Un gestionnaire de versions distribué : Git

Pourquoi un gestionnaire de version ?

- Travailler à plusieurs sur le même code
- Garder un historique de tout ce qui a été fait
- Revenir rapidement à une ancienne version
- Faciliter la sauvegarde du code source

On faisait comment avant Git ?

- Système D: copies du code source, FTP, clé USB, email, ...

Chapitre Ier

Dispositions relatives au mariage

I. – Le chapitre Ier du titre V du livre Ier du code civil est ainsi modifié :

1° Il est rétabli un article 143 ainsi rédigé :

« *Art. 143.* – Le mariage est contracté par deux personnes de sexe différent ou de même sexe. » ;

2° L'article 144 est ainsi rédigé :

« *Art. 144.* – Le mariage ne peut être contracté avant dix-huit ans révolus. » ;

3° L'article 162 est complété par les mots : « , entre frères et entre sœurs » ;

4° L'article 163 est ainsi rédigé :

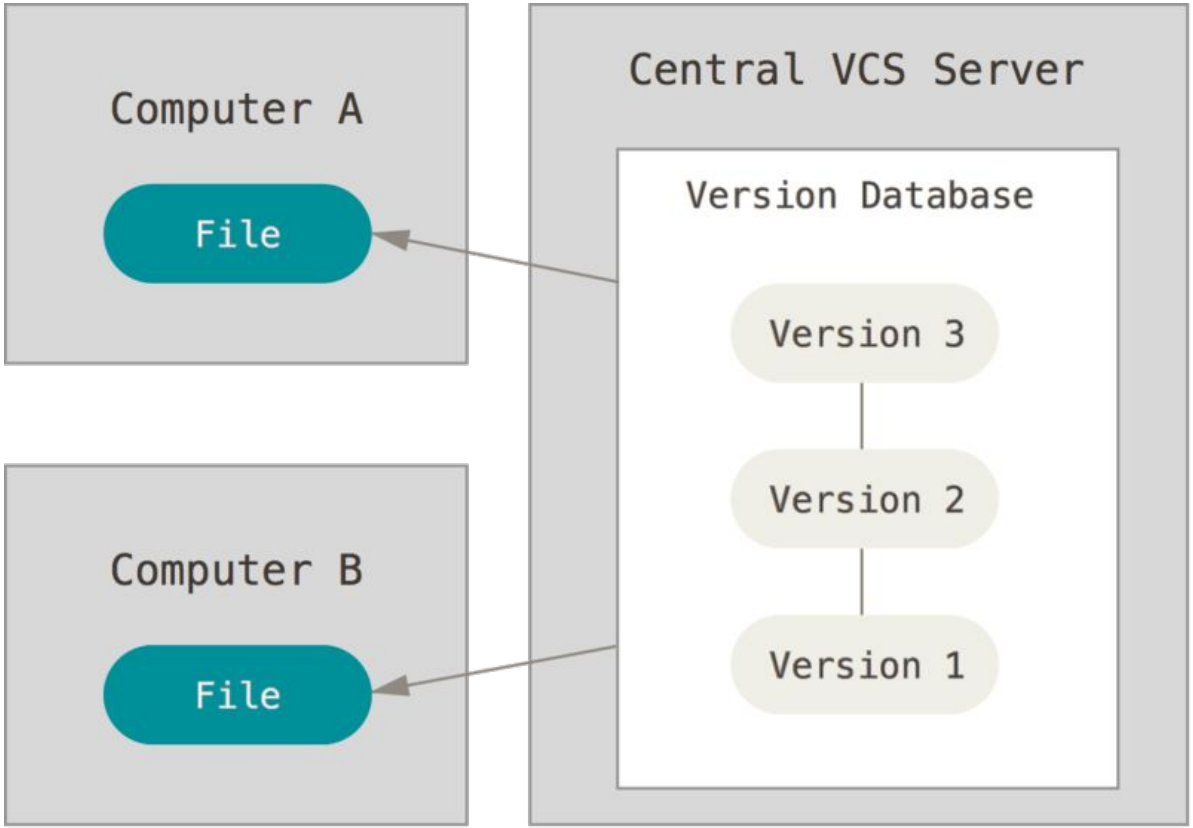
« *Art. 163.* – Le mariage est prohibé entre l'oncle et la nièce ou le neveu, et entre la tante et le neveu ou la nièce. » ;

Extrait du projet de loi sur le mariage pour tous.
Source: http://www.assemblee-nationale.fr/14/ta/ta0120.asp

On faisait comment avant Git ?

- Système D: FTP, clé USB, email, …

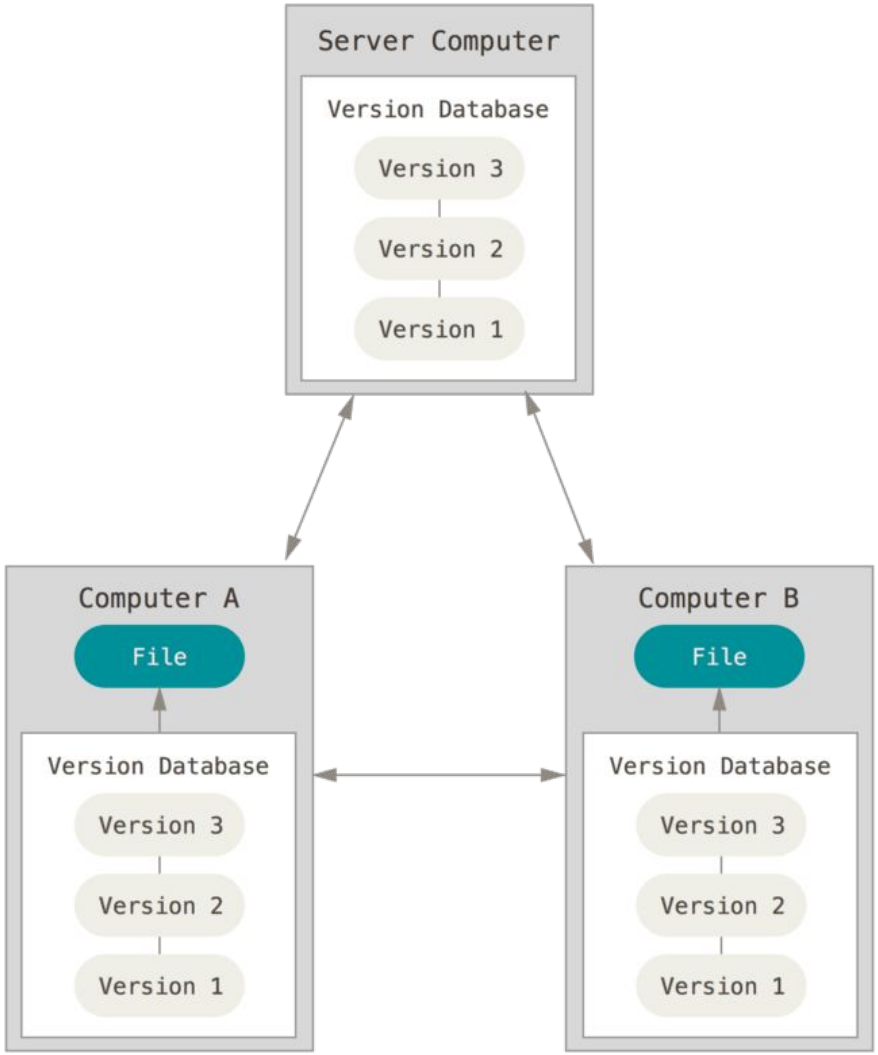
- Gestionnaires de version centralisés: SVN, CVS, Perforce



Fonctionnement d'un système de gestion de version centralisé.
Source: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

On faisait comment avant Git ?

