Utilisation de Git

Ce document contient :

- L’utilisation de Git à travers un terminal.

Gestionnaire de versions

(Gestionnaire de révisions, gestionnaire de sources)

Logiciel permettant de gérer les modifications de documents, programme informatique, la majorité des types d'information.

Il permet aussi de restaurer un fichier vers une version antérieure, on peut revenir ainsi en arrière quand cela est nécessaire.

Il permet de garder un journal (log) de toutes les modifications (dates, auteurs, motifs indiqués, etc.) et de visualiser ces différentes modifications.

Ainsi, il sera possible de déterminer grâce à ce mécanisme la ou les modifications qui ont pu causer des problèmes au bon fonctionnement d’un programme quelconque.

Dépôts d'un gestionnaire de versions

Local : le dépôt est sur la machine de l'utilisateur. Les autres utilisateurs intéressés par ce travail ne pourront pas participer localement.

Centralisé : le dépôt est sur un serveur central. La personne va travailler sur une copie locale sur sa machine et va envoyer les modifications sur le serveur.

Distribué : le dépôt est décentralisé sur plusieurs serveurs, y compris les ordinateurs des utilisateurs. Chacun a sa propre copie. Si un serveur est perdu, la copie reste disponible à travers les autres serveurs. Les utilisateurs ont ainsi une copie locale de l'historique complet du projet.

Logiciels libres de gestion de versions

Centralisé : CVS (1986/1990), Subversion (SVN, 2000)

Distribué : Git (2005), Mercurial (2005)

Git

Développer en 2005 pour la gestion des modifications du noyau Linux.

Développer avec comme objectifs : rapidité, simple d'utilisation, gestion de projets complexes (milliers de branches parallèles et une grosse quantité de données) et avoir un fonctionnement complètement distribué.

Quand vous modifiez un fichier, et entérinez ces modifications, Git prend un instantané (« snapshot ») de ce fichier. Il enregistre par la suite une référence vers cet instantané.

Toutes les opérations de Git sont locales, vu que l'utilisateur travaille sur une copie locale et tous les outils sont installés localement sur la machine de l'utilisateur.

L'intégrité des données est assurée par Git à travers l'utilisation de l'empreinte SHA-1.

Cette empreinte représente une chaîne de 40 caractères hexadécimaux (0 à 9 et A à F). Elle est calculée en fonction du contenu du fichier.

Cette empreinte est unique à un fichier. Elle permet de sauvegarder un fichier avec un contenu différent, peu importe son nom du fichier. Il s'agit donc d'une nouvelle version.

Installation

Vous disposez de plusieurs options :

https://git-scm.com/downloads

Puis installer la version associée à votre système d'exploitation.

Sur Windows, vous pouvez cliquer sur « bash.exe » si vous êtes à l'aise avec les commandes « bash ». Il s'agit d'une fenêtre « dos » classique. Ces deux fenêtres configurent par défaut le chemin pour trouver l'exécutable « git ».

Pour valider que le programme s’est installé correctement :

```
git --version
```

```
git version 2.47.1.windows.2
```

En date de l’écriture de ce document, la version disponible est :

```
2.47.1
```

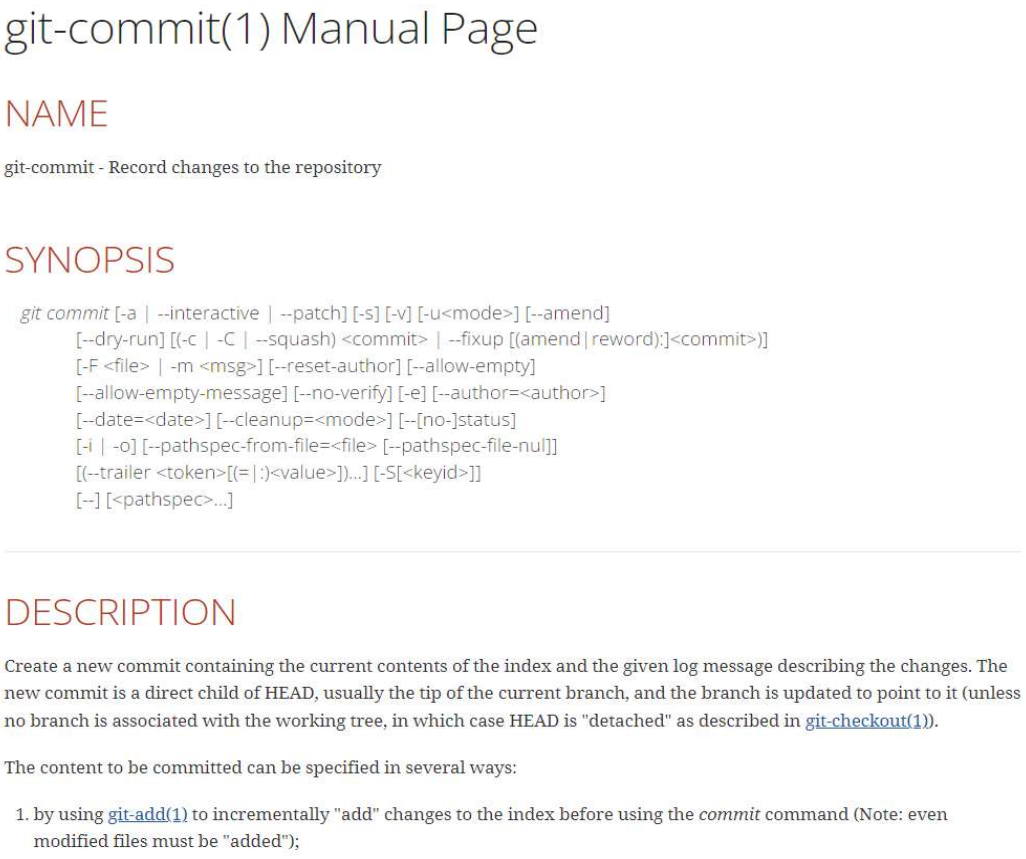
Pour obtenir de l’aide :

```
git --help
```

Pour obtenir de l'aide sur une commande en particulier :

```
git commit --help
```

Le système va ouvrir dans ce cas une nouvelle fenêtre dans un navigateur :



git-commit(1) Manual Page

NAME

git-commit - Record changes to the repository

SYNOPSIS

git commit [-a | --interactive | --patch] [-s] [-v] [-u<mode>] [--amend]
[--dry-run] [(-c | -C | --squash) <commit> | --fixup [(amend|reword):]<commit>)]
[-F <file> | -m <msg>] [--reset-author] [--allow-empty]
[--allow-empty-message] [--no-verify] [-e] [--author=<author>]
[--date=<date>] [--cleanup=<mode>] [--[no-]status]
[-i | -o] [--pathspec-from-file=<file> [--pathspec-file-nul]]
[(--trailer <token>[(=|:)<value>])...] [-S[<keyid>]]
[--] [<pathspec>...]

