Git : mode d'emploi pour un usage seul, sans dépôt distant, sur une seule branche

Florian Legendre

Université de Poitiers

Année 2020 - 2021

Table of Contents

Université de Poitiers 1431

Table of Contents

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

git help <nomDeLaCommande>

Les commandes Git ont souvent des options qui modifient leurs comportements. Si vous utilisez souvent une commande vous aurez certainement envie d'ouvrir son manuel grâce à *git help <nomCmd>* pour connaître ses options et syntaxes alternatives :

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

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

Listing 2 – Extrait d'un appel de git help sur la commande "add"

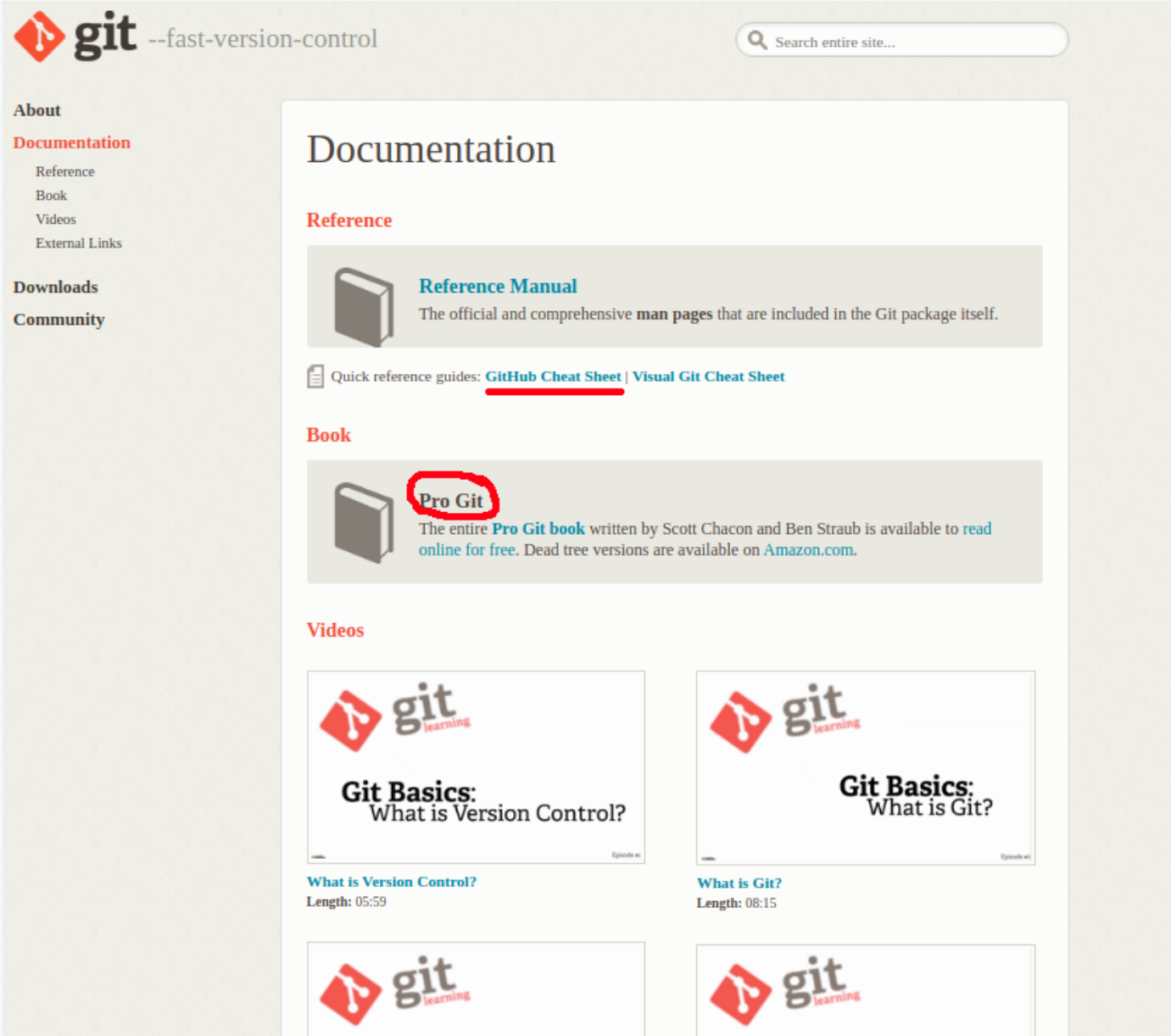
Note : Pour les utilisateurs de Linux, *git help <nomCmd>* est l'équivalent du *man <nomCmd>* mais pour Git !

git help [-a]

IMPORTANT : Le nom de la commande est optionnel avec *git help*. Si vous entrez simplement *git help* vous aurez un résumé des commandes les plus courantes.

Si vous entrez uniquement "**git help -a**" vous aurez toutes les commandes git qui existent. Celles que nous aborderont sont dans la premère catégorie 'Main Porcelain Command'

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



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 !

Table of Contents

Commande : git config −−global user.name/user.email

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.email "monMail@Mailbox.com"

Commande : git config ––global core.editor

Par défaut Git ouvrira l'éditeur de texte Vim ou Nano lorsque vous écrirez les messages de vos commits (sauf si vous utilisez l'option -m, cf. *git commit -m "..."*).