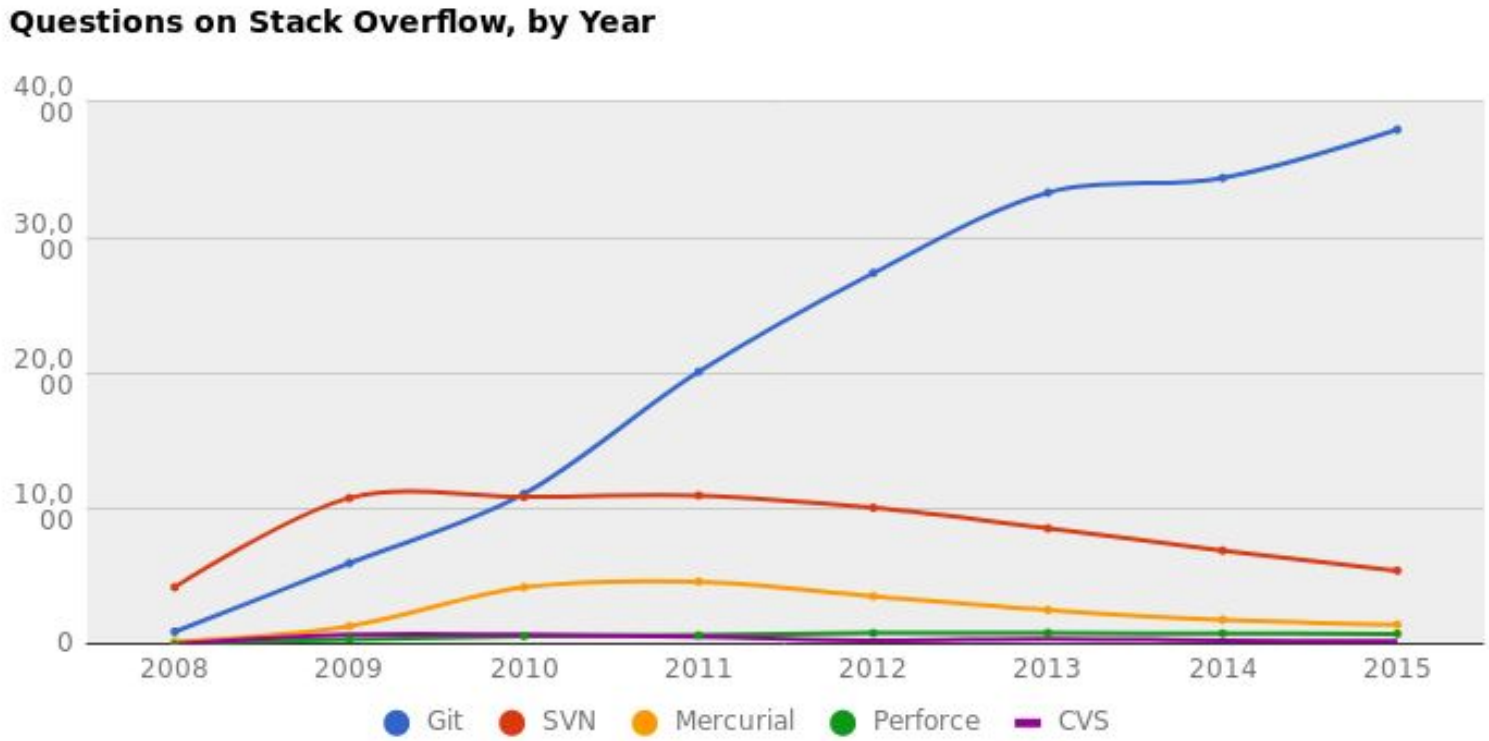
- Système D: FTP, clé USB, email, …
- Gestionnaires de version centralisés: SVN, CVS, Perforce
- Gestionnaires de version décentralisés: BitKeeper, Bazaar



Fonctionnement d'un système de gestion de version décentralisé.
Source:
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Un peu d’histoire

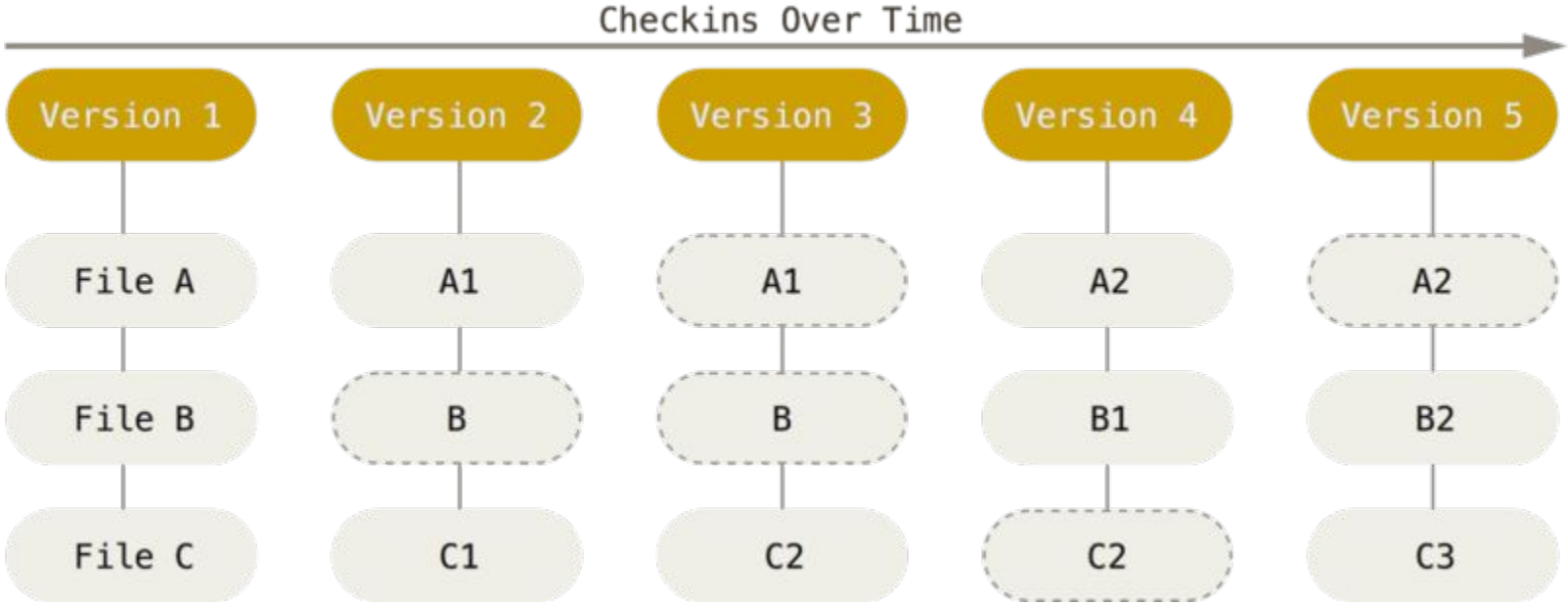
- Développé commencé par Linus Torvalds en Avril 2005
- Conçu pour gérer les sources du noyau Linux
- Popularisé par la sortie de Github en 2008
- Très populaire dans l’Open Source
- VCS (Version Control Software) le plus utilisé dans le monde en 2018



Nombre de questions Stackoverflow à propos des principaux VCS entre 2008 et 2015.
Source: https://rhodecode.com/insights/version-control-systems-2016

Fondamentaux de Git: it’s only snapshots!

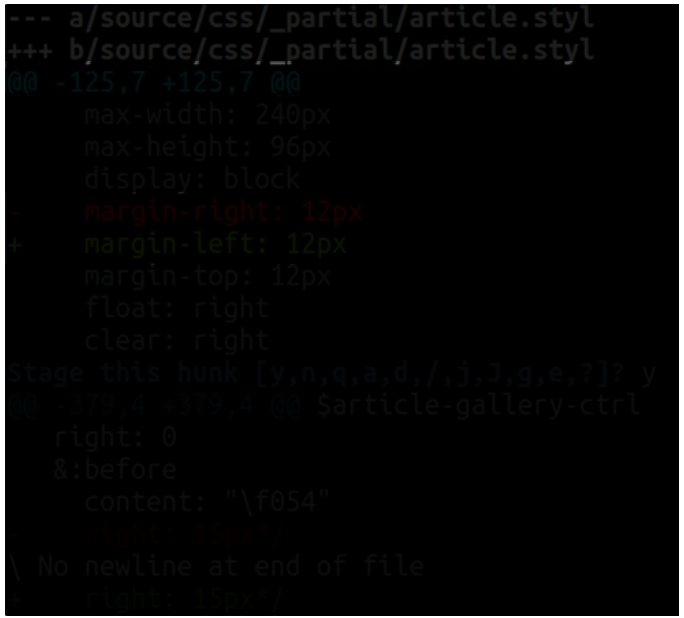
- Une version = un snapshot complet des sources



Stockage des sources sous la forme de snapshots
Source: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

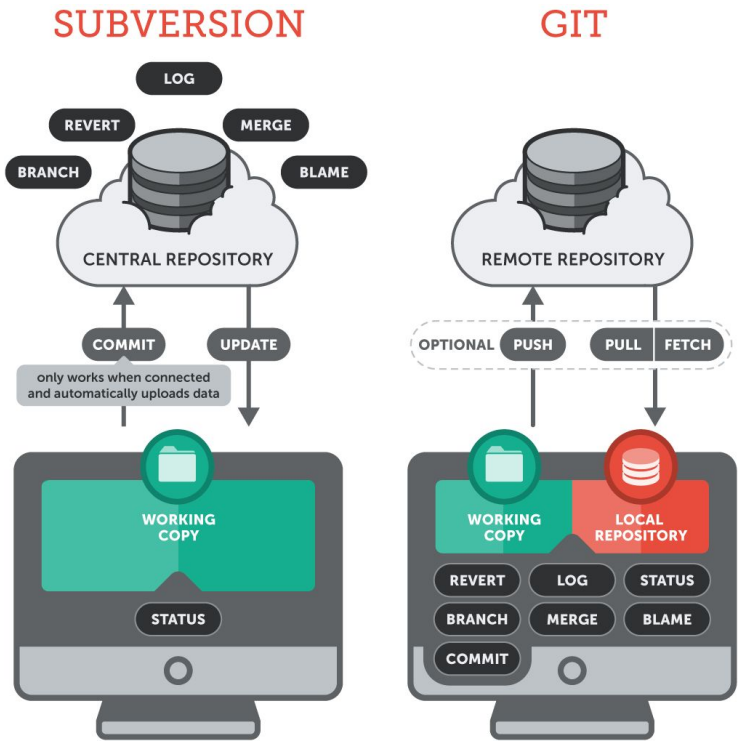
Fondamentaux de Git: it’s only patches!

