

Git : mode d'emploi pour un usage courant

Florian Legendre

Université de Poitiers

Année 2020 - 2021

Table of Contents

- 1 L'interface CLI de Git
- 2 Préparer son espace de travail
- 3 La boucle de travail local
- 4 Synchroniser son travail
- 5 Gérer les branches
- 6 Utiliser la remise
- 7 Les ressources de Git

Table of Contents

- 1 L'interface CLI de Git
 - Pourquoi préférer un terminal de commandes à une UI ?

CLI vs UI

Interface Commande	Interface Graphique
<ul style="list-style-type: none">+ Les commandes sont plus explicites+ Messages d'erreur très explicites+ De nombreuses ressources en ligne pour s'aider+ Rapide à lancer, pas de bug d'interface- Il faut se souvenir des commandes	<ul style="list-style-type: none">+ Moins besoin de mémoriser des commandes+ Meilleure visualisation des branches+ Meilleure comparaison de versions sur les images- Lent et parfois des bugs- Moins de ressources en ligne- L'apprentissage est tout aussi long

Des commandes dans les messages d'erreurs

```
crex@crex:~/projects/test$ git push
fatal: No configured push destination.
Either specify the URL from the
command-line or configure a remote
repository using

    git remote add <name> <url>

and then push using the remote name

    git push <name>
```

Listing 1 – Exemple de message d'erreur

Table of Contents

- 2 Préparer son espace de travail
 - Configurer Git
 - Initialiser le suivi des modifications ou cloner un projet

Commande : `git config --global user.name/user.mail`

On ne peut pas faire de commit sans nom d'utilisateur et de mail !
C'est une question de responsabilité.

Syntaxe : `git config --global user.name "monNomUtilisateur"`

Syntaxe : `git config --global user.mail "monMail@boiteMail.com"`

Les alias de commandes

Quelques alias recommandés :

```
git config --global alias.co checkout
```

```
git config --global alias.br branch
```

```
git config --global alias.ci commit
```

```
git config --global alias.st status
```

Lister ses alias existants :

```
git config --get-regexp alias
```


Commande : git init

La commande "git init" crée le dossier caché .git/ contenant tous les fichiers dont Git a besoin pour offrir ses services :

```
crex@crex:~/projects/test$ git init
Initialized empty Git repository in /home/crex/projects/test/.git/

crex@crex:~/projects/test/.git$ ls
branches  config  description  HEAD  hooks  info  objects  refs
```

Listing 2 – Contenu du dossier .git/

Commande : git clone <url>

```
crex@crex:~/projects$ git clone https://github.com/torvalds/linux.  
git  
Cloning into 'linux'...  
remote: Enumerating objects: 7886755, done.  
remote: Total 7886755 (delta 0), reused 0 (delta 0), pack-reused  
7886755  
Receiving objects: 100% (7886755/7886755), 2.98 GiB | 1.12 MiB/s,  
done.  
Resolving deltas: 100% (6548657/6548657), done.  
Updating files: 100% (71277/71277), done.
```

Listing 3 – Exemple de git clone

Commande : git log

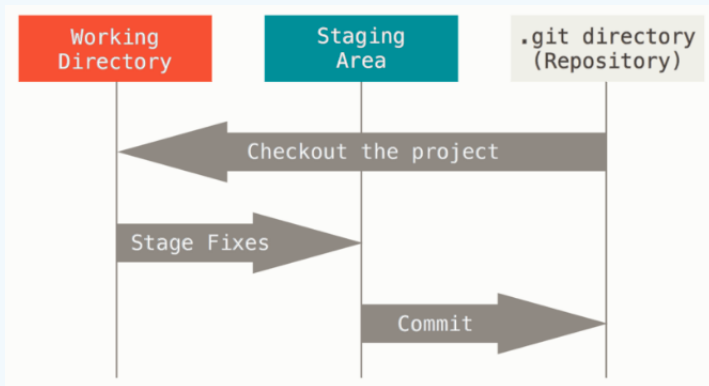
```
crex@crex:~/projects/linux$ git log
commit 291009f656e8eaebbfd3a8d99f6b190a9ce9deb (HEAD -> master,
    origin/master, origin/HEAD)
Merge: a3961497bd9c d11a1d08a082
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date:   Wed Feb 10 12:03:35 2021 -0800
[...]
```

