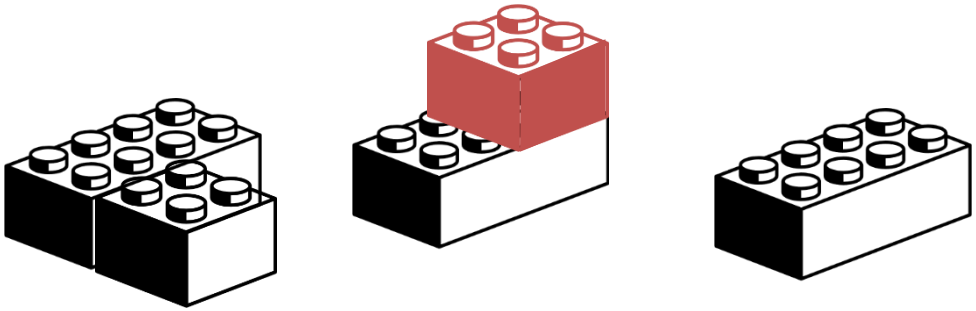
Versionning

J. Saraydaryan, G. Morel



Retrouver ses documents ou productions
Ne pas avoir à refaire un travail déjà fait
Annuler une erreur
Lutter contre la Fatalité (ordi HS, disque dur HS, clef usb perdu, Ordinateur volé , cloud corrompu, cloud HS !)
Combien d'heures de travail, êtes vous prêts à perdre ?
Certaines pertes sont irrattrapables !
Pouvez vous vous le permettre ?

Bonnes pratiques

Règle de bases:
Faires des copies locales au moins toutes les deux heures
Faire des copies sur au moins 2 supports pour les choses importantes
Sauvegarder dans un drive/ mail/ cloud tous documents représentants quelques jours de travails
Dans le cas d'un travail collaboratif, tout le monde doit avoir une copie du travail réalisé par le groupe.

Enfin
Un document vraiment important doit être sauvegarder à quatre endroits (ordinateur, clef usb, disque dur externe (copie globale ordi) et solution cloud (drive, mail …))

Si je dois par exemple projeter ou partager un travail à partir d'une clef usb (ou ordinateur), je prévois un plan B si problème avec le plan A

Conserver des informations pertinentes sous forme de dossier et noms de fichiers . Inclure un numéro itératif dans les noms ou une référence aux dates de dernière modification (Ne pas faire confiance aux dates des fichiers)

Cas du développement d'un code ou d'un schéma électronique ou équivalent.

Je conserve des versions dès que j'ai quelques choses qui marchent.
Dans le cas de projet complexe ou passant par un environnement de développement ou l'on paramètre beaucoup de choses . Je fais des copies des répertoires et je les archives (tar-gz, zip) et je conserve les archives pour etre sur de ne jamais les écraser.

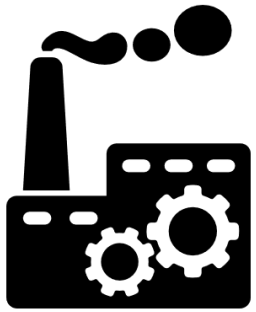
Quand un code complexe ne marche plus on peut ainsi revenir à un code qui marche .
On est toujours capable de relancer un code qui tourne même si on est en train de la modifier

PS , je commente mes codes, je laisse des README et des TODO associés aux répertoires

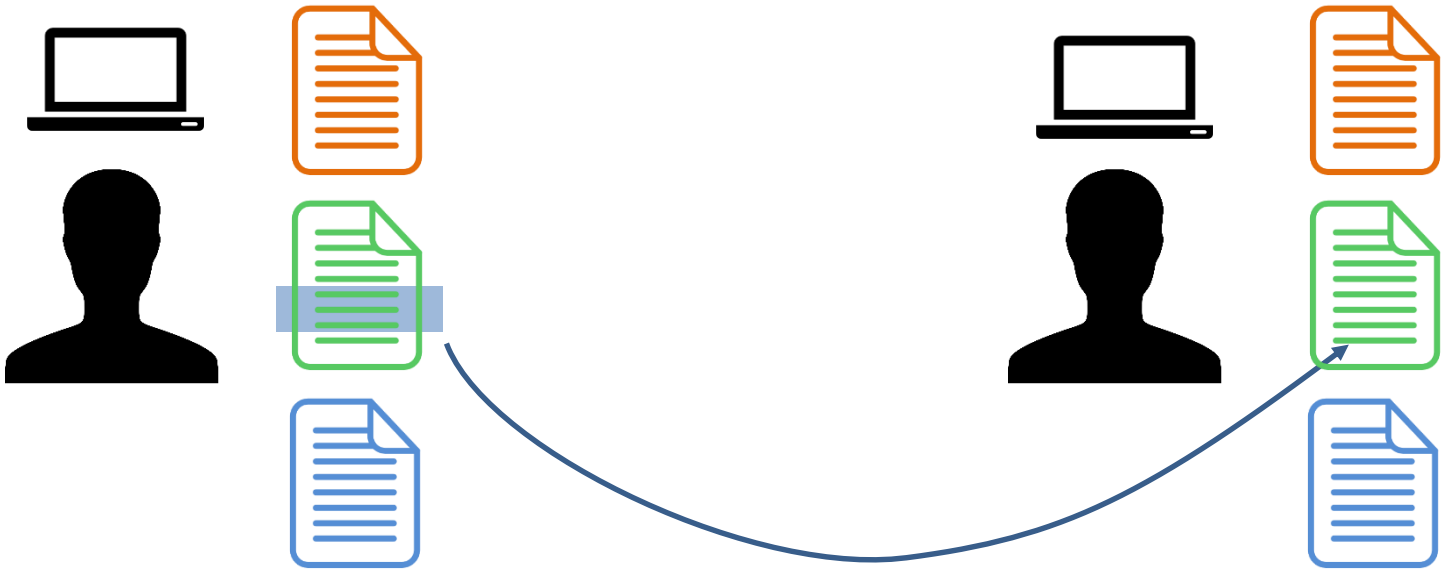
La collaboration dans le dev.

- Suivre l'évolution d'un code source
  - Revenir à une version antérieure
  - Comprendre les différentes modifications effectuées
  - Sauvegarder le travail effectué
  - Documentation du code

- Collaboration, travailler à plusieurs
  - Partager les modifications effectuées
  - Savoir qui a effectué des modifications
  - Fusionner les modifications de plusieurs développeurs.