- Conserve la différence ligne par ligne entre les fichiers



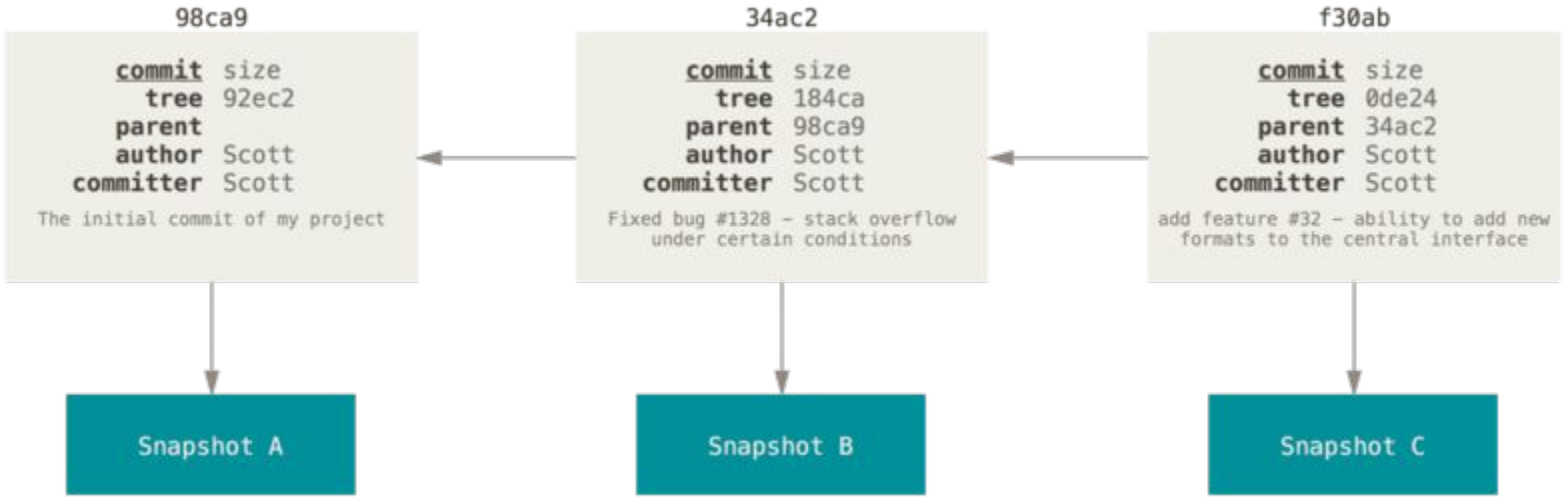
Fondamentaux de Git: it’s only local!

- La majorité des opérations se fait uniquement en local



Fondamentaux de Git: it’s maintaining integrity!

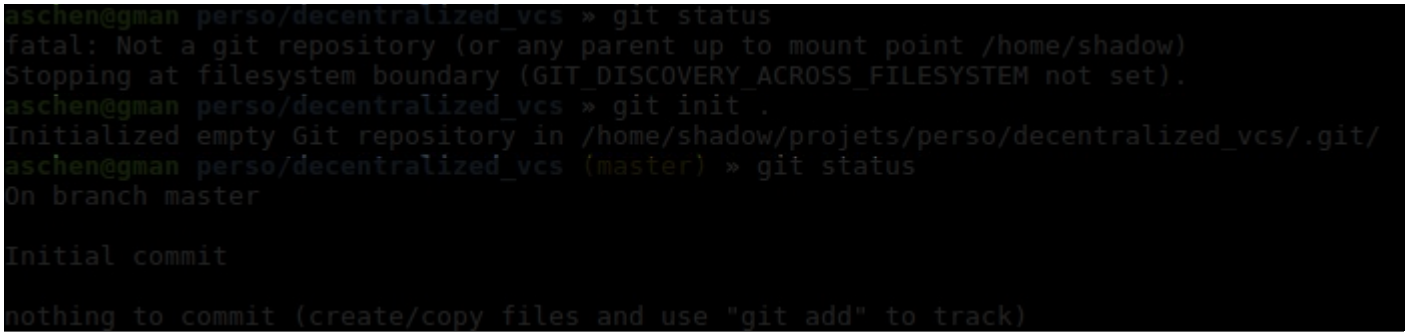
- Utilisation d’une somme de contrôle (Hash SHA1) pour assurer l’intégrité de l’arbre des modifications



Visualisation de l’arbre des commits
Source: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Initialisation d’un dépôt

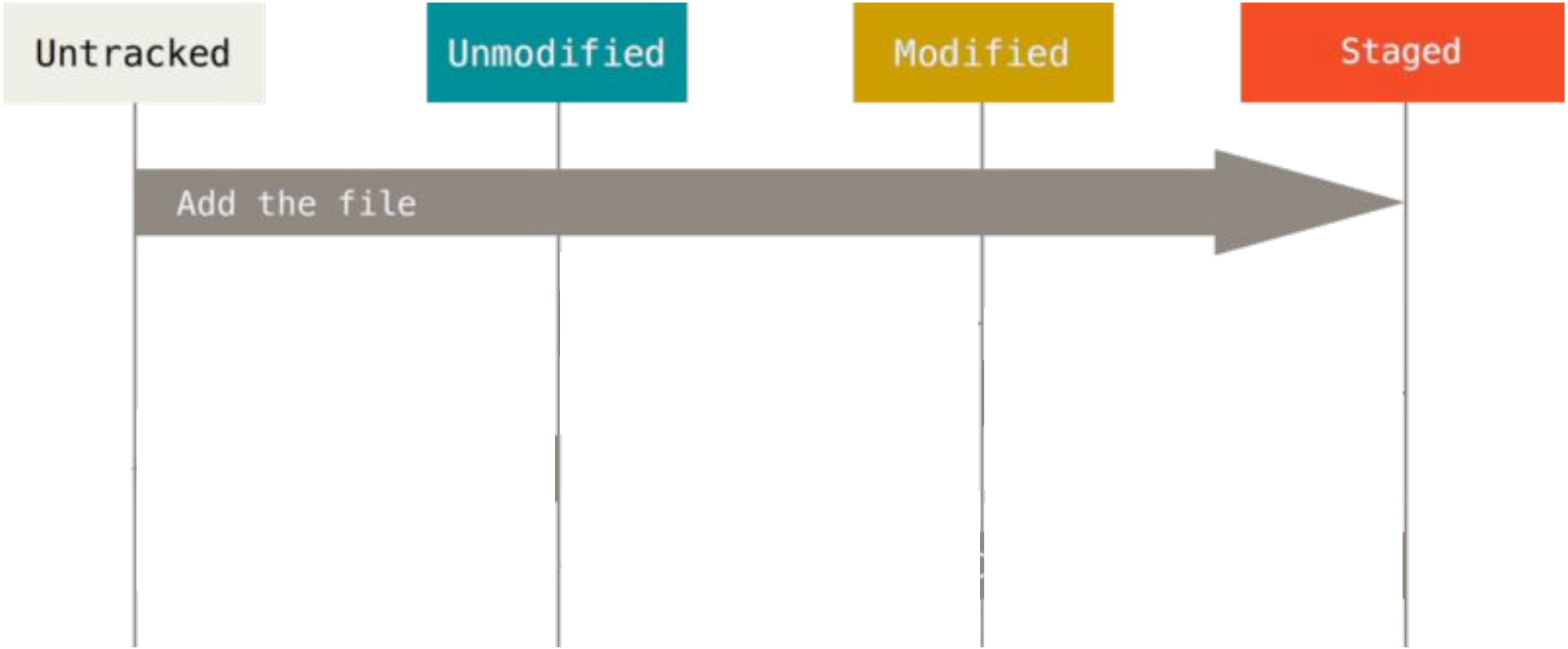
- Création en local avec ***git init***



Le cycle de vie des fichiers

- 2 types de fichiers: ***tracked*** et ***untracked***

- 3 états pour les fichiers suivis (***tracked***) par Git:
  - non-modifié (***unmodified***): fichier n'ayant pas subi de modifications
  - modifié (***modified***): fichier ayant été modifié
  - indexé (***staged***): fichier dont les modifications seront sauvegardées au prochain ***commit***