Pour changer d'éditeur vous pouvez suivre les instructions de cette adresse : `https://git-scm.com/book/fr/v2/Commandes-Git-Installation-et-configuration`

Exemple : git config –global core.editor "'C :\Program Files (x86)\Windows NT\Accessories\wordpad.exe'"

IMPORTANT : Remarquez dans cet exemple que les guillemets " entourent les simple quotes '

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 toutes ses clefs de configuration :

```
git config --list
```

Lister toutes les clefs de configuration possibles de Git :

```
git help --config
```

Commande : git init

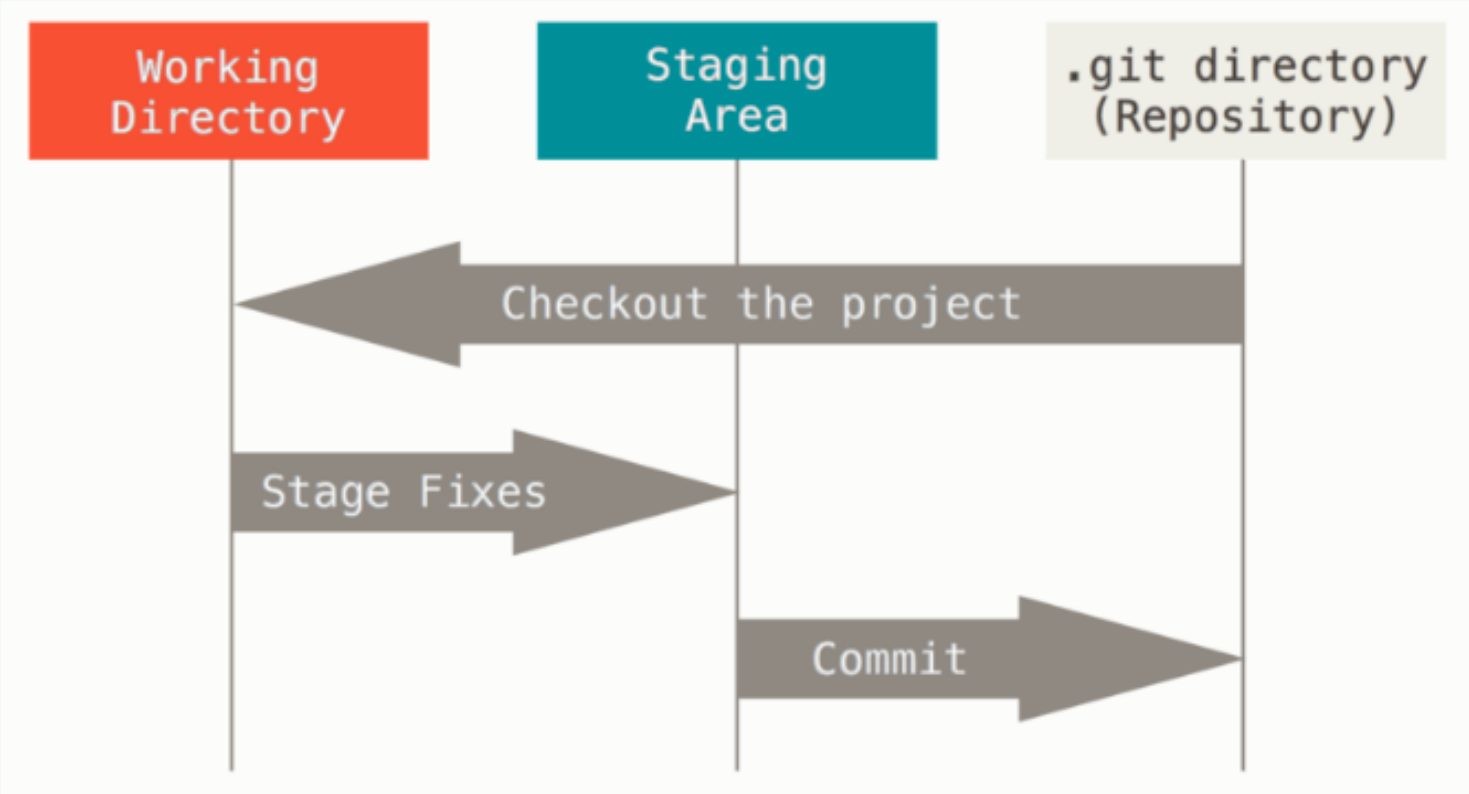
La commande "git init" crée le dossier caché .git/ contenant tous les fichiers dont Git a besoin pour offrir ses services :

```
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 3 – Contenu du dossier .git/

Table of Contents

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 4 – 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 5 – 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 6 – 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 7 – git status modification de test.txt

Le fichier .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 8 – Contenu du dossier .git/

Ajouter/Retirer du staging ou du suivi

Pour ajouter au staging :

```
git add <nom_fichier>
git add -A
```

Pour retirer du staging :