DESCRIPTION

Create a new commit containing the current contents of the index and the given log message describing the changes. The new commit is a direct child of HEAD, usually the tip of the current branch, and the branch is updated to point to it (unless no branch is associated with the working tree, in which case HEAD is "detached" as described in git-checkout(1)).

The content to be committed can be specified in several ways:

1. by using git-add(1) to incrementally "add" changes to the index before using the *commit* command (Note: even modified files must be "added");

Si on veut que l'aide soit affichée dans la fenêtre « dos » ou « bash » :

`git commit -h`

```
C:\MesLogiciels\PortableGit>git commit -h
usage: git commit [-a | --interactive | --patch] [-s] [-v] [-u<mode>] [--amend]
                  [--dry-run] [(-c | -C | --squash) <commit> | --fixup [(amend|reword):]<commit>]
                  [-F <file> | -m <msg>] [--reset-author] [--allow-empty]
                  [--allow-empty-message] [--no-verify] [-e] [--author=<author>]
                  [--date=<date>] [--cleanup=<mode>] [--[no-]status]
                  [-i | -o] [--pathspec-from-file=<file> [--pathspec-file-nul]]
                  [(--trailer <token>[(=|:)<value>])...] [-S[<keyid>]]
                  [--] [<pathspec>...]

    -q, --[no-]quiet      suppress summary after successful commit
    -v, --[no-]verbose    show diff in commit message template
```

Configuration

Git garde une traçabilité des modifications en y incluant le nom de l'utilisateur ayant apporté ces modifications et ses coordonnées.

Il faut donc ajouter ces coordonnées dans la configuration de Git, comme suit :

```
git config --global user.name "Mohamed Lokbani"
git config --global user.email "lokbani@iro.umontreal.ca"
```

Pour lister vos paramètres :

```
git config --list
```

Il est possible de configurer son éditeur préféré pour Git à l'aide de cette commande :

```
$ git config --global core.editor "c:/Windows/System32/notepad.exe"
```

L'exemple ci-dessus est pour l'éditeur « notepad ».

Le paramètre « global » permet d'étaler la configuration à l'ensemble des dépôts créés avec Git. Nous pouvons restreindre la configuration à l'aide du paramètre « local » pour cibler uniquement un dépôt en particulier.

Processus de suivi de versions sous Git

L’archivage des versions sous Git passe par plusieurs étapes :

1- Étape d’initialisation

Il faut d’abord dire à Git d’initialiser les paramètres d’un dépôt à créer dans un répertoire donné. On utilise pour cela la commande :

```
git init
```

Ouvrir une fenêtre «bash » disponible dans le répertoire « bin » associé à l'installation de « git ».

On crée un répertoire « Test » :

```
lokbani@dirot32 MINGW64 ~/Desktop
$ mkdir Test
```

On se déplace dans ce répertoire :

```
lokbani@dirot32 MINGW64 ~/Desktop
$ cd Test
```

On vérifie le contenu, on voit bien qu'il est vide :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test
$ ls -la
total 4
drwxr-xr-x 1 lokbani 1049089 0 Mar  4 10:25 ./
drwxr-xr-x 1 lokbani 1049089 0 Mar  4 10:25 ../
```

On initialise les paramètres de Git pour le répertoire « Test » :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test
$ git init
```

On obtient ce message :

```
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in C:/Users/lokbani/Desktop/Test/.git/

lokbani@dirot32 MINGW64 ~/Desktop/Test (master)
```

On vérifie de nouveau le contenu :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (master)
$ ls -la
total 8
drwxr-xr-x 1 lokbani 1049089 0 Mar  4 10:26 ./
drwxr-xr-x 1 lokbani 1049089 0 Mar  4 10:25 ../
drwxr-xr-x 1 lokbani 1049089 0 Mar  4 10:26 .git/
```

On constate la présence du répertoire caché « .git ». Ce répertoire est utilisé par Git pour stocker toutes les informations relatives à ce dépôt.

Si ce répertoire est effacé, on perd toutes les traces des révisions effectuées dans ce répertoire. Il faut donc éviter de détruire ce répertoire.

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (master)
```

La branche s'appelle « master ». Nous avons reçu un avertissement, nous demandant de lui donner un autre nom :

```
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change.
…
hint:   git branch -m <name>
```

On va suivre les recommandations :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (master)
$ git branch -m main

lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
```

La branche s'appelle maintenant « main ».

2- Étape d'indexation

On doit dire à Git quels sont les fichiers à surveiller afin de les inclure par la suite dans le dépôt. On parle ici d'indexation.

On crée le fichier « LISEZMOI.txt » en y copiant le texte « Mon Projet » :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ echo 'Mon Projet' > LISEZMOI.txt
```

On liste le contenu du répertoire :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ ls -la
total 9
drwxr-xr-x 1 lokbani 1049089  0 Mar  4 11:02 ./
drwxr-xr-x 1 lokbani 1049089  0 Mar  4 10:25 ../
drwxr-xr-x 1 lokbani 1049089  0 Mar  4 10:52 .git/
-rw-r--r-- 1 lokbani 1049089 11 Mar  4 11:02 LISEZMOI.txt
```

On voit la présence du fichier « LIEZMOI.txt ». On peut examiner le contenu :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ cat LISEZMOI.txt
Mon Projet
```

On demande le statut du dépôt à l'aide de la commande :

```
git status
```

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git status
On branch main

No commits yet

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

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

lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
```

Cette commande nous informe :

`On branch main` <= Nous sommes sur la branche « main ».

`No commits yet` <= Aucune validation pour l'instant (nous allons détailler ce point plus tard).

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

Le fichier « LISEZMOI.txt » (signalé en rouge) dans le répertoire en question n'est pas suivi (indexé). Si nous voulons préserver les différentes versions de ce fichier par Git, il faut l'indexer, sinon on ignore le message.

En résumé …

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

Il n'y a aucun fichier à valider, mais il y a un fichier à indexer. On utilise pour cela la commande « git add ».

On peut ajouter un seul fichier à l'aide de la commande :

```
git add  LISEZMOI.txt
```

Si nous avons plusieurs fichiers à indexer :

```
git add  .
```

On obtient ce message :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git add LISEZMOI.txt
warning: LF will be replaced by CRLF in LISEZMOI.txt.
The file will have its original line endings in your working directory
```

L'avertissement est en rapport avec l'encodage du retour à la ligne. Windows a décidé d'en coder différemment ce retour à la ligne et bonjour les problèmes cachés quand on échange les fichiers entre différentes plateformes.

