derniers.

Avant de continuer, sache que je suis bilingue français-sarcasme, si tu es du genre à t'énerver pour un rien, cette page est à haut risque pour toi.

Toujours là ? Tu auras été prévenue.

2 Ça a l'air cool, comment ça s'utilise ?

2.1 Et surtout, comment ça s'installe ?

Très bonne question Kévin. Tout d'abord, il faut t'assurer que git soit installé sur ton système et utilisable depuis le terminal. Sous GNU/Linux, tu peux l'installer via ton gestionnaire de paquet, ce qui rendra la commande accessible directement depuis le terminal.

```
$ apt install git           # Debian, Ubuntu et les distros basées dessus
$ yum install git           # CentOS
$ dnf -y install git        # Fedora
$ pacman -S git             # ArchLinux et les distros basées dessus
$ emerge --ask --verbose dev-vcs/git # >install Gentoo
```

Si tu n'es pas sous GNU/Linux mais que tu as au moins le goût d'être sous un OS de type Unix, tu peux exécuter la commande correspondante à ton OS suivant :

```
$ pkg install git           # FreeBSD
$ brew install git          # macOS avec brew
$ port install git +svn +doc +bash_completion +gitweb # macOS avec MacPorts
```

Si tu es sous Windows... Bonne chance. Toutes les commandes seront en syntaxe Unix dans ce tutoriel, mais si tu as bien deux neurones, tu devrais pouvoir tout de même suivre le tutoriel.

2.2 Ok c'est bon, et il y a une configuration à faire ?

Tu peux configurer Git si tu le souhaites, oui. En général, il est recommandé de paramétrer au moins son nom et son email. Tu peux les paramétrer via la ligne de commande :

```
$ git config --global user.name "Ton Nom"
$ git config --global user.email "ton@email.truc"
```

Tu peux aussi éditer le fichier ~/.gitconfig comme suit :

```
[user]
  email = ton@email.truc
  name = Ton nom
```

Cela permettra d'associer ton nom et ton adresse mail à tes commits. Par défaut, ceux qui sont enregistrés avec ton compte utilisateur de ton PC sont mis par défaut dans ces paramètres, mais on met quasiment tous un nom à la con quand on le crée. Et ça permet d'avoir les mêmes paramètres si tu es sur un autre ordinateur.

Il y a encore pas mal de paramètres que tu peux gérer avec ce fichier, je reparlerai de certains plus tard, mais pour le reste, la documentation en ligne sur gitconfig ne manque pas.

3 Ok très bien, mais je veux utiliser git moi !

Du calme Jean-Kévin, ralentis un peu. Comme le dit ce vieux dicton Chinois :

Celui qui marche trop vite..... marche..... trop... vite...? Tu peux tomber et te faire mal je suppose.

Bon, c'est une contrefaçon, donc la qualité de la citation n'est pas extraordinaire. Bref.

3.1 Je commence comment ?

Si tu souhaites créer un dépôt git, rien de plus simple : créé ton répertoire dans lequel tu travailleras, et déplace-y-toi. Ensuite, tu pourras initialiser ton dépôt via la commande `git init`.

```
$ mkdir monsuperprojet
$ cd monsuperprojet
$ git init
Initialized empty Git repository in /tmp/monsuperprojet/.git/
```

Si tu obtiens à peu près le même message après la dernière commande, félicitations ! Tu viens de créer ton premier dépôt git. En l'occurrence, j'ai créé mon dépôt dans `/tmp`, mais toi tu peux voir un truc du genre `/home/corentin/monsuperprojet` à la place. Tu peux vérifier que tout va bien en rentrant la commande `git status`

```
$ git status
On branch master
```

```
No commits yet
```

```
nothing to commit (create/copy files and use "git add" to track)
```

Parfait !

3.2 Et pour ajouter des fichiers ?

Maintenant tu peux commencer à travailler sur ton projet. Mais tout d'abord, on va voir ce qu'il se passe si jamais on crée un fichier dans le dépôt. Créé un fichier `main.c` dans lequel tu vas entrer ce code :

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello World!\n");
    return 0;
}
```

En exécutant à nouveau `git status`, on peut voir la sortie suivante :

```
$ git status
On branch master
```

```
No commits yet
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

```
main.c
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

Tu commences à comprendre un peu le bail ? Git vient de détecter qu'un nouveau fichier a été créé qu'il ne connaissait pas avant. Suivons ses bon conseils et ajoutons le fichier au dépôt.