Listing 4 – On peut directement travailler

Table of Contents

- 3 La boucle de travail local
 - Gérer le suivi des modifications
 - Enregistrer une version
 - Optionnel : Consulter ses versions

Les trois (+1) états du suivi des modifications



Commande : git status

La commande "git status" permet de savoir dans quel état se situent tous les fichiers du projet depuis la racine du projet. On remarque plusieurs choses :

- Le suivi des modifications concerne par défaut tous les fichiers du projet en partant du dossier où se trouve le dossier .git/
- Par défaut aucun fichier n'est suivi, il faut préciser à Git les fichiers dont on souhaite suivre les modifications
- Si un fichier n'a pas été modifié et qu'il est suivi, il n'apparaît pas dans l'affichage de "git status"
- Un fichier non suivi continuera d'apparaître à chaque affichage de "git status"

Commande : git status

```
crex@crex:~/projects/test$ git st
On branch master

No commits yet

Untracked files: (use "git add <file>..." to include in what will
    be committed)
    test.txt
    test2.txt

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

Listing 5 – Tout premier appel à git status

Commande : git status

```
crex@crex:~/projects/test$ git st
On branch master
No commits yet

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

Untracked files: (use "git add <file>..." to include in what will
    be committed)
    test2.txt
```

Listing 6 – git status après ajout au suivi de test.txt

Commande : git status

```
crex@crex:~/projects/test$ git st
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test2.txt
nothing added to commit but untracked files present (use "git add"
to track)
```

Listing 7 – git status après commit

Commande : git status

```
crex@crex:~/projects/test$ git st
On branch master
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:   test.txt

Untracked files: (use "git add <file>..." to include in what will
    be committed)
    test2.txt
```

Listing 8 – git status modification de test.txt

Commande : .gitignore

Le fichier .gitignore permet d'ignorer des fichiers du suivi des modifications. Les "wildcards" y sont autorisées :

```
crex@crex:~/projects/avlTreesL3Project$ cat .gitignore
*##
*~

# Pour l'exemple :
*.cmi
!toto.cmi
```

Listing 9 – Contenu du dossier .git/

Ajouter/Retirer du staging

Pour ajouter au suivi des modifications :

```
git add <nom_fichier>
```

```
git add -A
```

Pour retirer un fichier du suivi des modifications :

```
git rm --cached <nom_fichier>
```

Commande : git commit

La commande git commit permet de sauvegarder l'ensemble des fichiers suivis en une version nommée automatiquement et ajoutée à l'historique des versions :

- On doit toujours indiquer un message lors d'un commit (cf. **git commit -am "mon message"**)
- Ce message doit expliquer ce qui a été fait
- Ce message sera visible de tous ceux qui auront votre projet (y compris sur le dépôt distant)
- Un "commit" est donc un engagement, vous engagez votre responsabilité

Commande : git log

La commande "git log" permet d'afficher l'historique des versions. Ci-dessous un extrait d'un historique de version.