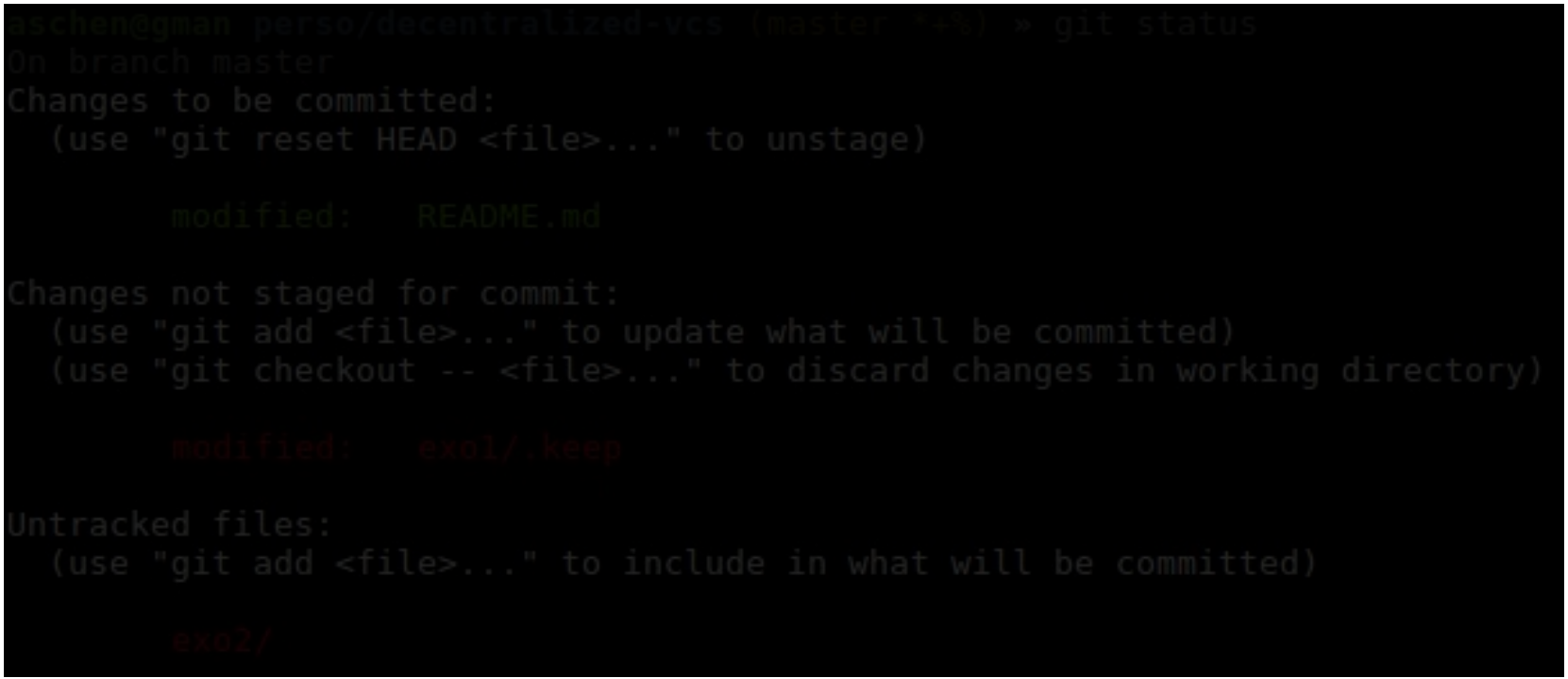
Les différents états du cycle de vie d'un fichier géré par Git
Source: https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

Le cycle de vie des fichiers

- 2 types de fichiers: ***tracked*** et ***untracked***

- 3 états pour les fichiers suivis (***tracked***) par Git:
  - non-modifié (***unmodified***): fichier n'ayant pas subi de modifications depuis le dernier **commit**
  - modifié (***modified***): fichier ayant été modifié
  - indexé (***staged***): fichier dont les modifications seront sauvegardées au prochain ***commit***

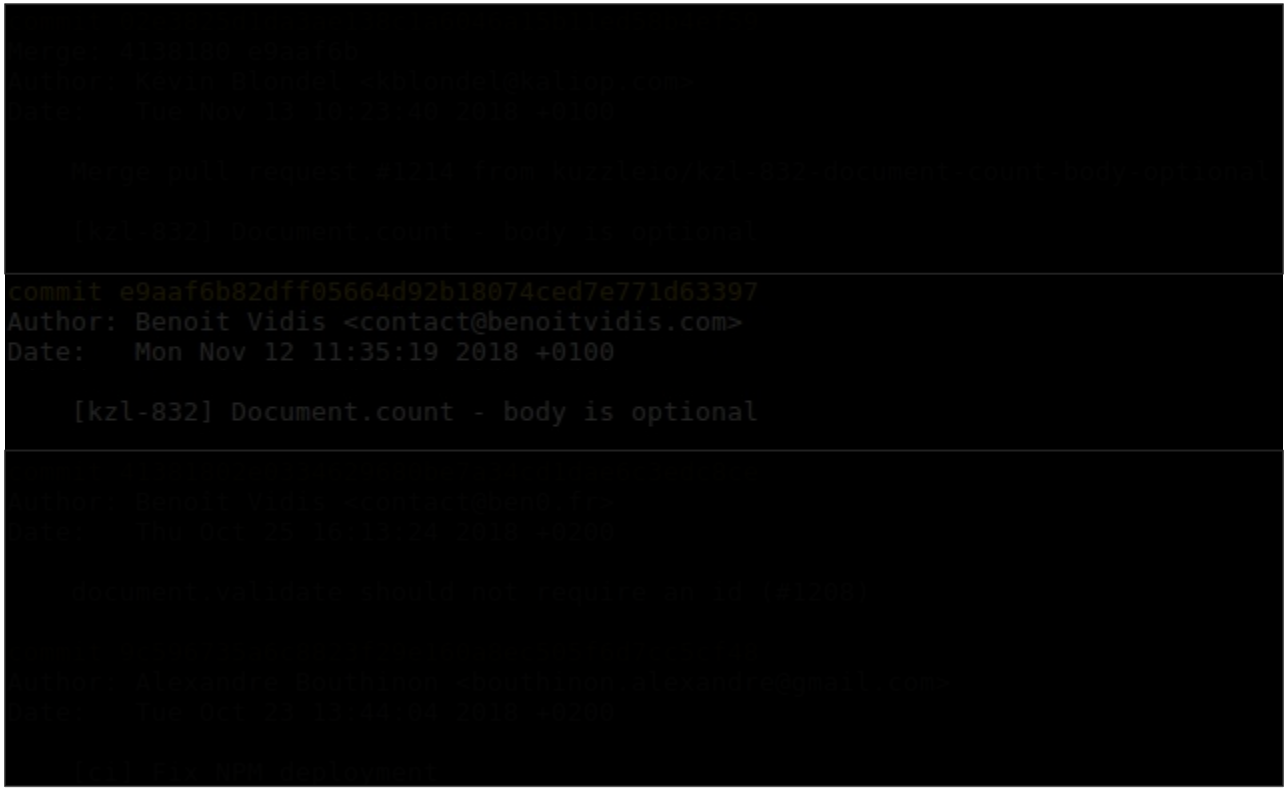
- Connaître l'état actuel du dépôt: ***$ git status***



*Résultat de la commande **git status***

Enregistrer des modifications

- Modifier son fichier



*Historique des commits par la commande **git log***

TD: Comprendre les différentes zones

- Initialiser un dépôt
- Créer un nouveau fichier
- Afficher git status
- Ajouter le fichier à la zone de staging
- Afficher git status
- Commit le fichier
- Afficher git status

- Modifier le fichier
- Ajouter le fichier à la zone de staging
- Annuler les modifications

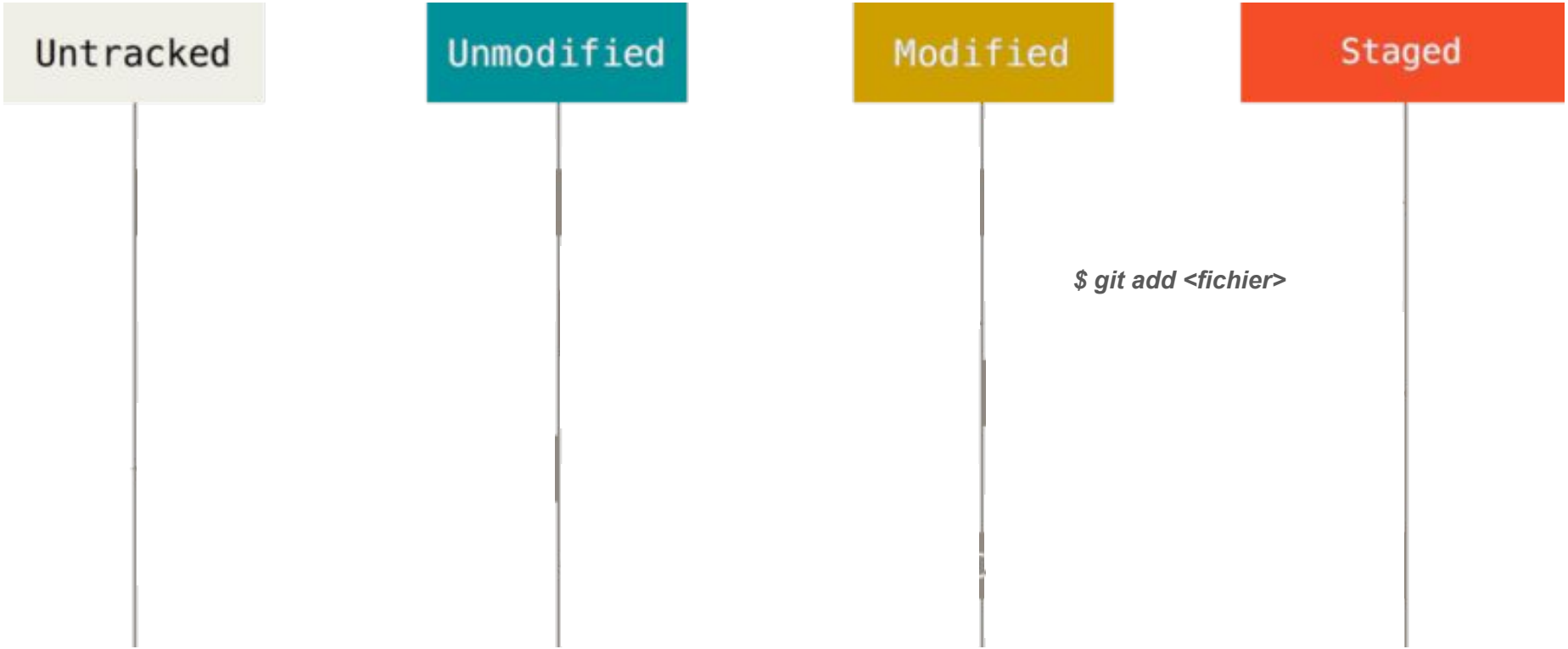
- Modifier le fichier
- Ajouter le fichier à la zone de staging
- Commit le fichier
- Annuler le commit mais conserver les modifications
- Annuler les modifications

