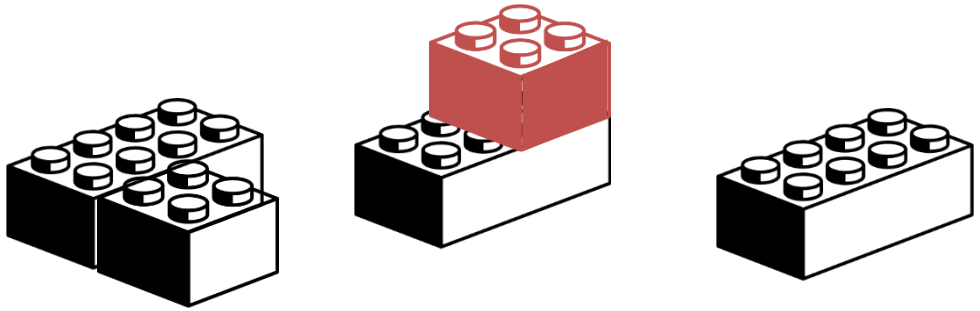
Versioning GIT

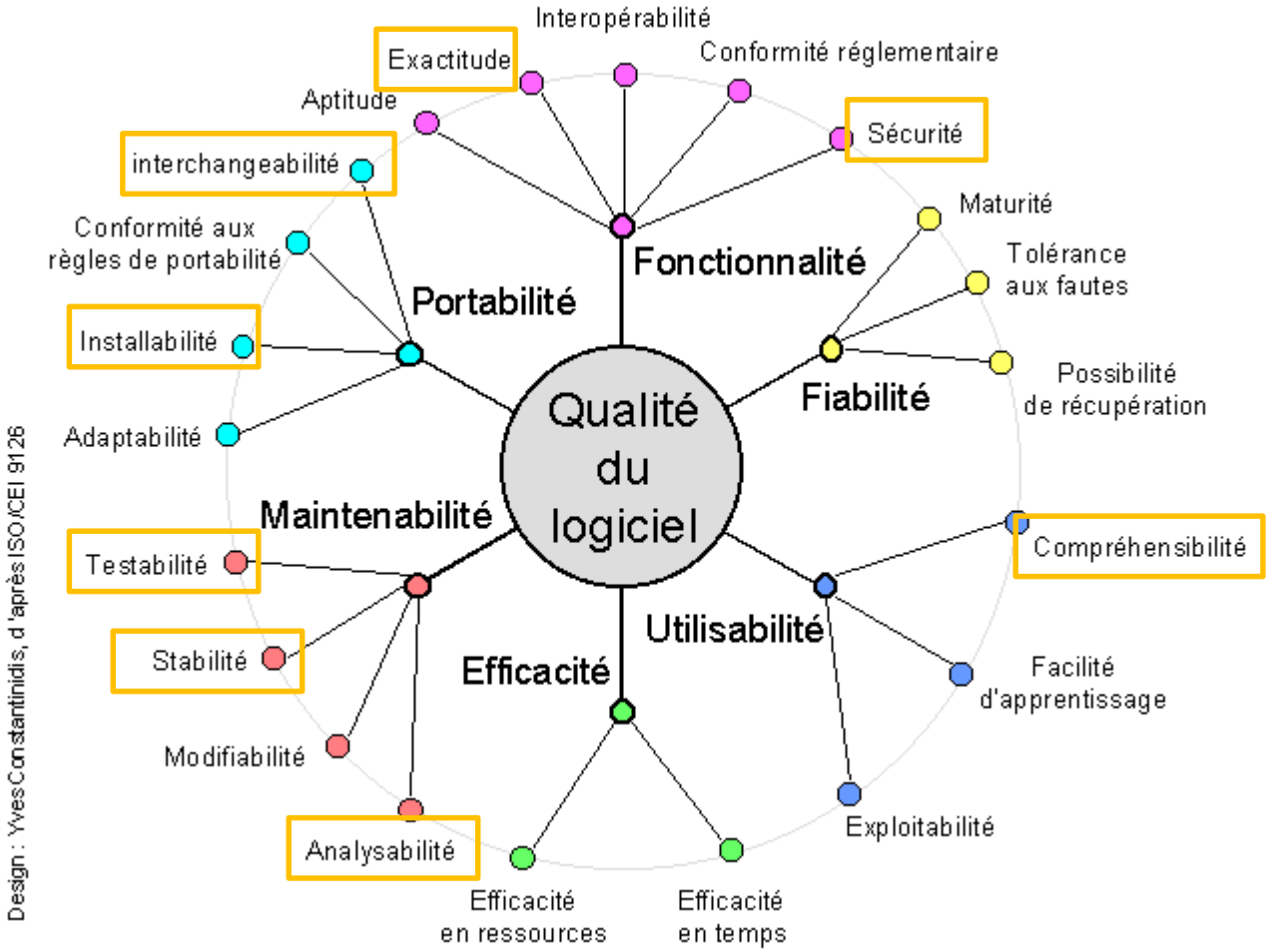
Vers l'intégration continue

J. Saraydaryan

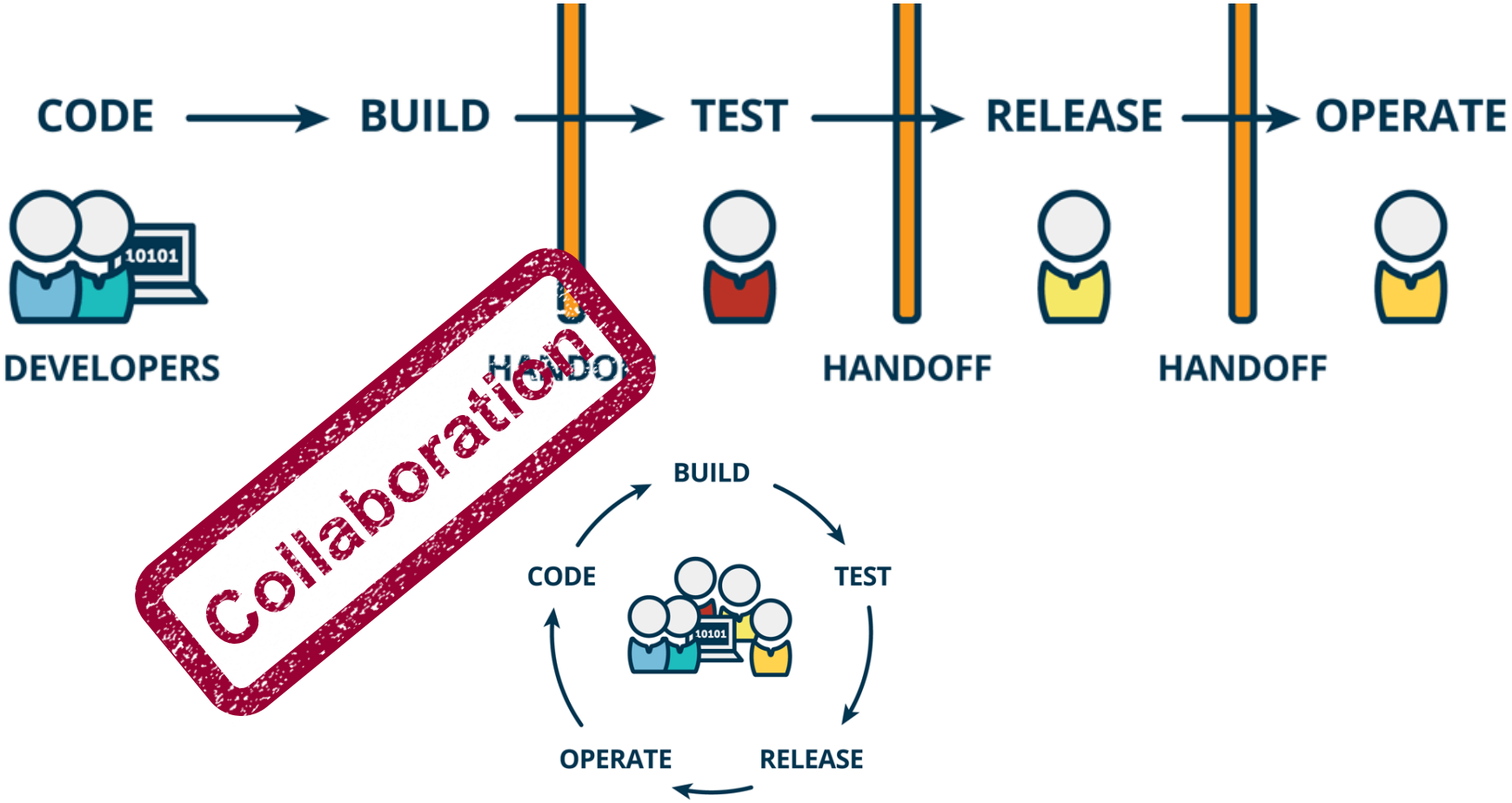


Motivation

Qualité d'un logiciel



Cycle de production

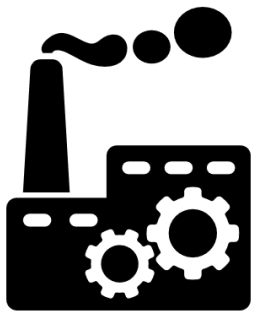


https://www.mindtheproduct.com/2016/02/what-the-hell-are-ci-cd-and-devops-a-cheatsheet-for-the-rest-of-us/

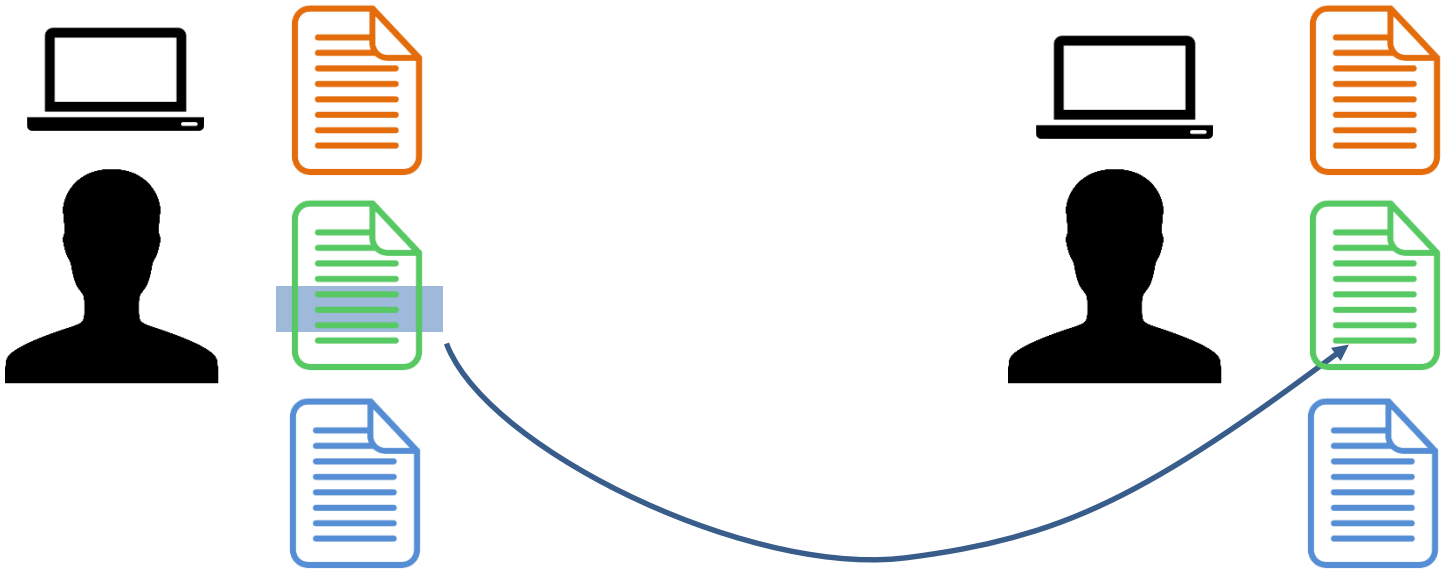
La collaboration dans le dev.

- 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.