```
git reset -- <nom_fichier>
git reset
```

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
- 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 commit

La commande *git commit* admet plusieurs options et syntaxes :

- git commit => ouvre l'éditeur de texte indiqué dans la clé de configuration *core.editor* pour vous permettre d'écrire votre message et enregistre ce qui était dans le staging area
- git commit -m "..." => enregistre ce qui était dans le staging area avec le message passé en paramètre
- git commit -a => combinaison *git add -A* et *git commit*
- git commit -am "..." => combinaison *git add -A* et *git commit -m "..."*

Commande git commit −−amend

Du fait de l'importance de ces commits et des messages qui y sont associés, Git fournit un moyen de corriger le message du dernier commit (et uniquement le message du dernier commit) grâce à l'option −−*amend* :

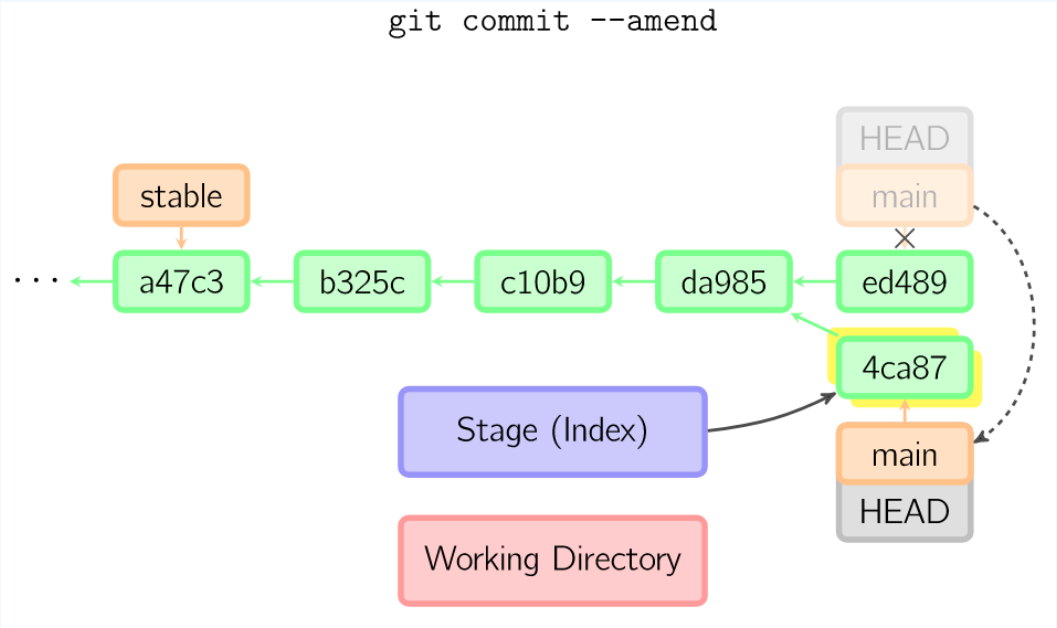


Table of Contents

Commande : git blame <nom_fichier>

La commande *git blame* suivi du nom d'un fichier permet d'afficher, ligne par ligne, qui a fait quoi (et quand) sur un fichier donné. Exemple :

```
e53bccf9 codes/..  (Florian 2021-04-11) public function construct()
f9f89eca codes/..  (Florian 2021-04-03) {
f9f89eca codes/..  (Florian 2021-04-03)  $em = $entityMan;
f9f89eca codes/..  (Florian 2021-04-03)  $user = getUser();
dc1bdd2a codes/..  (Florian 2021-04-08)  $globalUser->check($user);
f9f89eca codes/..  (Florian 2021-04-03) }
5e648da3 codes/..  (Florian 2021-03-31)
eb8986af codes/..  (frafraA 2021-04-03) /**
268e35ac codes/..  (frafraA 2021-04-03)  * @Route("/manageAccount")
eb8986af codes/..  (frafraA 2021-04-03)  */
268e35ac codes/..  (frafraA 2021-04-03)  public function f(): Resp
```

Listing 9 – Exemple de git blame

Commande : git log

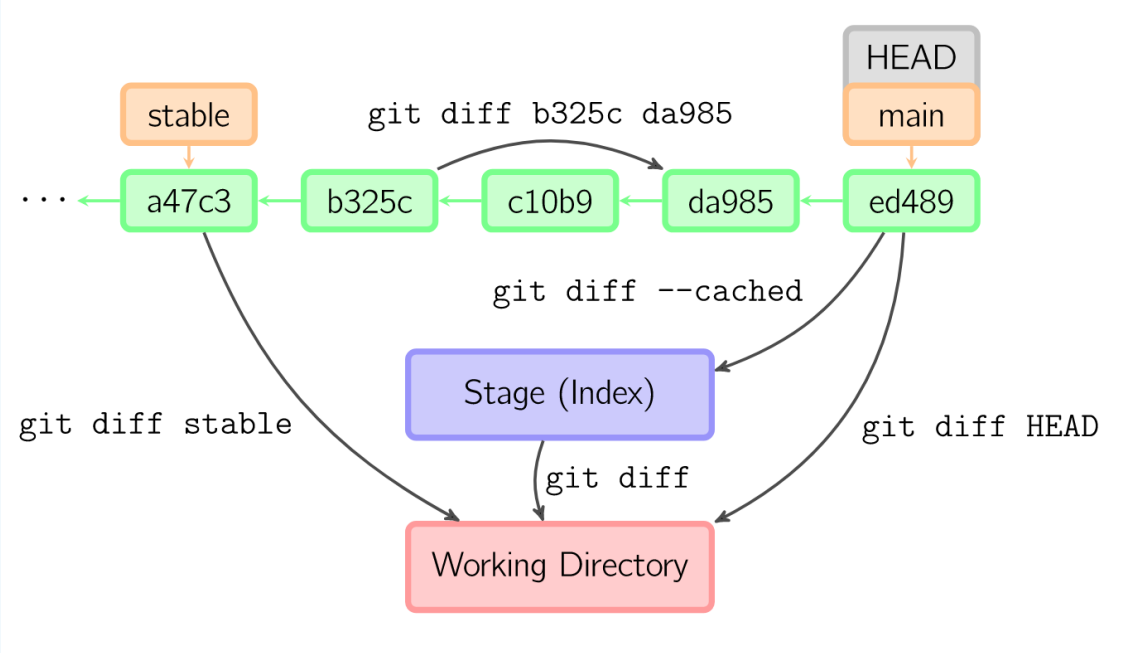
La commande *git log* permet d'afficher l'historique des versions.
Ci-dessous un extrait d'un historique de version.