```
commit a3961497bd9c7ca94212922a46729a9410568eb8
Merge: 708c2e418142 fe0af09074bf
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date:   Wed Feb 10 11:58:21 2021 -0800
    Merge tag 'acpi-5.11-rc8' of git://git.kernel.org/pub/scm/linux
    /kernel/git/rafael/linux-pm
    Pull ACPI fix from Rafael Wysocki:
    "Revert a problematic ACPIICA commit that changed the code to
    attempt to update memory regions which may be read-only on some
    systems (Ard Biesheuvel)" [...]
```

Listing 10 – Exemple de bon message de commit

Commande : `git reset --hard <numero_commit>`

La commande "`git reset --hard`" permet d'annuler tous les commits qui ont été faits après le commit spécifié et de faire revenir ses fichiers à l'état du commit spécifié.

Il n'est pas utile d'indiquer l'entièreté du numéro du commit, seuls les 6 premiers caractères suffisent.

ATTENTION ! C'est une commande très dangereuse... Ce qui est perdu est perdu définitivement ! À n'utiliser qu'en cas de dernier recours.

Commande : git diff

Il y a plusieurs utilisations possibles de la commande "git diff" :

- git diff + aucun paramètre : montre la différence entre la version dans le "staging area" et le dernier commit
- git diff <nom_commit1> <nom_commit2> : montre la différence entre les deux versions spécifiées

Table of Contents

4 Synchroniser son travail

- Récupérer les modifications d'un serveur distant
- Ajouter ses mises à jour sur un serveur distant

Commandes : git fetch / git pull

git fetch	git pull
<ul style="list-style-type: none">❶ Récupère seulement la dernière mise à jour de l'historique❷ Il faut ensuite faire un git merge pour incorporer les changements❸ Utile quand on veut anticiper des conflits ou qu'on n'a pas tout à fait fini son travail	<ul style="list-style-type: none">❶ Récupère seulement la dernière mise à jour de l'historique et incorpore les changements simultanément❷ Elle est plus utilisée que git fetch

Commande : git push

La commande "git push" permet de mettre à jour la branche distante en incorporant dans la branche distante l'historique et les fichiers locaux.

ATTENTION :

- ❶ Il faut avoir fait un git pull avant de l'utiliser