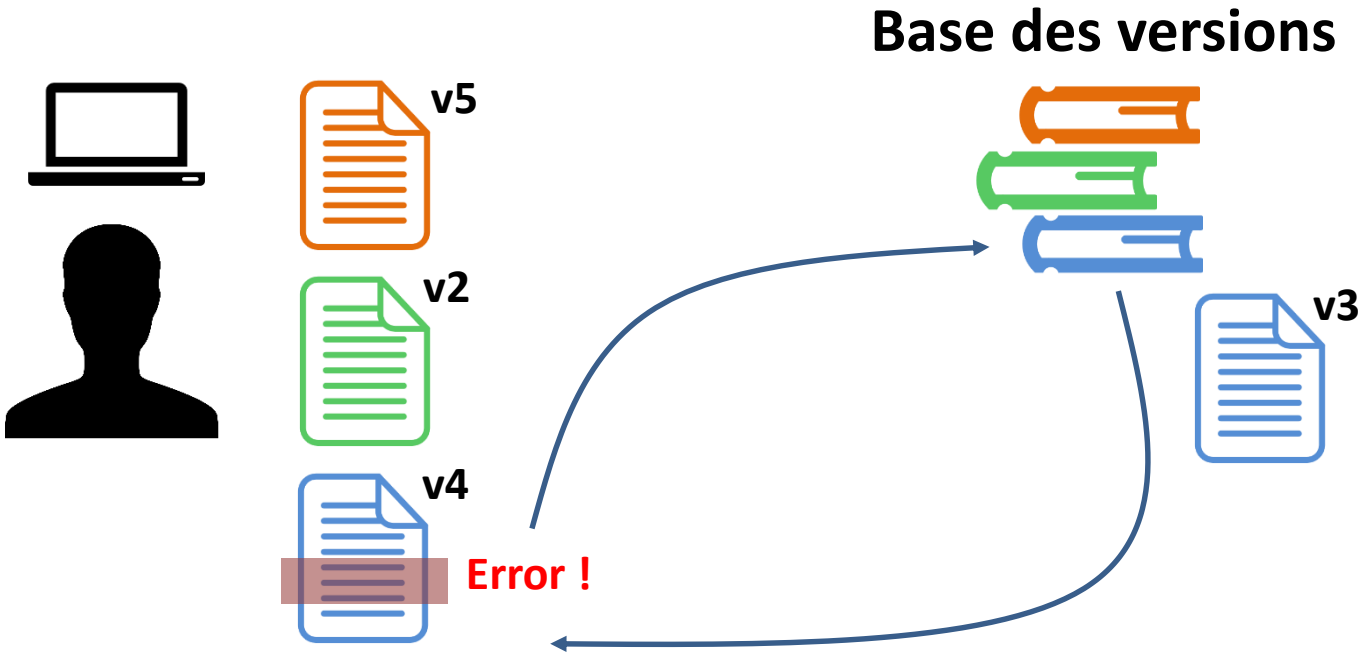


Les situations types



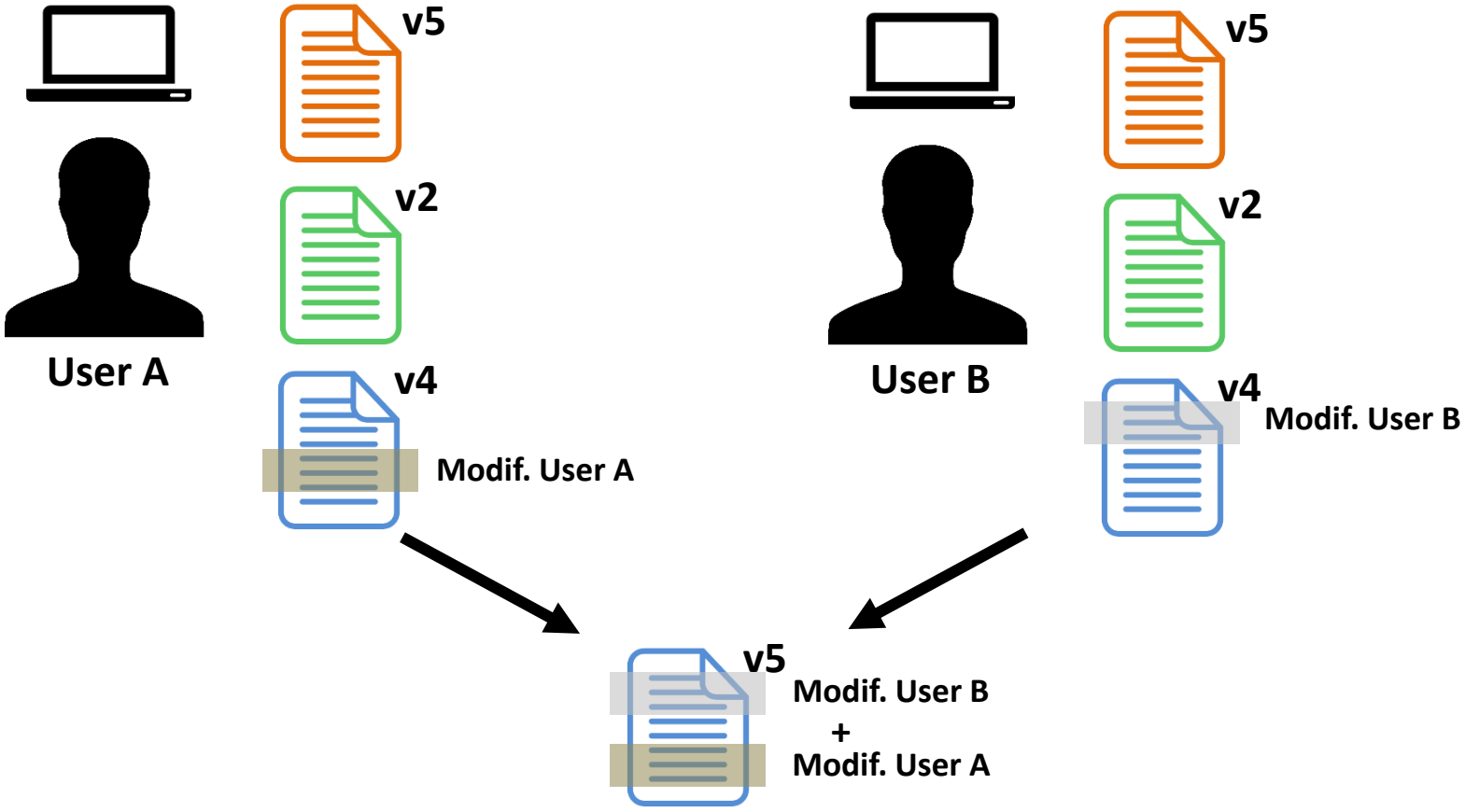
Partager les modifications

Les situations types



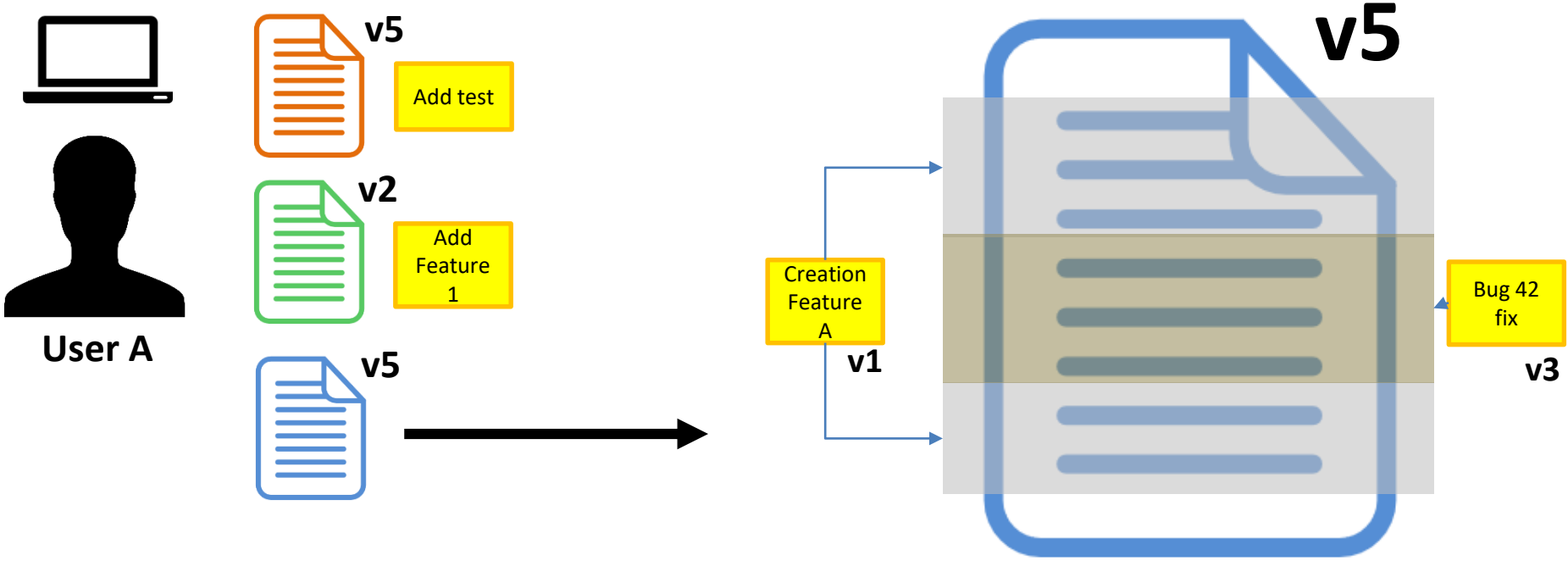
Restaurer une ancienne version

Les situations types

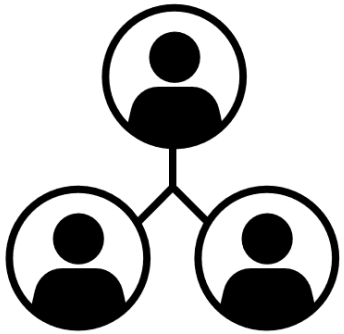


Fusionner des modifications

Les situations types



Comprendre l'historique des modifications



Les logiciels de gestions de version

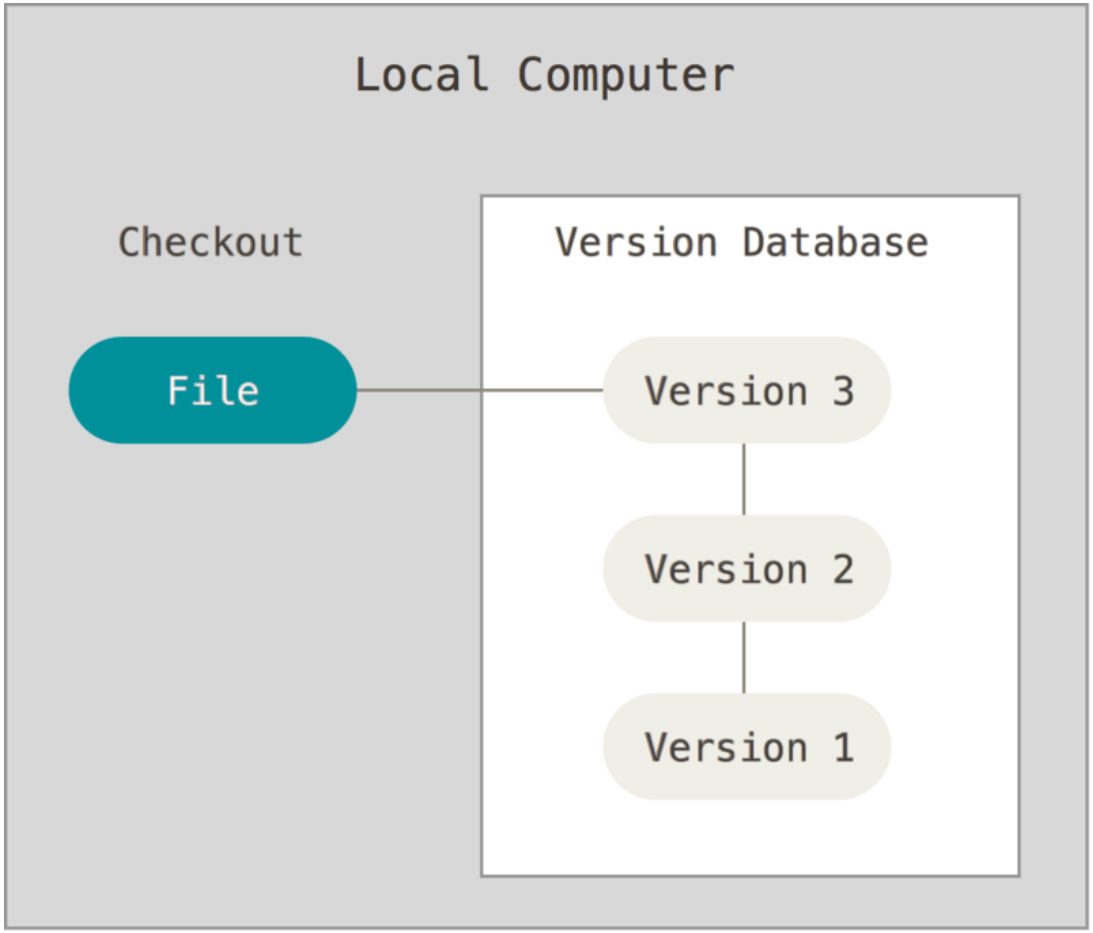
Définition

- 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**.