```
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 ACPICA 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

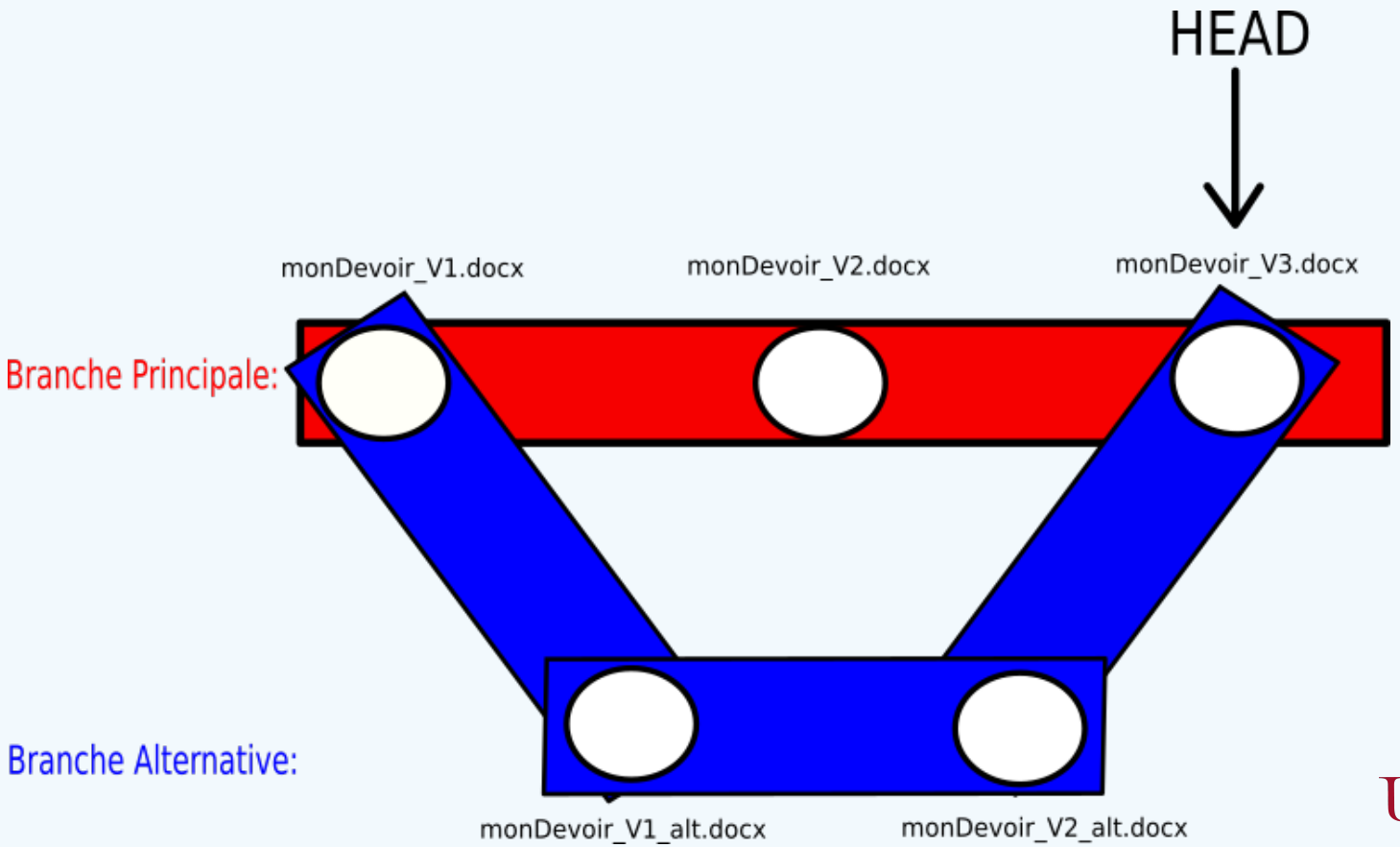
Commande : git diff

git diff montre ce qui a été ajouté/supprimé en partant du <nom_commit1> jusqu'au <nom_commit2>. *git diff* sans aucun paramètre montre la différence en partant de la version dans le "staging area" et les fichiers actuels du dossier, etc. :



Commande : git checkout <nom_commit>

La commande "git checkout <nom_commit>" permet de déplacer le pointeur "HEAD". Lorsque le pointeur "HEAD" est déplacé cela met à jour les fichiers pour correspondre la version pointée :