Comment faire un bon commit ?

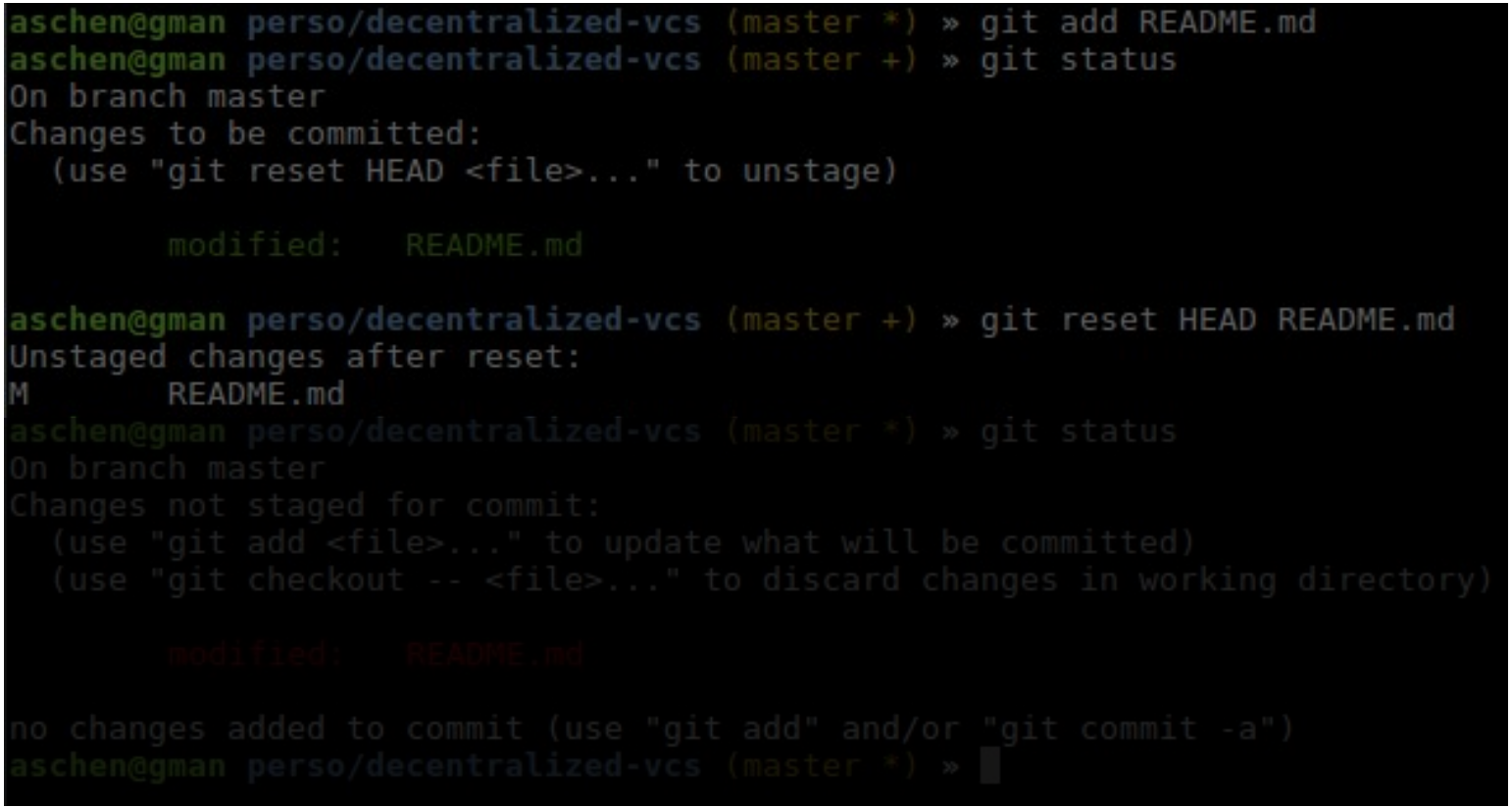
- Garder le même contexte pour les modifications entrant dans le commit

Revenir en arrière

- Enlever un fichier des fichiers **staged**
  - ***$ git reset HEAD <fichier>*** *(<fichier> ne sera pas intégré au prochain commit)*