On peut masquer l'avertissement sous Windows à l'aide de la commande :

```
git config core.autocrlf true
```

Il y a d'autres possibilités en fonction des situations (Mac, Linux, il y a collaboration sur le projet ou pas, etc.). Nous vous invitons à consulter la documentation mentionnée dans la section Bibliographie pour examiner les autres possibilités.

Je demande de nouveau le statut de mon dépôt :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git status
On branch main

No commits yet

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

Quand le fichier « LISEZMOI.txt » a été indexé, sa couleur a changé du rouge au vert. Git nous informe qu'il reste l'étape de validation « commit ». Et si on change d'avis, on peut désindexer le fichier à l'aide de la commande « git rm ... ».

3- Étape de validation

Git a indexé le fichier, mais il n'a pas pris un instantané du fichier (n'a pas enregistré une version/copie). Pour ce faire, nous devons valider le fichier à l'aide de la commande « commit ». Par ailleurs, un message court doit accompagner la commande de validation. Il informe des changements apportés dans le fichier.

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git commit -m "Projet initial"
[main (root-commit) 4d35698] Projet initial
 1 file changed, 1 insertion(+)
 create mode 100644 LISEZMOI.txt
```

Si l'option « -m » est omise, Git ouvre un éditeur dans lequel la personne doit écrire son message.

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git status
On branch main
nothing to commit, working tree clean
```

Nous sommes à jour avec le gestionnaire de versions!

Les trois états

Git gère trois états dans lesquels le fichier peut résider :

Répertoire de travail (ou Working directory) : il s'agit du répertoire courant dans lequel se trouve le fichier.

Pour l'exemple, « LISEZMOI.txt » :

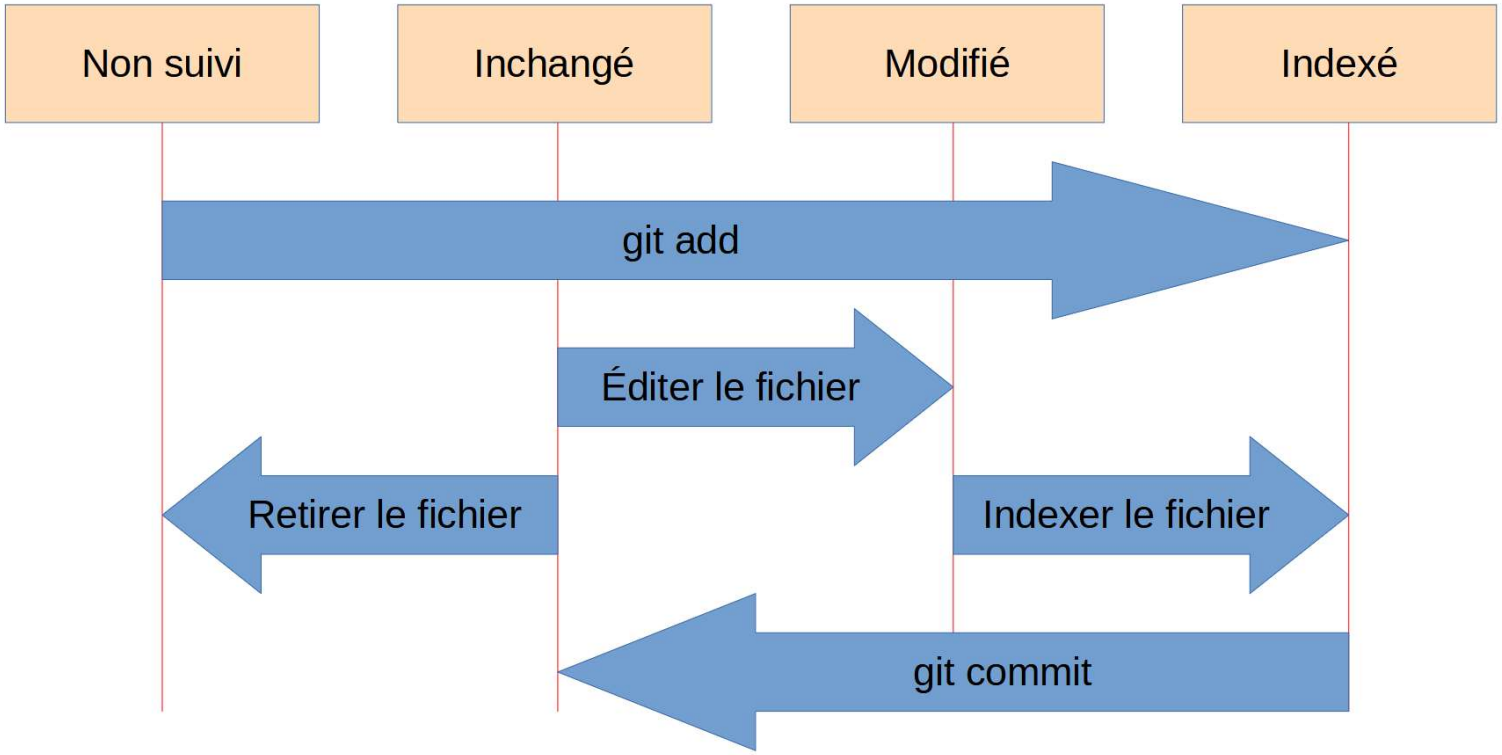
```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ pwd
/c/Users/lokbani/Desktop/Test
```

« pwd » donne le répertoire courant.