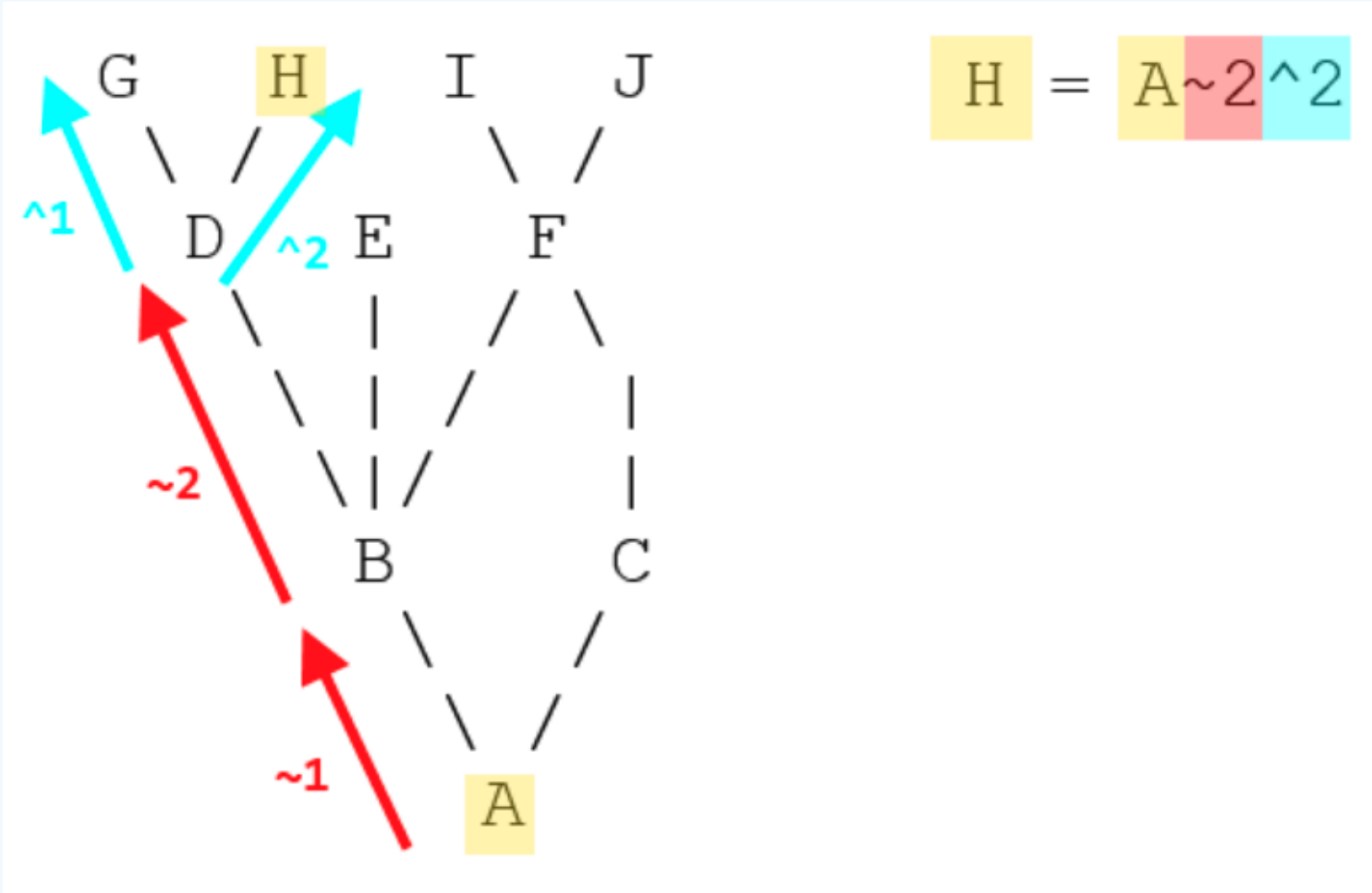


À propos des noms de commit

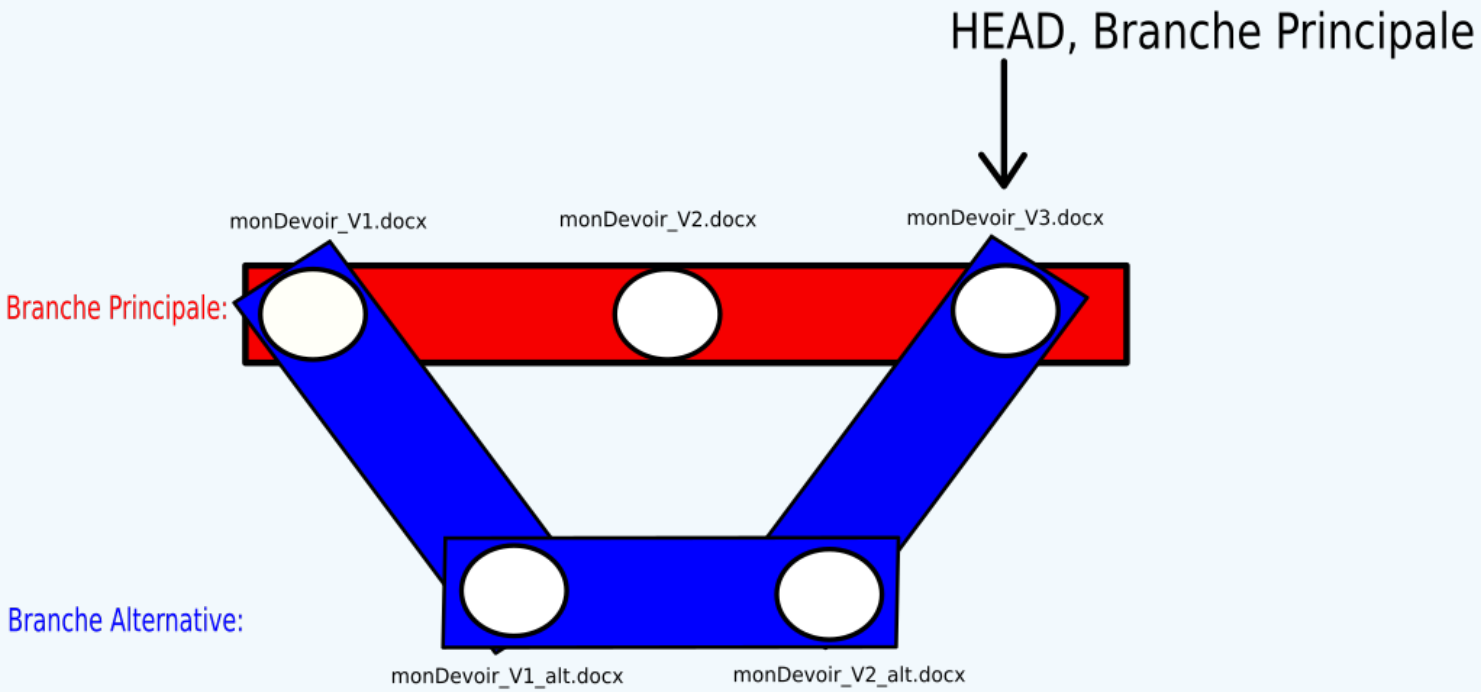
Il y a plusieurs façons d'obtenir un nom de commit :

- Le nom absolu : "git log" + vous prenez les 4 ou 5 premiers caractères (inutile de donner toute la clef !)