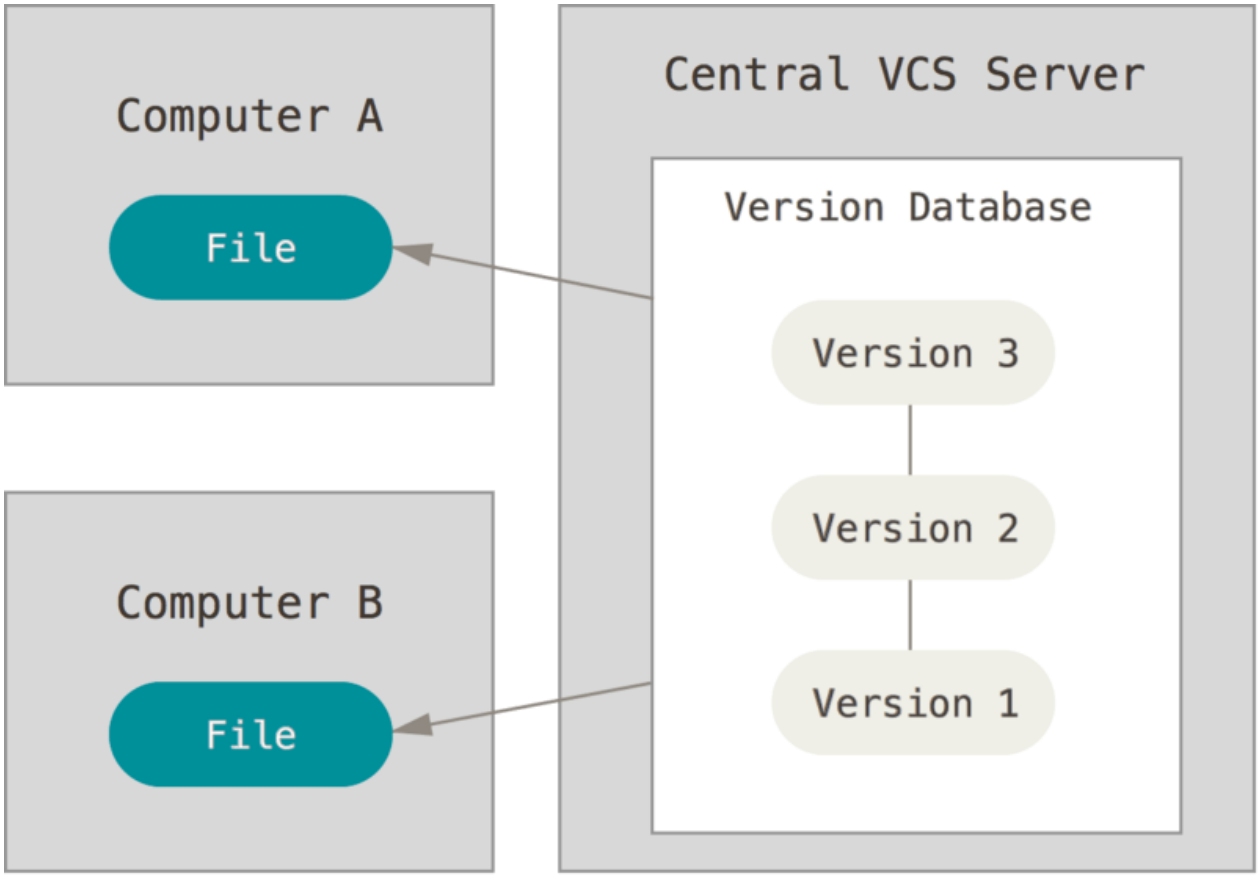
- Architecture
  - Local
    Le suivi des versions des fichiers est uniquement assuré sur la machine locale
  - Centralisée:
    Le suivi et le maintien de version est garanti par un serveur central
  - Décentralisée :
    Chaque développeur possède l'ensemble des versions des documents et informe le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

Architecture Locale



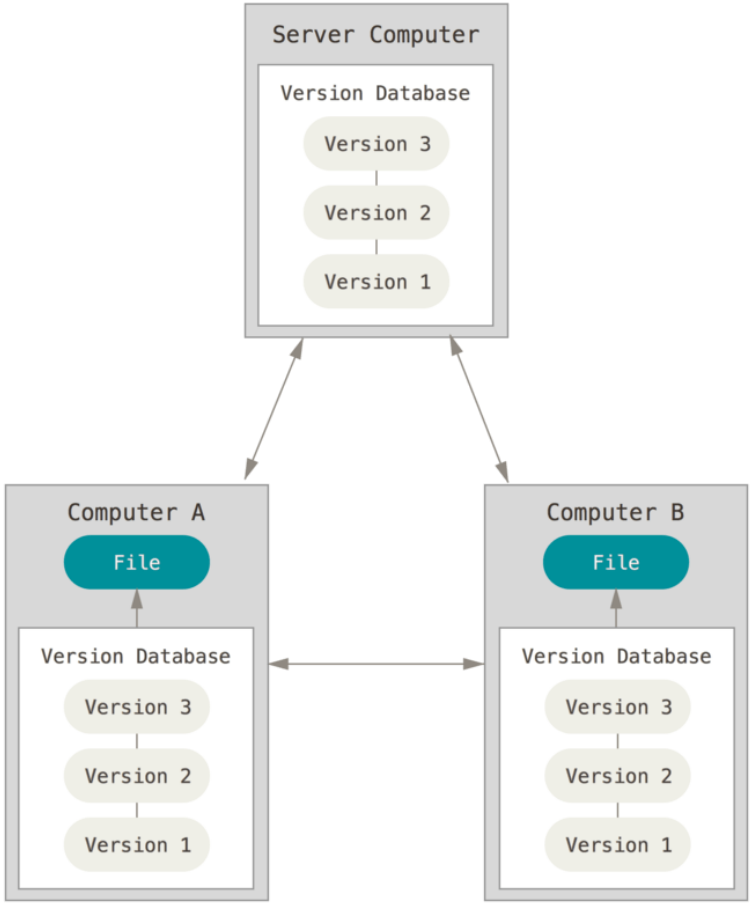
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Architecture centralisée



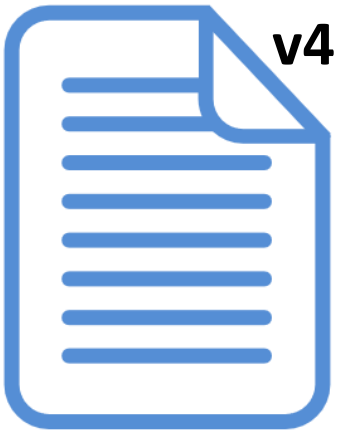
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Architecture décentralisée