```
$ git add main.c
$ git status
On branch master
```

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

```
new file:   main.c
```

Super, maintenant git va surveiller les changements du fichier, mais attention, il n'a pas encore enregistré son état. Pour l'instant il sait juste que le fichier est là, dans un certain état, mais rien ne garanti encore qu'on pourra retrouver cet état plus tard. On appelle ça le *staging*. Pour ce faire, il faut créer ce qu'on appelle un commit. En gros, il s'agit d'un enregistrement des modifications apportées à un ou plusieurs fichiers (dans leur globalité ou partiellement, on verra ça plus tard), le tout avec un commentaire.

```
$ git commit -m "Un petit pas pour moi, un grand pas pour mon projet"
[master (root-commit) 89139ef] Un petit pas pour moi, un grand pas pour mon projet
1 file changed, 6 insertions(+)
create mode 100644 main.c
```

Parfait ! Certains éléments peuvent être un peu différent chez toi, comme par exemple la référence du commit juste avant le message. Ça, c'est un truc qui est géré automatiquement par git. Et voilà, on a l'état de notre répertoire qui est enregistré et qui sera disponible plus tard. Maintenant, tu sais comment enregistrer des état de ton dépôt via les commits.

3.3 Cool, mais j'ai accidentellement mis un fichier en staging

Si jamais tu as un staging que tu veux annuler, tu peux utiliser la commande `git reset HEAD nomdetonfichier voiredautresfichiers` pour annuler le staging. Une fois le fichier qui n'est plus dans ton staging, tu peux même annuler toutes les modifications que tu as apporté au fichier depuis ton dernier commit avec la commande `git checkout -- tonfichier voiredautresfichiers`. Par exemple, si j'ai modifié mon `main.c` en modifiant ainsi les arguments du `main()` :

```
#include <stdio.h>

int main(void) {
    printf("Hello World!\n");
    return 0;
}
```

Je peux annuler tout ça via ces commandes :

```
$ git reset HEAD main.c
Unstaged changes after reset:
M      main.c
$ git checkout -- main.c
$ git status
On branch master
nothing to commit, working tree clean
```

Si je fait un cat du fichier, je vois qu'il est revenu à son état initial.

3.4 En fait, j'ai juste oublié un truc dans mon commit précédent

Si jamais tu veux à la place ajouter la modification d'un fichier au dernier commit (mettons, tu as oublié d'ajouter également un fichier texte), tu peux utiliser l'option `--amend` lors du commit du fichier oublié.

```
$ git add main.c # J'ai refait les modifications annulées plus tôt
$ git commit -m "second commit"
[master 97f698a] second commit
 1 file changed, 1 insertion(+), 1 deletion(-)
$ echo "C'est un super projet !" > projet.txt
$ git add projet.txt
$ git commit --amend -m "second commit + oubli"
[master 9aff4c0] second commit + oubli
Date: Fri Oct 5 11:10:56 2018 +0200
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 projet.txt
```

En gros, le commit que tu viens de faire a remplacé le précédent en conservant les informations du commit précédent, mis à part son commentaire. Si tu ne met pas l'option `-m "ton texte"` lors de l'amendement du commit, ton éditeur texte par défaut va s'ouvrir pour que tu puisses modifier le texte du commit précédent si tu le souhaites. Si jamais `vi` ou `vim` s'ouvre et que tu n'as aucune idée de comment sortir de cet enfant du démon, tu as juste à appuyer sur la touche Échap (au cas où), puis à taper `:wq` (w pour écrire le fichier, q pour quitter), puis tu appuie sur la touche Entrée. Si tu as Nano qui s'est ouvert, alors il faut taper `Ctrl-X`.

3.5 Euh, j'ai oublié ce que j'ai changé lors du dernier commit

Pas de panique ! Tu peux entrer la commande `git diff` afin de voir ce que tout ce que tu as modifié lors de ton dernier commit. Et si tu ne souhaite voir les modifications que d'un certain fichier, tu peux ajouter le nom de ton fichier à la fin de la commande.