- Le nom relatif : Exemple HEAD~2^3 => il vaut pour un numéro de commit ! Seule différence : il est relatif

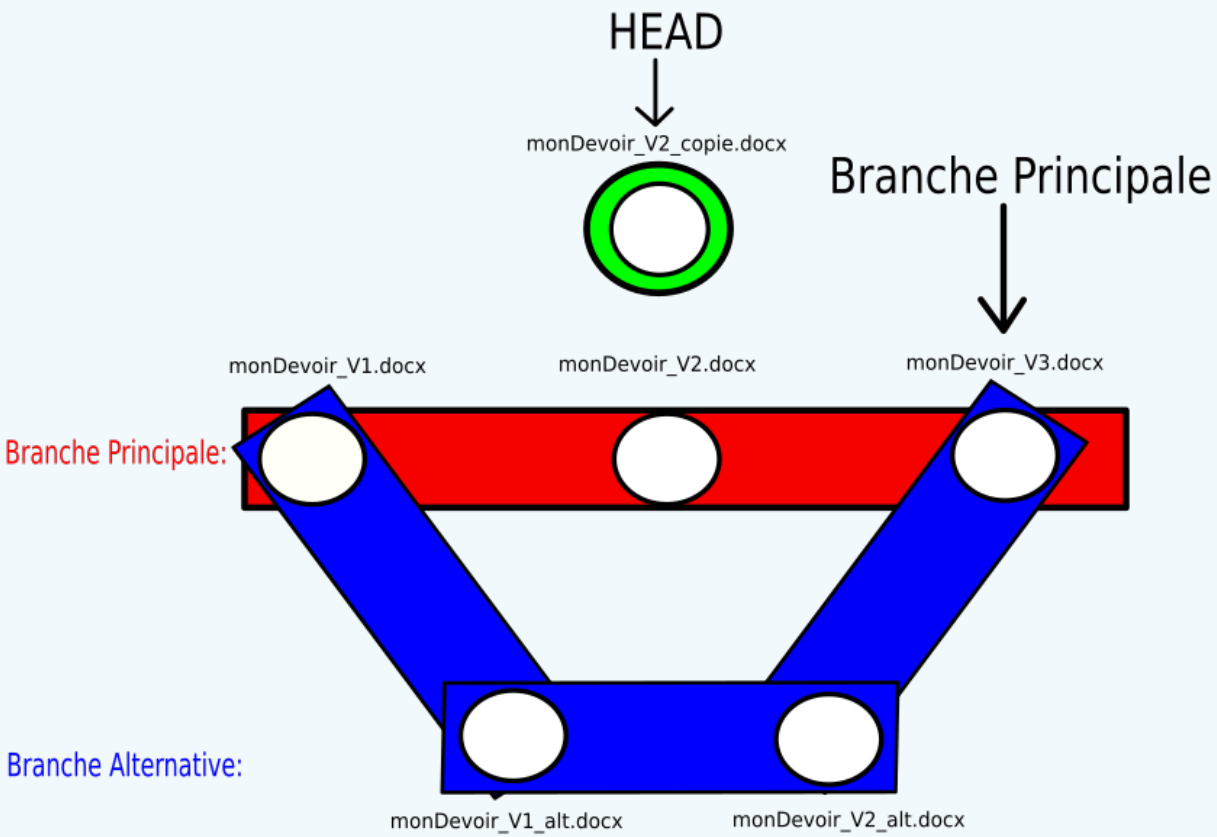
Illustration des noms de commits relatifs



Que signifie "la tête est détachée" ?