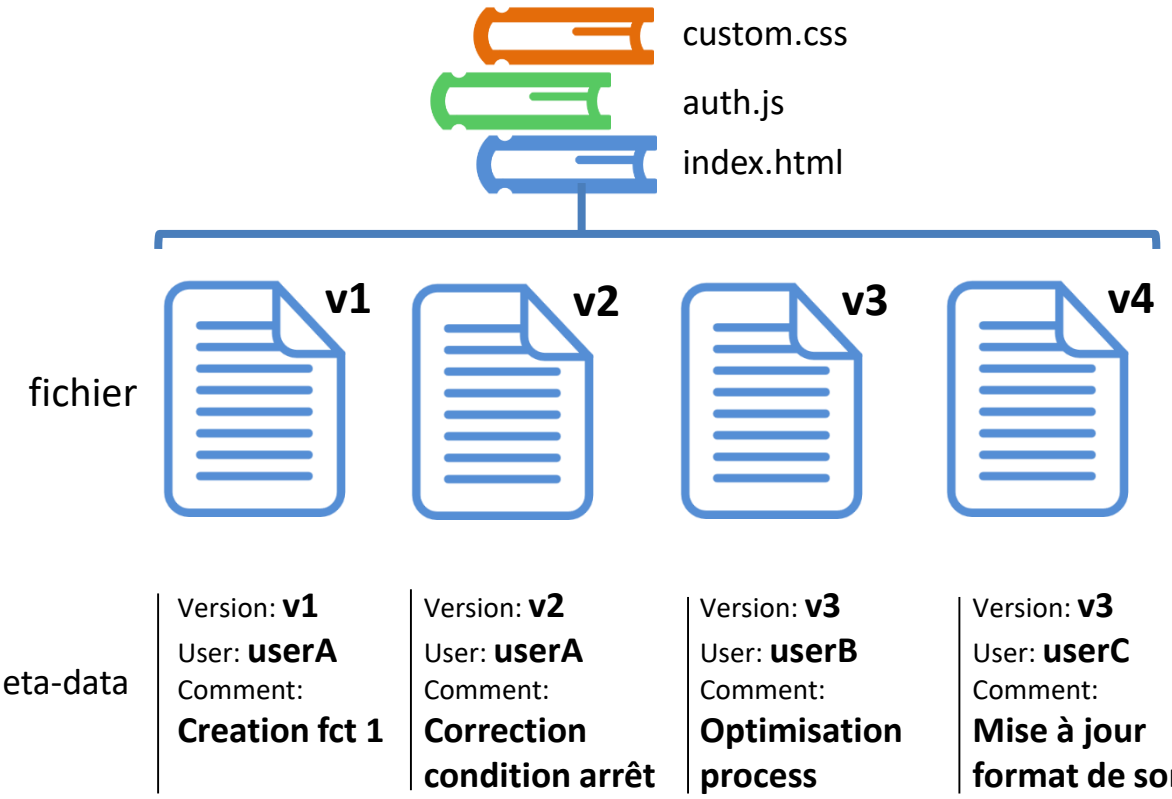


https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

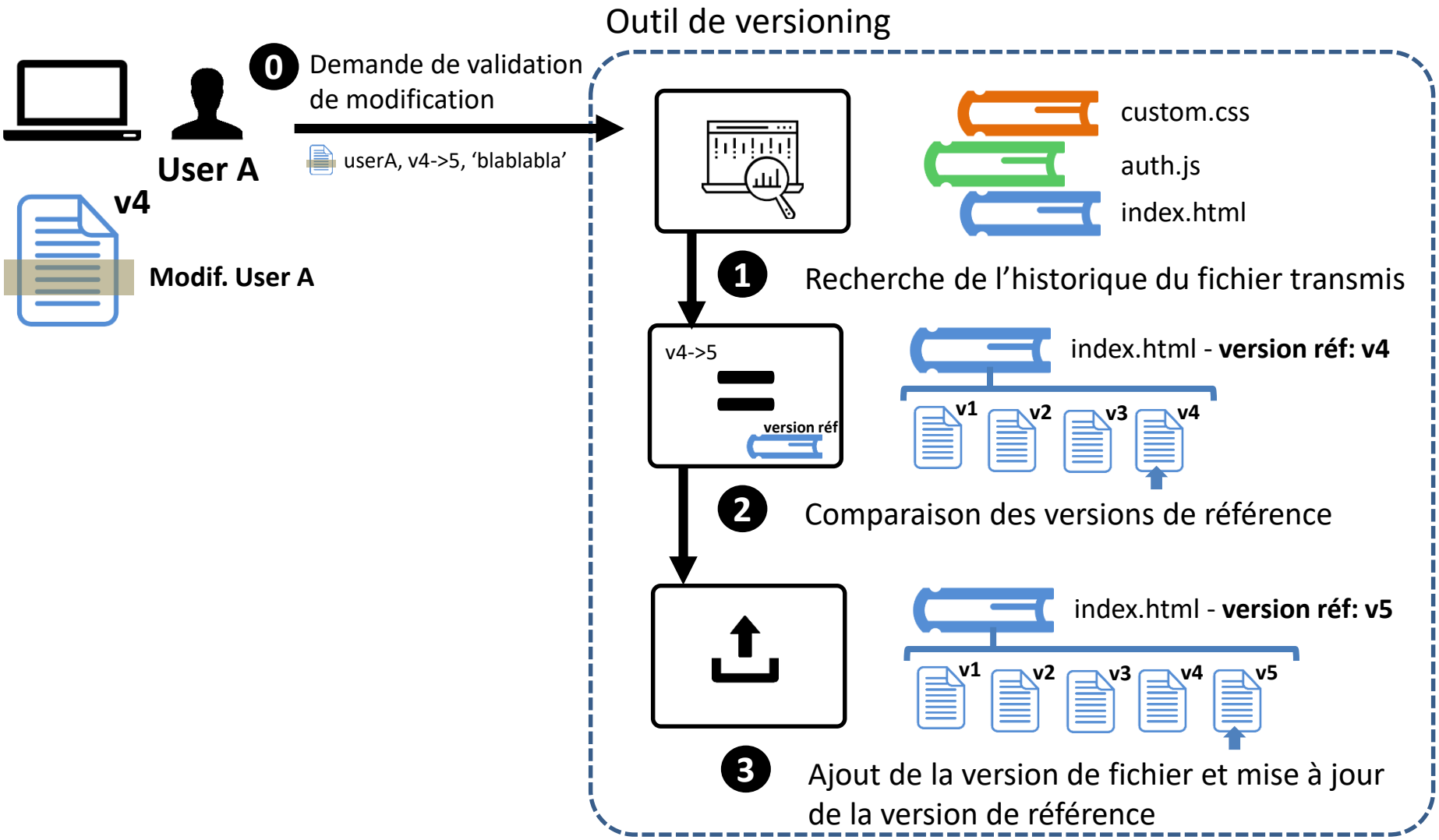
Fonctionnement



Historique des modifications



Fonctionnement



Les outils

- 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

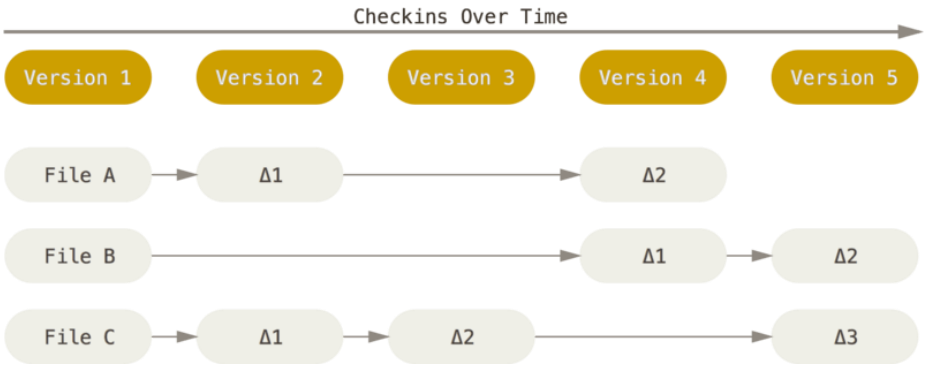
L’outil de versoning Git

Pourquoi Git ?

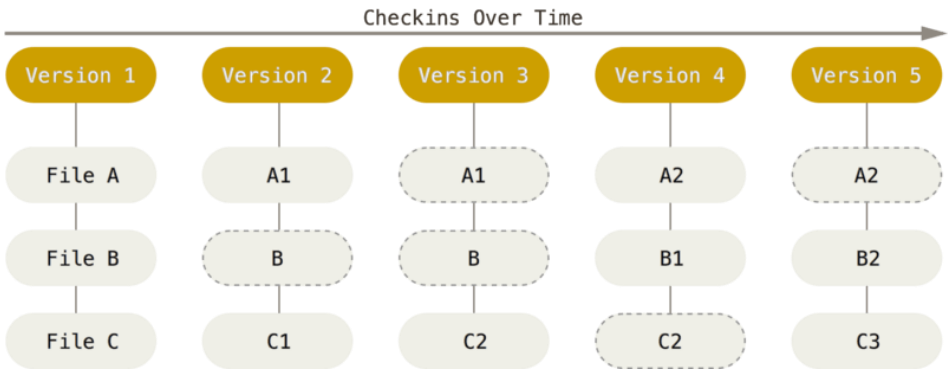
- Date de création 2005
- Objectif:
  - Gratuit
  - Système rapide
  - Design simple
  - Usage possible de développement non linéaire (branches de développement en parallèle)
  - Pleinement distribué
  - Support de gros projets possible (e.g kernel linux)
- Propriétés
  - Snapshot (pas de delta)
  - 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
  - Delta (e.g svn) versus Snapshot (e.g git)



