

# Mémo Git

Thomas Clavier

## Introduction

L'ensemble de ce cours est disponible sur <http://github.com/tclavier/tp-git-blog/> vous pouvez participer à son amélioration en proposant des «pull request».

## Présentation

Git est un logiciel de gestion de versions décentralisé.

Ce mémo est un extrait du livre : <https://git-scm.com/book/fr>

## Des objets

Un dépôt Git peut être vu comme une collection d'objets liés entre eux. Chaque objet est identifié par une chaîne de 40 caractères hexadécimaux correspondant à la somme de contrôle de son contenu. Il y a 3 types d'objets :

- blob : les données
- tree : les arborescences
- commit : une version du répertoire de travail

Un objet de type «arborescence» peut contenir des objets «données» et «arborescence». Le répertoire de travail est lui-même un objet de type «arborescences».

L'historique d'un dépôt Git c'est l'ensemble des versions du répertoire de travail. Pour identifier une version, Git s'appuie sur un objet de type «commit». L'objet de type «commit» associe de nombreuses informations comme l'auteur, un message, une version de l'objet répertoire de travail mais aussi les «commits» parents.

## Les espaces

Git utilise 3 espaces différents pour manipuler toutes ces données.

- le répertoire de travail
- l'index
- l'historique

Le répertoire de travail présente dans un dossier de la machine, l'ensemble des fichiers du dépôt. C'est le point d'entrée de l'utilisateur.

L'index contient les données en préparation pour le commit

La tête (HEAD) de l'historique est un pointeur vers le commit qui sera utilisé comme prochain commit parent.

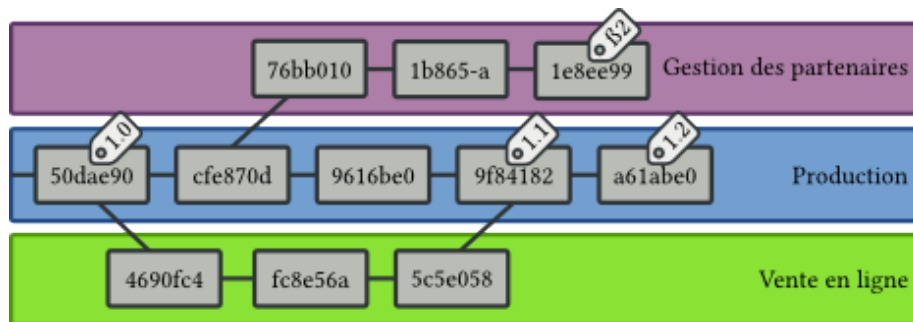


Figure 1: Historiques de commits

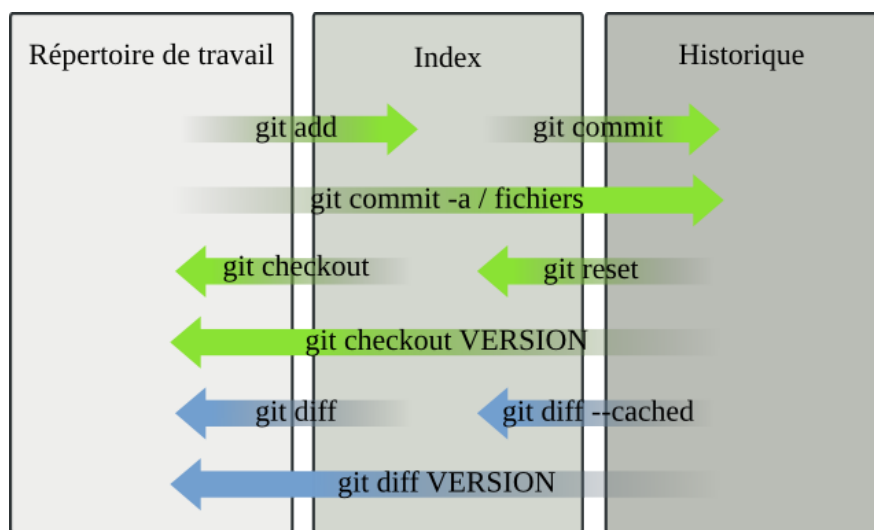


Figure 2: 3 espaces de données

## À distance

Avec les commandes push, fetch et pull, il est possible d'envoyer ou de recevoir la totalité de l'historique d'une branche vers un serveur distant. Ce qui permet de travailler à plusieurs sur le même projet.

## Resources

Pour avoir de l'aide sur git, il est possible de lancer "man git". Pour avoir de l'aide sur la commande "git log" il est possible de lancer "man git-log".