- ❷ Il faut avoir les droits

Gérer les droits de mise à jour

Il y a trois scénarios possibles, qui dépendent de la configuration du ou des dépôts distants.

Premier scénario :

Vous êtes indiqué comme **collaborateur** sur la branche distante. En tant que collaborateur pour "pusher" les changements que vous voulez sans restriction !

ATTENTION pour les chefs de projet : Cela nécessite d'avoir une très grande confiance en ses collaborateurs. En cas de doute une des solutions alors d'avoir une branche distante de développement

Gérer les droits de mise à jour

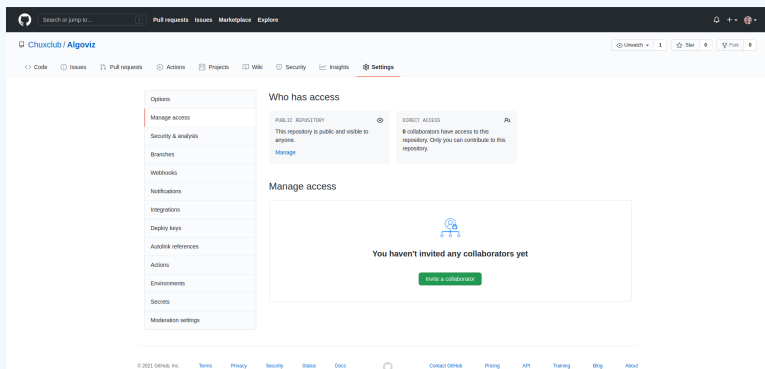


Figure – Définir des collaborateurs sous GitHub

Gérer les droits de mise à jour

Second scénario :

- 1 Vous avez "**forké**" (càd copié) sous Github le projet d'une personne.
- 2 Vous avez ensuite cloné ce fork.
- 3 Vous avez travaillé sur ce clone et "pushé" vos projets sur votre fork distant.
- 4 Vous pouvez ensuite faire un pull request directement sur l'interface GitHub.

Gérer les droits de mise à jour

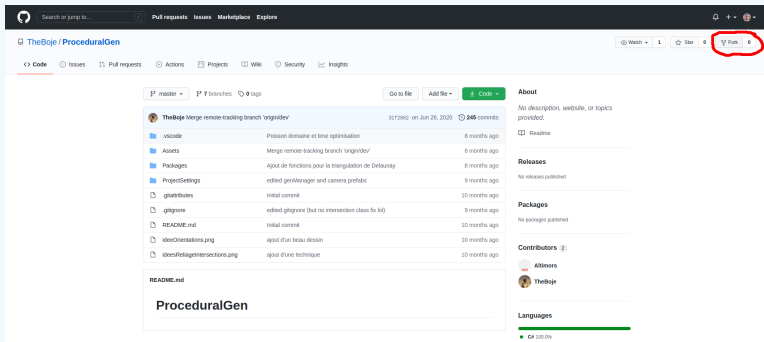


Figure – Forker sous GitHub

Gérer les droits de mise à jour

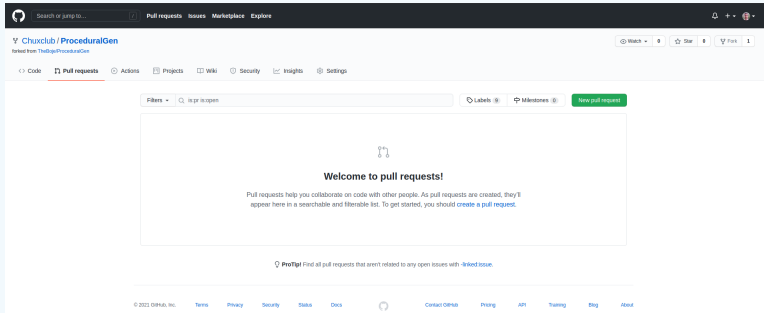


Figure – Faire un pull request sous GitHub

Gérer les droits de mise à jour

Dernier scénario :

Vous avez **cloné** le projet d'une personne. Vous avez ensuite travaillé dessus et vous vous rendez compte que vous souhaiteriez faire part de vos modifications à l'auteur...

- ❶ Comme précédemment vous forkez
- ❷ Vous ajoutez une branche distante à votre clone : "git remote add <nom> <urlDeVotreDepot>" ou, plus brutal : "git remote rm origin" suivi de "git remote add origin <urlDeVotreDepot>"
- ❸ Vous poussez vos changements sur ce fork
- ❹ Vous faites un pull request comme précédemment

Table of Contents

- 5 Gérer les branches
 - Sur quelle branche suis-je ?
 - Créer/Supprimer une branche
 - Changer de branche
 - Fusionner des branches
 - Suivre une branche distante

Commandes : git branch

Il est important de connaître la branche sur laquelle on travaille. Pour cela, il suffit d'entrer dans le terminal "git branch" sans argument :