```
$ echo "C'est un super projet !" > projet.txt
$ git diff
diff --git a/projet.txt b/projet.txt
index 03b0f20..b93413f 100644
--- a/projet.txt
+++ b/projet.txt
@@ -1,1 @@
-projet
+C'est un super projet !
```

Tu peux également voir les différences de fichiers entre deux commits en entrant leur référence. Pour avoir la référence, tu peux rentrer la commande `git log` pour avoir un petit historique des commits.

```
$ git log
commit 4380d8717261644b81a1858920406645cf409028 (HEAD -> master)
Author: Phuntsok Drak-pa <phundrak@phundrak.fr>
Date: Fri Oct 5 11:59:40 2018 +0200
```

new commit

```
commit 59c21c6aa7e3ec7edd229f81b87becbc7ec13596
Author: Phuntsok Drak-pa <phundrak@phundrak.fr>
Date: Fri Oct 5 11:10:56 2018 +0200
```

nouveau texte

```
commit 89139ef233d07a64d3025de47f8b6e8ce7470318
Author: Phuntsok Drak-pa <phundrak@phundrak.fr>
Date: Fri Oct 5 10:56:58 2018 +0200
```

Un petit pas pour moi, un grand pas pour mon projet

Bon, c'est un peu long et un peu trop d'infos d'un coup, généralement je préfère taper `git log --oneline --graph --decorate` afin d'avoir un affichage comme suit :

```
$ git log --oneline --graph --decorate
* 4380d87 (HEAD -> master) new commit
* 59c21c6 nouveau texte
* 89139ef Un petit pas pour moi, un grand pas pour mon projet
```

Plus propre, non ? Et les références sont plus courtes, ce qui est plus agréable à taper. Allez, comparons les deux derniers commits.

```
$ git add .
$ git commit -m "new commit"
$ git log --oneline --graph --decorate
* 4380d87 (HEAD -> master) new commit
* 59c21c6 nouveau texte
* 89139ef Un petit pas pour moi, un grand pas pour mon projet
$ git diff 59c21c6 4380d87
diff --git a/projet.txt b/projet.txt
index 03b0f20..b93413f 100644
--- a/projet.txt
+++ b/projet.txt
@@ -1,1 @@
-projet
+C'est un super projet !
```

3.6 Il y a des fichiers dont je me fiche dans mon dépôt

Dans ce cas, il est grand temps de te présenter le fichier `.gitignore`. Comme son nom l'indique, il permet au dépôt d'ignorer des fichiers selon ce que tu lui indiqueras. Par exemple, si tu veux ignorer tous les fichiers qui se terminent en `.out` (ou `.exe` sous Windows), tu peux éditer (ou créer) ton `.gitignore` et entrer ces lignes :

```
*.out
*.exe
```

Maintenant, si tu crées un fichier en `.out` ou `.exe`, il sera complètement ignoré par git et ne sera pas stocké dans l'historique des versions.

4 J'ai entendu parler de GitHub...

Tu commences à me plaire Enzo ! GitHub est un site web sur lequel tu peux héberger des projets libres ou open-source (si tu ne connais pas la différence, voici [un article](#) pour t'aider à comprendre, et [un autre](#) pour la route). C'est en particulier orienté pour les projets gérés par git, ce qui tombe bien car c'est ce qu'on utilise. Cela a pour avantage de pouvoir aisément partager ton code et d'assurer qu'il est bien sauvegardé quelque part d'autre que ton disque dur (un `rm -rf` est si vite arrivé). Et surtout, ça peut te permettre de collaborer avec d'autres personnes sur le même projet sans te casser la tête.

4.1 J'ai téléchargé un projet en zip

Ou bien, tu peux télécharger le projet directement via git. Eh oui ! git permet de gérer les dépôts dits distants, c'est à dire ceux qui sont hébergés sur un serveur en ligne, comme par exemple sur GitHub. Pour cela, il te faut te munir du lien vers le dépôt git, et le passer en argument de `git clone`. Par exemple, si tu veux télécharger de dépôt du petit logiciel de chat en réseau que j'ai codé durant ma L2 d'informatique, tu peux exécuter ceci :