Que signifie "la tête est détachée" ?

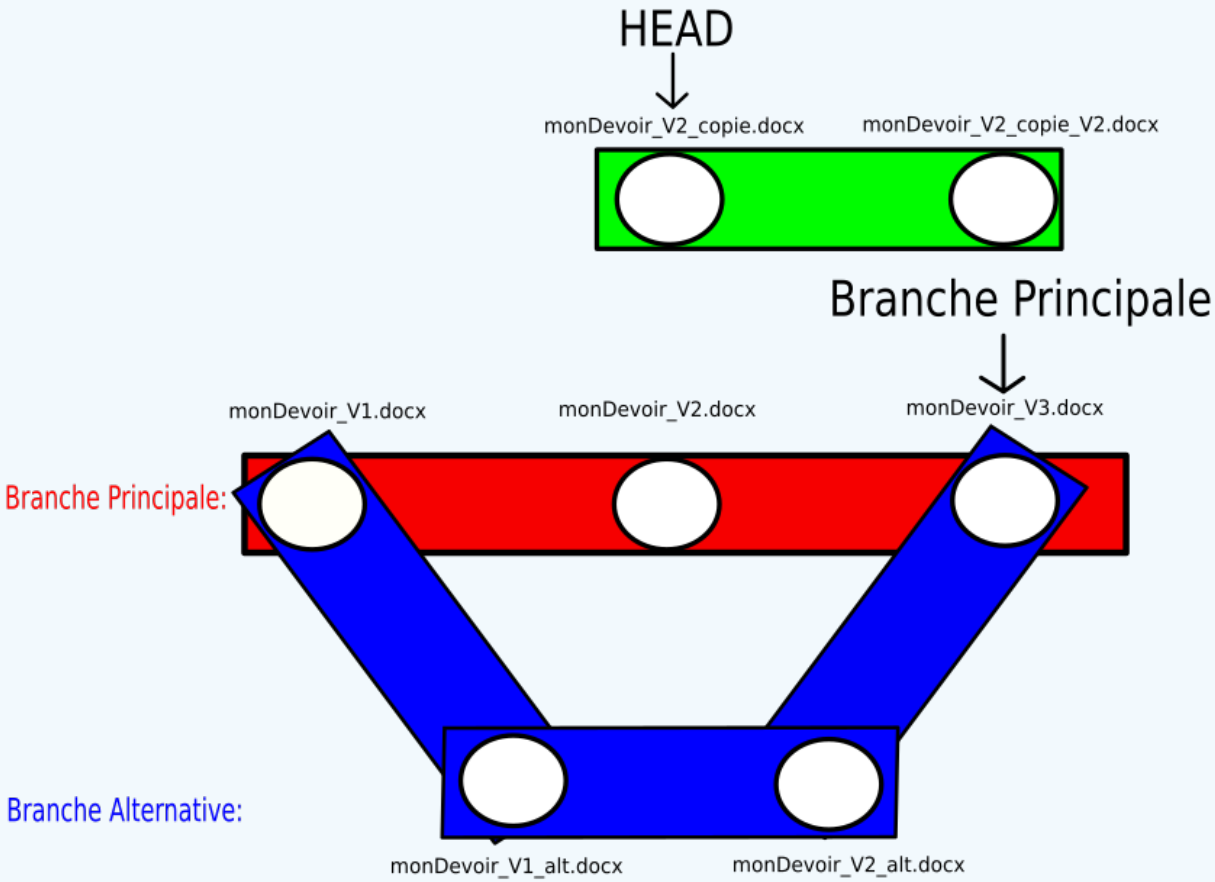


Qu'ai-je le droit de faire avec une tête détachée ?

```
crex@crex:~/projects/matlab(main)$ git checkout a1a8a9
A    test
Note: switching to 'a1a8a9'.

You are in 'detached HEAD' state. You can look around, make
experimental changes and commit them, and you can discard
any commits you make in this state without impacting any branches
by switching back to a branch.

If you want to create a new branch to retain commits you create,
you may do so (now or later) by using -c with the switch command.
Example:
  git switch -c <new-branch-name>
Or undo this operation with:
  git switch -

Turn off this advice by setting config variable advice.detachedHead
      to false
HEAD is now at a1a8a9d Initial commit
```

Listing 11 – Des droits spéciaux avec une tête détachée

Qu'ai-je le droit de faire avec une tête détachée ?



Qu'ai-je le droit de faire avec une tête détachée ?