Zone index : il s'agit de l'endroit où vont résider les fichiers marqués (« indexés ») pour qu'ils fassent partie d'un prochain instantané. Nous avons utilisé pour cela la commande « git add ».

Répertoire git (dépôt) : il s'agit du répertoire git dans lequel les différentes versions du fichier sont stockées. Nous avons utilisé pour cela la commande « git commit ».

En résumé un fichier passe par plusieurs étapes …



Les fichiers peuvent-être suivis ou non suivis. Pour les fichiers suivis, ils peuvent être à l’état « inchangé », « modifié » ou « indexé ».

La commande « git status » vous donne un aperçu de l’état du fichier.

Les Log de git

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git log
commit 4d35698e32574f05a96613244a958d1d16b20c51 (HEAD -> main)
Author: Mohamed Lokbani <lokbani@iro.umontreal.ca>
Date:   Fri Mar 4 11:58:48 2022 -0500

    Projet initial
```

Pour le moment, nous n'avons fait qu'une seule validation « commit » dans ce dépôt. Cette validation est référencée par l'empreinte SHA-1 4d35698e32574f05a96613244a958d1d16b20c51. La validation contient le nom et l'adresse courriel de la personne qui l'a réalisée, ainsi que la date, l'heure et le message.

Mise à jour d'un fichier

Nous allons mettre à jour le contenu du fichier « LISEZMOI.txt » et reprendre les commandes git précédemment utilisées.

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ echo "Nous testons présentement les commandes Git" >> LISEZMOI.txt
```

Nous avons ajouté une nouvelle ligne (à l'aide de « >> »). Vous pouvez aussi éditer le fichier de manière classique avec votre éditeur préféré et ajouter la ligne en question (ou un texte de votre choix).

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ cat LISEZMOI.txt
Mon Projet
Nous testons présentement les commandes Git
```

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
directory)
        modified:   LISEZMOI.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Si on veut garder ces changements, il faut indexer de nouveau ce fichier dans git. Si on veut ignorer ces changements dans le répertoire de travail, on peut restaurer le fichier d'origine du dépôt git.

On peut aussi demander les différences qui existent entre les deux versions, comme suit :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git diff
warning: LF will be replaced by CRLF in LISEZMOI.txt.
The file will have its original line endings in your working directory
diff --git a/LISEZMOI.txt b/LISEZMOI.txt
index 1e17b0c..f8e014f 100644
--- a/LISEZMOI.txt
+++ b/LISEZMOI.txt
@@ -1 +1,2 @@
 Mon Projet
+Nous testons présentement les commandes Git
```

L'affichage du résultat de la commande « diff » est à la sauce « Linux ».

```
--- a/LISEZMOI.txt <= fichier d'origine
+++ b/LISEZMOI.txt <= fichier modifié (nouveau fichier)

index 1e17b0c..f8e014f 100644 <= étiquettes uniques pour ces versions

@@ -1 +1,2 @@
```

-1 le changement commence à la ligne 1 du fichier d'origine.
+1,2 le changement commence à la ligne 1 et implique 2 lignes du fichier destination.

```
 Mon Projet   <= aucun signe => cette ligne reste la même
+Nous testons présentement les commandes Git <= signe +, cette ligne a
besoin d'être ajoutée dans le fichier d'origine pour qu'il soit le
même que le fichier destination.
```

On indexe le fichier :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git add LISEZMOI.txt
warning: LF will be replaced by CRLF in LISEZMOI.txt.
The file will have its original line endings in your working directory
```

On prend un instantané :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git commit -m "Tester un 2e commit"
[main d404ccb] Tester un 2e commit
 1 file changed, 1 insertion(+)
```

Les logs :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git log
commit d404ccb13ae41709fc7a7ab783899ec8b24ddcee (HEAD -> main)
Author: Mohamed Lokbani <lokbani@iro.umontreal.ca>
Date:   Fri Mar 4 17:46:20 2022 -0500

    Tester un 2e commit

commit 4d35698e32574f05a96613244a958d1d16b20c51
Author: Mohamed Lokbani <lokbani@iro.umontreal.ca>
Date:   Fri Mar 4 11:58:48 2022 -0500

    Projet initial


lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git status
On branch main
nothing to commit, working tree clean
```

Dans le dépôt, nous avons donc deux versions du fichier « LISEZMOI.txt » :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git log --oneline
d404ccb (HEAD -> main) Tester un 2e commit
4d35698 Projet initial
```

HEAD référence le dernier « commit » (le plus récent).
HEAD~1 référence l'avant-dernier « commit ».

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git diff HEAD~1 LISEZMOI.txt
diff --git a/LISEZMOI.txt b/LISEZMOI.txt
index 1e17b0c..f8e014f 100644
--- a/LISEZMOI.txt
+++ b/LISEZMOI.txt
@@ -1 +1,2 @@
 Mon Projet
+Nous testons présentement les commandes Git
```

HEAD~2 référence l'avant l'avant-dernier « commit », etc.

Mais, s'il y a beaucoup de « commit », nous pouvons spécifier l'empreinte SHA-1 du fichier.

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git diff 4d35698e32574f05a96613244a958d1d16b20c51 LISEZMOI.txt
diff --git a/LISEZMOI.txt b/LISEZMOI.txt
index 1e17b0c..f8e014f 100644
--- a/LISEZMOI.txt
+++ b/LISEZMOI.txt
@@ -1 +1,2 @@
 Mon Projet
+Nous testons présentement les commandes Git
```

On peut réduire la taille de l'empreinte à quelques caractères (7) :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git log --oneline
d404ccb (HEAD -> main) Tester un 2e commit
4d35698 Projet initial


lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git diff 4d35698 LISEZMOI.txt
diff --git a/LISEZMOI.txt b/LISEZMOI.txt
index 1e17b0c..f8e014f 100644
--- a/LISEZMOI.txt
+++ b/LISEZMOI.txt
@@ -1 +1,2 @@
 Mon Projet
+Nous testons présentement les commandes Git
```

Restaurer un fichier

Le sommet (HEAD) se déplace avec les « commit ». Comme nous l'avons vu, il référence le dernier « commit ».

Nous sommes en présence de plusieurs scénarii.

1- Vous avez apporté plusieurs modifications dans le fichier « LISEZMOI.txt » qui se trouve dans le répertoire de travail et vous voulez revenir en arrière avec la dernière copie sauvegardée.

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ echo "Ne perdons pas la tête!" >> LISEZMOI.txt
```

On utilise la commande « restore » pour restaurer un fichier du dépôt.