*Enlever un fichier de la zone de staging avec **git reset HEAD <fichier>***

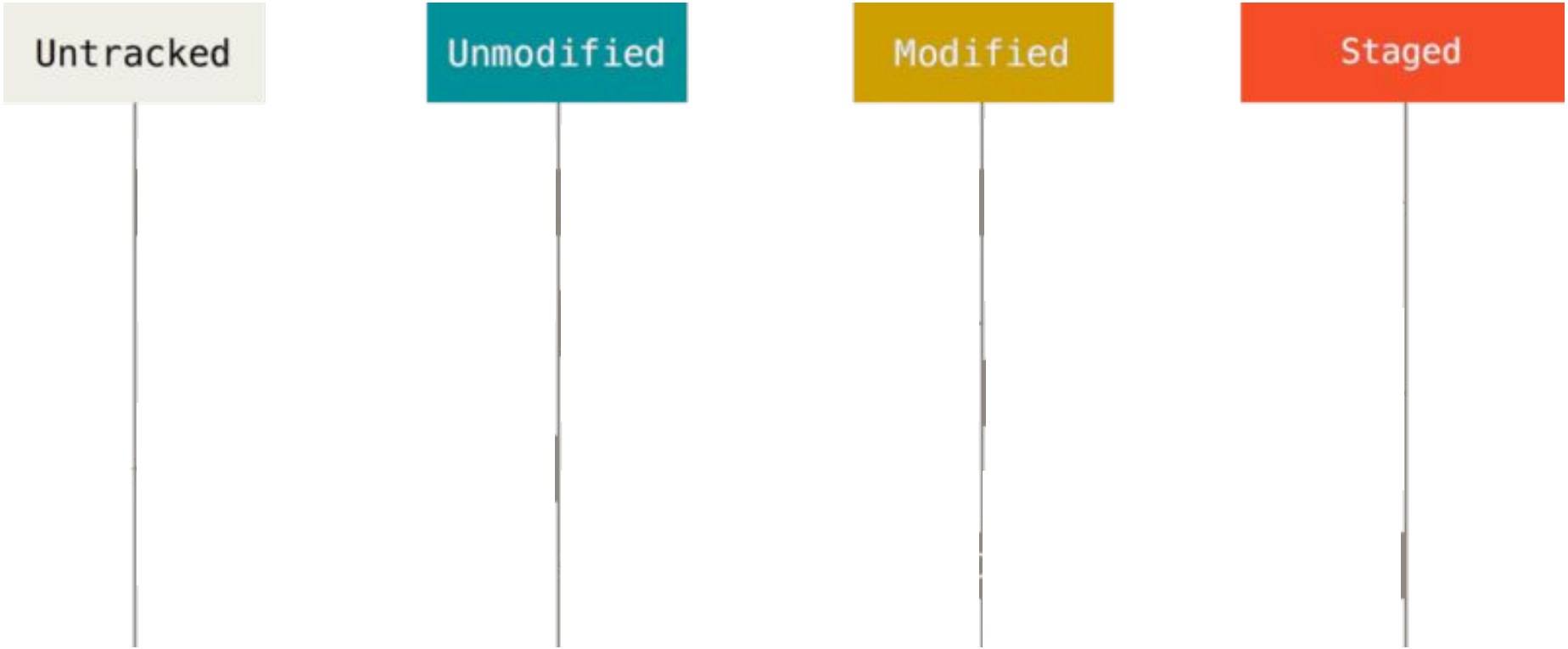


*Enlever un fichier de la zone de staging avec **git reset HEAD <fichier>***

Revenir en arrière

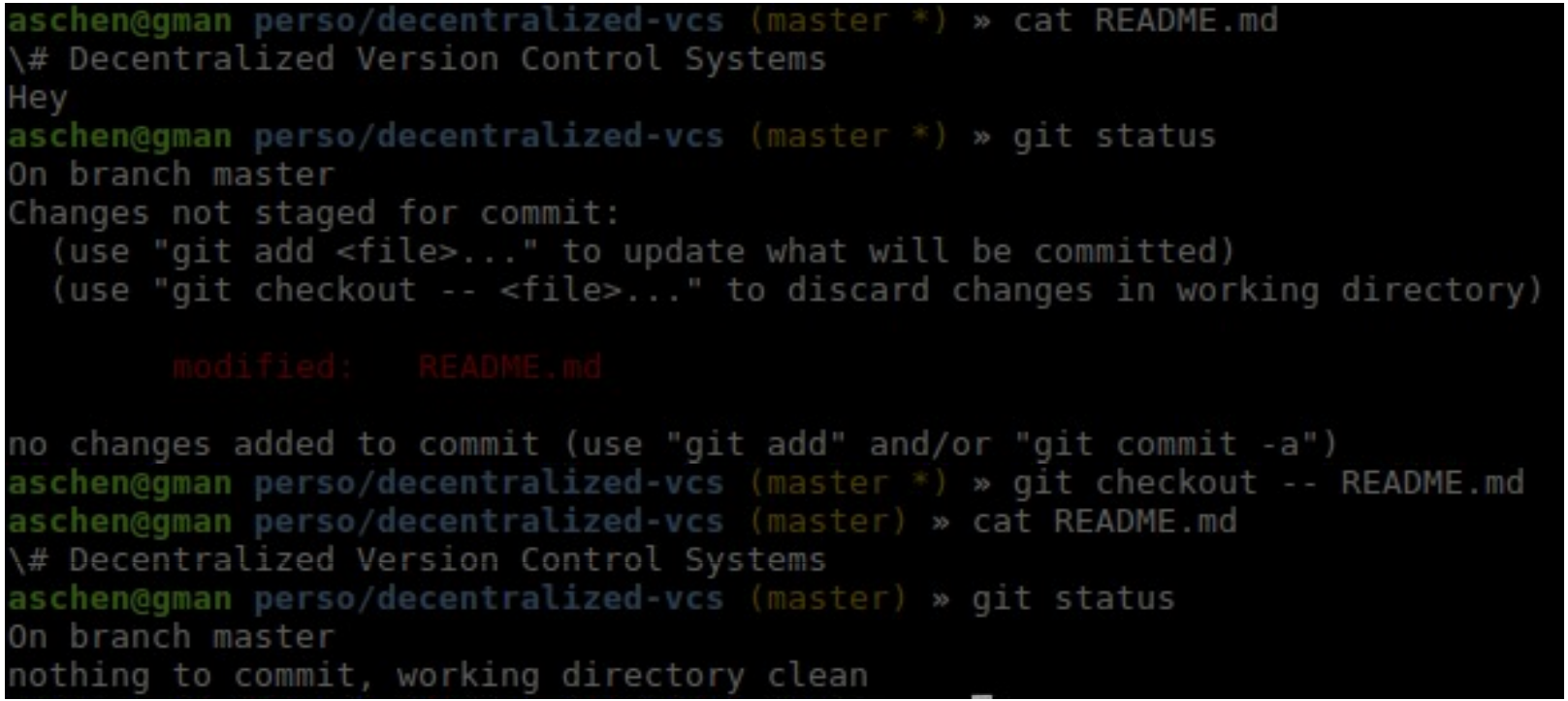
- Retirer un fichier des fichiers du prochain commit (**staged** => **modified**)
  - ***$ git reset HEAD <fichier>***

- Annuler les modifications faites sur un fichier (**modified** => **unmodified**)
  - ***$ git checkout -- <fichier>***

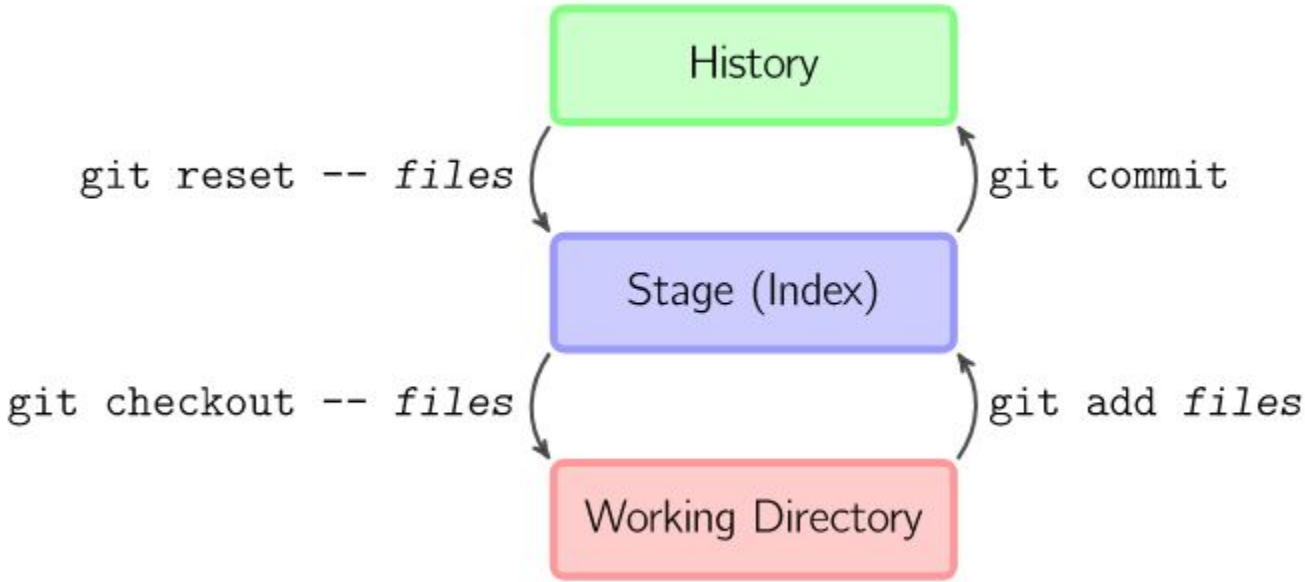


$ git checkout -- <fichier>

*Enlever un fichier de la zone de staging avec **git reset HEAD <fichier>***



*Annuler les modifications faites sur un fichier avec **git checkout -- <fichier>***



Résumé des processus d'ajout de fichiers et des retours en arrière possibles
Source: https://marklodato.github.io/visual-git-guide/index-fr.html

Développer en parallèle avec les branches

- Une branche est un historique de commits qui se suivent
  - Par défaut, les commits sont sur la branche **master**
  - Chaque branche avance de manière indépendante

- Création d'une branche à partir du commit courant
  - ***$ git branch <nom de la branche>***

- Utiliser une branche
  - ***$ git checkout <nom de la branche>***

- Créer + utiliser une branche
  - ***$ git checkout -b <nom de la branche>***


Free Explore
Have fun!
Local Repository
HEAD: master
e137e9b..
first commit
master
HEAD
$ enter git command

Se déplacer dans l’arborescence des commits

- Emplacement courant symbolisé par **HEAD**

- Déplacer la **HEAD**
  - ***$ git checkout <branche|commit>***

- Déplacer la **HEAD** relativement
  - ***$ git checkout <branche|commit>^* : un commit en arrière**
  - ***$ git checkout <branche|commit>~2* : deux commits en arrière**


Free Explore
Have fun!
Local Repository
HEAD: master
e137e9b..
first commit
master
HEAD
$ enter git command

Annuler des commits: git reset

- Modifications locales seulement
  - ***$ git reset <commit>***
  - ***$ git reset HEAD~1***
- **Conserve les modifications mais supprime les commits**

- Utilisable sur un fichier à la fois
  - ***$ git reset HEAD <fichier>***

```
aschen@gman perso/decentralized-vcs (master) » cat README.md
\# Decentralized Version Control Systems
Hey
aschen@gman perso/decentralized-vcs (master) » git log --pretty=oneline | head
f8479a6ef61f70d675cf900c6d50019f955e9301 Add hey to README.md
d8a2ce1677a092c0a0ce486cc91088d747c5f262 add exo2
4b1c40b4a0b853ca8b3e7fe100e8aa9656f318b9 work
91ee56ed7af5cbb6262f4a6f94fa0ac35b0f889b first commit
aschen@gman perso/decentralized-vcs (master) » git reset --soft d8a2ce
aschen@gman perso/decentralized-vcs (master +) » git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

aschen@gman perso/decentralized-vcs (master +) » git reset HEAD README.md
Unstaged changes after reset:
M       README.md
aschen@gman perso/decentralized-vcs (master *) » git checkout -- README.md
aschen@gman perso/decentralized-vcs (master) » cat README.md
\# Decentralized Version Control Systems
```