```
crex@crex:~/projects/avlTreesL3Project$ git branch
dev
* main
```

Listing 11 – Exemple d'affichage de la commande "git branch"

Commandes : git branch [-d] <nom_branche>

Pour créer une branche on entre "git branch" suivi du nom de la branche qu'on souhaite donner. L'option -d permet au contraire de supprimer la branche désigné par le nom donné en paramètre :

```
crex@crex:~/projects/avlTreesL3Project$ git branch
dev
* main
crex@crex:~/projects/avlTreesL3Project$ git branch exp
crex@crex:~/projects/avlTreesL3Project$ git branch
dev
exp
* main
crex@crex:~/projects/avlTreesL3Project$ git branch -d exp
Deleted branch exp (was 9985cdb).
crex@crex:~/projects/avlTreesL3Project$ git branch
dev
* main
```

Listing 12 – Exemple de création et de suppression de branche

Commandes : git checkout <nom_branche>

Pour changer de branch on entre "git checkout" suivi du nom de la branche :

```
crex@crex:~/projects/avlTreesL3Project$ git branch
dev
* main
crex@crex:~/projects/avlTreesL3Project$ git checkout dev
Switched to branch 'dev'
crex@crex:~/projects/avlTreesL3Project$ git branch
* dev
main
```

Listing 13 – Exemple de création et de suppression de branche

Remarque : Les fichiers sont automatiquement changés pour correspondre à l'état de la version la plus récente de la branche sur laquelle on bascule.

Commandes : git checkout <nom_branche>

ATTENTION : Le changement de branche vous sera refusé si des modifications ont été détectées mais n'ont pas encore été "commitées". Deux solutions s'offrent alors à vous : 1) commiter vos changements puis changer de branche ou 2) utiliser la remise dont nous détaillerons le fonctionnement ultérieurement.

Commandes : git merge <nom_branche>

La commande "git merge" suivi du nom de la branche permet d'incorporer les changements d'une branche dans une autre branche.

ATTENTION : git merge <nom_branche> incorpore les changements de la branche désignée dans la branche courante.

Ce qui signifie, par exemple dans le cas où vous voulez incorporer les changements d'une branche de développement dans une branche "main" ou "master" que vous devez d'abord changer de branche pour aller sur la branche "main" ou "master" puis exécuter la commande.

Commandes : git remote add <name> <url> / git remote rm <name>

La première commande "git remote add" suivi d'un nom et d'une url permettent de suivre la branche distante désignée par l'url.

Le nom est un alias donné à l'url il permet notamment :

- 1 De spécifier où faire le push d'une branche (Exemple : git push <nom> <nomDeMaBrancheLocale>)
- 2 De spécifier quelle branche du dépôt distant on veut "pull" (Exemple : git pull <nom>)
- 3 De supprimer un suivi de branche distante avec "git remote rm <name>"
- 4 De spécifier une branche où on veut push/pull par défaut avec "git push -u <nom> <branch>"

Table of Contents

- 6 Utiliser la remise
 - Découverte et Gestion de la remise

Intérêt de la remise

La remise permet de revenir au dernier commit tout en sauvegardant votre travail en cours. Ceci est très utile dans certains cas, par exemple :

- Quand on veut changer de branche rapidement sans "commiter" des modifications en cours
- Quand on a commencé un travail mais qu'on s'est trompé de branche
- Et probablement d'autres cas auxquels je n'ai pas encore pensé...

Commandes : git stash [pop/list/drop]

La commande "git stash" fonctionne de la manière suivante :

- git stash + aucun argument : Revient au dernier commit en sauvegardant le travail en cours
- git stash list : Liste les différents WIP ("Work In Progress") sauvegardés dans la remise
- git stash apply <numéro> : Applique les modifications de <numéro> de la remise
- git stash pop : Applique les modifications du dernier stash de la remise puis le supprime
- git stash drop <numéro> : Retire le stash <numéro> de la remise

Commandes : git stash [pop/list/drop]

```
crex@crex:~/projects/test$ git st
On branch master
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:   test.txt

crex@crex:~/projects/test$ git stash
Saved working directory and index state WIP on master: 1a0b2eb Dé
but du suivi de test.txt

crex@crex:~/projects/test$ git stash list
stash@{0}: WIP on master: 1a0b2eb Début du suivi de test.txt

crex@crex:~/projects/test$ git stash apply 0
On branch master
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:   test.txt
```

Listing 14 – Exemple de fonctionnement du stash

Table of Contents

- 7 Les ressources de Git
 - Ressources pour apprendre Git ou recevoir de l'aide

Un lien à connaître : <https://git-scm.com/doc>

git --fast-version-control

Search entire site...

About

Documentation

Reference

Book

Videos


External Links

Downloads

Community


Documentation

Reference


 **Reference Manual**
The official and comprehensive **man pages** that are included in the Git package itself.


Quick reference guides: [GitHub Cheat Sheet](#) | [Visual Git Cheat Sheet](#)

Book

 **Pro Git**
The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to read [online for free](#). Dead tree versions are available on [Amazon.com](#).

Videos


Git Basics:
What is Version Control?
Length: 05:59


Git Basics:
What is Git?
Length: 08:15

git help <nomDeLaCommande>

```
crex@crex:~/projects/test$ git help add
GIT-ADD(1)                  Git Manual

NAME
    git-add - Add file contents to the index
```

Listing 15 – Extrait d'un appel de git help sur la commande "add"

man giteveryday/gittutorial

La commande "man giteveryday" vaut également le détour car elle est comme un petit "cheatsheet" intégré au terminal de commande. La commande "man gittutorial" recense de nombreux tutoriels sur de nombreuses notions du versionnement par Git. Je recommande vivement d'explorer les suggestions de la section "SEE ALSO" en fin de page de ce manuel !