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git restore LISEZMOI.txt
```

On vérifie le contenu du fichier :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ cat LISEZMOI.txt
Mon Projet
Nous testons présentement les commandes Git
```

Si on veut, restaurer à partir d'un autre « commit » :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git restore --source 4d35698 LISEZMOI.txt
```

On vérifie le contenu du fichier :

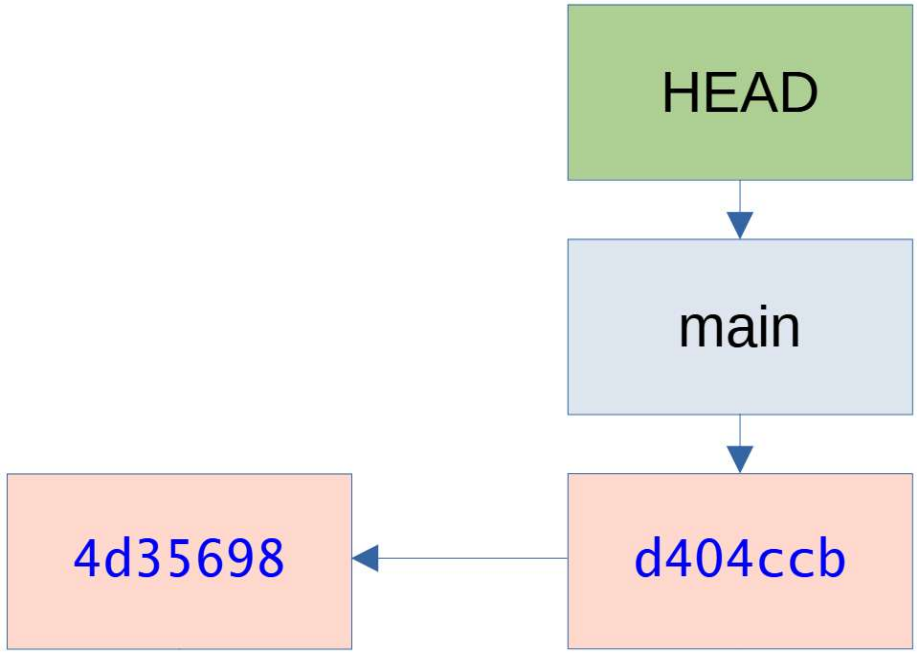
```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ cat LISEZMOI.txt
Mon Projet
```

Sauf qu'en faisant cela, le fichier restauré est en retard d'une version par rapport à celui qui se trouve à « HEAD ».

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
directory)
        modified:   LISEZMOI.txt

no changes added to commit (use "git add" and/or "git commit -a")


lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git diff
diff --git a/LISEZMOI.txt b/LISEZMOI.txt
index f8e014f..1e17b0c 100644
--- a/LISEZMOI.txt
+++ b/LISEZMOI.txt
@@ -1,2 +1 @@
 Mon Projet
-Nous testons présentement les commandes Git
```

Les branches

Chaînage de départ :



On ajoute la nouvelle branche « dev » :

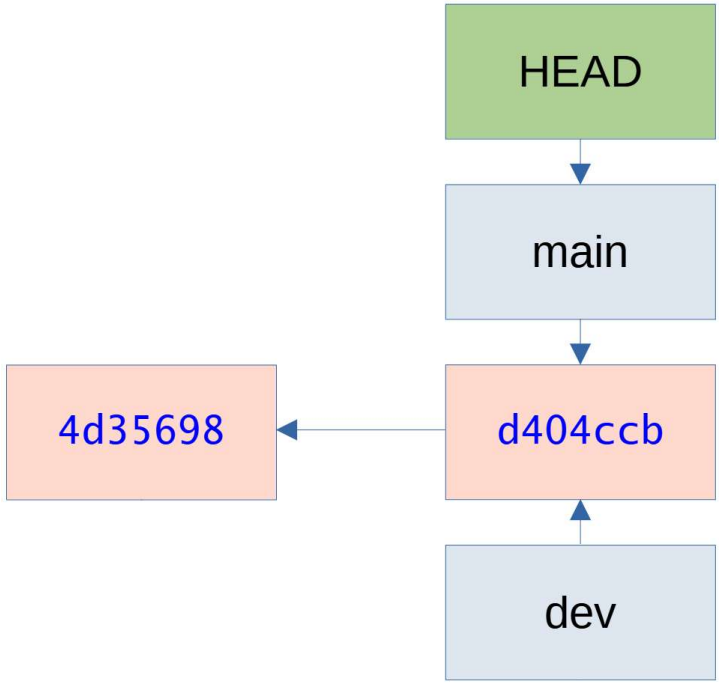
```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git branch dev
```

Pour lister les branches disponibles :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git branch -l
  dev
* main
```

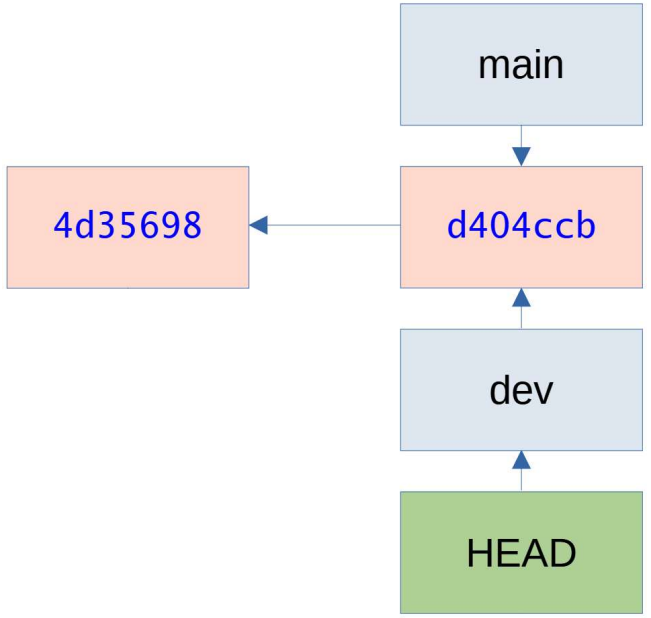
On constate l’astérisque sur la branche « main » pour signifier que le HEAD pointe vers elle.

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git log
commit d404ccb13ae41709fc7a7ab783899ec8b24ddcee (HEAD -> main, dev)
Author: Mohamed Lokbani <lokbani@iro.umontreal.ca>
Date:   Fri Mar 4 17:46:20 2022 -0500

    Tester un 2e commit

commit 4d35698e32574f05a96613244a958d1d16b20c51
Author: Mohamed Lokbani <lokbani@iro.umontreal.ca>
Date:   Fri Mar 4 11:58:48 2022 -0500

    Projet initial
```