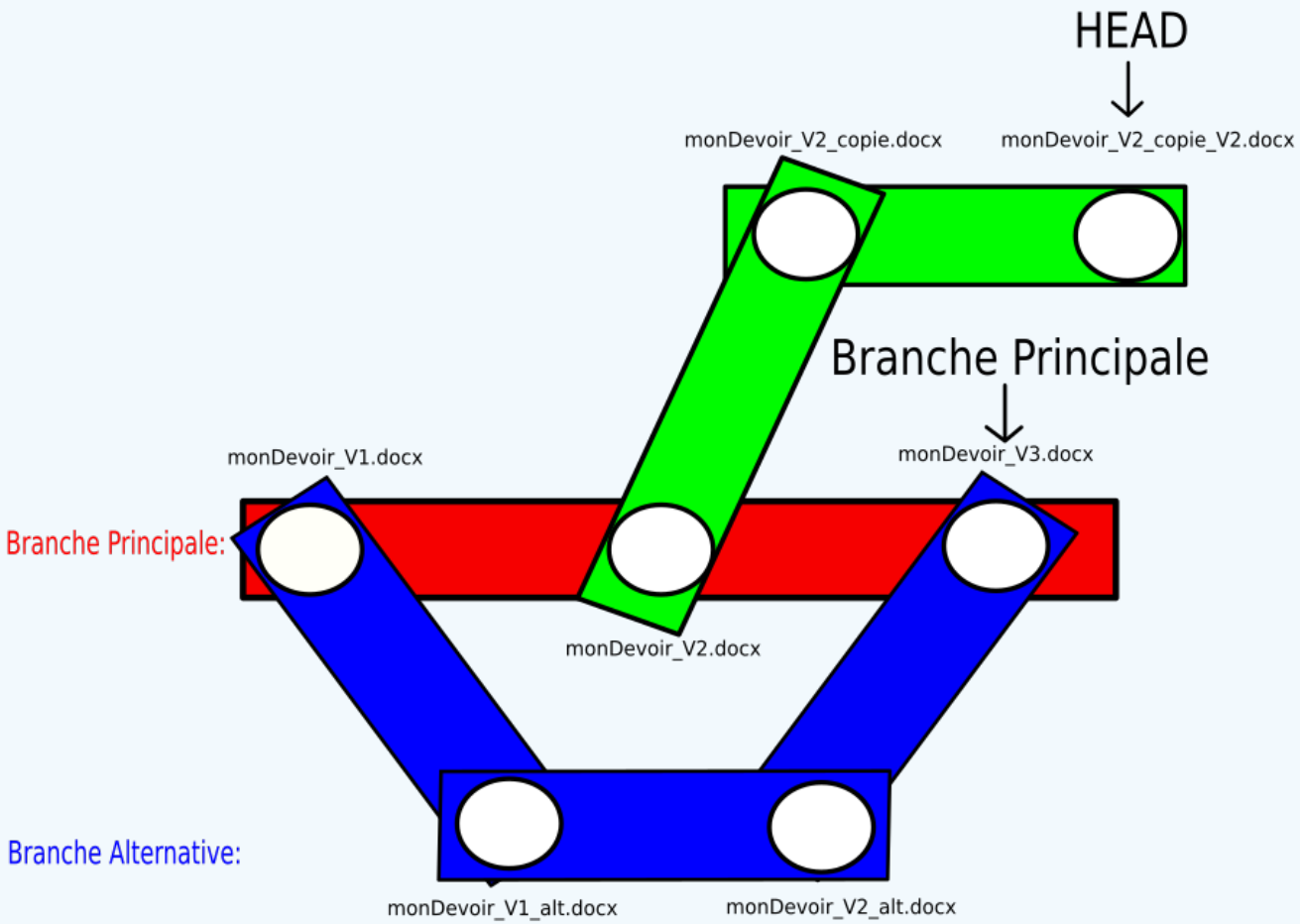


Table of Contents

Commande : git revert -n <older_id>^..<newer_id>

Cette commande crée un commit qui contient l'annulation de tous les changements des commits spécifiés dans l'intervalle *<older_id>^..<newer_id>*

C'est-à-dire l'intervalle : [<older_id> ; <newer_id>]

IMPORTANT : L'option "-n" évite la création automatique d'un commit. Cela présente plusieurs avantages :

1. Ça laisse le temps de réfléchir à son annulation
2. Ça évite une "cascade" de commits d'annulations
3. Si vous voulez "annuler l'annulation" vous pouvez le faire très rapidement avec la commande *git revert HEAD*

Annuler un changement apporté par une version donnée

Cette question ne sera pas abordée ici car elle demande une assez bonne connaissance de Git et notamment de la résolution des conflits. Elle est abordée dans le module "bonus" associé à ce module.

Table of Contents

git restore <file>

Permet de restaurer un seul fichier à l'état de son dernier commit.

Cas concret : Vous travaillez sur votre code, vous sauvegardez et la fin de la journée arrive. Vous ne voulez pas "commiter" vos changements car vous n'avez pas encore eu le temps de tester vos ajouts...

Le lendemain matin vous reprenez votre code et il est truffé de bug. Vous voulez "revert" uniquement ce fichier (les autres ont l'air bon) => *git restore <nom_fichier>*

Commande : git reset ––hard (sans paramètre)

La commande *git reset ––hard* sans paramètre permet d'annuler tous les changements en cours dans votre dossier de travail.
L'historique des modifications n'est pas touché !

C'est une façon très pratique de repartir de 0 quand on n'est pas satisfait de ce qu'on a écrit...

Commande : git reset −−hard <numero_commit>

La commande *git reset −−hard <numero_commit>* 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 4 (ou 5) 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.