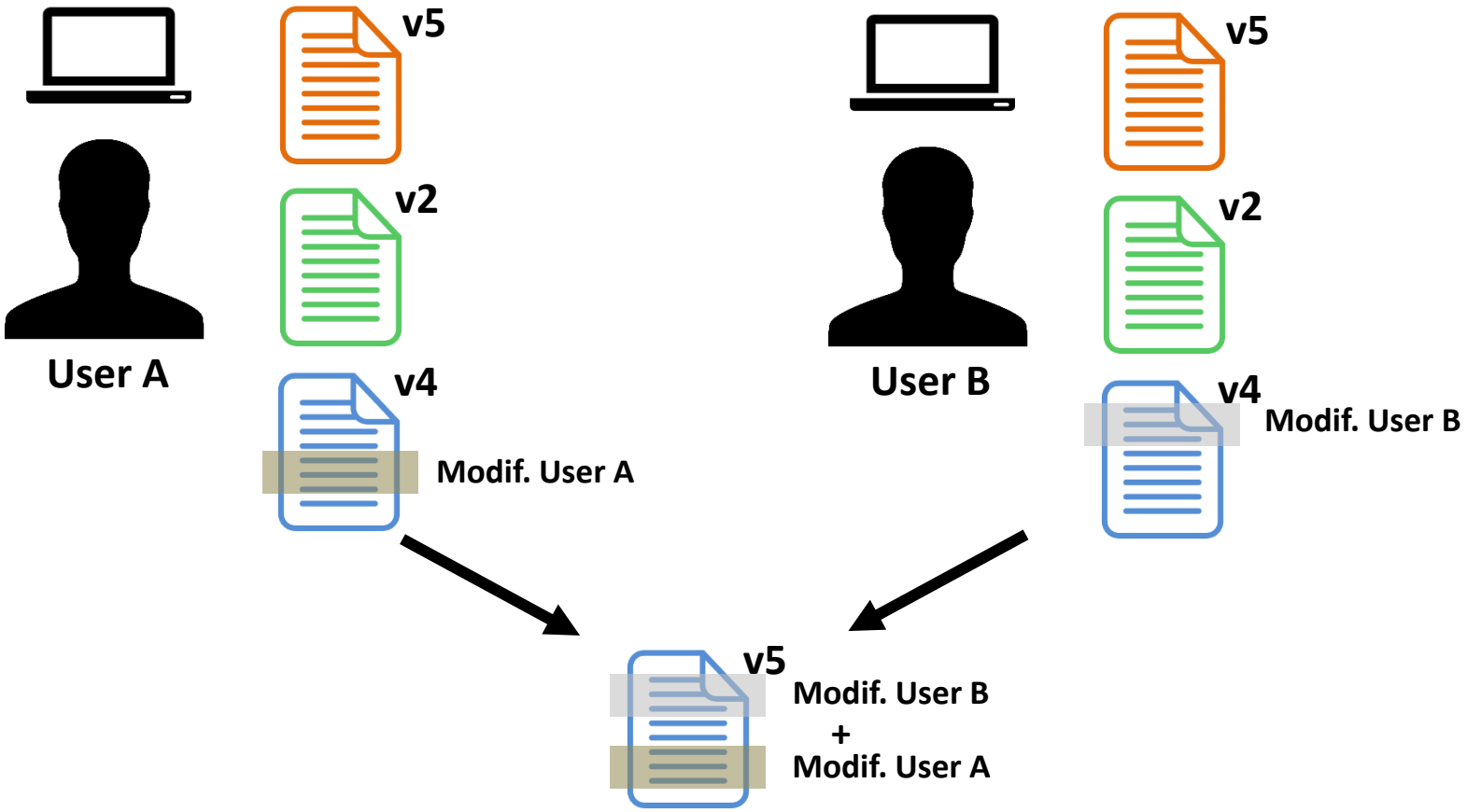


Les situations types



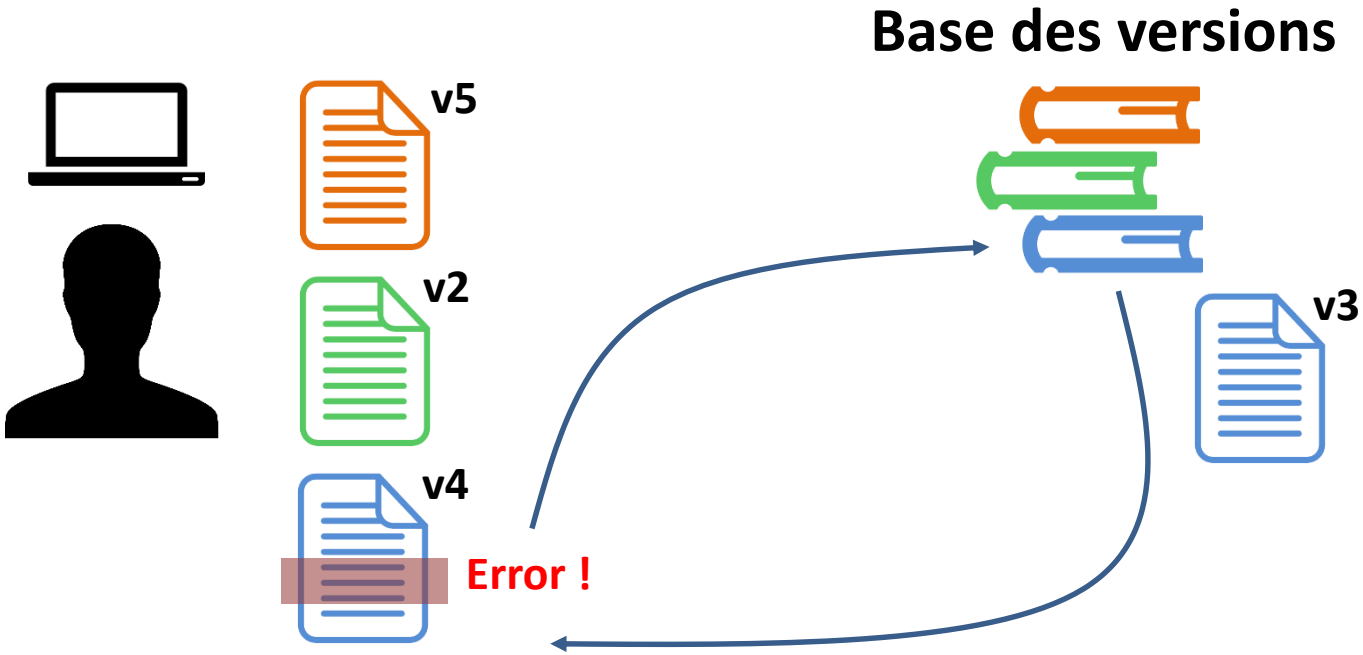
Partager des modifications

Les situations types



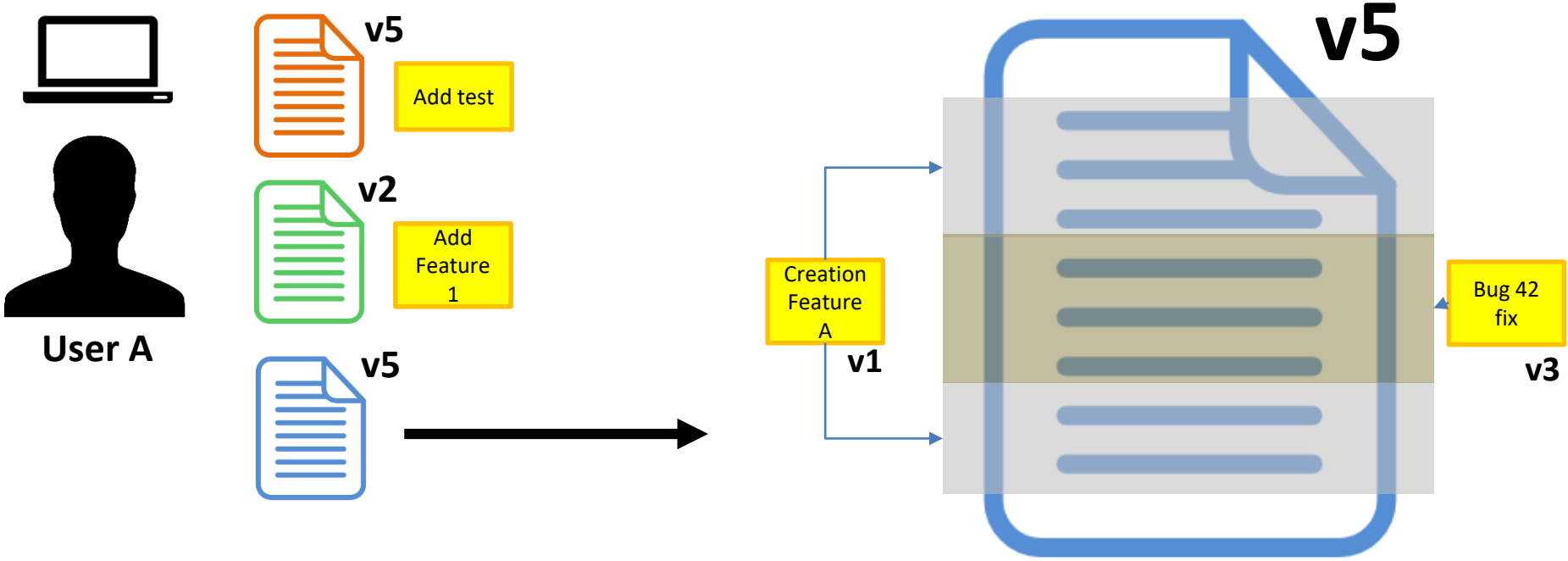
Fusionner des modifications

Les situations types

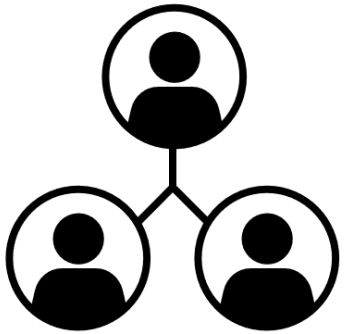


Restaurer une ancienne version d'un fichier

Les situations types



Comprendre l'historique des modifications



Les logiciels de gestions de version

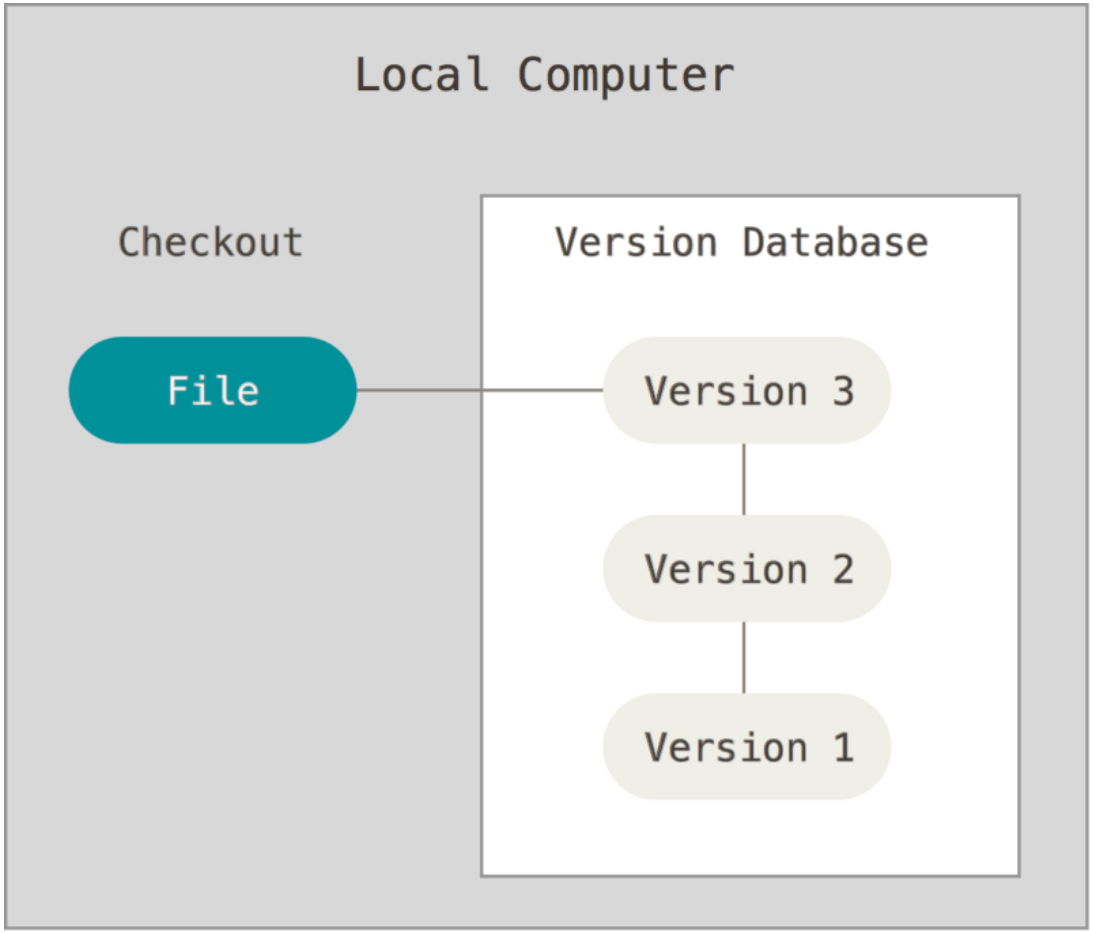
Définition

- Définition

Outil permettant de garantir le **suivi de versions** d'un ensemble de fichiers et fournissant des outils permettant de **naviguer** / **commenter** / **fusionner** les différentes **versions de fichiers**.

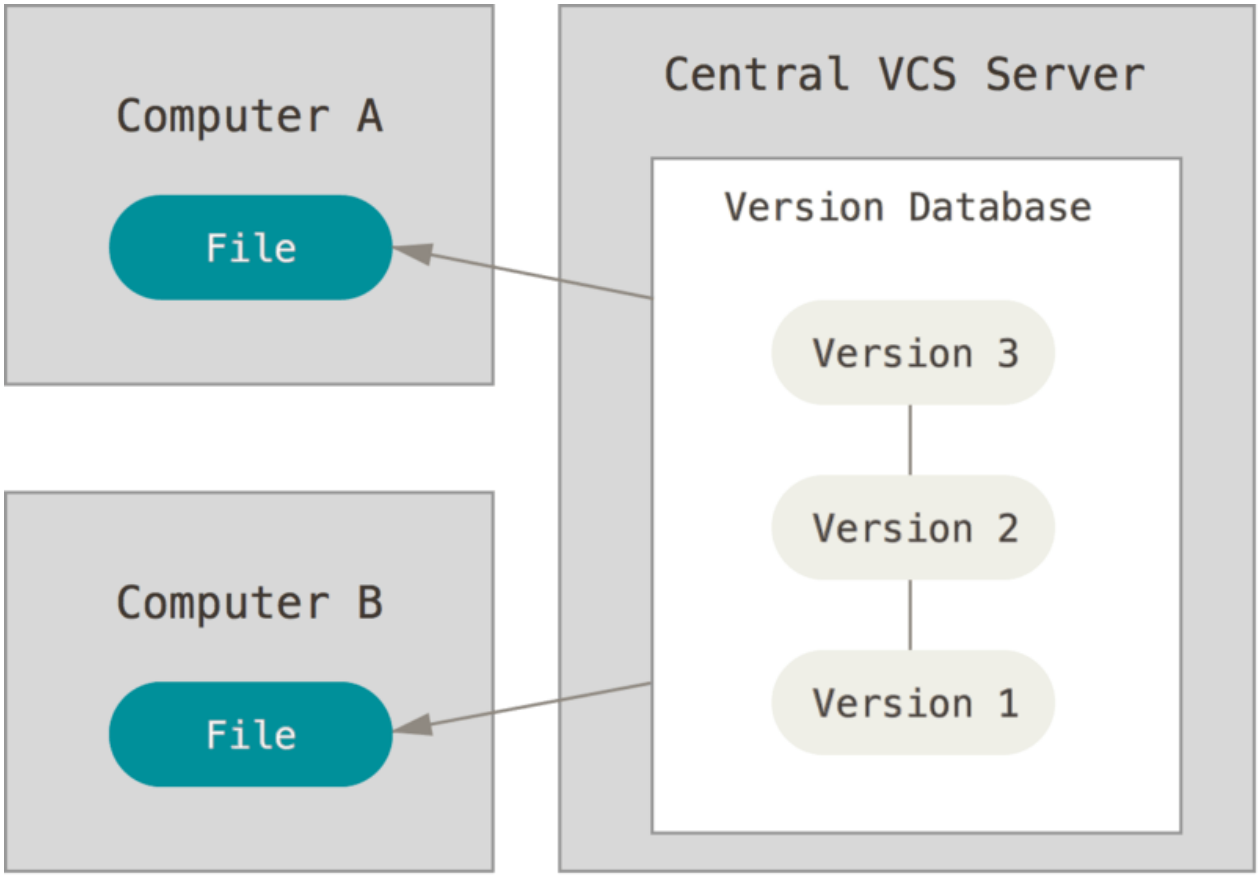
- Architecture
  - Locale :
    le suivi des versions des fichiers est **uniquement** assuré sur la machine **locale**
  - Centralisée:
    le suivi et le maintien de version est garanti par un **serveur central**
  - Décentralisée :
    chaque développeur possède l'ensemble des versions des documents et **informe le réseau de collaborateurs de toutes modifications**

Architecture Locale



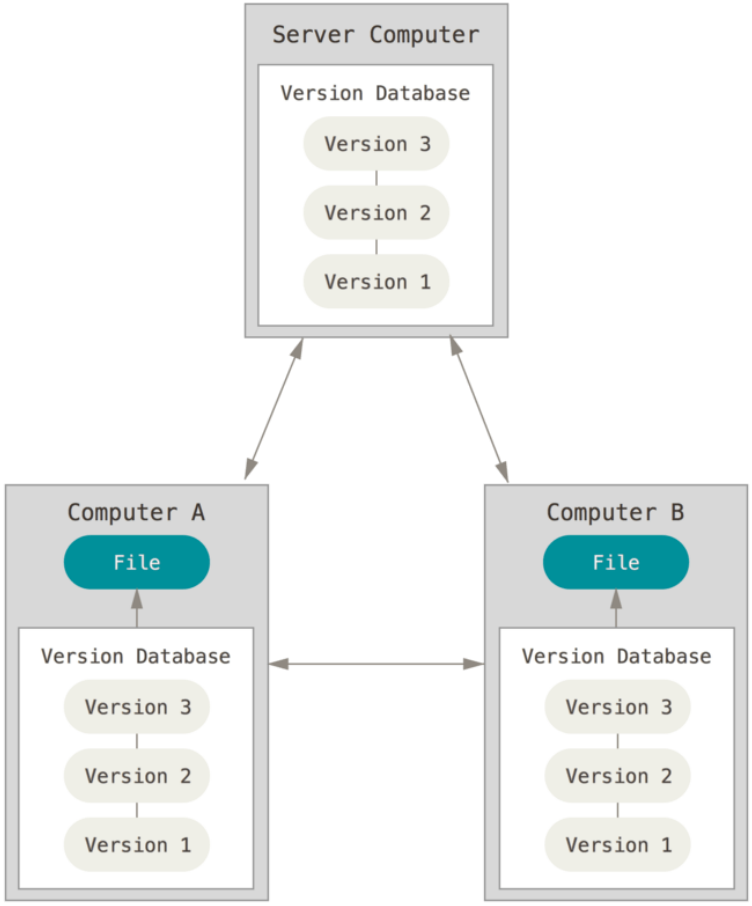
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Architecture centralisée



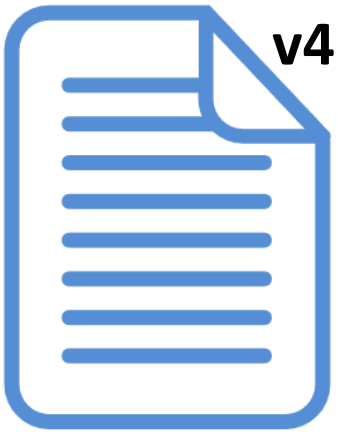
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Architecture décentralisée