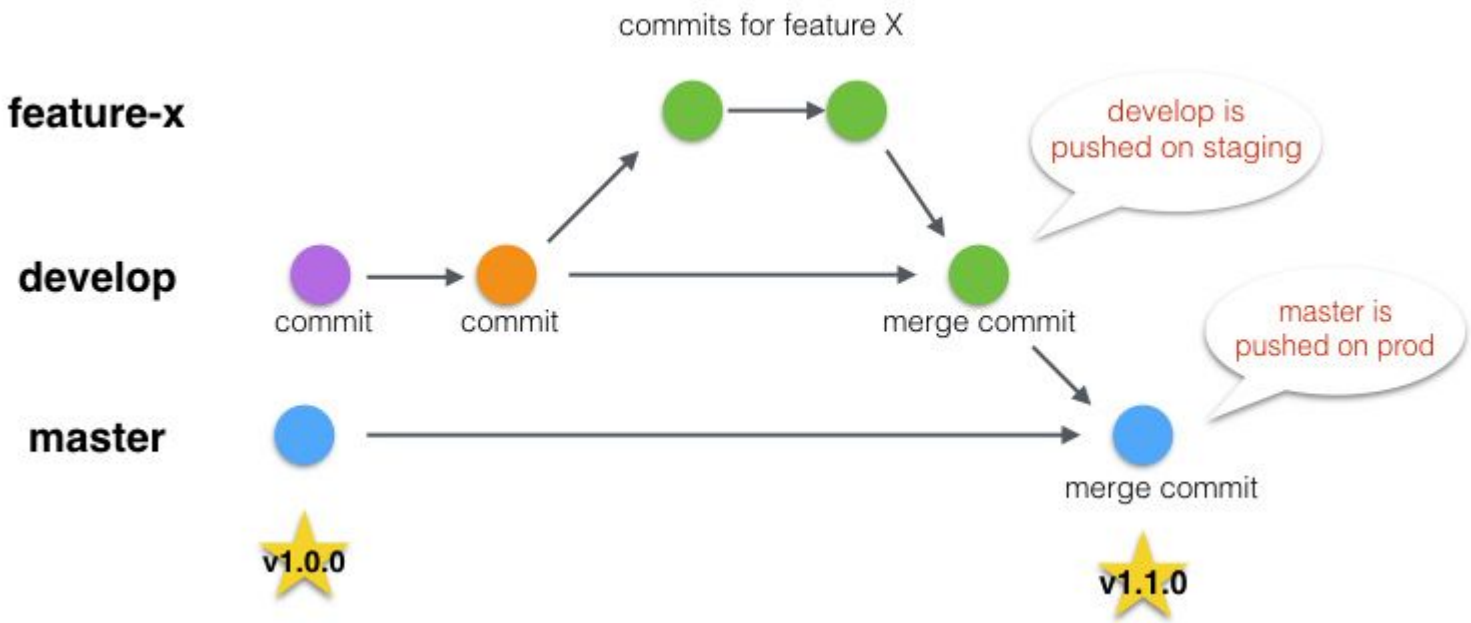
*Revenir à un état antérieur du dépôt avec **git reset --soft <commit>***

*Utilisation de **git revert <commit>** pour annuler un commit tout en conservant l'historique*


Free Explore
Have fun!
Local Repository
HEAD: master
e137e9b..
first commit
master
HEAD
$ enter git command

Fusion de branches

- Une branche = une fonctionnalité en cours de développement

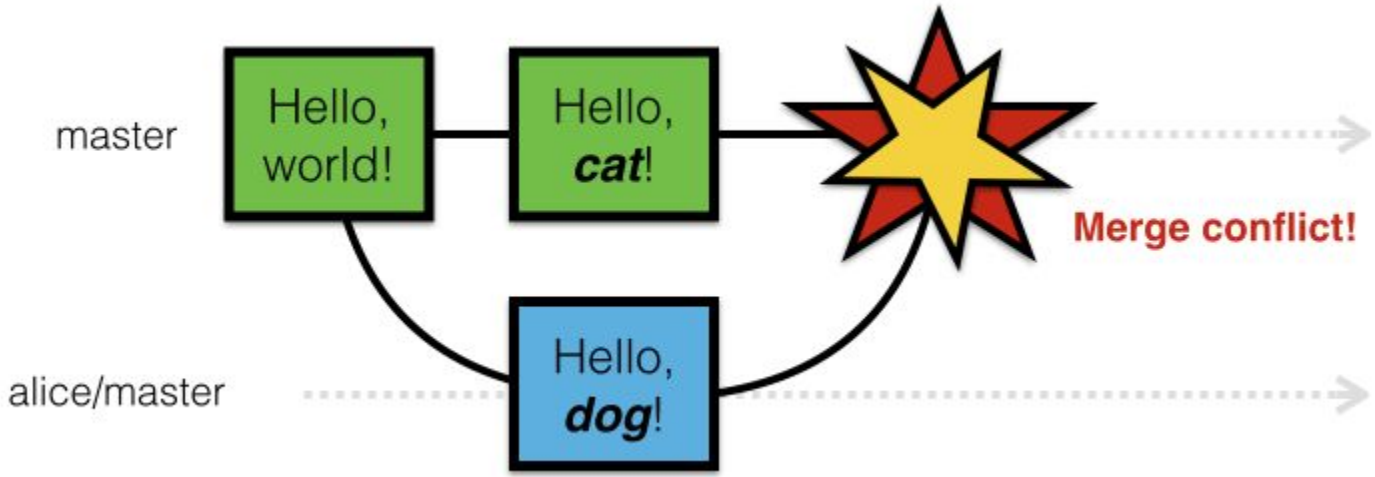


http://git-school.github.io/visualizing-git/


Free Explore
Have fun!
Local Repository
HEAD: master
e137e9b..
first commit
master
HEAD
$ enter git command