On change de branche :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main) <= Nom actuel
$ git checkout dev
Switched to branch 'dev'
M       LISEZMOI.txt


lokbani@dirot32 MINGW64 ~/Desktop/Test (dev) <= Nom a changé
$ git log --oneline
d404ccb (HEAD -> dev, main) Tester un 2e commit
4d35698 Projet initial
```



Pour rappel le répertoire de travail contient la première version du fichier :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (dev)
$ cat LISEZMOI.txt
Mon Projet
```

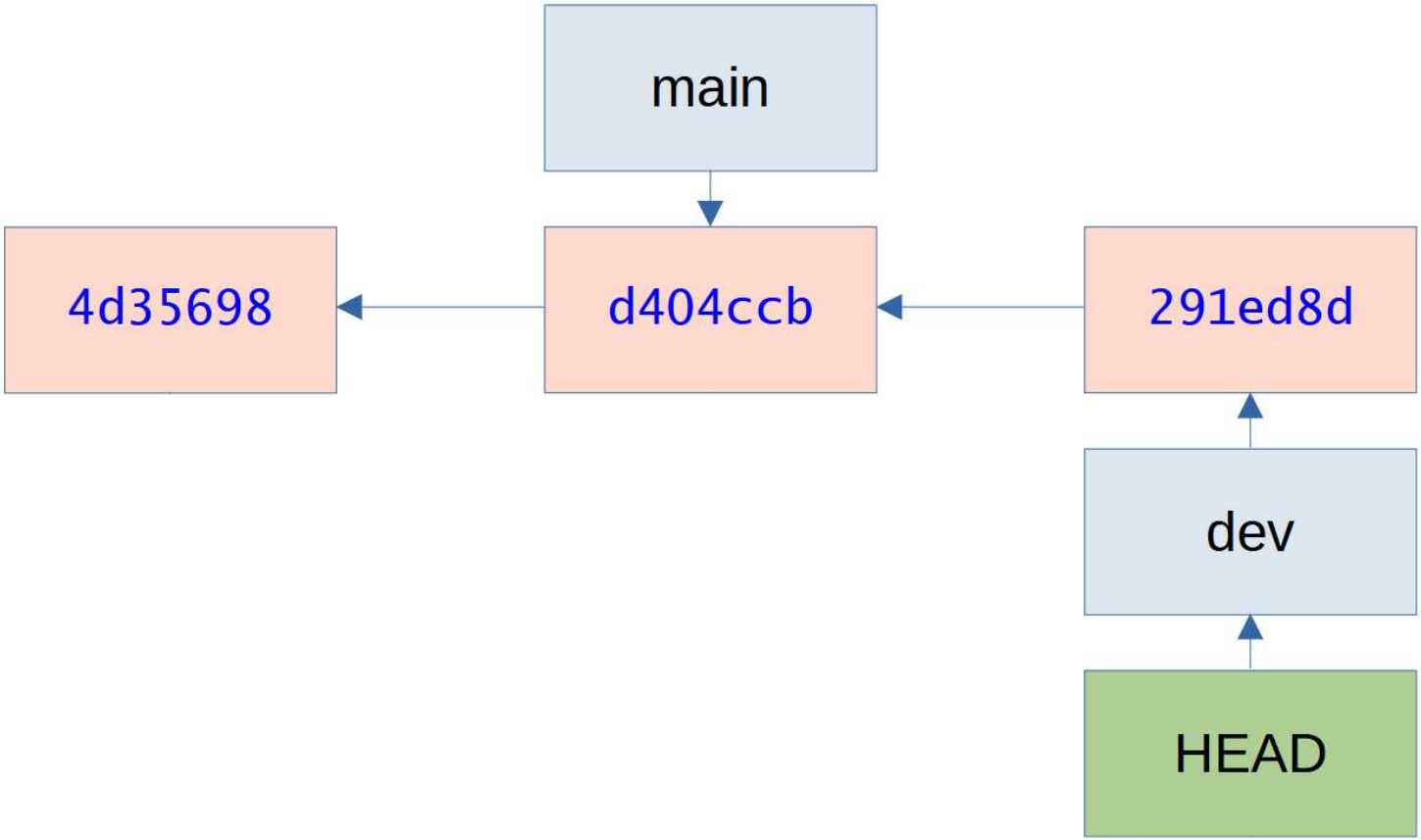
Nous restaurons avec la dernière version disponible :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (dev)
$ git restore LISEZMOI.txt

lokbani@dirot32 MINGW64 ~/Desktop/Test (dev)
$ cat LISEZMOI.txt
Mon Projet
Nous testons présentement les commandes Git
```

Maintenant, nous allons modifier le contenu du fichier :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (dev)
$ echo "L'exemple a une branche de plus!" >> LISEZMOI.txt

lokbani@dirot32 MINGW64 ~/Desktop/Test (dev)
$ cat LISEZMOI.txt
Mon Projet
Nous testons présentement les commandes Git
L'exemple a une branche de plus!

lokbani@dirot32 MINGW64 ~/Desktop/Test (dev)
$ git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
directory)
        modified:   LISEZMOI.txt

no changes added to commit (use "git add" and/or "git commit -a")

lokbani@dirot32 MINGW64 ~/Desktop/Test (dev)
$ git add LISEZMOI.txt
warning: LF will be replaced by CRLF in LISEZMOI.txt.
The file will have its original line endings in your working directory
```

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (dev)
$ git commit -m "Tester une branche"
[dev 291ed8d] Tester une branche
 1 file changed, 1 insertion(+)


lokbani@dirot32 MINGW64 ~/Desktop/Test (dev)
$ git log
commit 291ed8dcaa05bcc74b421bf04be6157081c9442d (HEAD -> dev)
Author: Mohamed Lokbani <lokbani@iro.umontreal.ca>
Date:   Fri Mar 4 22:56:31 2022 -0500