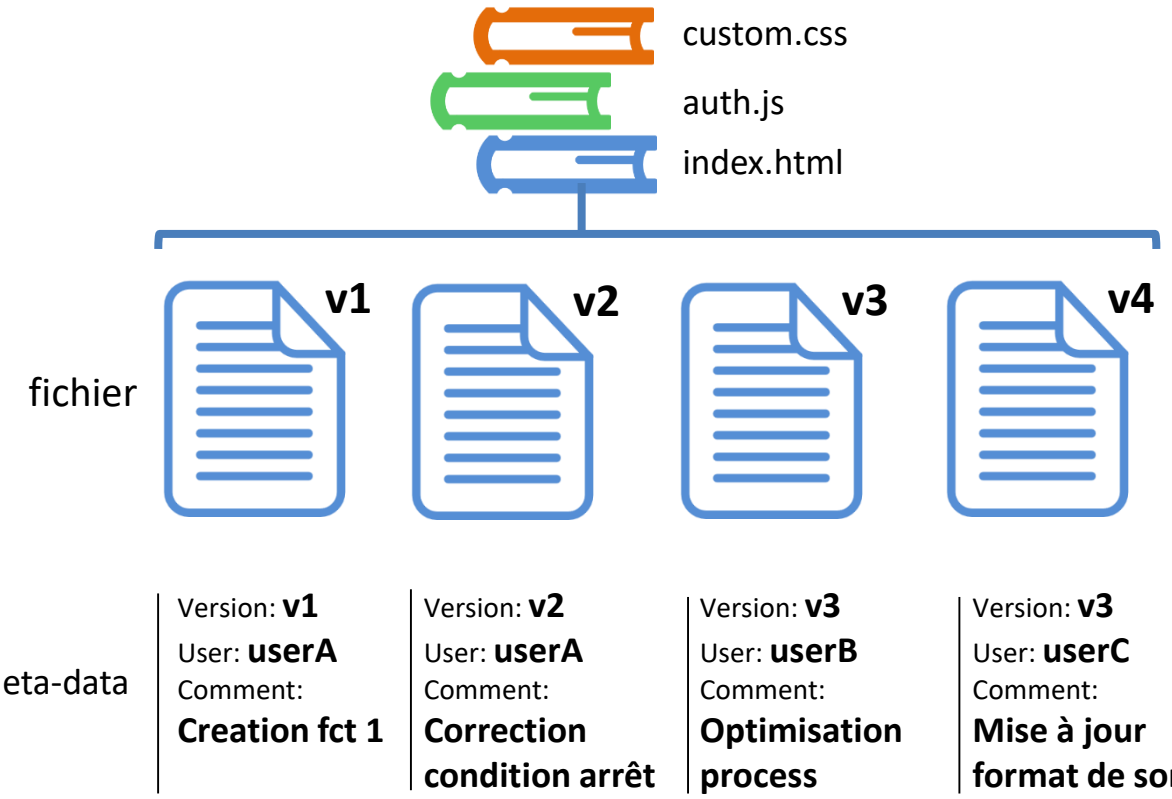


https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

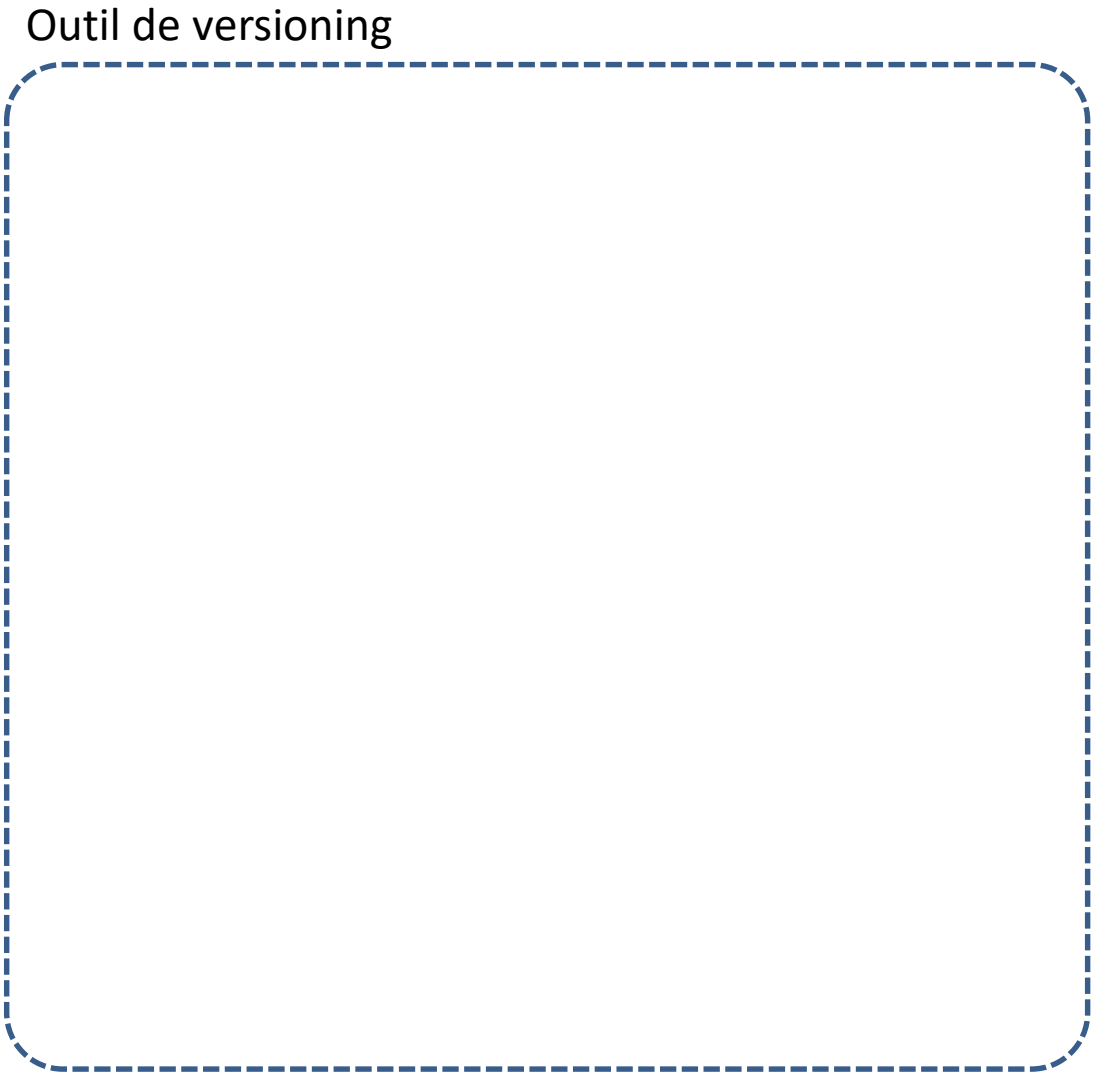
Fonctionnement



Historique des modifications



Fonctionnement



Fonctionnement

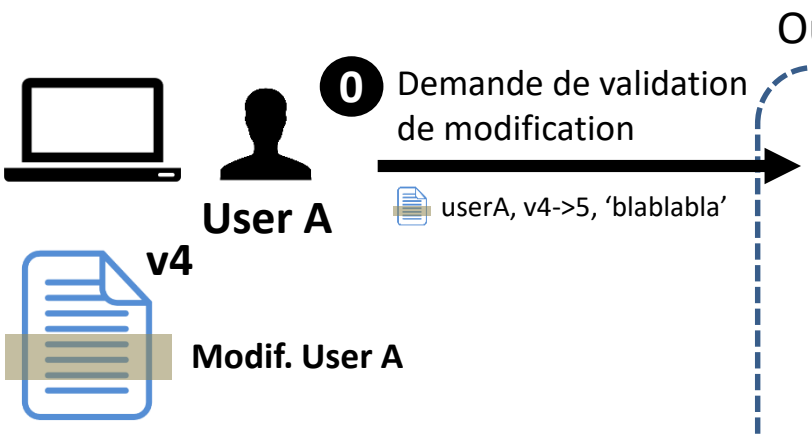
User A

v4

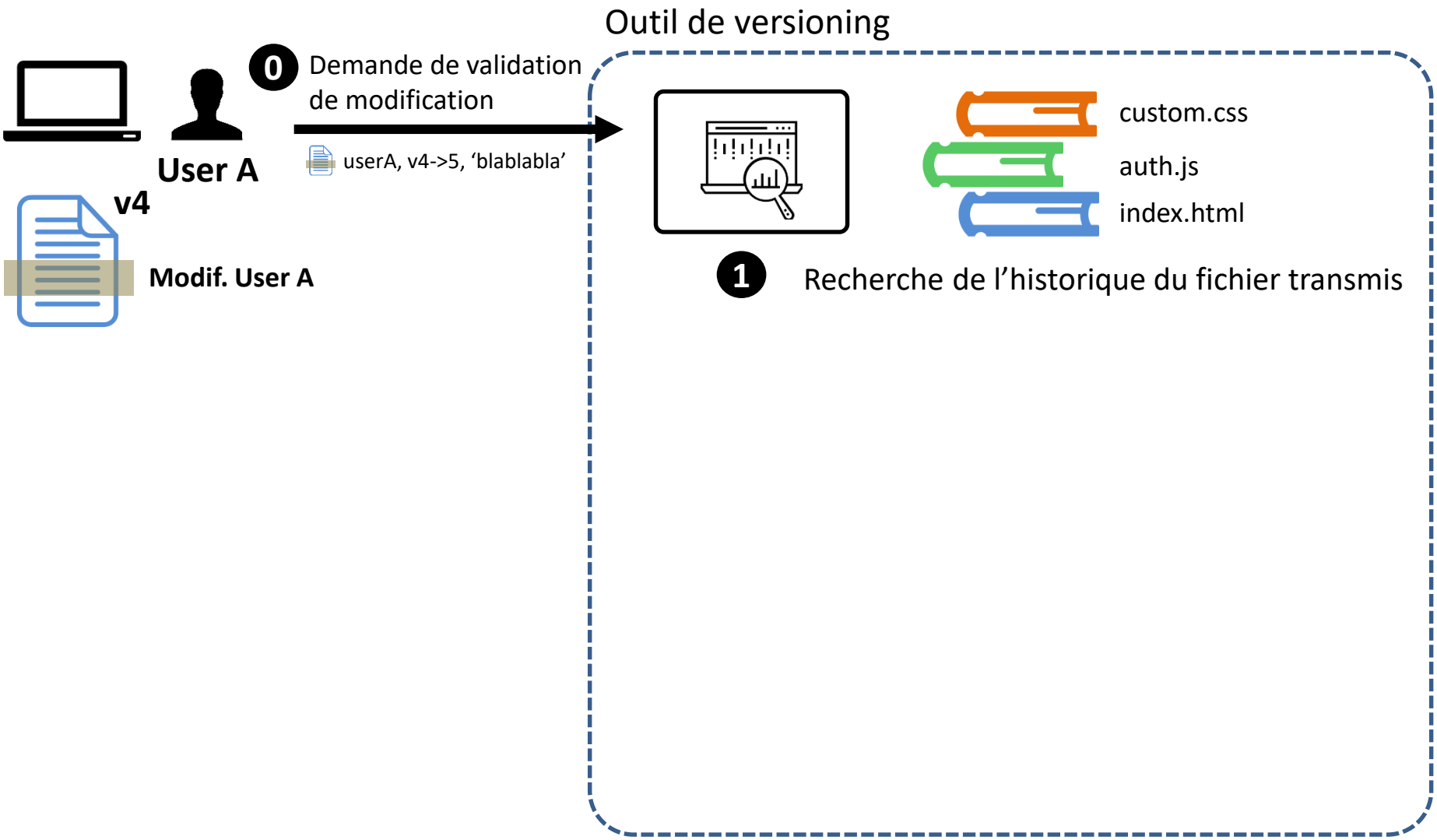
Modif. User A

Outil de versioning

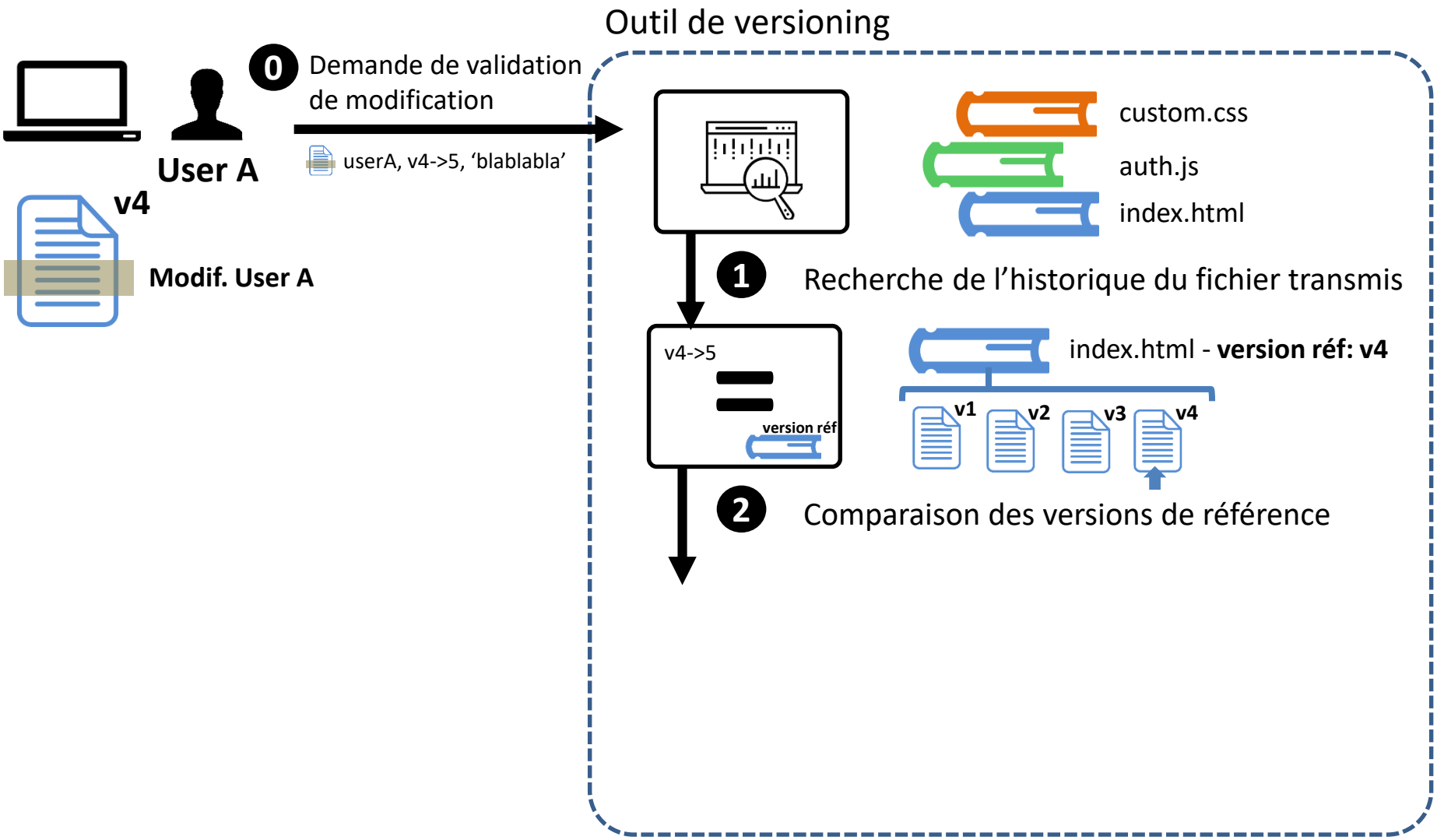
Fonctionnement



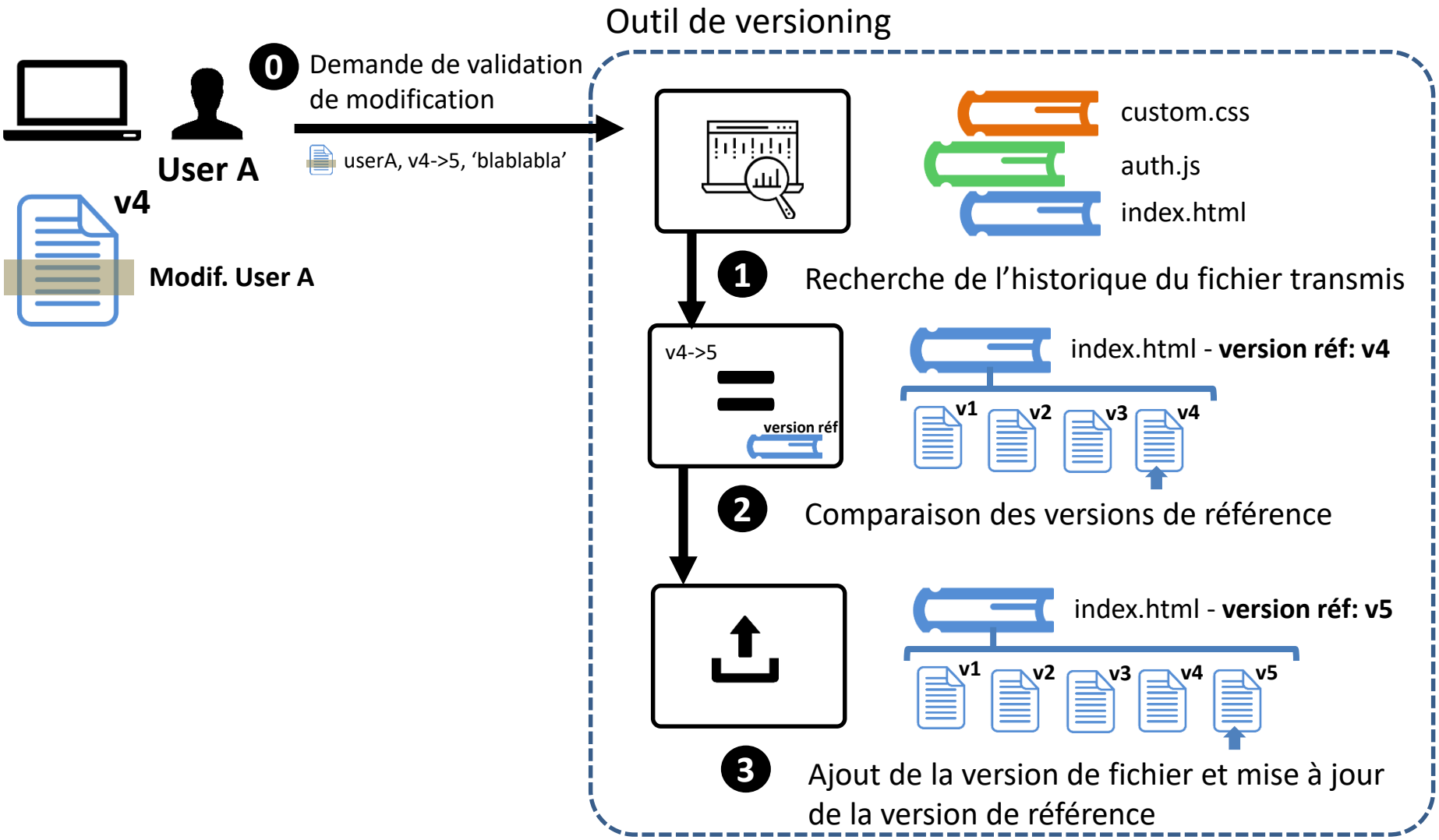
Fonctionnement



Fonctionnement



Fonctionnement



Les systèmes de gestion de version

- Beaucoup d'outils disponibles ! (gratuits et payants)

Outil	Type	Description	Projets qui l'utilisent
CVS	Centralisé	C'est un des plus anciens logiciels de gestion de versions. Bien qu'il fonctionne et soit encore utilisé pour certains projets, il est préférable d'utiliser SVN (souvent présenté comme son successeur) qui corrige un certain nombre de ses défauts, comme son incapacité à suivre les fichiers renommés par exemple.	OpenBSD…
SVN (Subversion)	Centralisé	Probablement l'outil le plus utilisé à l'heure actuelle. Il est assez simple d'utilisation, bien qu'il nécessite comme tous les outils du même type un certain temps d'adaptation. Il a l'avantage d'être bien intégré à Windows avec le programme Tortoise SVN, là où beaucoup d'autres logiciels s'utilisent surtout en ligne de commande dans la console. Il y a un tutoriel SVN sur OpenClassrooms.	Apache, Redmine, Struts…
Mercurial	Distribué	Plus récent, il est complet et puissant. Il est apparu quelques jours après le début du développement de Git et est d'ailleurs comparable à ce dernier sur bien des aspects. Vous trouverez un tutoriel sur Mercurial sur OpenClassrooms.	Mozilla, Python, OpenOffice.org…
Bazaar	Distribué	Un autre outil, complet et récent, comme Mercurial. Il est sponsorisé par Canonical, l'entreprise qui édite Ubuntu. Il se focalise sur la facilité d'utilisation et la flexibilité.	Ubuntu, MySQL, Inkscape…
Git	Distribué	Très puissant et récent, il a été créé par Linus Torvalds, qui est entre autres l'homme à l'origine de Linux. Il se distingue par sa rapidité et sa gestion des branches qui permettent de développer en parallèle de nouvelles fonctionnalités.	Kernel de Linux, Debian, VLC, Android, Gnome, Qt…

Source: https://openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git

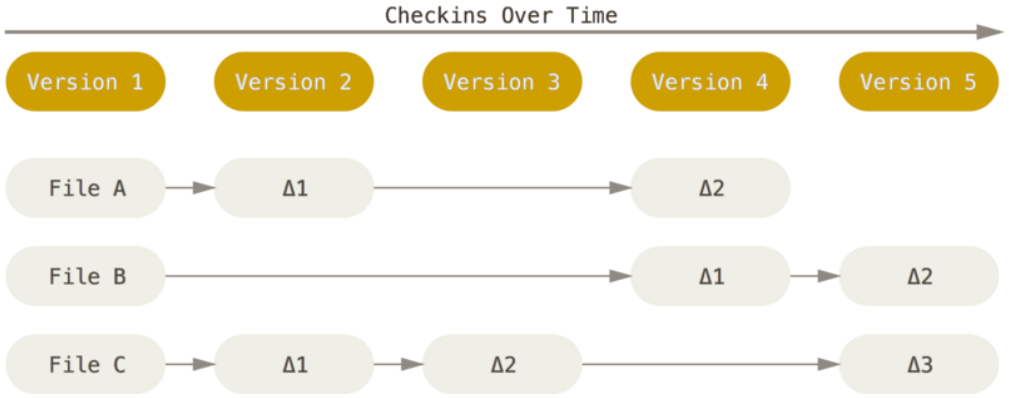
L'outil de versioning Git

Git

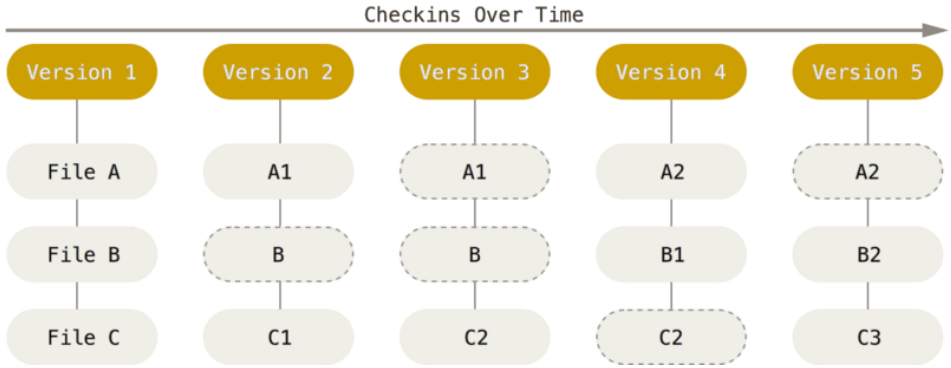
- Créé en 2005 par Linus Torvalds (le créateur de Linux)
- Objectifs
  - Gratuit
  - Système rapide
  - Conception simple
  - Usage possible de développement non linéaire (branches de développement en parallèle)
  - Pleinement distribué
  - Support de gros projet possible (e.g. kernel linux)
- Propriétés
  - « Snapshot and not delta »
  - Les opérations de git sont principalement locales
  - L'intégrité des composants est garantie (checksum)
  - Principalement que des fonctions d'ajout (très difficile de modifier des éléments validés avec git)
  - 3 états principaux pour des fichiers : **modified - staged - commited**

Pourquoi Git ?