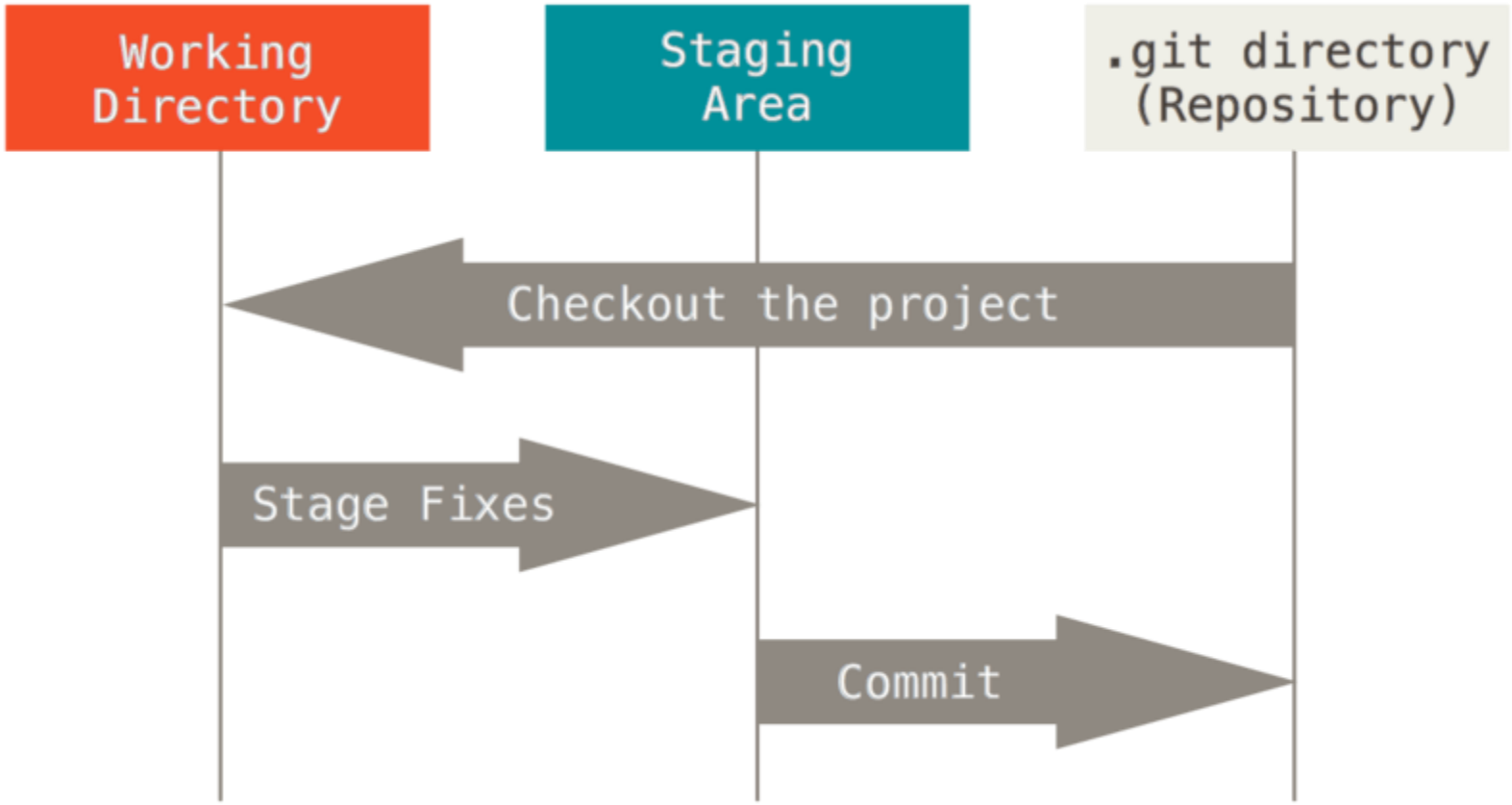
VS



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

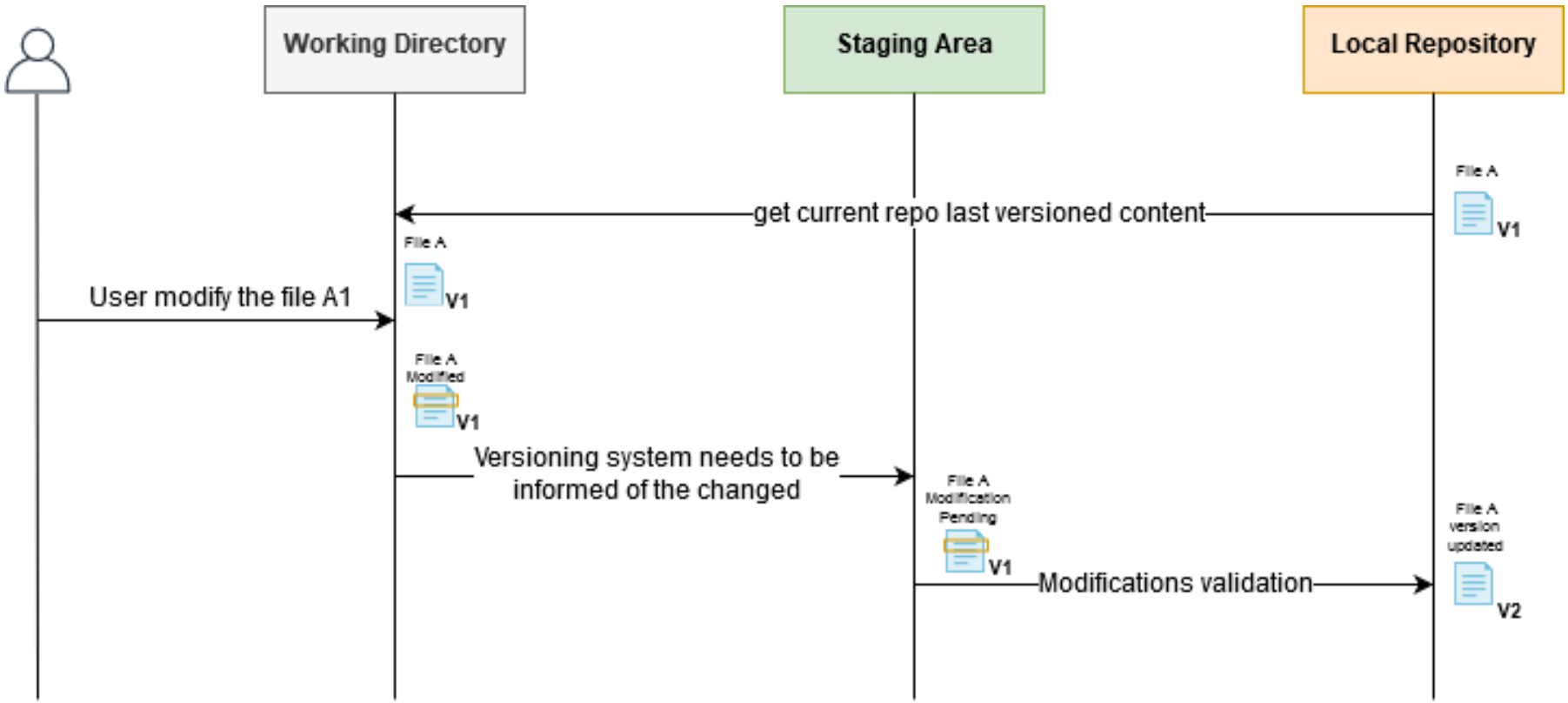
Workflow général d'usage

- Fonctionnement de git
  - 3 états de fichiers



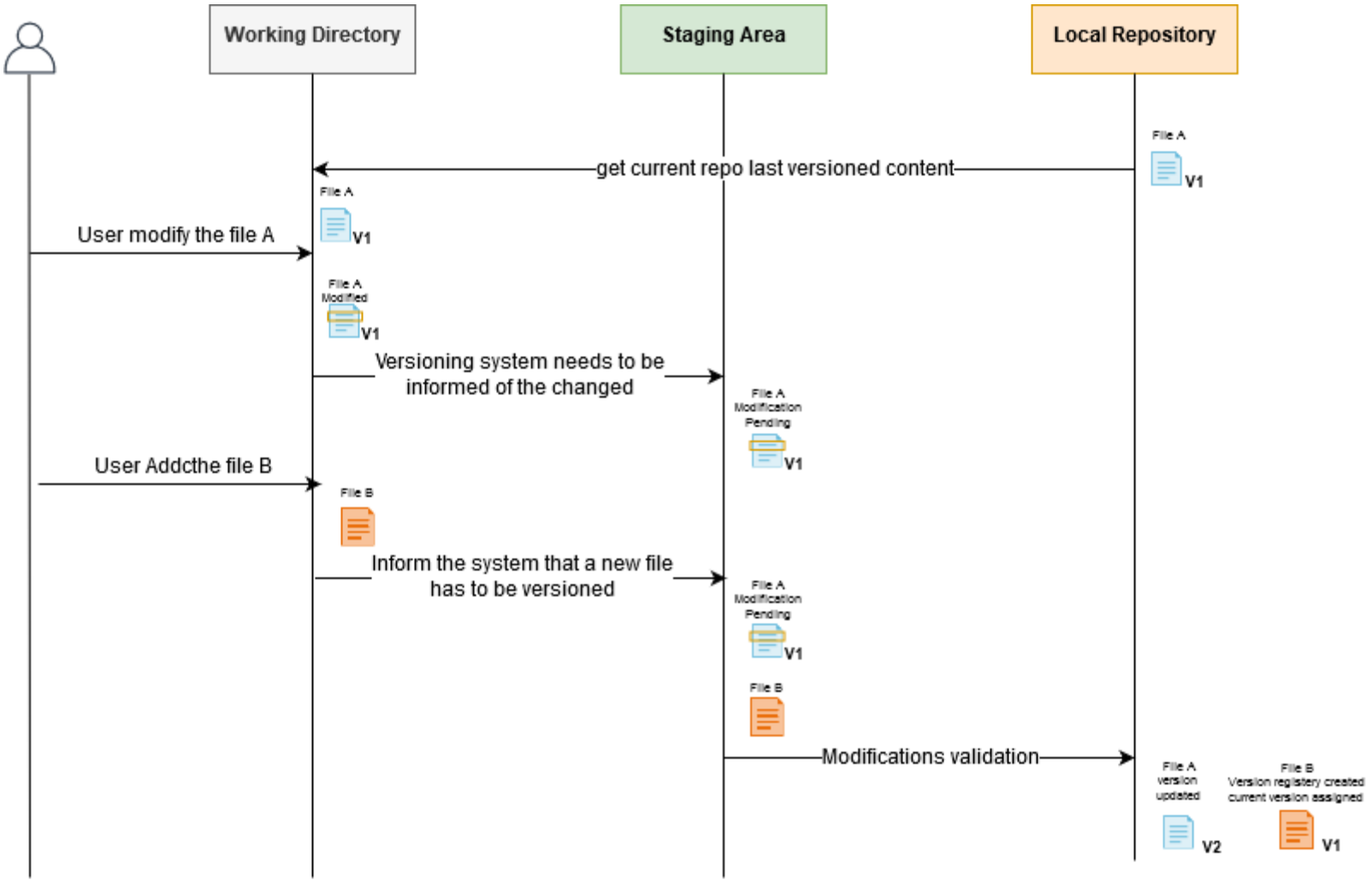
Workflow général d'usage

- Fonctionnement de git
  - 3 états de fichiers



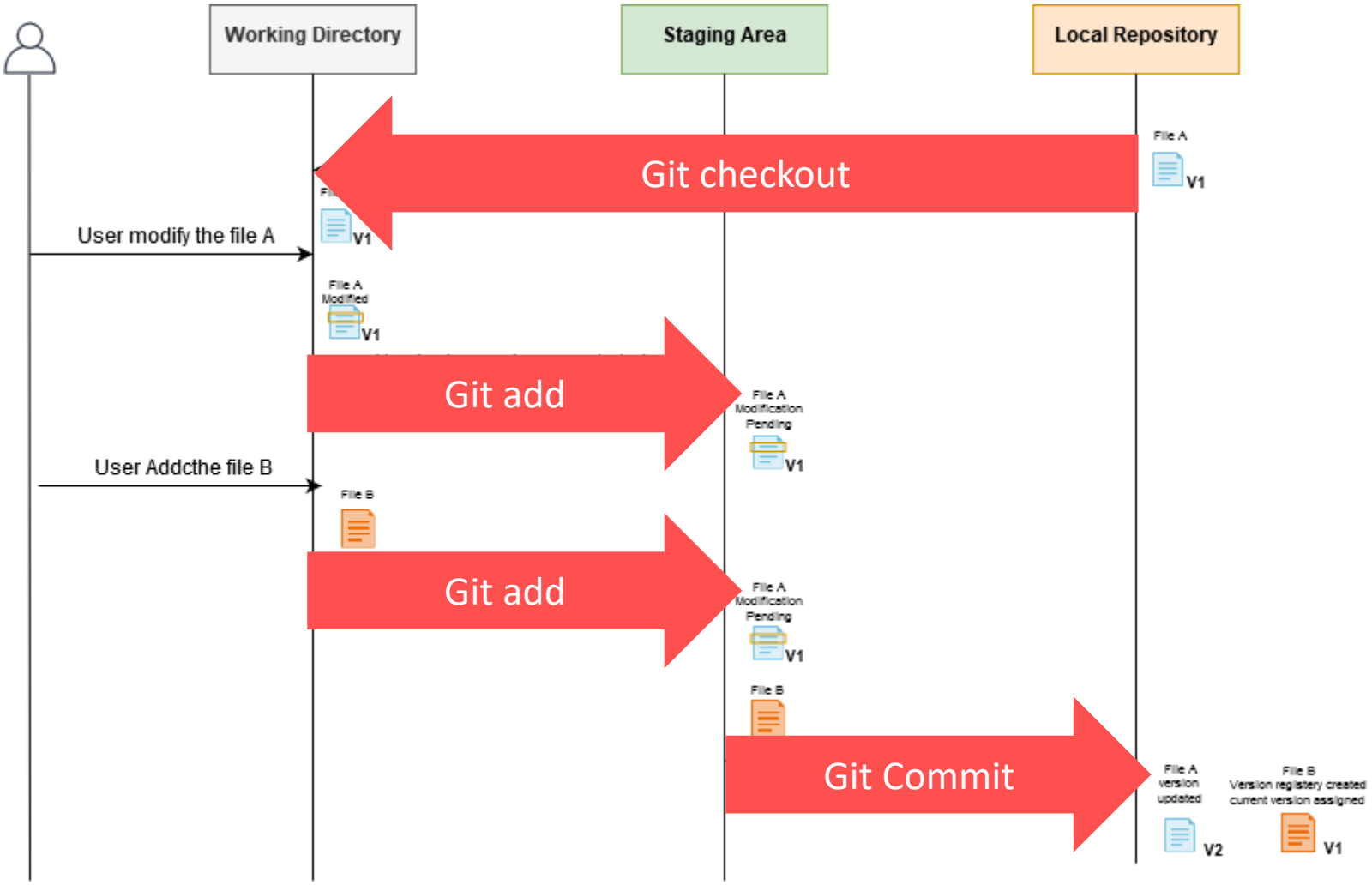
Pourquoi Git ?

- Fonctionnement de git

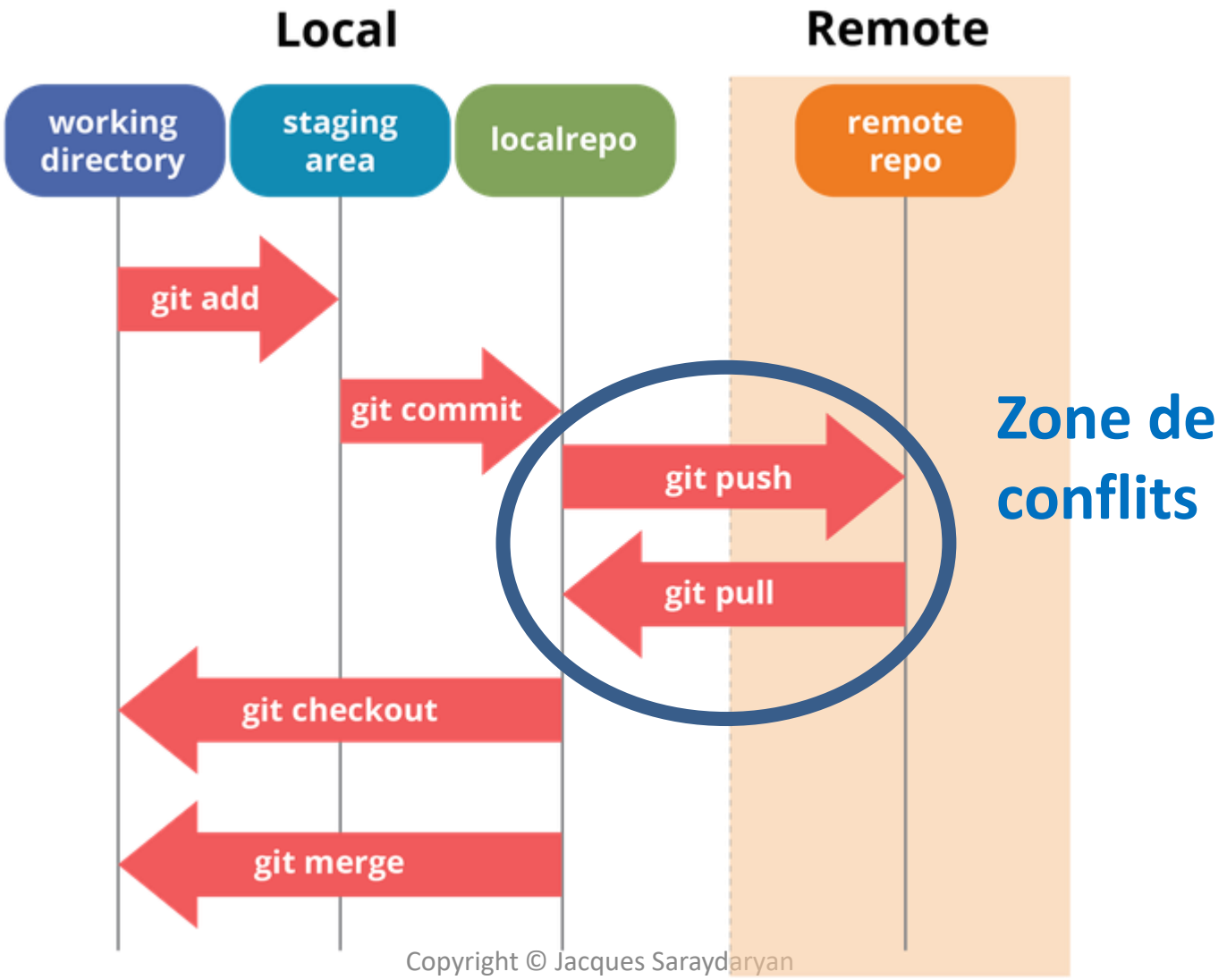


Pourquoi Git ?