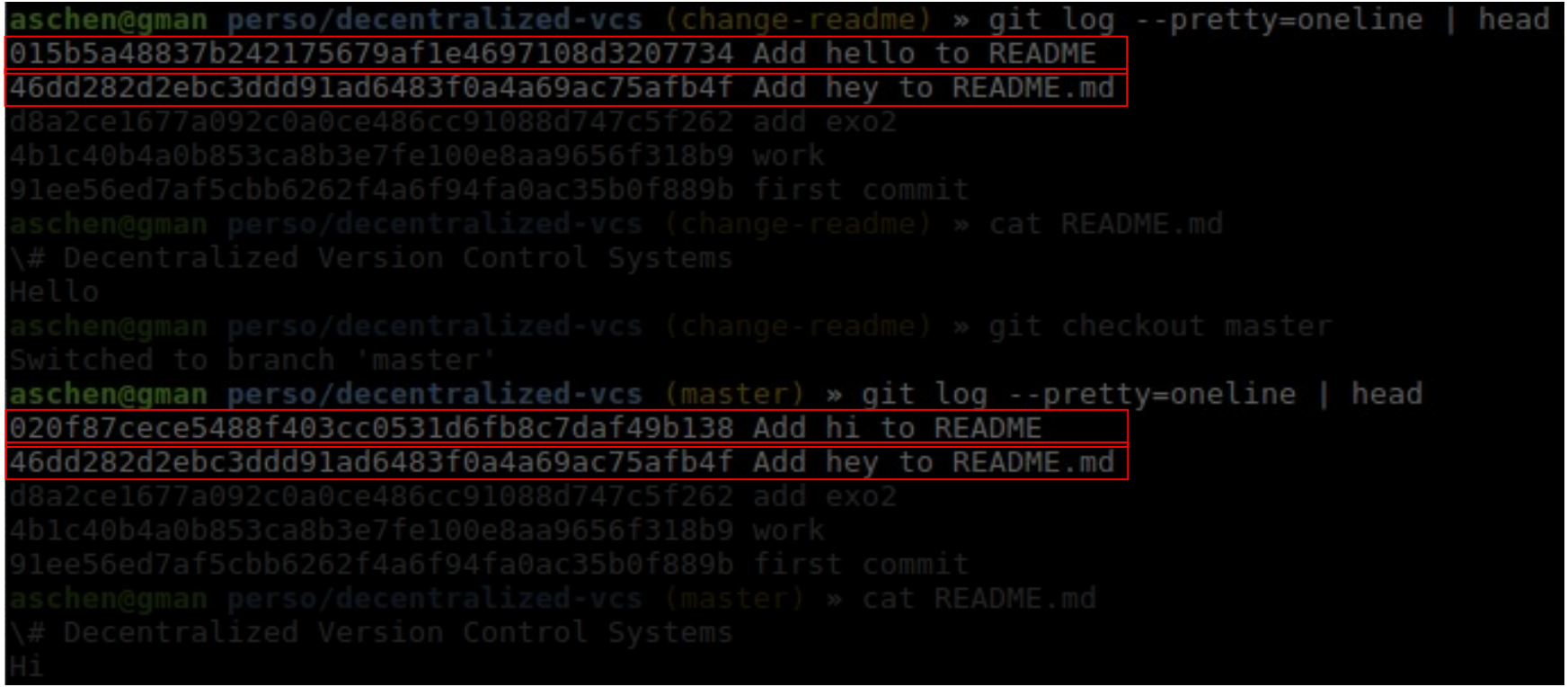
Résoudre les conflits de fusion

- Apparaissent lorsqu'une même ligne est modifiée sur les deux branches

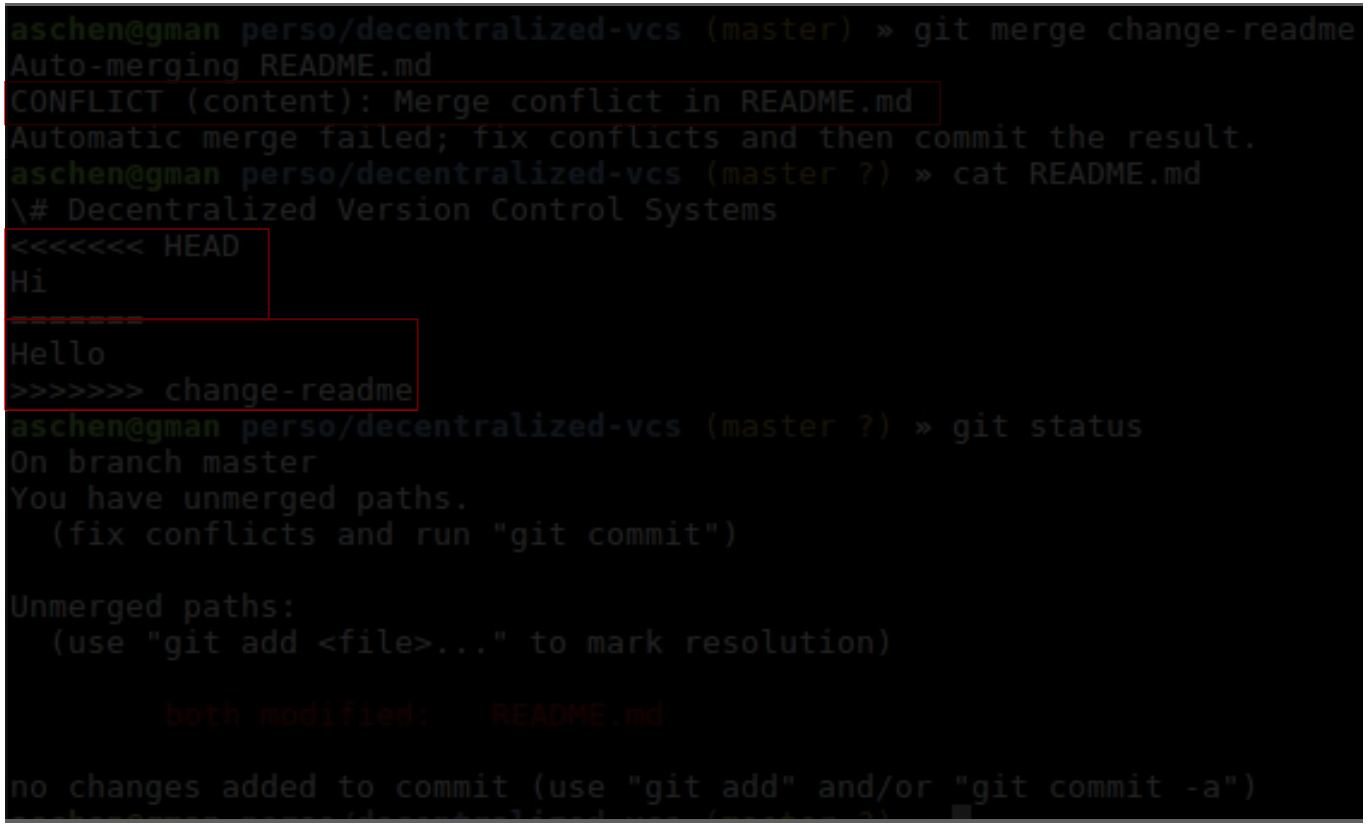


Source: https://developer.atlassian.com/blog/2015/01/a-better-pull-request/

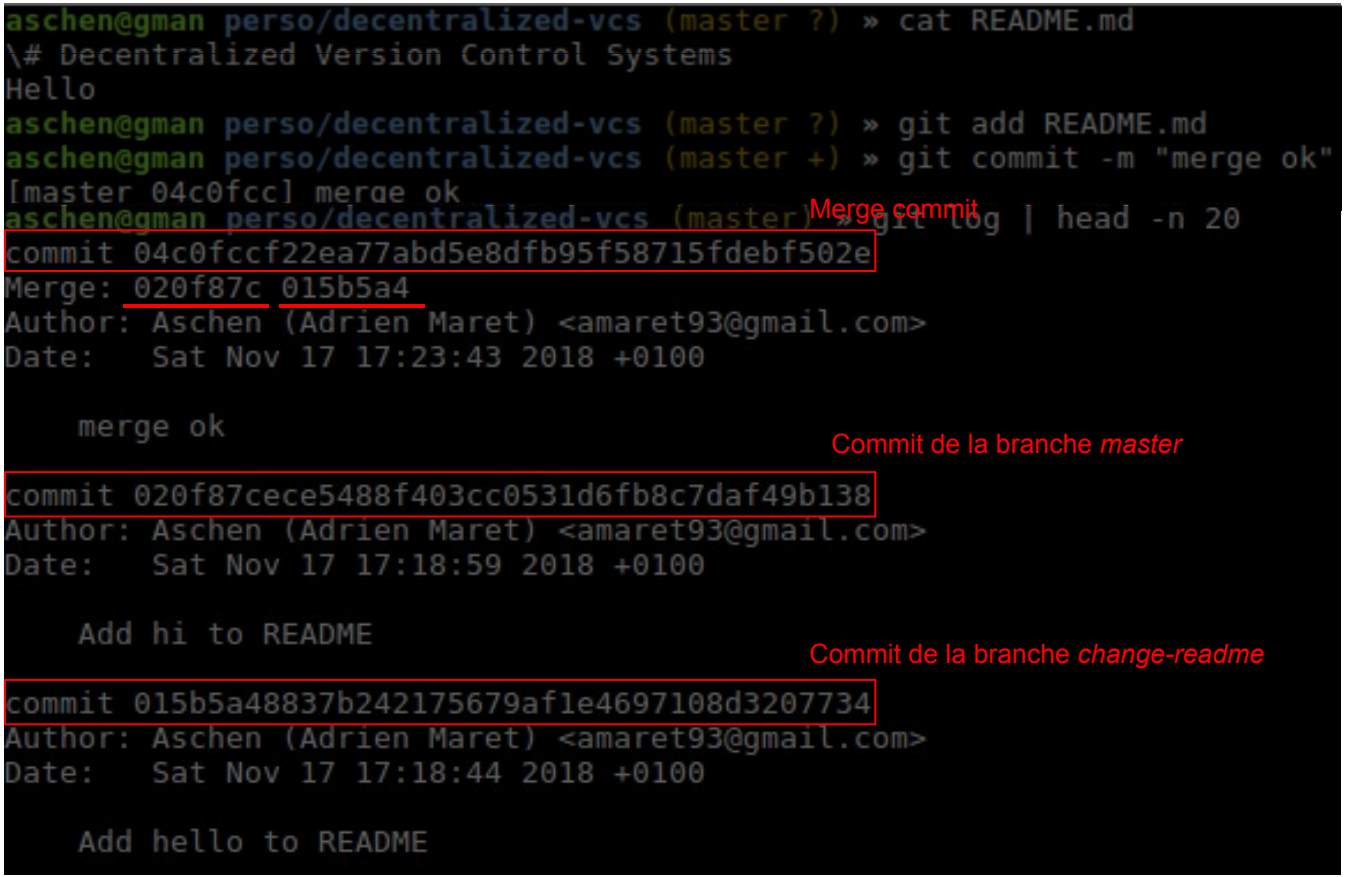

Free Explore
Have fun!
Local Repository
HEAD: master
e137e9b..
first commit
master
HEAD
$ enter git command



Exemple d'un conflit entre deux branche.
Ici chaque branche a avancé d'un commit ayant modifié la même ligne du même fichier.



Exemple d'un conflit entre deux branches.
Ici Git nous indique à quel endroit il y a un conflit pour que nous puissions le résoudre.



Exemple d'un conflit entre deux branches.
*Ici nous avons résolu le conflit avec un **merge commit**.*
L'historique contient bien tous les commits de chaque branche

1. Créer un fichier avec plusieurs lignes de contenu
2. Changer de branche, modifier une des ligne puis commiter
3. Retourner sur master
4. Changer de branche, modifier la même ligne puis commiter
5. Retourner sur master
6. Merge la première puis la deuxième branche dans master
7. Comprendre et résoudre le conflit