- Fonctionnement de git : « Snapshot and not delta »



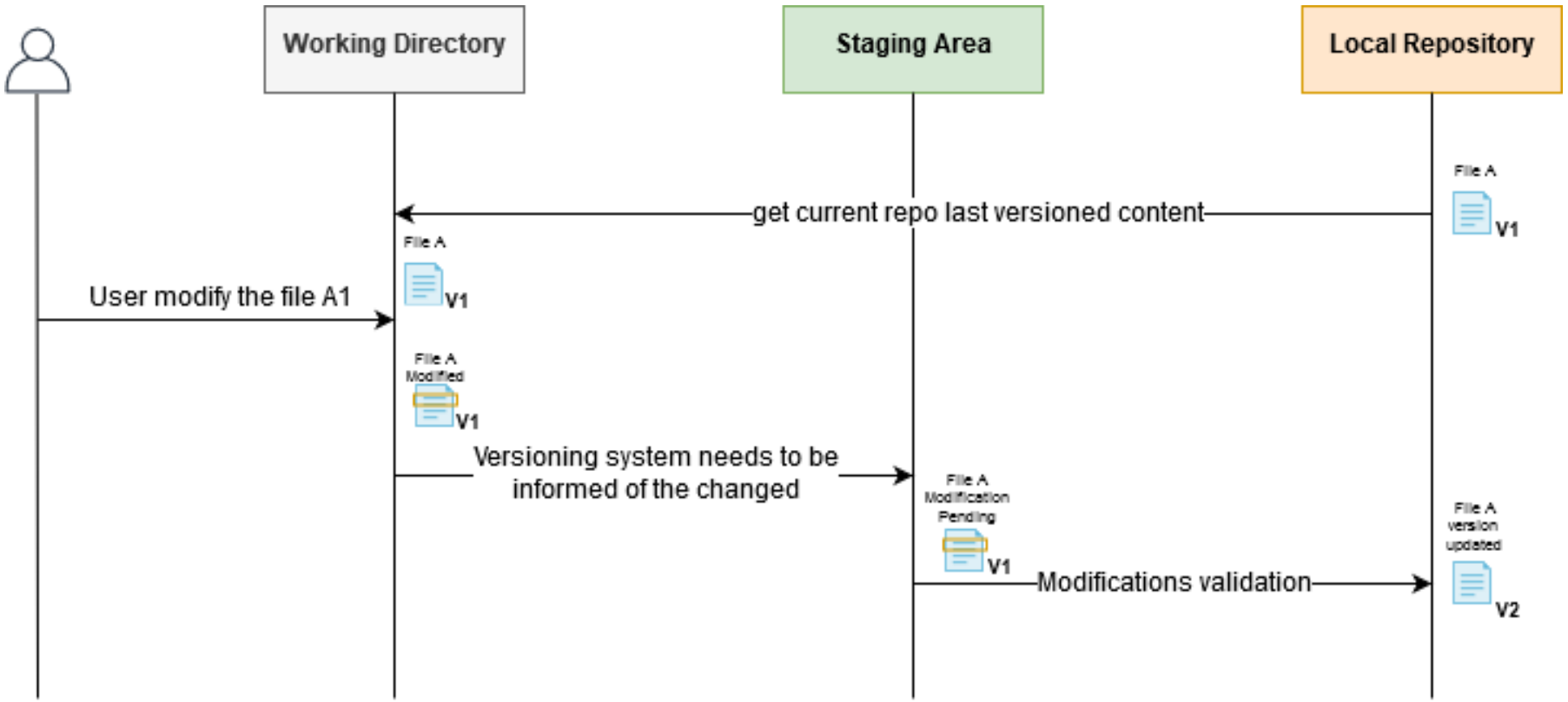
Deltas VS snapshots



https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

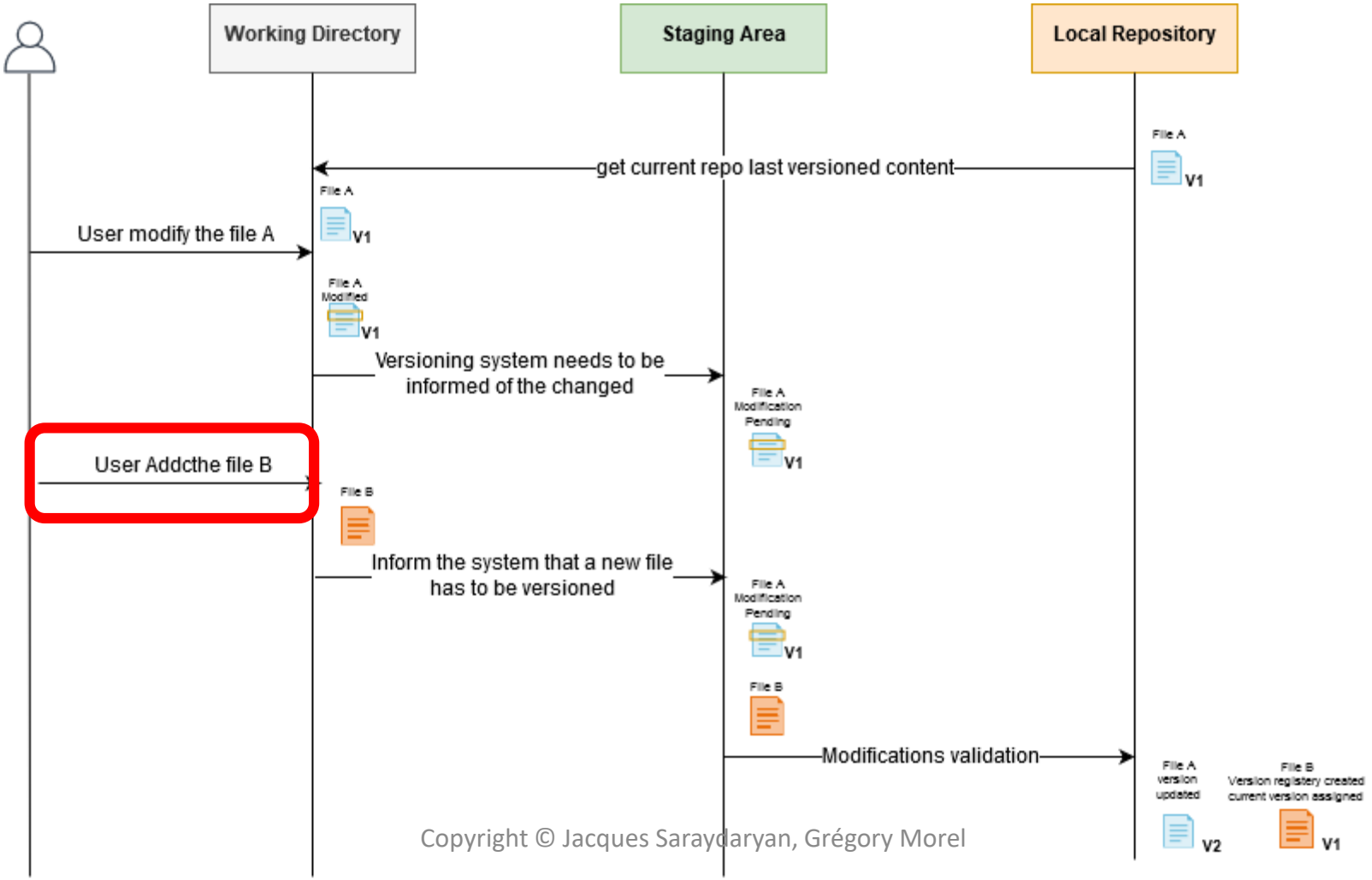
Workflow général d'usage

- 3 états de fichiers (*modified, staged, commited*)



Workflow général d'usage

- Les nouveaux fichiers peuvent aussi être versionnés



Usage de Base

- Première utilisation de git : définir son identité **`git config`**

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

`--global` = pour tous les projets

- Définir l'éditeur de texte par défaut pour gérer les fusions (ici, VS Code)

```
$ git config --global merge.tool code
```

- Afficher les propriétés de la configuration utilisée dans ce repo.

```
$ git config -l
    merge.tool=code
    user.name=John Doe
    user.email=johndoe@example.com
    core.repositoryformatversion=0
    core.filemode=true
    core.bare=false
    …
```

Usage de Base

- Initialiser un dépôt local **`git init`**

```
$ git init monDossier
```

- Partir d'un dépôt existant **`git clone`**

```
$ git clone https://gitlab.com/jsaraydaryan/test-ci.git
```

- Afficher l'état des travaux en cours **`git status`**

```
$ git status
     Sur la branche master
     Modifications qui seront validées :
       (utilisez "git restore --staged <fichier>..." pour désindexer)
          nouveau fichier : file3

     Modifications qui ne seront pas validées :
       (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
       (utilisez "git restore <fichier>..." pour annuler les modifications dans le
     répertoire de travail)
          modifié :         file1

     Fichiers non suivis:
       (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
          file2
```

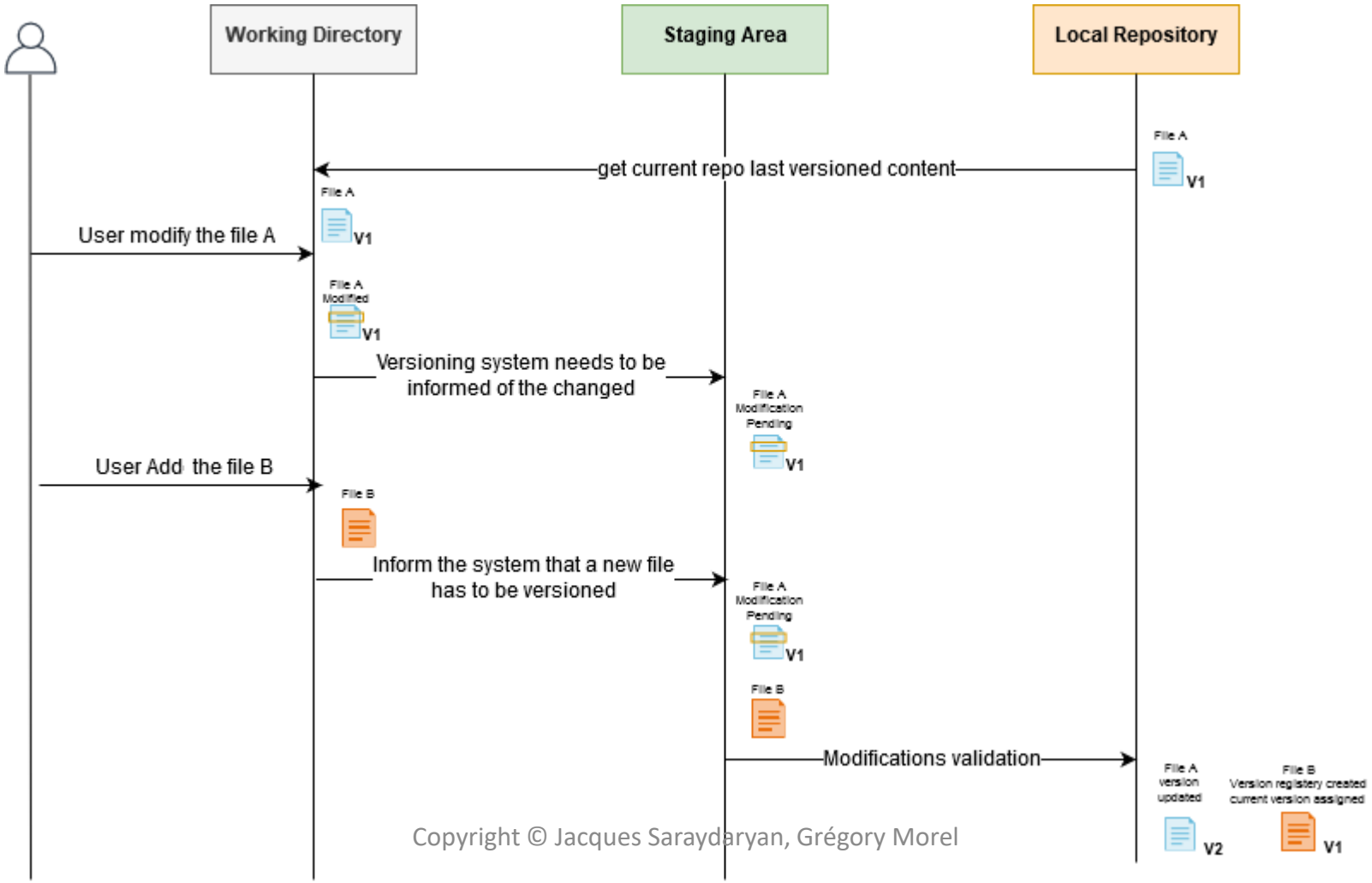
Usage de Base

- Historique des différents commits effectués **`git log`**

```
$ git log
    commit 8ad5d971dde365b6093f22e5921c7cf1b05e6530 (HEAD -> master)
    Author: John Doe <johndoe@example.com>
    Date:   Wed Sep 4 09:28:35 2019 -0400

        update file1 and add file3

    commit d3e8d4cc10d95cac2a643c28e9ec69b2e22f2fc0
    Author: John Doe <johndoe@example.com>
    Date:   Wed Sep 4 09:27:56 2019 -0400

        add file 2

    commit 75e6a57ceb9b7551d71a7599522d5f885c097263
    Author: John Doe <johndoe@example.com>
    Date:   Wed Sep 4 09:04:31 2019 -0400

        first file added
```

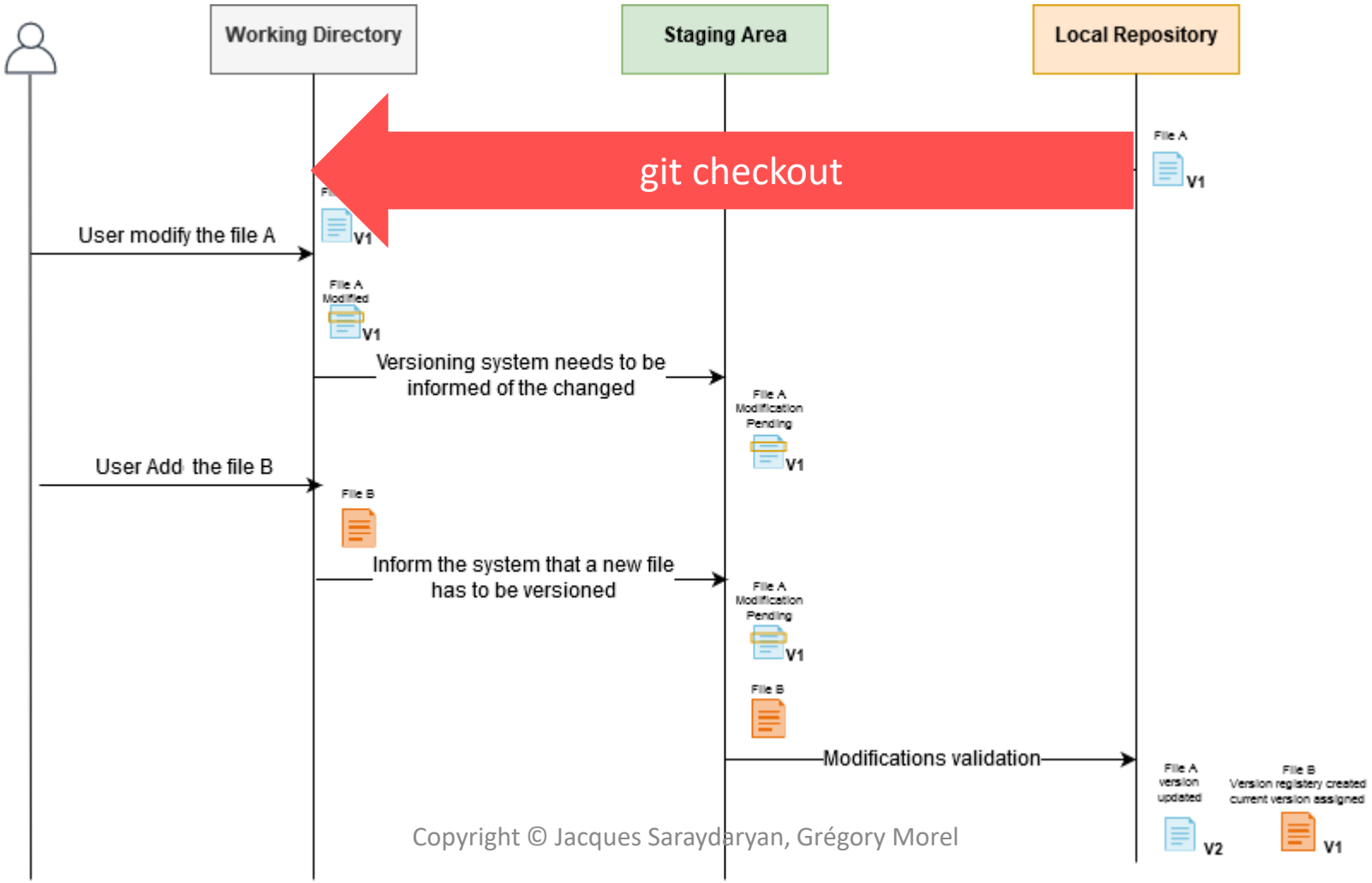
Workflow général d'usage

- Concrètement, avec git :