    Tester une branche

commit d404ccb13ae41709fc7a7ab783899ec8b24ddcee (main)
Author: Mohamed Lokbani <lokbani@iro.umontreal.ca>
Date:   Fri Mar 4 17:46:20 2022 -0500

    Tester un 2e commit

commit 4d35698e32574f05a96613244a958d1d16b20c51
Author: Mohamed Lokbani <lokbani@iro.umontreal.ca>
Date:   Fri Mar 4 11:58:48 2022 -0500

    Projet initial
```


main
4d35698
d404ccb
291ed8d
dev
HEAD

On vérifie l'état de la branche « dev », rien à signaler.

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (dev)
$ git status
On branch dev
nothing to commit, working tree clean
```

On se déplace vers la branche « main » :

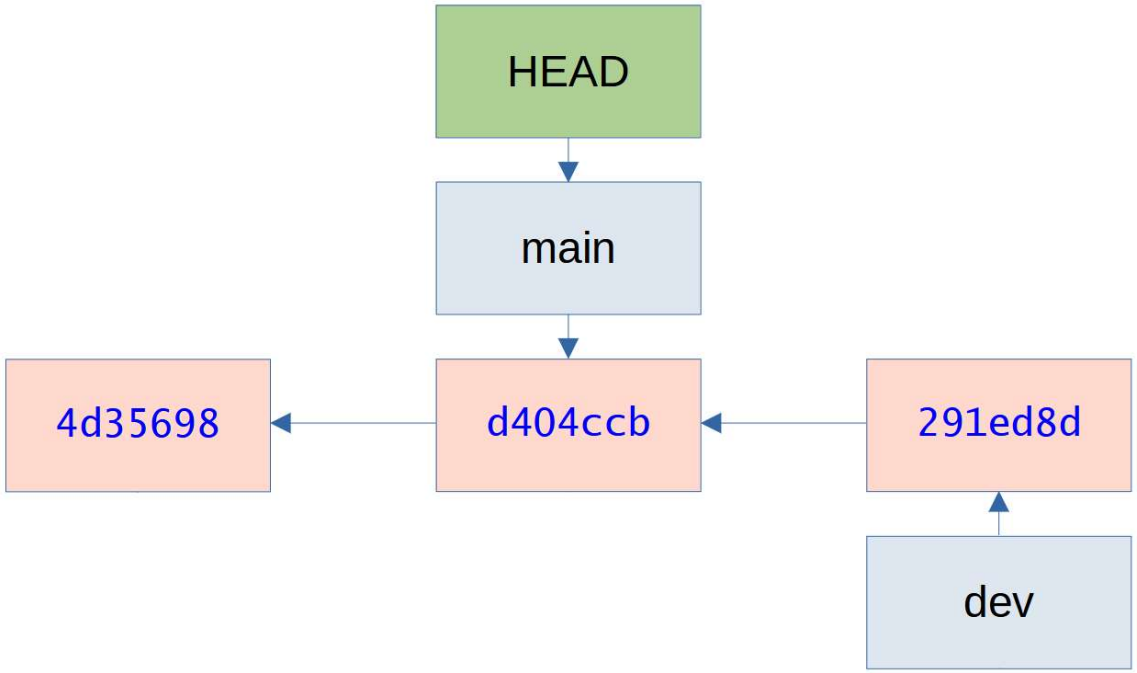
```
lokbani@dirot32 MINGW64 ~/Desktop/Test (dev)
$ git checkout main
Switched to branch 'main'
```

On vérifie l'état de la branche « main », rien à signaler.

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git status
On branch main
nothing to commit, working tree clean
```

On examine le contenu du fichier :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ cat LISEZMOI.txt
Mon Projet
Nous testons présentement les commandes Git
```