- Fonctionnement de git



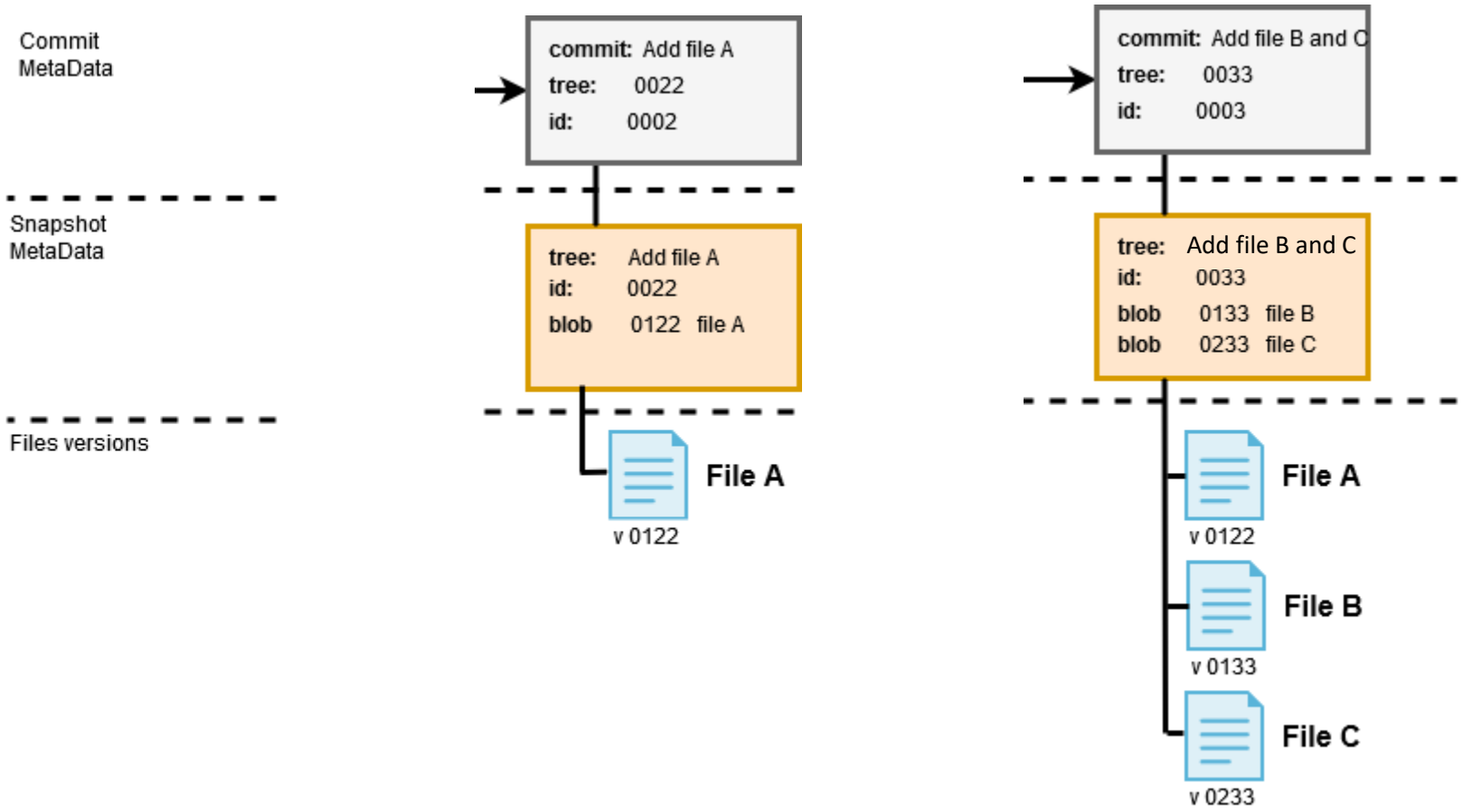
Workflow général d'usage



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

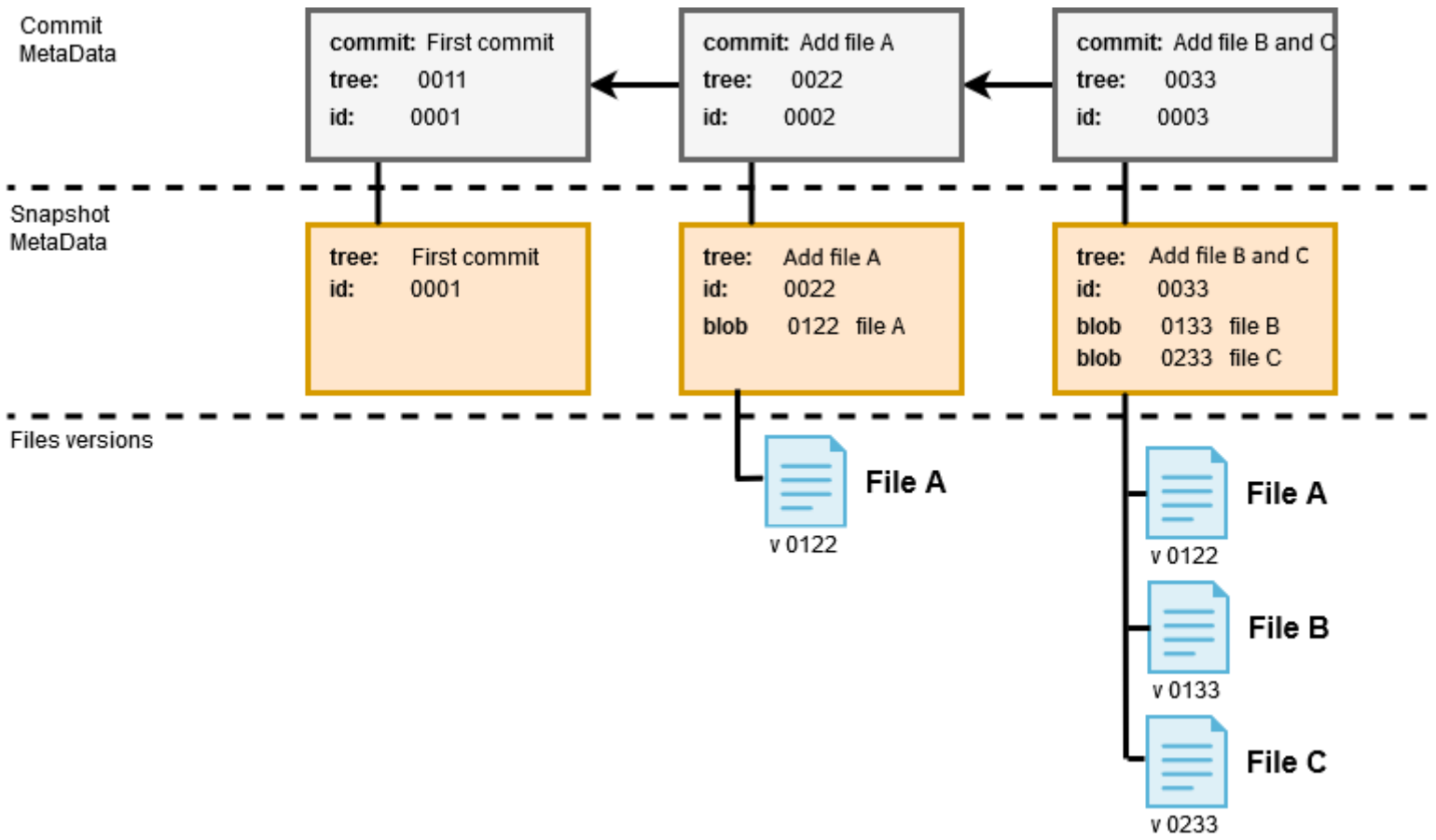
Workflow général d'usage

- Gestion des branches



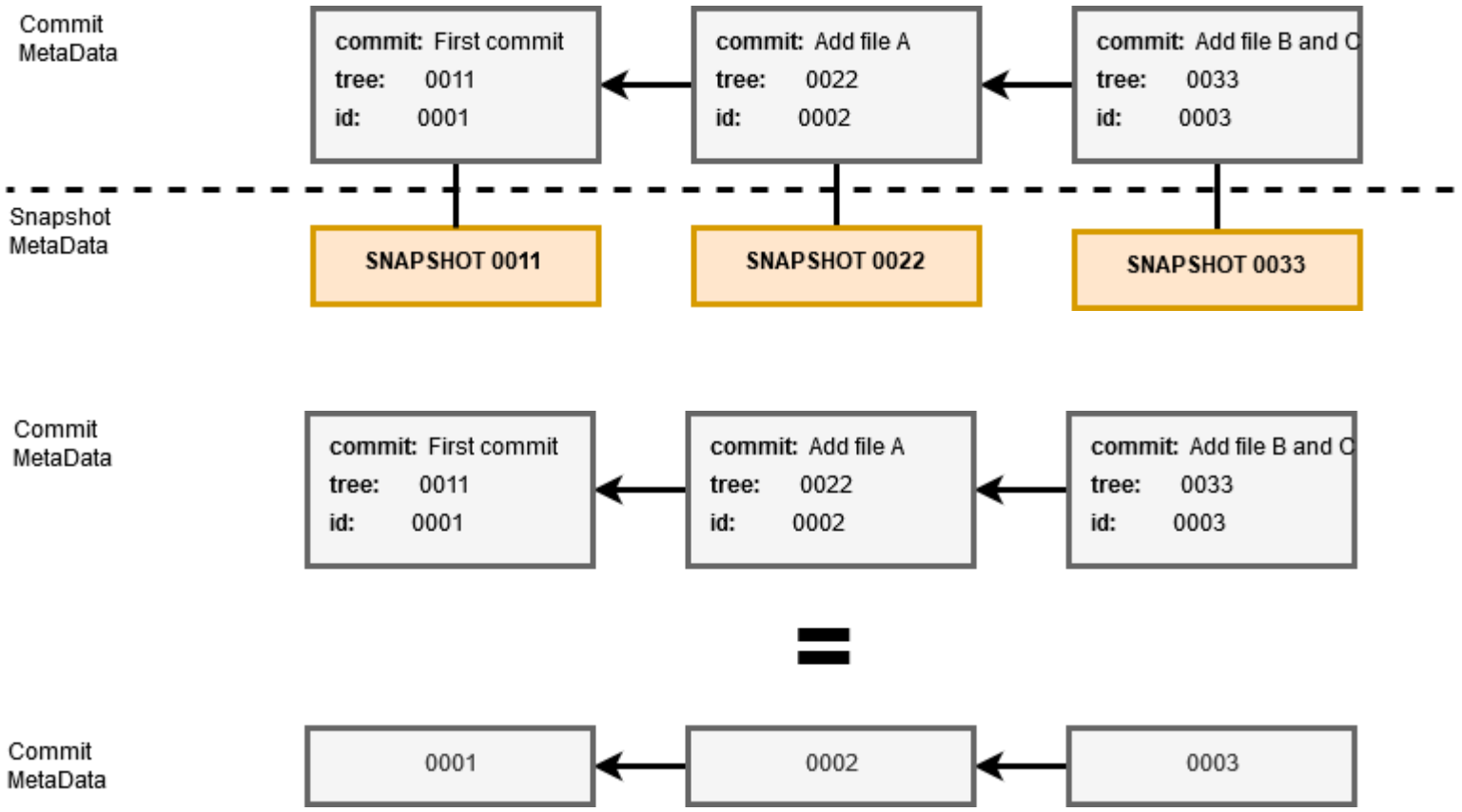
Workflow général d'usage

- Gestion des branches



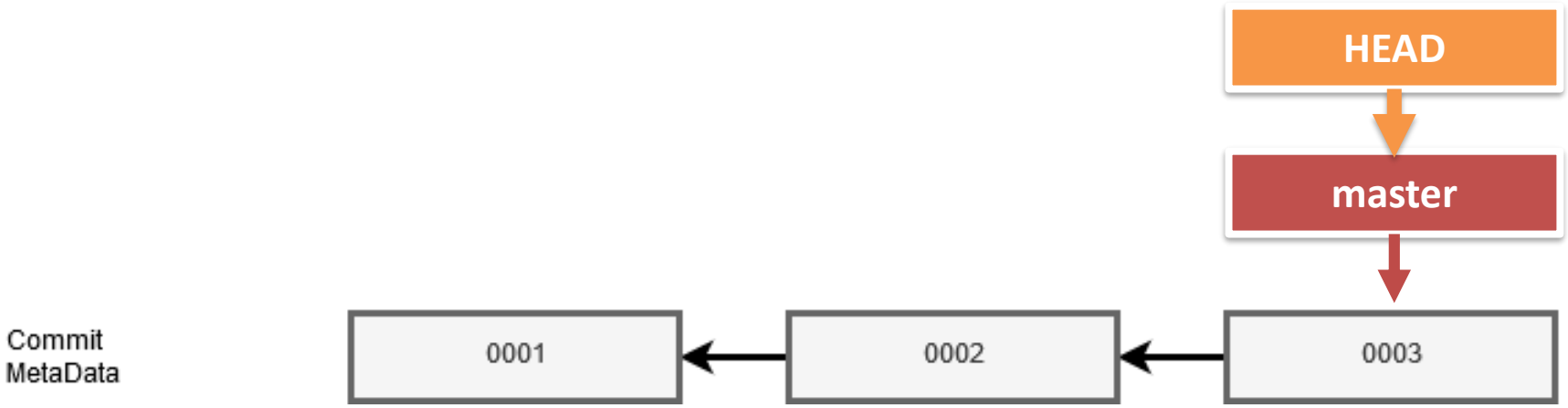
Workflow général d'usage

- Gestion des branches



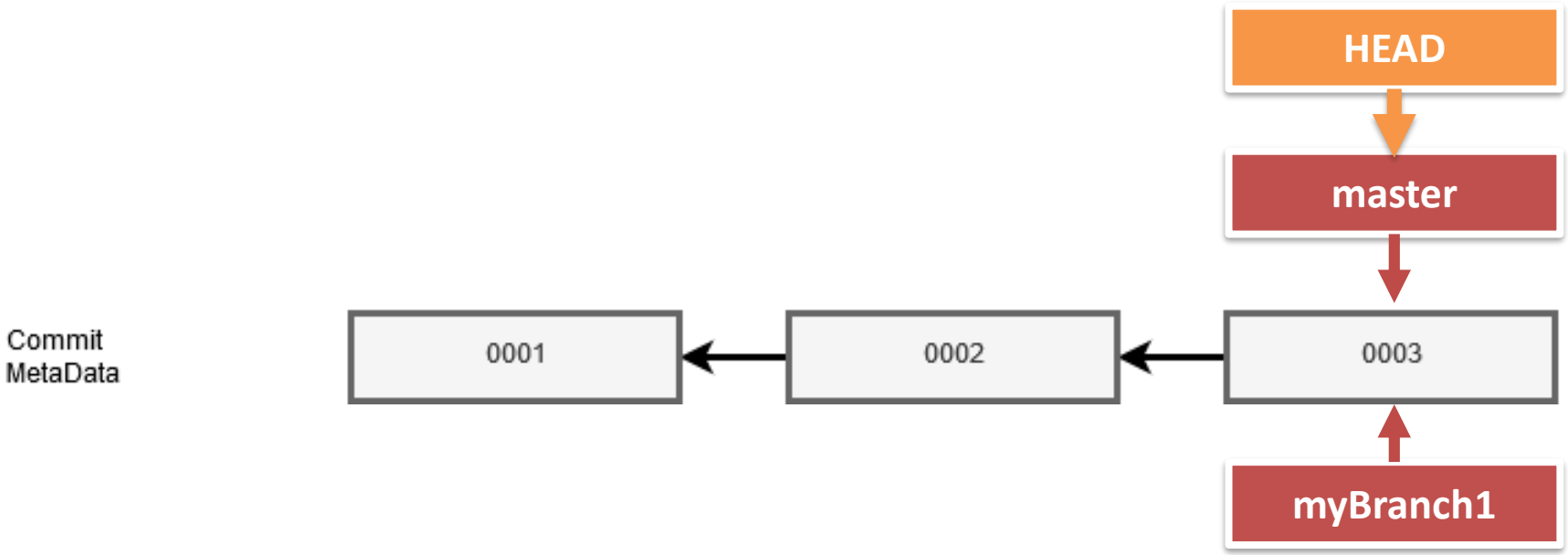
Workflow général d'usage

- Gestion des branches



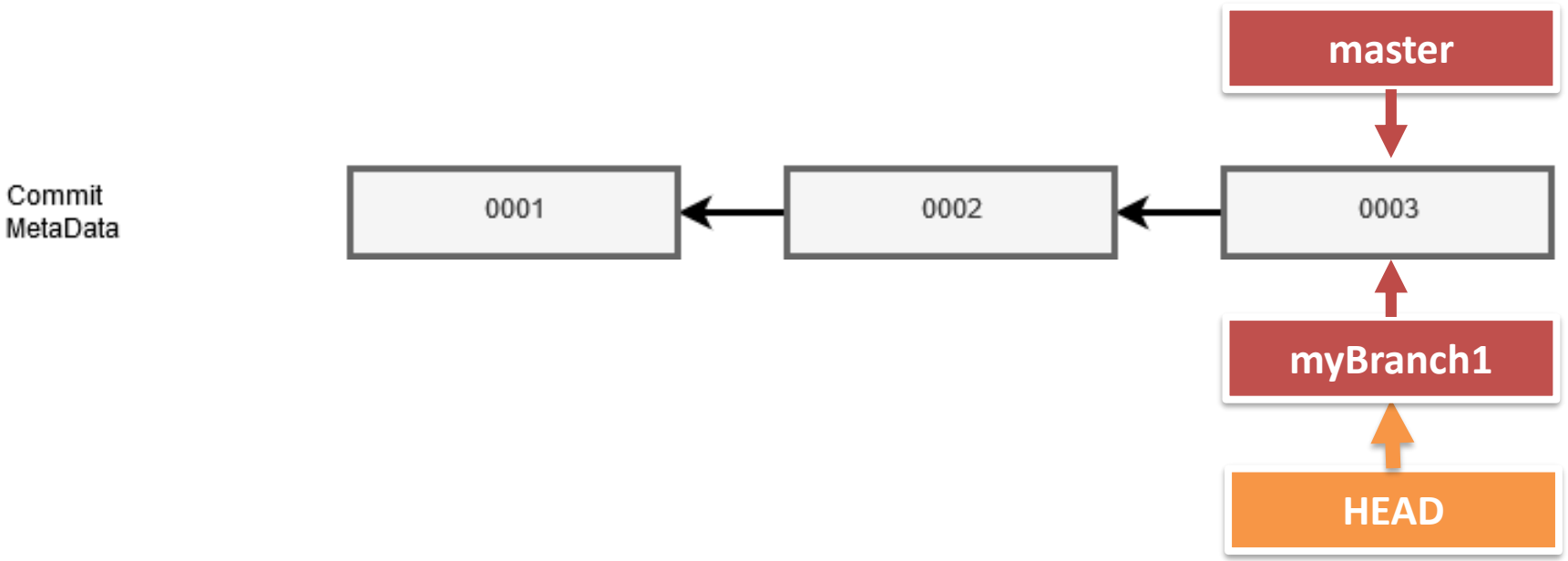
Workflow général d'usage

- Création d'une branche



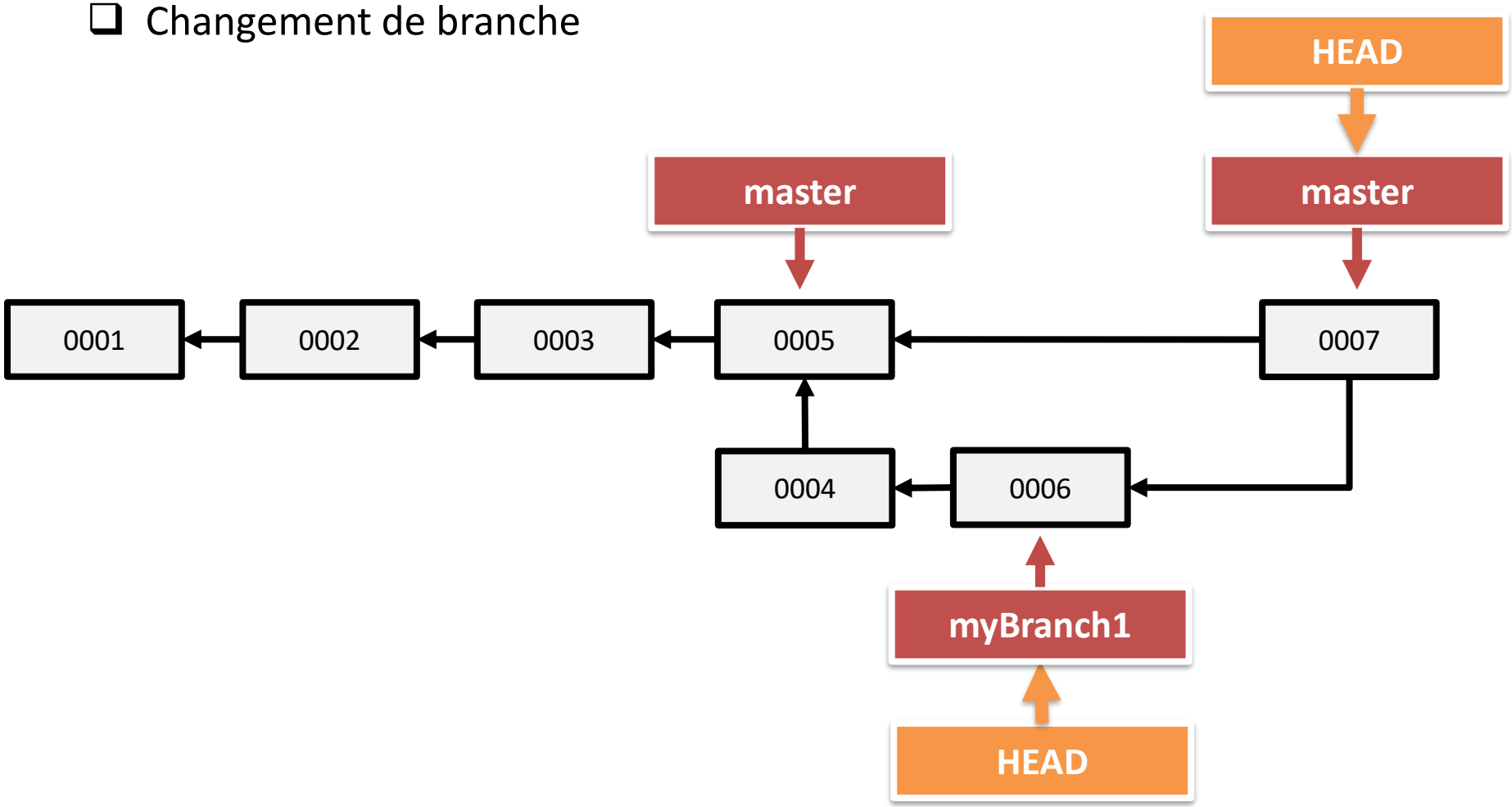
Workflow général d'usage

- Changement de branche

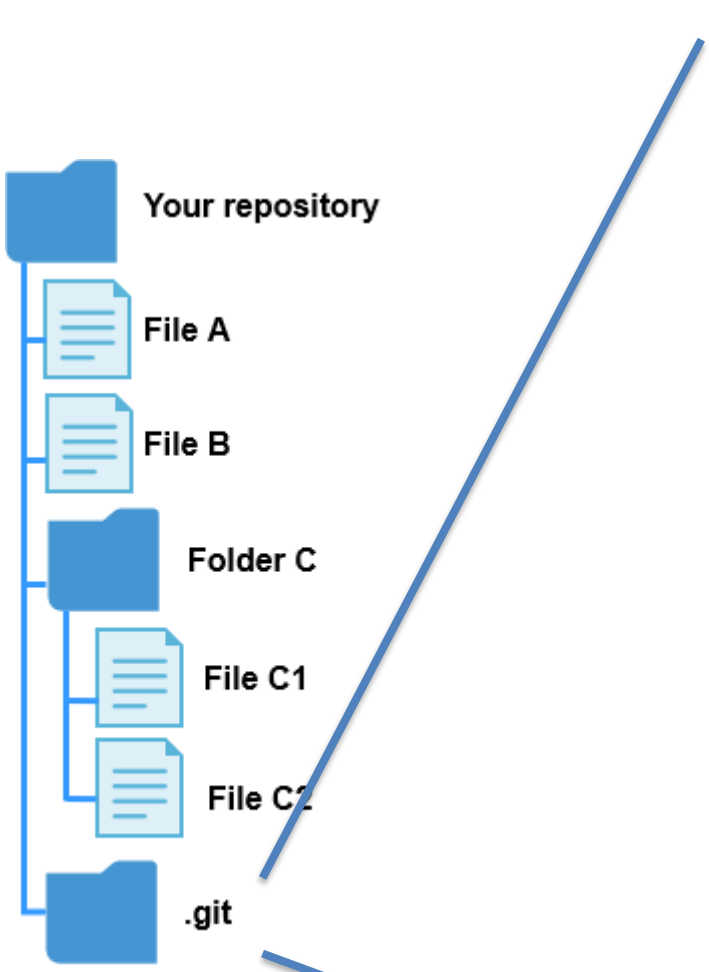


Workflow général d'usage

- Changement de branche



Fonctionnement



- **branches** : Deprecated
- **config** : Stocke les configurations git spécifiques au projet
- **description** : Utilisé uniquement par gitweb (représentation)
- **HEAD** : **Stocke la référence de la branche en cours d'utilisation**
- **hooks** : Contient des scripts invoqués après les commandes git (après commit → run a script)
- **index** : **Utilisé pour marquer vos modifications comme « stage »**
- **info** : Toutes les informations complémentaires au repository
- **logs** : **Stock les changements effectués sur les « refs »**
- **objects** : **Ensemble des objets git (files, commits, trees,…**
- **refs** : **Contient les références des branches et des tags (Id → Hash)**

Fonctionnement

https://youtu.be/2bwqMGkcXU0

Git Commande de bases

Usage de Base

- Premier usage de git: définir son profil

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

- Définir son identité (besoin pour valider les changements) **`git config`**

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

- Définir l'outil à lancer pour gérer les conflits

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

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

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

Usage de Base

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

https://youtu.be/yEdHU3F_odM

Usage de Base

- Utilisation d'un repo. existant **`git clone`**

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

- Créer un répo. Localement **`git init`**

```
$ git init myRepo
```

- Afficher l'état des travaux en cours **`git status`**
  - Vous avez la possibilité de vérifier l'état des travaux en cours (éléments versionnés ou non, modifications non validées …)

```
$ git status
      On branch master
      Changes to be committed:
        (use "git reset HEAD <file>..." to unstage)
           new file:   file3
      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:   file1
      Untracked files:
        (use "git add <file>..." to include in what will be committed)
           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
```