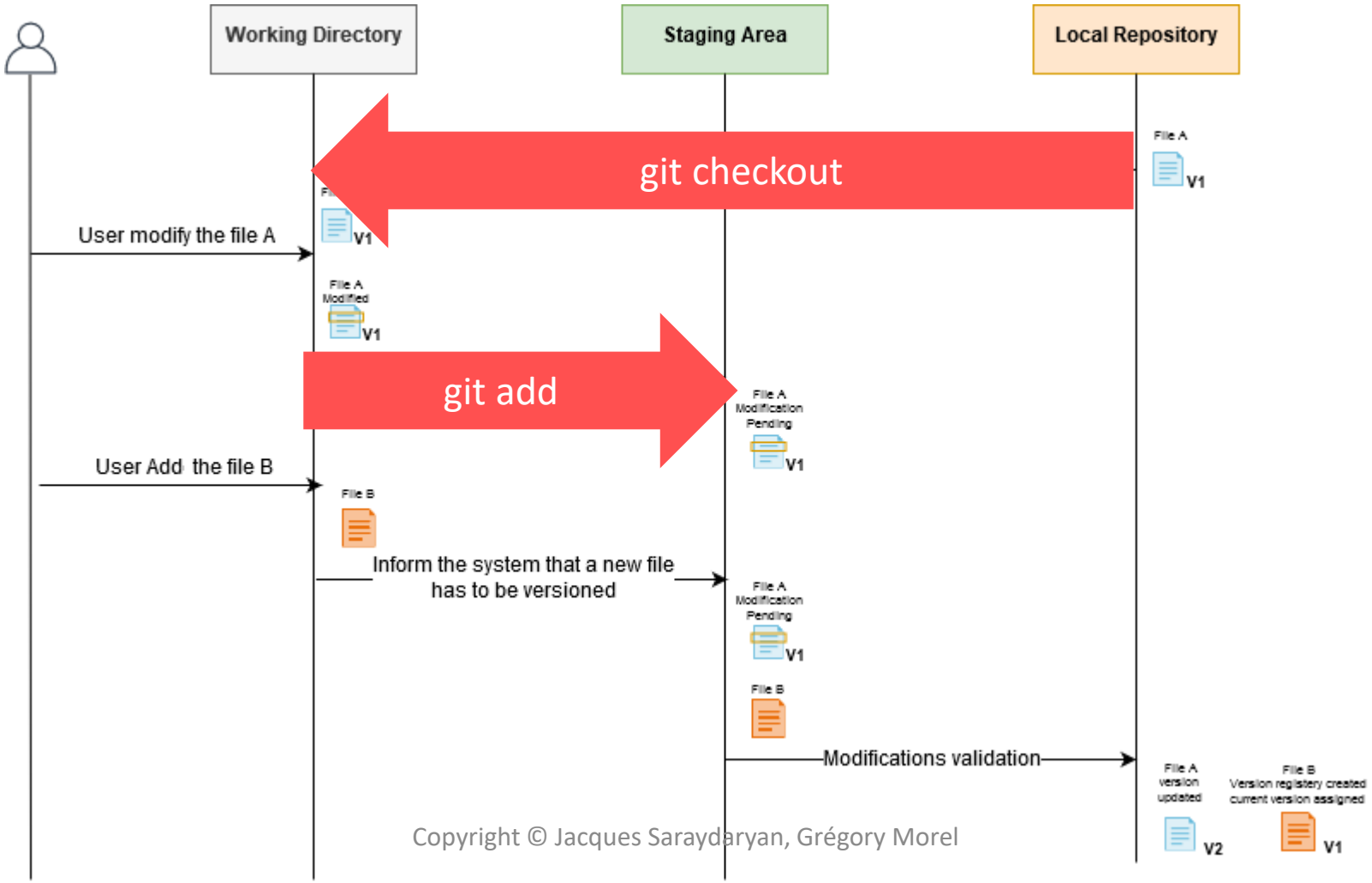
Workflow général d'usage

- Concrètement, avec git :



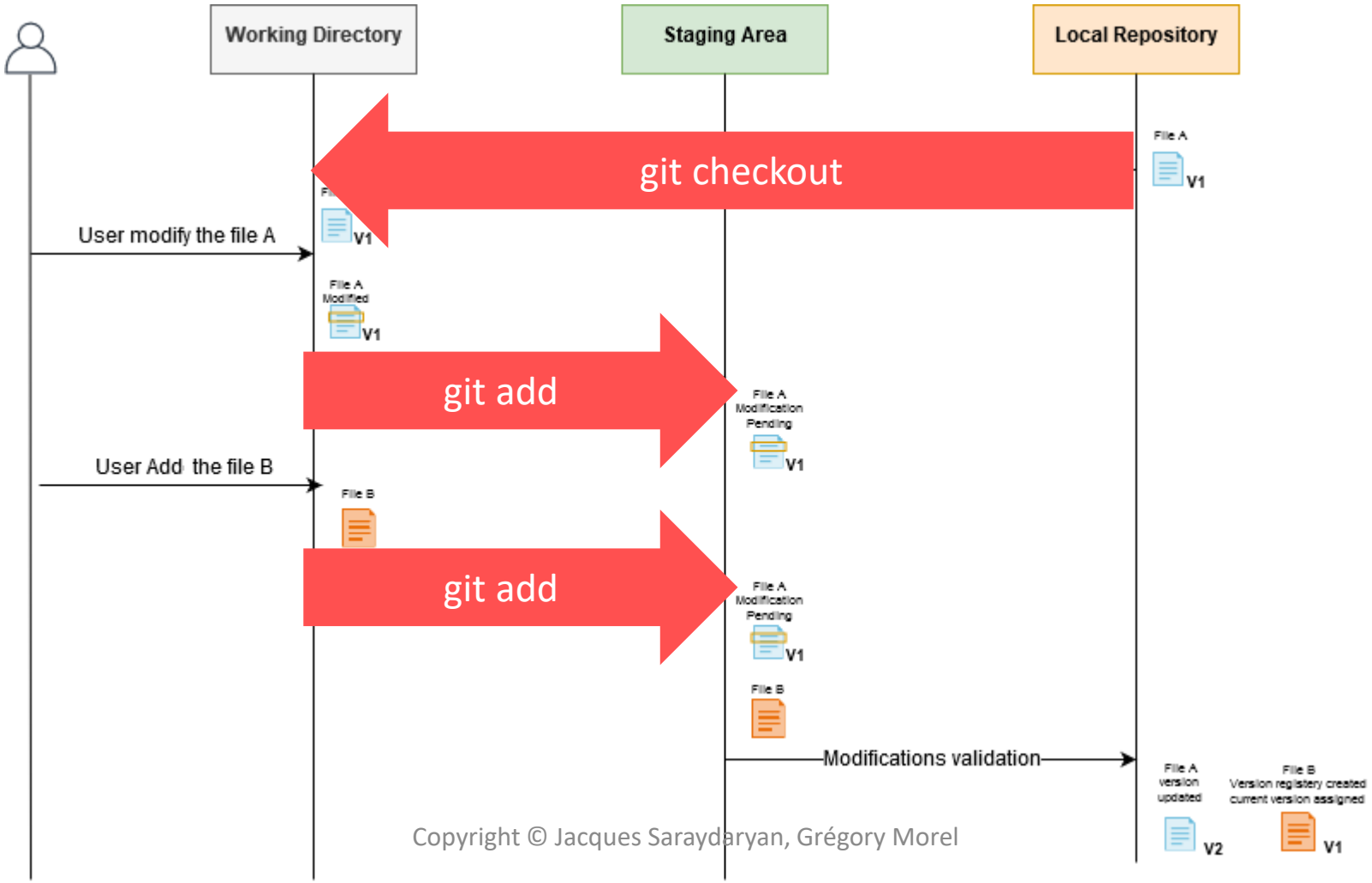
Workflow général d'usage

- Concrètement, avec git :



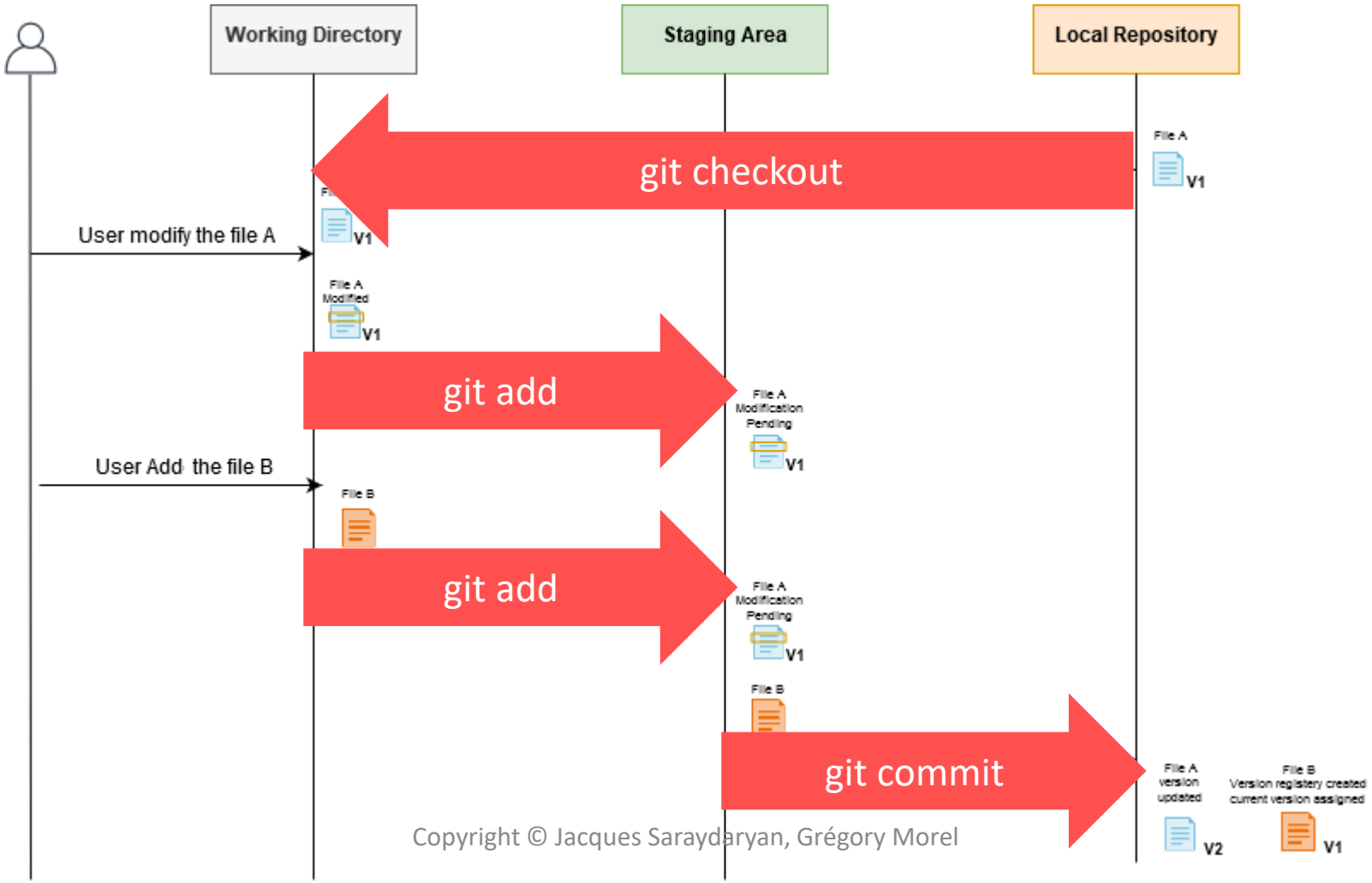
Workflow général d'usage

- Concrètement, avec git :



Workflow général d'usage

- Concrètement, avec git :



Usage de Base

- Ajout d'un fichier **`git add`**
  - Lorsqu'on souhaite versionner un *nouveau* fichier ou un fichier existant, il est nécessaire d'en informer git

```
$ git add myNewFile1
```

  - Plusieurs fichiers peuvent être ajoutés en même temps

```
$ git add myNewFile1 myNewFile2 myNewFile3
```

  - Tous les fichiers non versionnés peuvent être ajoutés d'un coup

```
$ git add --all
```

  - Une fois les fichiers ajoutés ils doivent être validés *(commités)*

```
$ git commit myNewFile1 myNewFile2 myNewFile3 –m " validation of new files"
```

Usage de Base

- Validation d'un fichier **`git commit`**
  - La modification d'un fichier existant nécessite une validation pour être versionnée

```
$ git commit file1 –m "my first file modification"
```

  - La validation de plusieurs fichiers est également possibles

```
$ git commit file1 file2 file3 –m " multi files commit operation"
```

  - Ainsi que la validation de l'ensemble des fichiers modifiés

```
$ git commit –a –m "commit operation of all files"
```

Usage de Base

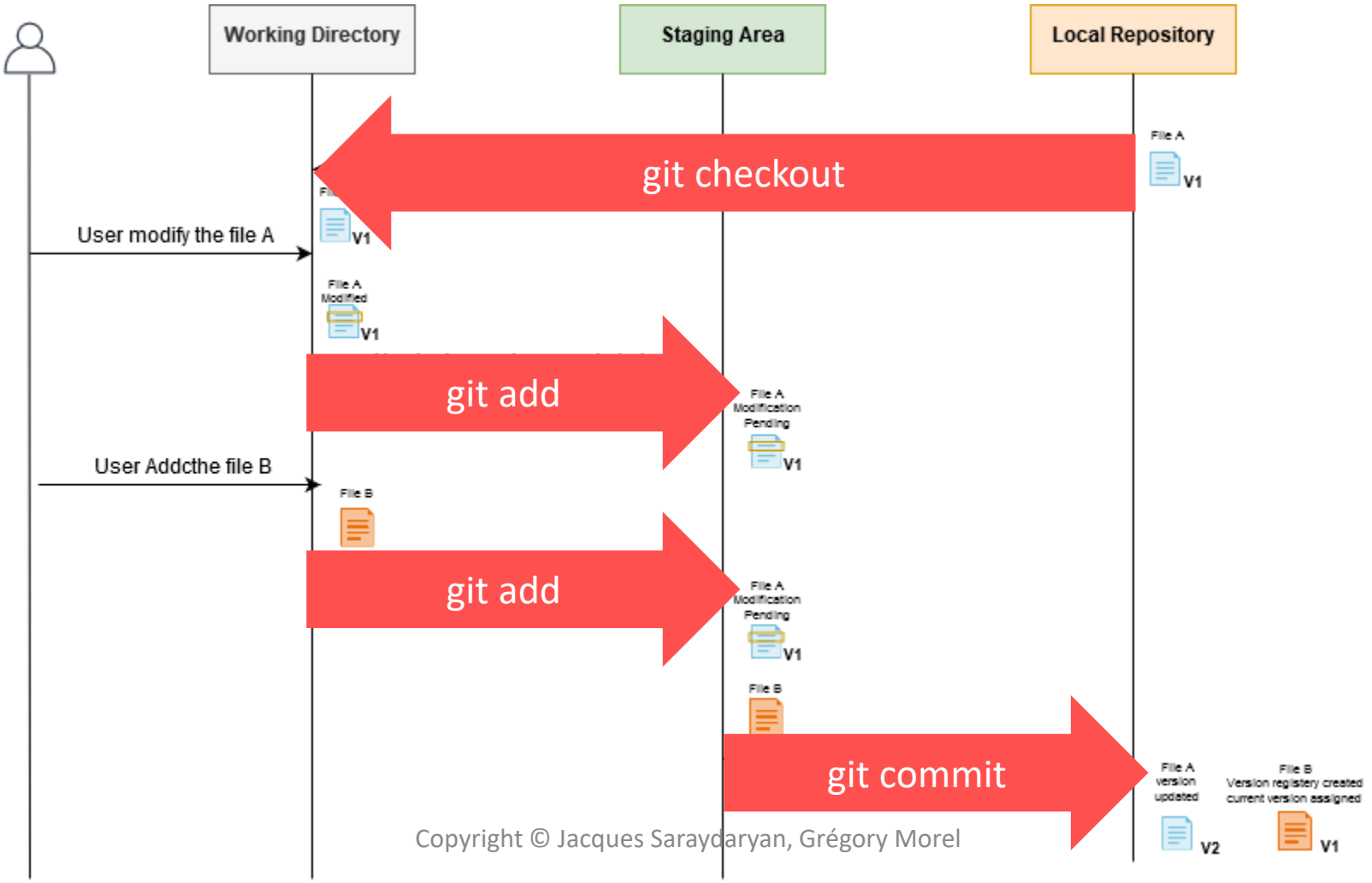
- Comparaison version courante/version validée **`git diff`**
  - Permet de visualiser les différences entre le fichier courant non validé et la version du fichier validée

```
$ git diff file1
```