```
$ git clone https://github.com/Phundrak/chat-reseau-P8.git
Cloning into 'chat-reseau-P8'...
remote: Enumerating objects: 345, done.
remote: Total 345 (delta 0), reused 0 (delta 0), pack-reused 345
Receiving objects: 100% (345/345), 63.91 KiB | 39.00 KiB/s, done.
Resolving deltas: 100% (107/107), done.
```

Et c'est bon, tu as accès au répertoire chat-reseau-P8 et au code source du projet.

4.2 Et si je veux créer mon propre dépôt sur GitHub

Dans ce cas là, c'est simple Brigitte. Il faut que tu te crées un compte sur GitHub, puis tu cliques sur le bouton + et New Repository. Tu lui donnes le nom que tu souhaites (en l'occurrence je le nomme temporary-repo car je vais le supprimer cinq minutes après l'écriture de ces lignes), et tu cliques sur Create Repository. Tu n'ajoutes rien avant, pas de description, pas de .gitignore, RIEN.

Et là, magie ! GitHub indique comment ajouter le dépôt distant à ton dépôt local.

```
$ git remote add origin https://github.com/Phundrak/temporary-repo.git
```

Et voilà, ton dépôt est lié au dépôt distant. Oui, juste comme ça.

4.3 Et du coup, comment je met tout ça en ligne ?

Bon ok, ce n'est pas aussi simple que ça. Une fois que tu as lié ton dépôt au dépôt distant, il faudra que tu mettes en ligne tes commits quand tu en auras l'occasion. Pour ce faire, tu n'as qu'à taper `git push` ; et la première fois, il faudra que tu indiques à ton dépôt où mettre en ligne précisément dans le dépôt distant, auquel cas tu ajoutes `-u origin master` pour cette première fois. Git te demandera donc tes identifiants GitHub pour pouvoir mettre tout ça en ligne.

```
$ git push -u origin master
Username for 'https://github.com': phundrak
Password for 'https://phundrak@github.com':
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (10/10), 940 bytes | 313.00 KiB/s, done.
Total 10 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/Phundrak/temporary-repo/pull/new/master
remote:
To https://github.com/Phundrak/temporary-repo.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Bon, là en nom d'utilisateur y'a le mien, faudra remplacer avec le tiens. Et ouais, ma vitesse de mise en ligne n'est pas fameuse, je suis sur une connexion 3G+ à l'heure où j'écris ces lignes, ne me juge pas. Bref, toujours est-il que je viens de mettre en ligne les fichiers du dépôt sur GitHub. Pas la peine de chercher le mien sur GitHub par contre, ça fera un bail que je l'aurai supprimé au moment où tu liras ces lignes.

4.4 Quelqu'un a fait des modifications depuis mon dernier commit, je récupère ça comment ?

Pour faire un exemple, je viens de créer un README.md sur GitHub directement. Ce type de fichiers est assez standard afin de présenter plus ou moins en détails le dépôt et le projet qui y est lié, et son contenu apparaîtra formaté sur la page du dépôt sur GitHub s'il est au format .md (Markdown) ou .org (org-mode, le Markdown d'Emacs avec lequel est écrit ce tutoriel). Mais il n'est pas présent dans mon dépôt local, du coup je vais devoir le récupérer. On va donc entrer `git pull`.

```
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```



```
From https://github.com/Phundrak/temporary-repo
 4380d87..8bd4896  master    -> origin/master
Updating 4380d87..8bd4896
Fast-forward
 README.md | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 README.md
```

4.5 Je suis en train de travailler sur le même fichier que Ginette

Là, c'est un problème qui aurait pu être évité avec l'usage des branches dont je t'avais parlé plus haut, mais visiblement, vous êtes sur la même branche. Pas bien. Dans ce cas-là, met-toi d'accord avec Ginette pour savoir qui fait ses push en premier. Si le choix tombe sur Ginette, ou si elle a imposé sa vision des choses et a fait son push avant toi, GitHub va râler car tu n'es pas à jour. Dans ce cas ne panique pas, si tu n'as pas fait tes commits, lance la commande `git stash` ; ça va sauvegarder tes modifications dans un coin à part et va annuler tes modifications