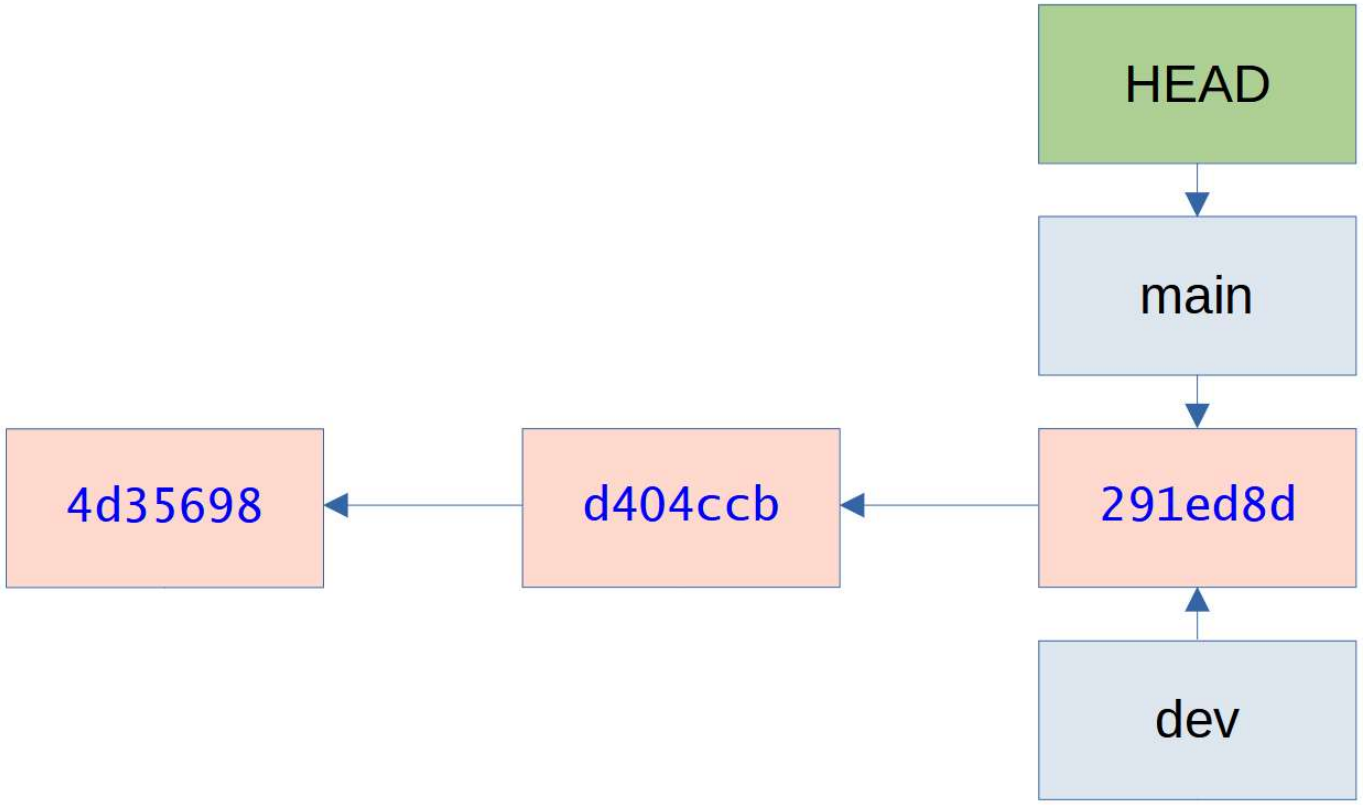


Pour fusionner les branches :

Il faut d’abord se positionner sur la branche principale, puis lancer la commande « git merge nom_branche » :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git merge dev
Updating d404ccb..291ed8d
Fast-forward
 LISEZMOI.txt | 1 +
 1 file changed, 1 insertion(+)


lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ cat LISEZMOI.txt
Mon Projet
Nous testons présentement les commandes Git
L'exemple a une branche de plus!
```

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git log
commit 291ed8dcaa05bcc74b421bf04be6157081c9442d (HEAD -> main, dev)
Author: Mohamed Lokbani <lokbani@iro.umontreal.ca>
Date:   Fri Mar 4 22:56:31 2022 -0500

    Tester une branche

commit d404ccb13ae41709fc7a7ab783899ec8b24ddcee
Author: Mohamed Lokbani <lokbani@iro.umontreal.ca>
Date:   Fri Mar 4 17:46:20 2022 -0500

    Tester un 2e commit

commit 4d35698e32574f05a96613244a958d1d16b20c51
Author: Mohamed Lokbani <lokbani@iro.umontreal.ca>
Date:   Fri Mar 4 11:58:48 2022 -0500

    Projet initial
```


HEAD
main
4d35698
d404ccb
291ed8d
dev

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git branch -l
  dev
* main
```

Pour effacer une branche :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git branch -d dev
Deleted branch dev (was 291ed8d).

lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git log --oneline
291ed8d (HEAD -> main) Tester une branche
d404ccb Tester un 2e commit
4d35698 Projet initial

lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git branch -l
* main
```

Ignorer des fichiers

Parfois nous voulons exclure des fichiers du système de versions.

Nous allons mentionner ces fichiers ou répertoires à exclure dans le fichier « .gitignore » dans le répertoire du projet.

Commençons par créer le fichier « .gitignore ». Utiliser l'éditeur qui vous convient le mieux. Attention quand même, Windows a tendance à ajouter par défaut l'extension « .txt ».

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ touch .gitignore
```

On valide la présence du fichier dans le répertoire :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ ls -la
total 9
drwxr-xr-x 1 support007 1049089  0 Mar  5 11:50 ./
drwxr-xr-x 1 support007 1049089  0 Mar  4 10:25 ../
drwxr-xr-x 1 support007 1049089  0 Mar  4 23:17 .git/
-rw-r--r-- 1 support007 1049089  0 Mar  5 11:50 .gitignore
-rw-r--r-- 1 support007 1049089 92 Mar  4 23:10 LISEZMOI.txt
```

On crée un fichier quelconque afin de tester l'exclusion. Dans l'exemple, nous avons créé le fichier « secret.txt » avec le contenu ci-dessous :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ echo "Nous allons ignorer ce fichier" >> secret.txt
```

On valide la présence du fichier dans le répertoire :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ ls -la
total 15
drwxr-xr-x 1 support007 1049089  0 Mar  5 11:53 ./
drwxr-xr-x 1 support007 1049089  0 Mar  4 10:25 ../
drwxr-xr-x 1 support007 1049089  0 Mar  5 11:54 .git/
-rw-r--r-- 1 support007 1049089 11 Mar  5 11:53 .gitignore
-rw-r--r-- 1 support007 1049089 92 Mar  4 23:10 LISEZMOI.txt
-rw-r--r-- 1 support007 1049089 31 Mar  5 11:50 secret.txt
```

On vérifie le statut du dépôt :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        secret.txt

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

Git nous informe de la présence de deux fichiers non tracés par le système de gestion de versions.

Éditer le fichier « .gitignore » avec votre éditeur préféré et ajouter la ligne :

```
secret.txt
```

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ cat .gitignore
secret.txt
```

On vérifie de nouveau le statut du dépôt :

```
lokbani@dirot32 MINGW64 ~/Desktop/Test (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

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

On constate que Git ne mentionne plus le fichier « secret.txt ». En effet, nous venons de l'exclure à l'aide du fichier « .gitignore ».

Ce fichier d'exclusion nous sera très utile quand nous allons utiliser le système de gestion de versions sur un projet Android.

Conclusion

Nous avons présenté dans ce document quelques fonctionnalités de Git, en ligne de commandes.

Le but du document est de vous introduire dans ce monde de gestion de versions.

Il est difficile de couvrir toutes les facettes de Git vu l'utilisation qui sera faite.

Nous vous suggérons au départ de faire une utilisation simple de l'outil.

Garder une arborescence simple pour commencer.

Faire régulièrement des « commit ».

Par la suite, au fur et à mesure que l'habitude s'installe, explorer les autres facettes de l'outil, comme :

- l'utilisation des branches

- la résolution de conflits dans le cas d'une fusion

- la mise dans une zone tampon des modifications en cours (la zone « stash »)

Et bien d'autres sujets …

Bibliographie

Livre Pro Git
https://git-scm.com/book/en/v2

Version française du livre Pro Git :
https://git-scm.com/book/fr/v2

Dépôt du livre Pro Git :
https://github.com/progit/progit2

Plusieurs exemples dans le cadre d'un travail collaboratif
https://cupofcode.blog/intro-to-git/