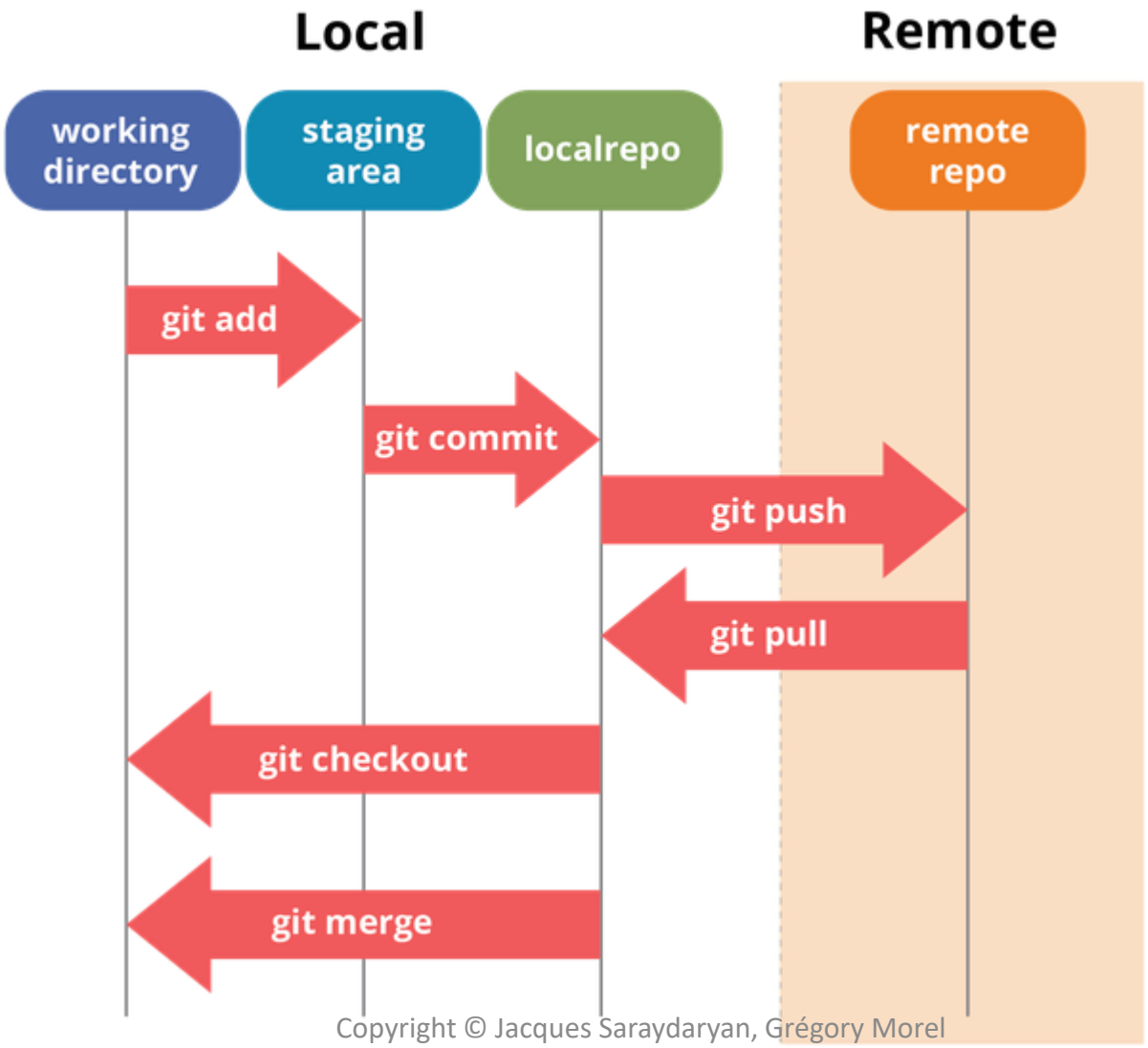
  - Exemple :

```
$ git diff Storage.java
index 781c9d1..4ff95cc 100644
--- a/src/com/ci/myShop/controller/Storage.java
+++ b/src/com/ci/myShop/controller/Storage.java
@@ -2,19 +2,21 @@ package com.ci.myShop.controller;
 import com.ci.myShop.model.Item;

 import java.util.HashMap;
-import java.util.Map;

+/**
+Class managing ItemList
+*/
 public class Storage {
+//Map of item
     private Map<Integer, Item> itemList;
```

Git permet de travailler en local…

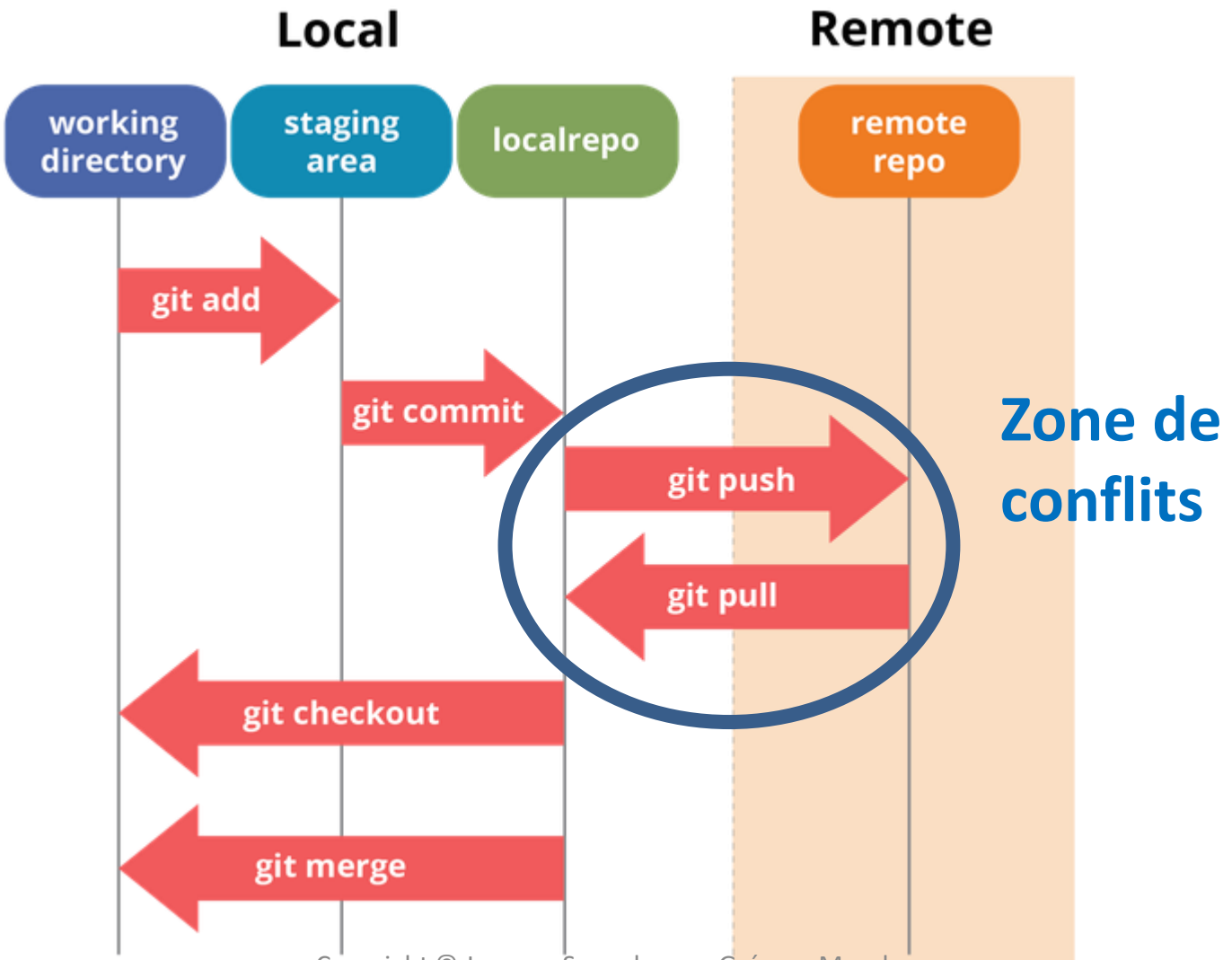


… ou avec un dépôt distant



https://www.quora.com/What-is-Git-and-why-should-I-use-it

… ou avec un dépôt distant



https://www.quora.com/What-is-Git-and-why-should-I-use-it

Usage de Base

- Récupération des données distantes **`git pull`**

```
$ git pull
     remote: Enumerating objects: 4, done.
     remote: Counting objects: 100% (4/4), done.
     remote: Compressing objects: 100% (2/2), done.
     remote: Total 3 (delta 0), reused 0 (delta 0)
     Unpacking objects: 100% (3/3), done.
     From https://gitlab.com/jsaraydaryan/test-ci
        cc2e658..afc40e7  master     -> origin/master
     Updating cc2e658..afc40e7
     Fast-forward
      fileA | 1 +
      1 file changed, 1 insertion(+)
      create mode 100644 fileA
```

- Envoi des données modifiées localement sur le serveur distant **`git push`**

```
$ git push
     Enumerating objects: 4, done.
     Counting objects: 100% (4/4), done.
     Compressing objects: 100% (2/2), done.
     Writing objects: 100% (3/3), 295 bytes | 98.00 KiB/s, done.
     Total 3 (delta 0), reused 0 (delta 0)
     To https://gitlab.com/jsaraydaryan/test-ci.git
        afc40e7..1a58ccf  master -> master
```

Usage de Base

- Modifier le *dernier* commit **`git commit --amend`**
  - Situation: erreur dans le commit, oubli d'éléments à valider

```
$ git commit --amend
```

  - E.g : typo dans un message du commit précédent plus oubli d'un fichier à ajouter

```
$ git commit –m 'my frt msg'
$ git add newFileX
$ git commit –-amend
```

```
my first message

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Sep 5 03:37:44 2019 -0400
#
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
# Changes to be committed:
#       modified:   fileA
#       new file:   newFileX
```

Usage de Base

- Annulation des modifications (working dir.) **`git checkout -- file`**

```
$ git checkout -- file
```

```
$ git status
     On branch master
     Your branch is ahead of 'origin/master' by 1 commit.
       (use "git push" to publish your local commits)
     Changes to be committed:
       (use "git reset HEAD <file>..." to unstage)
          modified:   fileB
     Changes not staged for commit:
       (use "git add <file>..." to update what will be committed)
       (use "git checkout -- <file>..." to discard changes in working directory)
          modified:   README.md
$ git checkout -- README.md
$ git status
     On branch master
     Your branch is ahead of 'origin/master' by 1 commit.
       (use "git push" to publish your local commits)
     Changes to be committed:
       (use "git reset HEAD <file>..." to unstage)
          modified:   fileB
```

Usage de Base

- Annulation des modifications (locales / commit privé) **`git reset HEAD`**
  - Annulation du dernier commit (soft)

```
$ git reset HEAD
```

Usage de Base

- Annulation des modifications (locales / commit privé) **`git reset HEAD`**
  - Annulation du dernier commit (soft)

```
$ git reset HEAD
```

  - Annulation d'autres commit

    `Head` : dernier commit
    `Head^` : avant dernier commit
    `Head^^` : avant avant dernier commit
    `Head~2` : avant avant dernier commit (autre notation)
    `D6d98923868578a7f38dea79833b56d0326fcba1` : numéro commit (précis)
    `D6d9892` : numéro commit (notation courte)

Usage de Base

- Annulation des modifications (locales / commit privé) **`git reset HEAD`**
  - Annulation du dernier commit (soft)

```
$ git reset HEAD
```

  - Annulation d'autres commit

    `Head` : dernier commit
    `Head^` : avant dernier commit
    `Head^^` : avant avant dernier commit
    `Head~2` : avant avant dernier commit (autre notation)
    `D6d98923868578a7f38dea79833b56d0326fcba1` : numéro commit (précis)
    `D6d9892` : numéro commit (notation courte)

➡ **Seuls les *commits* sont annulés, vos fichiers restent les mêmes**

  - Annulation du dernier commit (hard)

```
$ git reset --hard HEAD /!\ Annule les commits et perd tous les changements
```

Usage de Base

- Annulation des modifications (distantes / commit public) **`git revert`**
  - Annulation d'un commit publié sur le dépôt distant

```
$ git revert 5478cc1
```

  - Attention ! Remettre votre dossier de travail dans l'état du commit `5478cc1`

En réalité, crée une **nouvelle version** qui efface les modifications effectuées depuis la version spécifiée

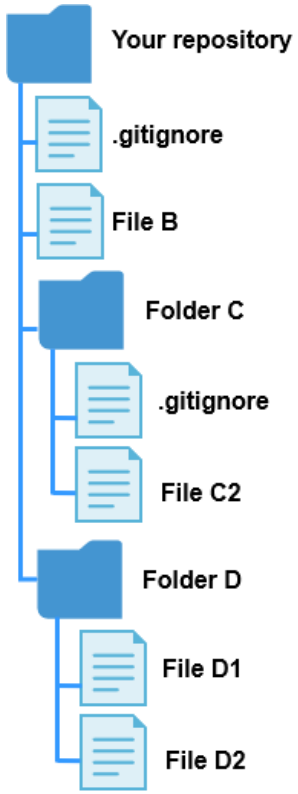
➡ **Les pushs sont définitifs, les modifications publiques sont enregistrées et ne peuvent pas être supprimées**

Ignorer certains fichiers

- .gitignore
  - Permet de sélectionner les fichiers qui ne seront pas suivis par Git
  - Chaque ligne définit un modèle (*pattern*) de format de fichiers/répertoires
  - Les références aux objets sont relatives à l'emplacement du fichier .gitignore

.gitignore

```
#ignore conf files
FileB

#ignore all the directory
FolderC/

#ignore a file into a directory
/FolderD/FileD1

#ignore all file with suffix .txt
*.txt
```

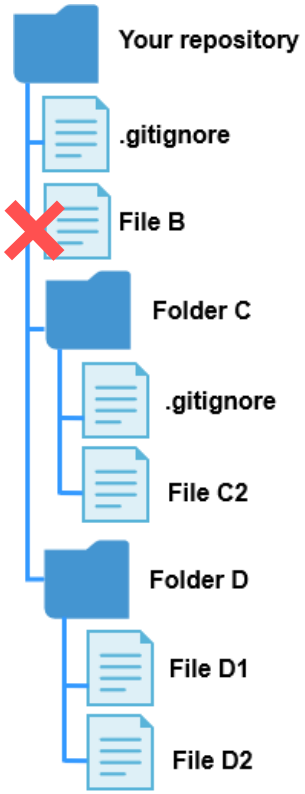


Ignorer certains fichiers

- .gitignore
  - Permet de sélectionner les fichiers qui ne seront pas suivis par Git
  - Chaque ligne définit un modèle (*pattern*) de format de fichiers/répertoires
  - Les références aux objets sont relatives à l'emplacement du fichier .gitignore

.gitignore

```
#ignore conf files
FileB

#ignore all the directory
FolderC/

#ignore a file into a directory
/FolderD/FileD1

#ignore all file with suffix .txt
*.txt
```

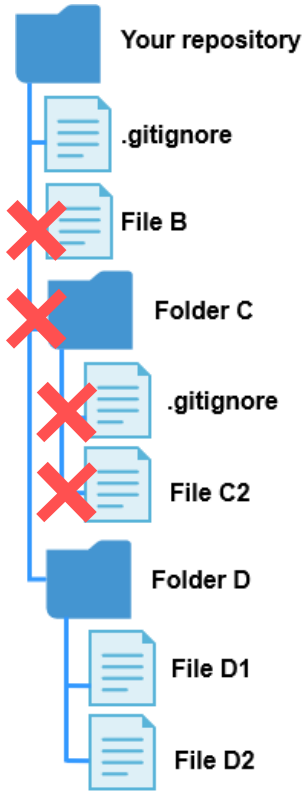


Ignorer certains fichiers

- .gitignore
  - Permet de sélectionner les fichiers qui ne seront pas suivis par Git
  - Chaque ligne définit un modèle (*pattern*) de format de fichiers/répertoires
  - Les références aux objets sont relatives à l'emplacement du fichier .gitignore

.gitignore

```
#ignore conf files
FileB

#ignore all the directory
FolderC/

#ignore a file into a directory
/FolderD/FileD1

#ignore all file with suffix .txt
*.txt
```

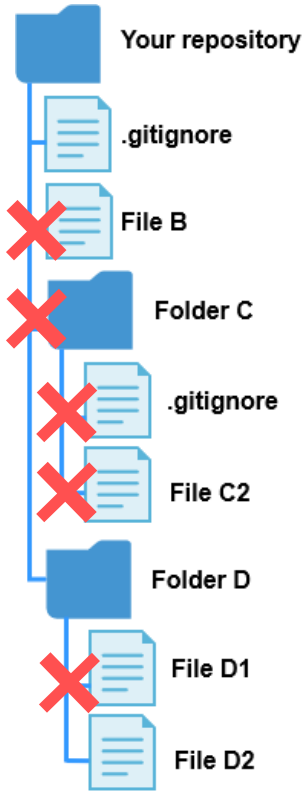


Ignorer certains fichiers

- .gitignore
  - Permet de sélectionner les fichiers qui ne seront pas suivis par Git
  - Chaque ligne définit un modèle (*pattern*) de format de fichiers/répertoires
  - Les références aux objets sont relatives à l'emplacement du fichier .gitignore

.gitignore

```
#ignore conf files
FileB

#ignore all the directory
FolderC/

#ignore a file into a directory
/FolderD/FileD1

#ignore all file with suffix .txt
*.txt
```



Ignorer certains fichiers

- .gitignore
  - Les modèles (*patterns*)

    < blanc> : pas interprété (ligne), interprété si échappé à l'aide d'un \ (e.g "\ ")

    # : commentaire pour le fichier

    ! : prend l'inverse du pattern

    / : délimiteur de répertoire,
      - si au début (/index.html), renvoie à la position du fichier gitignore
      - si à la fin (target/) indique un répertoire

    * : n'importe quelle suite de caractères (sauf /)

    ? : un seul caractère (sauf /)

    **/rep : tous les répertoires nommés rep, quel que soit leur emplacement

    rep/** : l'ensemble du contenu du dossier rep, **y compris les sous-dossiers**

Résolution des conflits

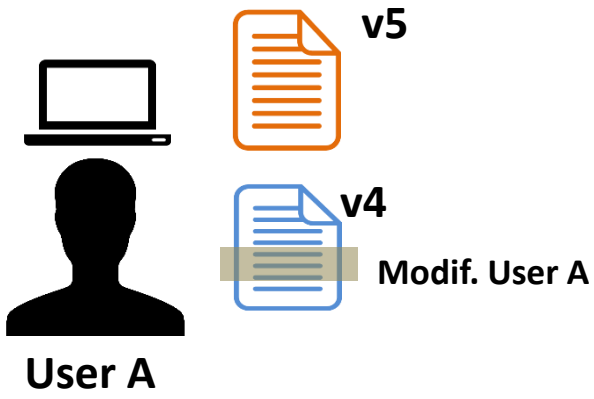
- Conflit

Un conflit survient lorsque 2 utilisateurs ont modifié la même zone d'un fichier et que le système de versioning n'est pas capable de fusionner ces éléments. Il n'est alors pas possible d'appliquer les modifications sur le serveur.