## Principales commandes

Toutes les commandes git sont de la forme suivante :

```
git action [options] argument
```

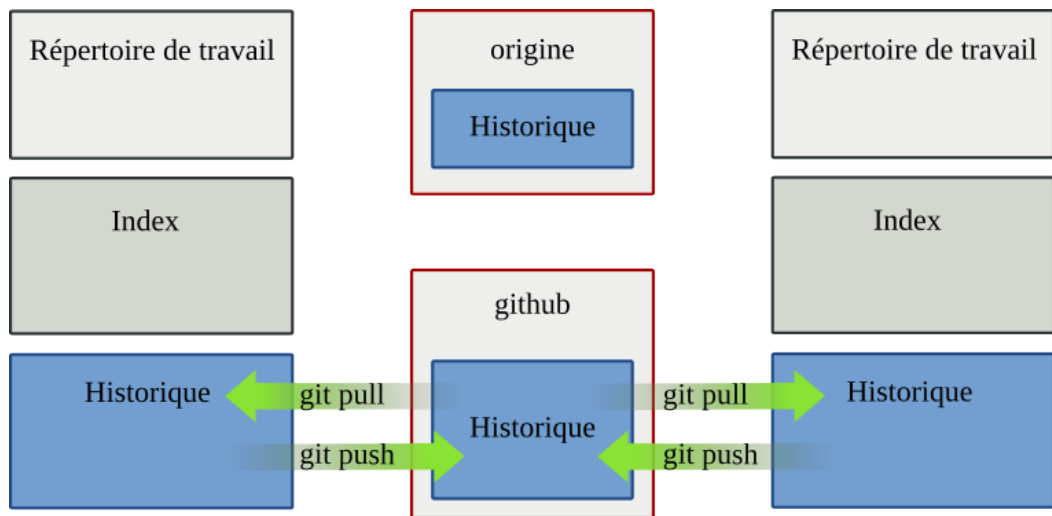


Figure 3: Serveurs distants

## add

Ajouter un fichier à suivre dans l'index. Pour préparer un commit.

```
git add chemin/vers/fichier
```

## commit

Valider l'index actuel pour créer un commit

```
git commit -m "Mon message de commit"
```

## config

Permet de configurer les variables de paramétrage de git

L'option *global* permet de configurer les variables de l'utilisateur, donc valable pour l'ensemble des projets de l'utilisateur.

L'option *system* permet de configurer les variables de tous les utilisateurs.

```
git config --global user.name "Thomas Clavier"
```

## init

Initialiser un répertoire de travail comme un dépôt local git. Permet d'initialiser l'ensemble des 3 espaces.

Exemple pour le répertoire courant :

```
git init .
```

## log

Observer l'historique des changements

```
git log
```

## **diff**

Observer la différence entre 2 commits, permet par exemple de voir quelles sont les différences entre 2 commits sur un fichier donné.

```
git diff
```

## **status**

Observer l'état de l'index et du répertoire de travail

```
git status
```

## **rm**

Supprimer un fichier de l'index

```
git rm chemin/vers/fichier
```

## **mv**

Déplacer un fichier. C'est équivalent à copier le fichier puis supprimer le fichier d'origine.

```
git mv chemin/vers/fichier chemin/vers/nouveau/fichier
```

## **remote**

Permet de manipuler les adresses des index distants.

```
git remote add origin https://github.com/tclavier/tp-git-blog.git
```

## **push**

Permet d'envoyer une copie de l'index des changements vers un serveur distant

```
git push origin master
```

## **pull**

Permet de recevoir une copie de l'index d'un serveur distant

```
git pull origin master
```

## **clone**

Prépare une copie de travail depuis un serveur distant

```
git clone URL
```

## **clean**

Supprime les fichiers non suivis par git

```
git clean -f
```

## **tag**

Permet de poser une étiquette sur un commit donné.

```
git tag 1.0.0
```

## **merge**

Fusionne 2 branches.

```
git merge ma-branche
```

## **rebase**

Concatène 2 branches.

```
git rebase ma-branche
```

## **branch**

Crée un branche de travail

```
git branch ma-branch
```

## **checkout**

Change la version de travail par une branche ou les fichiers du dernier commit

```
git checkout ma-branch
```

## **cherry-pick**

Permet de prendre un commit quelconque et de l'appliquer localement.

Exemple :

```
git cherry-pick ma-branche
```

Permet d'appliquer le dernier commit de la branche «ma-branche» dans la branche courante.

# **1 Fichiers particulier**

## **.gitignore**

Liste des fichiers ou répertoires que git doit ignorer

## **revert**

Crée un patch de retour en arrière afin d'annuler certains commits.

```
git revert HEAD~3
```

Va créer un patch pour annuler les 4 derniers commits.