Résolution des conflits

- Conflit

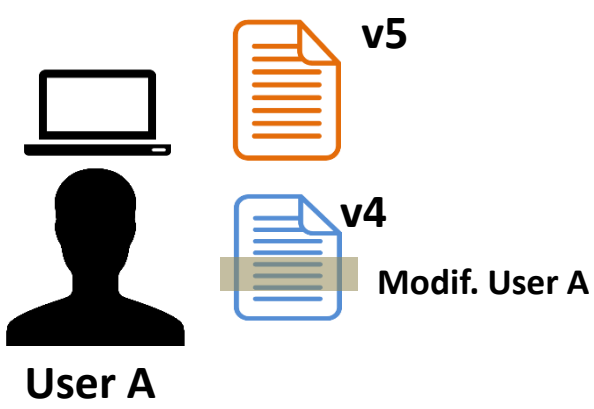
Un conflit survient lorsque 2 utilisateurs ont modifié la même zone d'un fichier et que le système de versioning n'est pas capable de fusionner ces éléments. Il n'est alors pas possible d'appliquer les modifications sur le serveur.

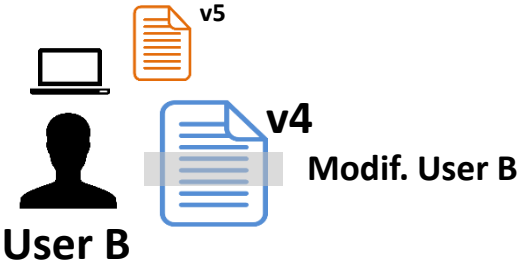


Résolution des conflits

- Conflit

Un conflit survient lorsque 2 utilisateurs ont modifié la même zone d'un fichier et que le système de versioning n'est pas capable de fusionner ces éléments. Il n'est alors pas possible d'appliquer les modifications sur le serveur.

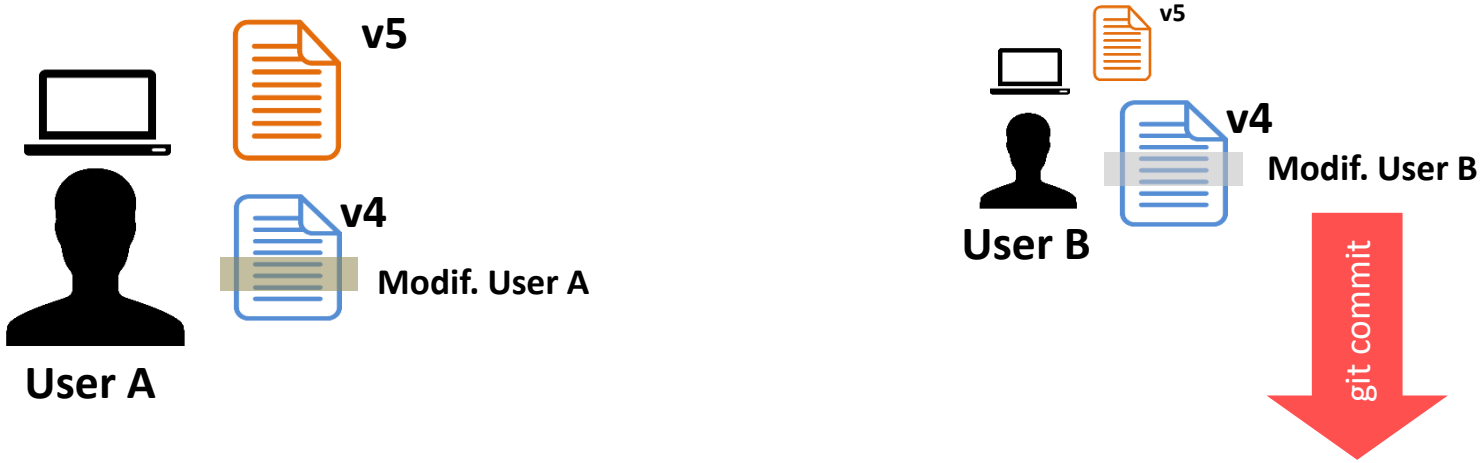




Résolution des conflits

- Conflit

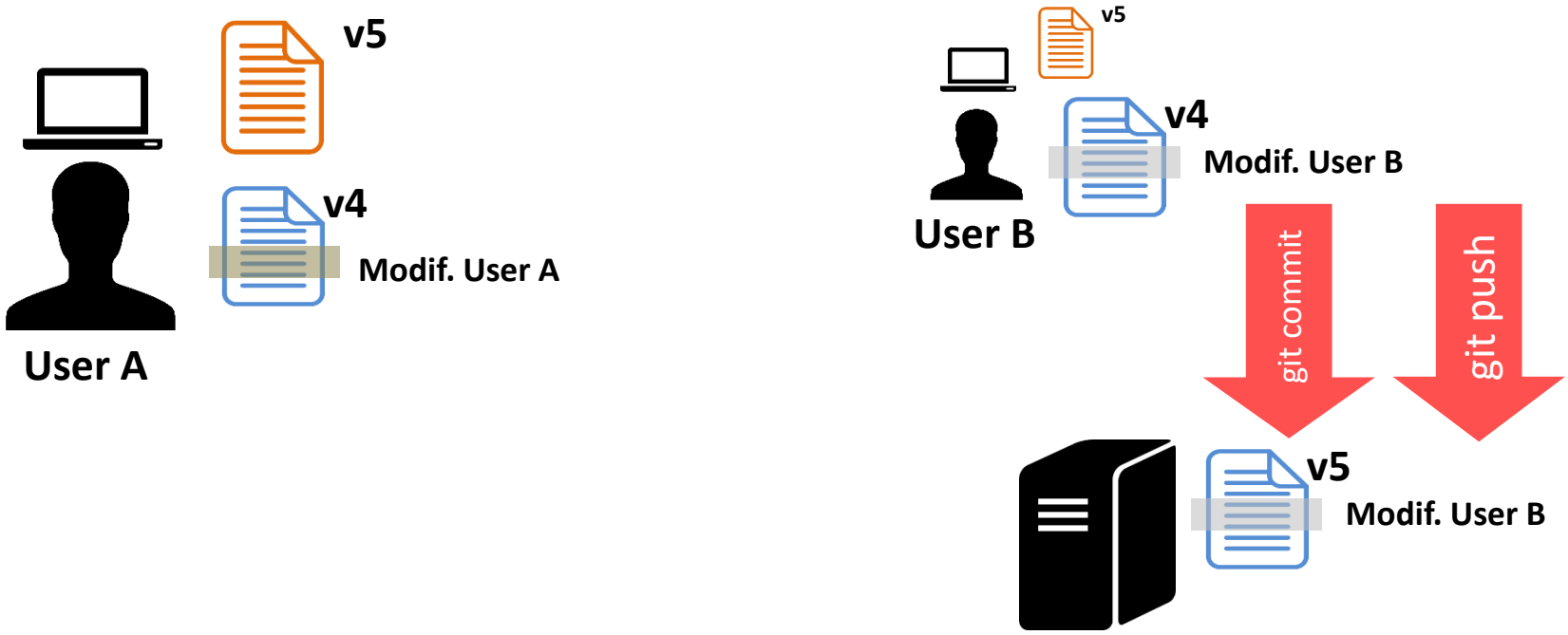
Un conflit survient lorsque 2 utilisateurs ont modifié la même zone d'un fichier et que le système de versioning n'est pas capable de fusionner ces éléments. Il n'est alors pas possible d'appliquer les modifications sur le serveur.



Résolution des conflits

- Conflit

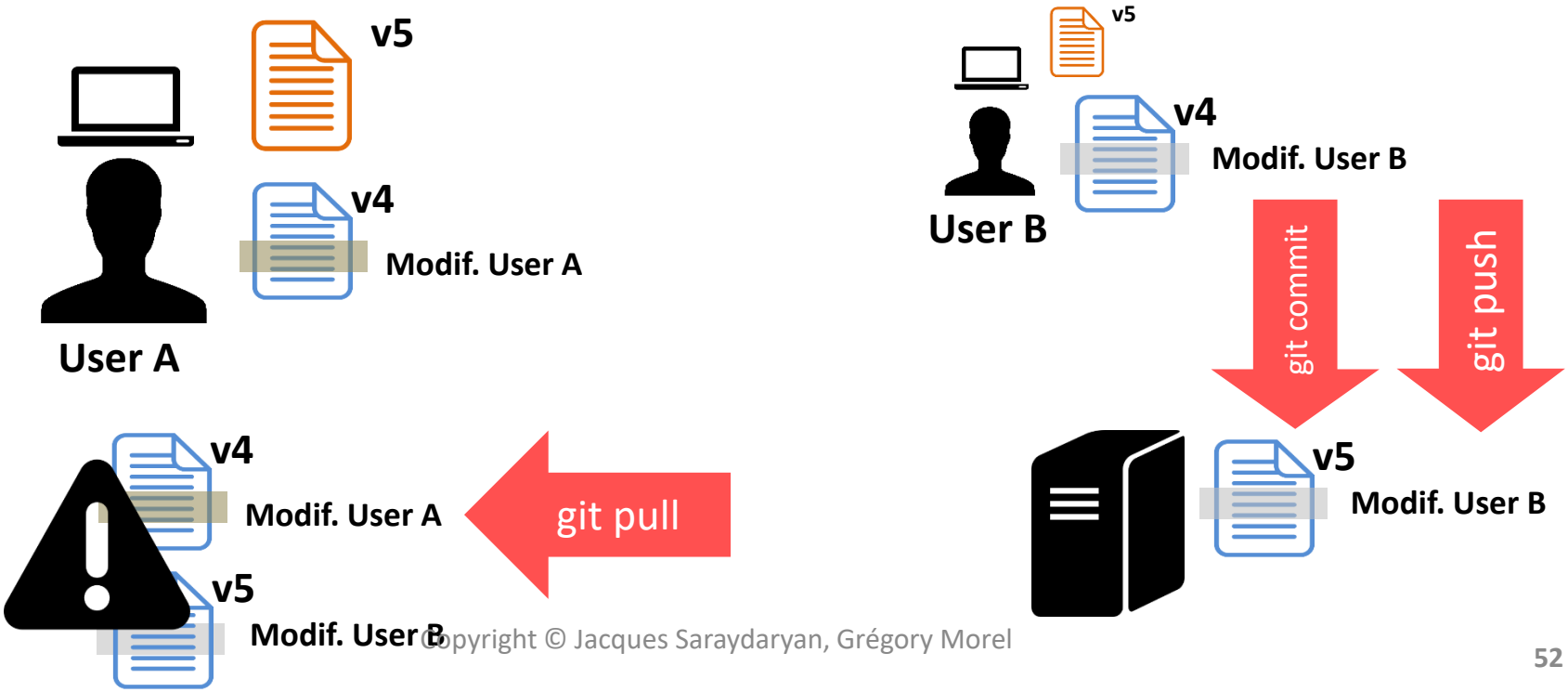
Un conflit survient lorsque 2 utilisateurs ont modifié la même zone d'un fichier et que le système de versioning n'est pas capable de fusionner ces éléments. Il n'est alors pas possible d'appliquer les modifications sur le serveur.



Résolution des conflits

- Conflit

Un conflit survient lorsque 2 utilisateurs ont modifié la même zone d'un fichier et que le système de versioning n'est pas capable de fusionner ces éléments. Il n'est alors pas possible d'appliquer les modifications sur le serveur.



Résolution des conflits

- Résolution du conflit
  - Exemple

User A

```
Hello the is the fileToMerge
I am A i modified the file here
This is the text of end of the
file
```

User B

```
Hello the is the fileToMerge
I am b and i modified the file
in the middle
This is the text of end of the
file
```

```
$ git commit fileToMerge “B modified file”
$ git push
```

```
$ git commit fileToMerge “A modified file”
$ git push
```

```
hint: Updates were rejected because the remote
contains work that you do
hint: not have locally. This is usually caused by
another repository pushing
hint: to the same ref. You may want to first
integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
```

Résolution des conflits

- Résolution du conflit
  - Exemple (suite)

```
$ git pull

From https://gitlab.com/jsaraydaryan/test-ci
   a26e7d4..6d7d209  master     -> origin/master
Auto-merging fileTomerged
CONFLICT (add/add): Merge conflict in fileTomerge
Automatic merge failed; fix conflicts and then commit the result.
```

fileToMerge

```
Hello the is the fileToMerge
<<<<<<< HEAD
I am A i modified the file here
=======
I am b and i modified the file in
the middle
>>>>>>>
6d7d209bf5cfcf05ff4d90790bc8f56fe64
68a00
This is the text of end of the file
```

Résolution des conflits

- Résolution du conflit
  - Exemple (suite)

```
$ git pull

From https://gitlab.com/jsaraydaryan/test-ci
   a26e7d4..6d7d209  master     -> origin/master
Auto-merging fileTomerged
CONFLICT (add/add): Merge conflict in fileTomerge
Automatic merge failed; fix conflicts and then commit the result.
```

fileToMerge

Elément commun

```
Hello the is the fileToMerge
<<<<<<< HEAD
I am A i modified the file here
=======
I am b and i modified the file in
the middle
>>>>>>>
6d7d209bf5cfcf05ff4d90790bc8f56fe64
68a00
This is the text of end of the file
```

Elément commun

Résolution des conflits

- Résolution du conflit
  - Exemple (suite)

```
$ git pull

From https://gitlab.com/jsaraydaryan/test-ci
   a26e7d4..6d7d209  master     -> origin/master
Auto-merging fileTomerged
CONFLICT (add/add): Merge conflict in fileTomerge
Automatic merge failed; fix conflicts and then commit the result.
```

fileToMerge

Elément commun

Modification locale (HEAD)

Elément commun

```
Hello the is the fileToMerge
<<<<<<< HEAD
I am A i modified the file here
=======
I am b and i modified the file in
the middle
>>>>>>>
6d7d209bf5cfcf05ff4d90790bc8f56fe64
68a00
This is the text of end of the file
```

Résolution des conflits

- Résolution du conflit
  - Exemple (suite)

```
$ git pull

From https://gitlab.com/jsaraydaryan/test-ci
   a26e7d4..6d7d209  master     -> origin/master
Auto-merging fileTomerged
CONFLICT (add/add): Merge conflict in fileTomerge
Automatic merge failed; fix conflicts and then commit the result.
```

fileToMerge

Elément commun

Modification locale (HEAD)

```
Hello the is the fileToMerge
<<<<<<< HEAD
I am A i modified the file here
=======
I am b and i modified the file in the middle
>>>>>>>
6d7d209bf5cfcf05ff4d90790bc8f56fe6468a00
This is the text of end of the file
```

Modification remote présente sur le serveur (commit num: 6d7d209..)

Elément commun

Résolution des conflits

- Résolution du conflit
  - Exemple (suite)

fileToMerge

```
Hello the is the fileToMerge
<<<<<<< HEAD
I am A i modified the file here
=======
I am b and i modified the file in
the middle
>>>>>>>
6d7d209bf5efef05ff4d90790be8f56fe6468a00
This is the text of end of the file
```

fileToMerge

```
Hello the is the fileToMerge
I am A i modified the file here
This is the text of end of the file
```

```
$ git add fileToMerge
$ git commit -m "merge the current file, with my modification“
[master 28bb9d3] merge the current file, with my modification
$ git push
```

Résolution des conflits

```
tp@tp-VM:~/project$ git clone https://gitlab.com/jsaraydaryan/test-ci.git
```

Travailler avec des branches

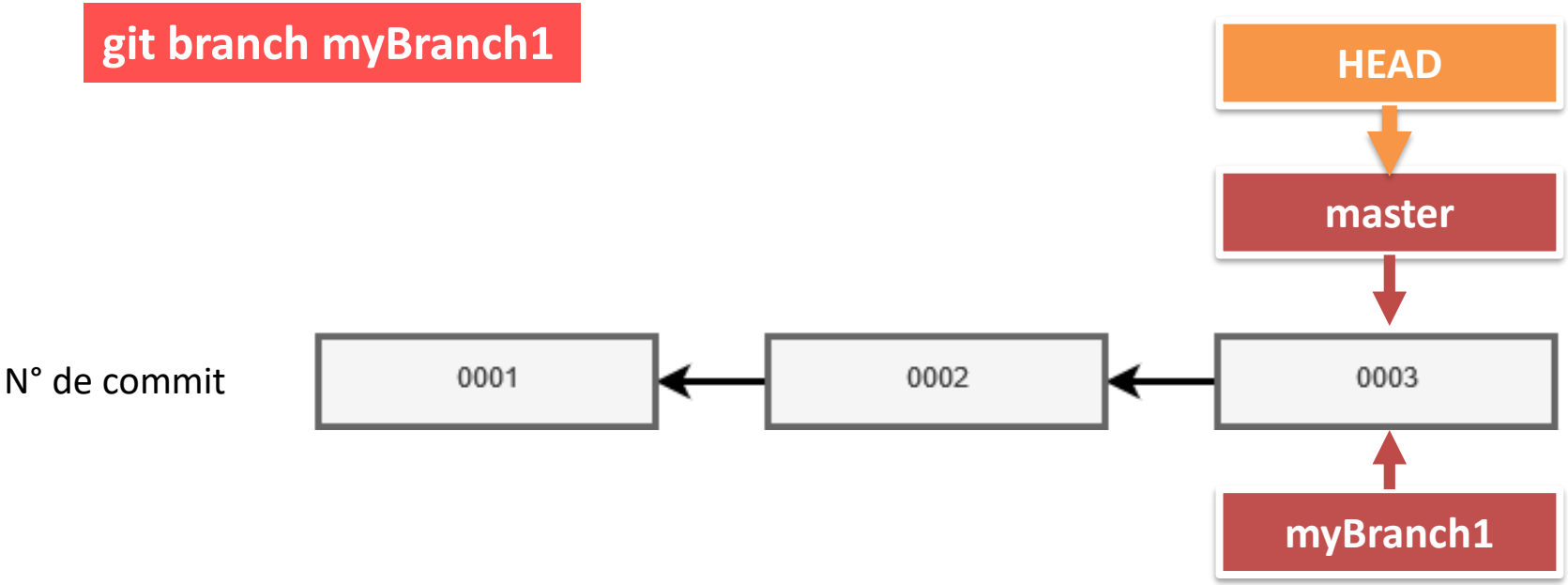
Workflow général d'usage

- Gestion des branches

Workflow général d'usage

- Création d'une branche

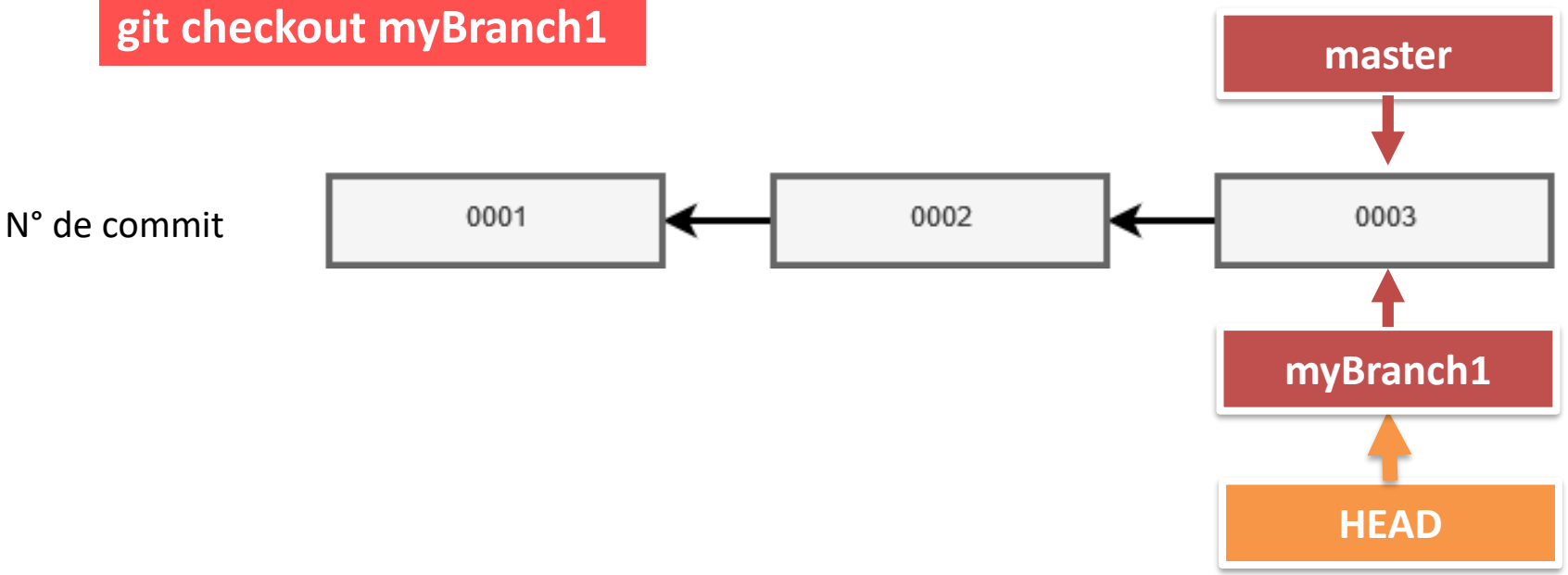
git branch myBranch1



Workflow général d'usage

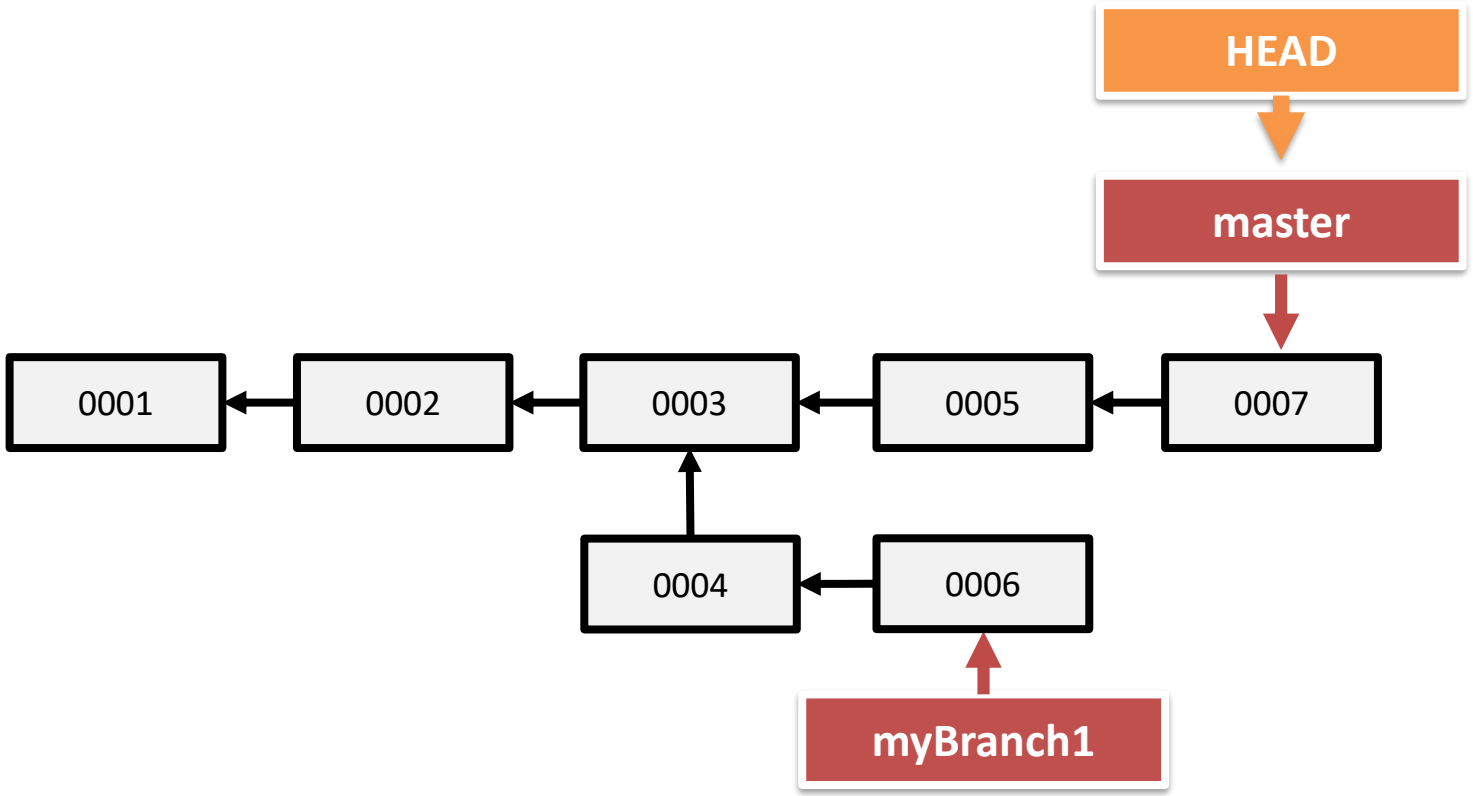
- Changement de branche

git checkout myBranch1



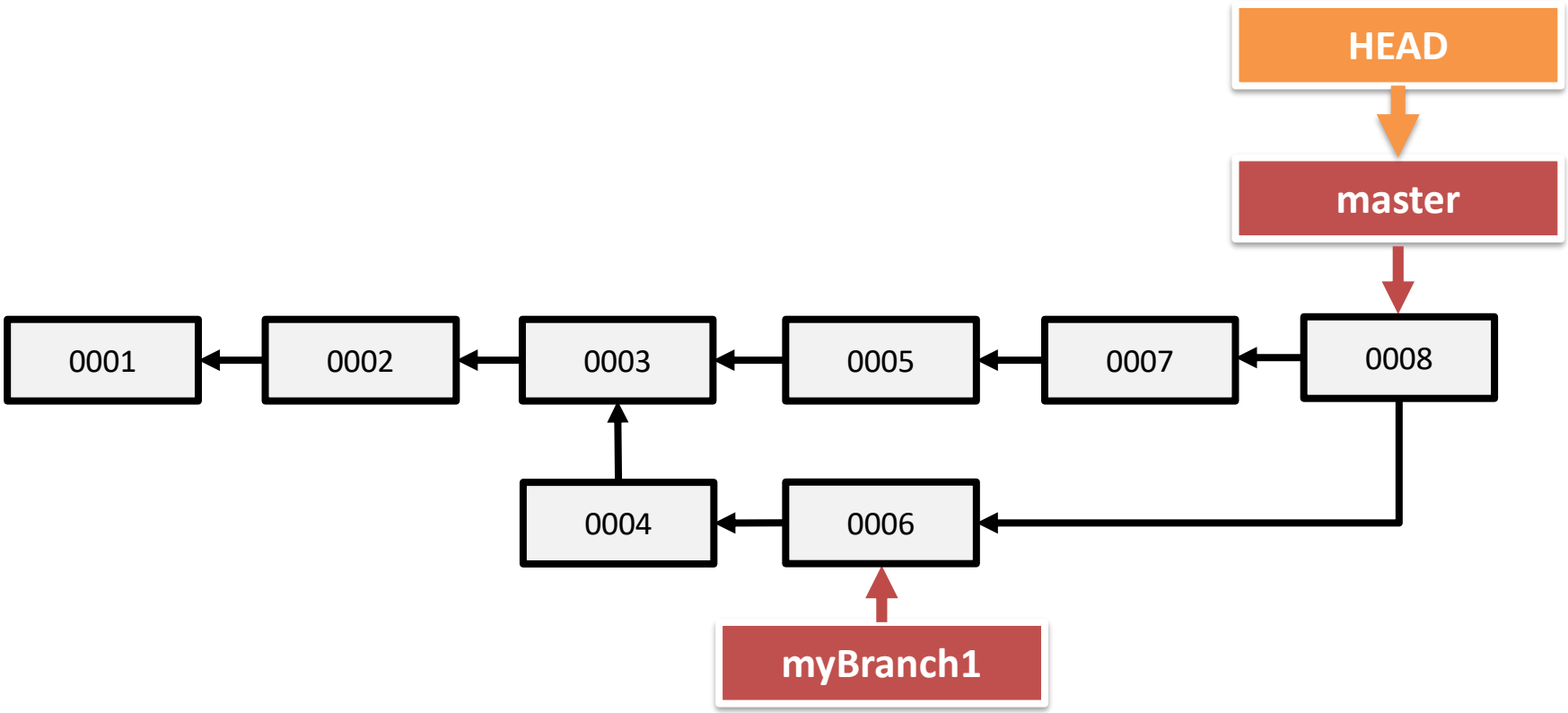
Workflow général d'usage

- Développement en parallèle



Workflow général d'usage

- Fusion de branches



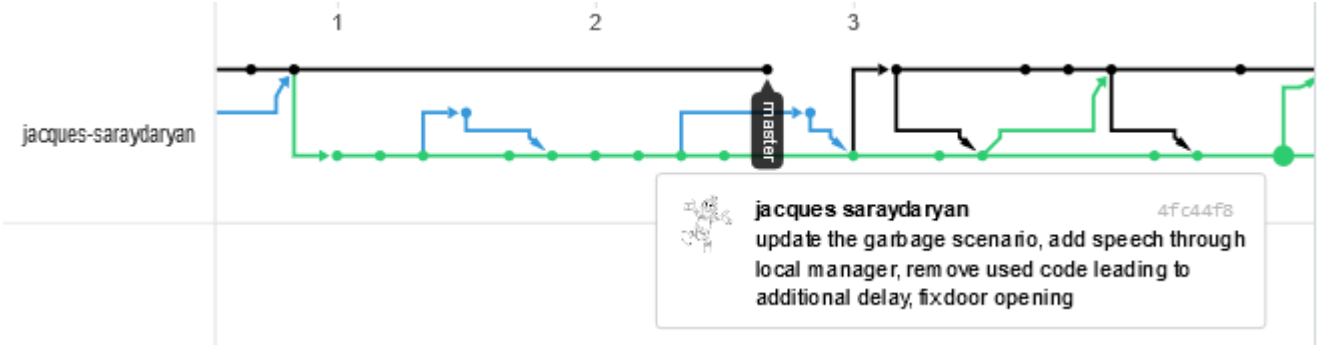
Les branches

- Visualisation des branches du repo.

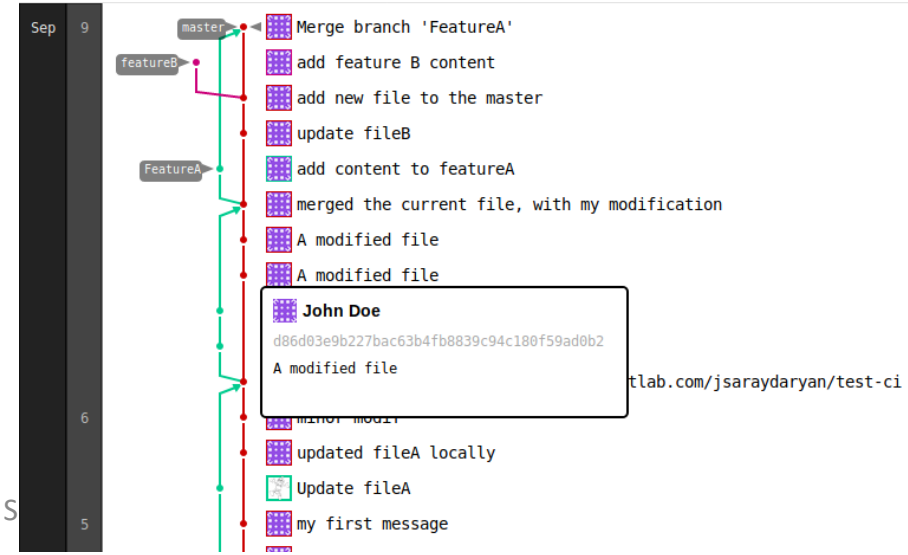
```
$ git log --graph --oneline -all
*   3308e57 (origin/dev, dev) merge featureB on dev and minor correction
|\
| * 284c683 (featureB) add buy feature and Launcher
| * 9499ab5 add buy function
* | b03b041 (featureA) add features and launch file
|/
* 740b5dd (origin/master) add shop class
* 6ddea2c (test) add shop object
* f2e8fd8 (HEAD -> master) update of classes
* 49ecbcc  add item and Storage
* 388b210 first commit
```

Les branches

Github branches graph



GitLab branches graph



Questions ?