Usage de Base

- Historique affichage compact

`git log`

```
$ git log --graph --decorate --pretty=oneline --abbrev-commit --all

*   01e5ca8 (HEAD -> master) Merge branch 'featureA' into master
|\
| * 7851ad9 (featureA) minor update fileC
| * bc0fbf6  make some modifications
* | 2c05e25 continue FileB update
* | cca3938 add modification on FileB
|/
* 9e85a68 (origin/master) first commit
```

Usage de Base

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

https://youtu.be/iL72gh-pCBU

Usage de Base

- Ajout d'un fichier (et indexation) **`git add`**
  - Lorsque qu'un nouveau fichier est ajouté il est nécessaire d'informer git que ce dernier doit être versionné

```
$ git add myNewFile1
```

  - Plusieurs fichiers peuvent être ajoutés en même temps

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

  - Tous les nouveaux fichiers peuvent être ajoutés d'un coup

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

  - Une fois les fichiers ajoutés ils doivent être validés

```
$ git commit myNewFile1 myNewFile2 myNewFile3 –m " valid of several files"
```

- Tracking d'un fichier pour validation (stage status) **`git add`**

```
$ git add myModifiedFile1
```

Usage de Base

- Validation de modification d'un fichier **`git commit`**
  - Validation de l'ensemble des éléments indexés dans l'état staged

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

  - La validation de plusieurs fichiers distincts dans l'état stage est également possibles

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

  - Tips: possibilité de valider directement un fichier modifié (préalablement indexés, dans l'état staged ou non)

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

  - Tips: Validation de l'ensemble des fichiers modifiés (préalablement indexés, dans l'état staged ou non)

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

Usage de Base

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


https://youtu.be/pkvA5S4-3Bs

Exercice

- Créer un compte sous gitlab
- Créer un projet: myFirstRepo
- Cloner votre répertoire sur votre machine
- Créer deux fichiers
  - A.txt
  - B.txt
- Ajouter du contenu dans les deux fichiers
- Vérifier l'état de votre repo. Local
- Valider vos modifications
- Afficher les logs des opérations effectuées
  - Expliquer les informations affichées

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 validé

```
$ git diff file1
```

  - E.g

```
$ 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;
```

Exercice

- Utiliser votre projet: myFirstRepo
- Créer deux fichiers
  - C.txt
  - D.txt
- Ajouter du contenu dans les deux fichiers
- Vérifier l'état de votre repo. Local
- Valider vos modifications
- Vérifier l'état de votre repo. Local
- Modifier les fichiers:
  - A.txt
  - B.txt
- Valider vos modifications
- Afficher les logs des opérations effectuées
  - Expliquer les informations affichées

Usage de Base

- Modification d'un commit **`git commit --amend`**
  - Situation: erreur dans le commit, oubli d'éléments à valider (commit)

```
$ 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 (Stage) **`git restore --staged`**

```
$ git restore --staged 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 restore --staged  fileB
$ git status
     On branch master
     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
           modified:   fileB
```

Usage de Base

- Annulation de toutes les modifications (Stage) **`git reset`**

```
$ git reset
```

```
$ 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
          modified:   README.md

$ git reset
$ git status
     On branch master
     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
          modified:   fileB
```

Usage de Base

- Annulation des modifications (repo dir.) **`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 précis notation courte

➡ **Seul les commits sont annulés, vos fichiers restent les mêmes (non suivi à la modification)**

  - Annulation du dernier commit (hard)

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

Exercice

- Dans votre repo. myFirstRepo
- **Modifier** votre fichier **B**
- **Annuler** la modification effectuée
- **Modifier** votre fichier **B**
- **Valider** votre modification **(staged)**
- **Annuler** votre modification **validée**
- **Modifier** votre fichier **B**
- **Valider** votre modification **(staged)**
- **Exporter** votre modification **(commited)**
- **Annuler** votre modification exportée

Ignorer certains fichiers

- .gitignore
  - Permet de sélectionner les fichiers qui ne seront pas suivis par GIT
    Chaque ligne définit un pattern de format de fichiers/répertoires
    Les références aux objets sont faites à partir de la localisation du fichier .gitignore

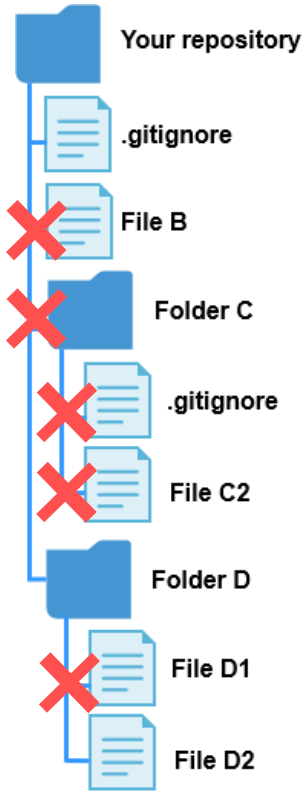
.gitignore

```
#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 fichier

- .gitignore
  - Les 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 (e.g /index.html) renvoie à la racine de la position du fichier git
    - si à la fin indique un répertoire (e.g target/)

    * : match n'importe quel caractère (sauf le /)

    ?: seulement un caractère

    **/: match tous les répertoires (e.g **/classes)

    /**: match l'ensemble du contenu (e.g /target/**)

Exercice

- Créer un repo local git
- Modifier le répo comme suit:

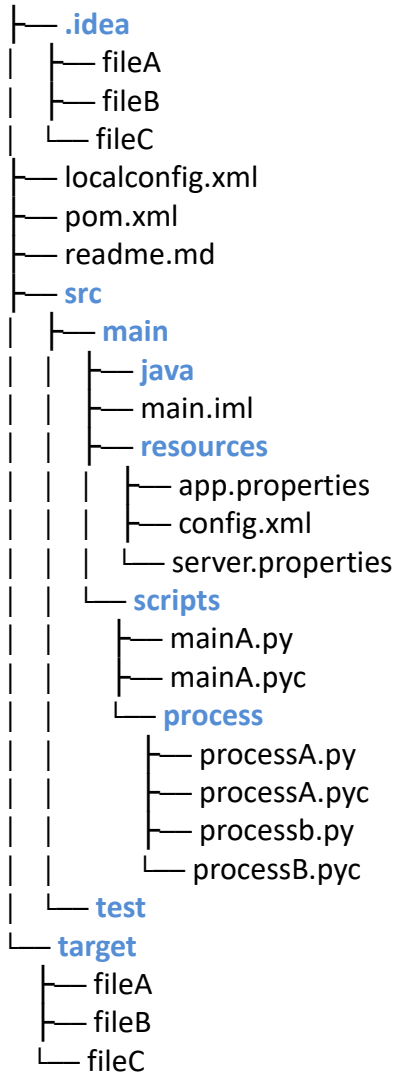
- Créer un fichier git .gitignore à la racine de votre répo répondant aux contraintes suivantes

```
├── .idea
│   ├── fileA
│   ├── fileB
│   └── fileC
├── localconfig.xml
├── pom.xml
├── readme.md
├── src
│   ├── main
│   │   ├── java
│   │   ├── main.iml
│   │   ├── resources
│   │   │   ├── app.properties
│   │   │   ├── config.xml
│   │   │   └── server.properties
│   │   └── scripts
│   │       ├── mainA.py
│   │       ├── mainA.pyc
│   │       └── process
│   │           ├── processA.py
│   │           ├── processA.pyc
│   │           ├── processb.py
│   │           └── processB.pyc
│   └── test
└── target
    ├── fileA
    ├── fileB
    └── fileC
```

Exercice

- Contraintes, les éléments suivants ne doivent pas être suivis:
  - Le répertoire **.idea**
  - Le fichier **localconfig.xml**
  - Tous les fichiers de type **.properties** du répertoire **resources**
  - Tous les fichiers de types **.pyc**
  - Le fichier **main.iml**
  - Tous les fichiers du répertoire **target**

- Réaliser des modifications afin de vérifier le bon fonctionnement du .gitignore

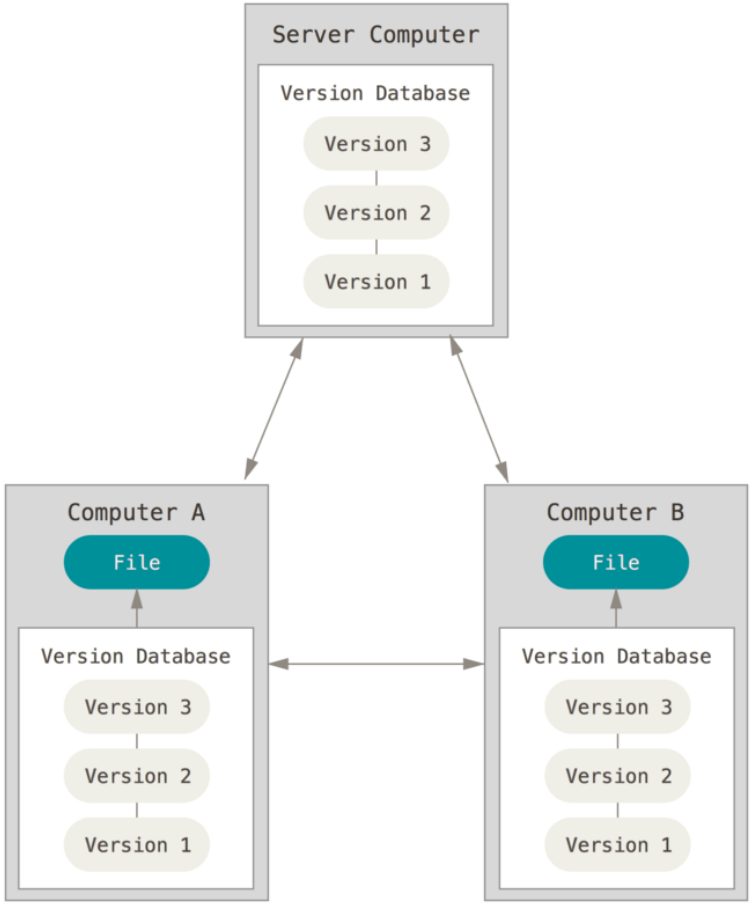


Exercice Synthèse

- **Créer** un repo local git
- **Ignorer** le suivi :
  - des fichiers suivants: *.pyc
  - Les dossiers suivants:
    .vscode/
    .history/
- **Créer** les répertoires: css , js
- **Créer** le fichier index.html à la racine de votre repo
- **Afficher** les **statut** du repo.
- **Valider** vos modifications
- **Modifier** le fichier index.html (en tête html, <h1> titre +<p> text)
- **Valider** vos modifications
- **Ajouter** et valider un fichier basic.css (h1 text en bleu et p background color noir et text en blanc)
- **Annuler** votre dernière validation (hard) et créer à nouveau fichier basic.css (h1 background color noir et text en blanc et <p> text en bleu)
- **Valider** vos modification
- **Ajouter** dans le répertoire js, basic.js
- **Modifier** basic.js afin qu'il ajouter un paragraphe lors d'un click sur le Header

Git Remote

Architecture décentralisée



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

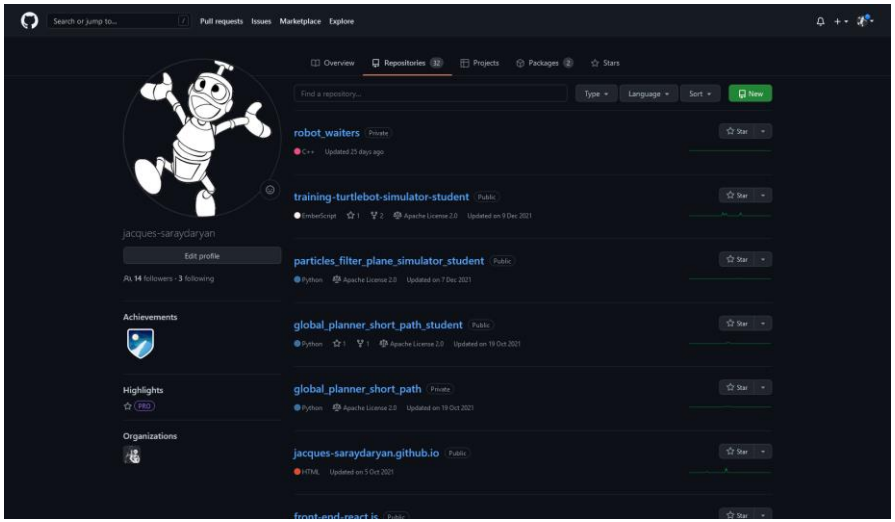
Remote

GIT Hub

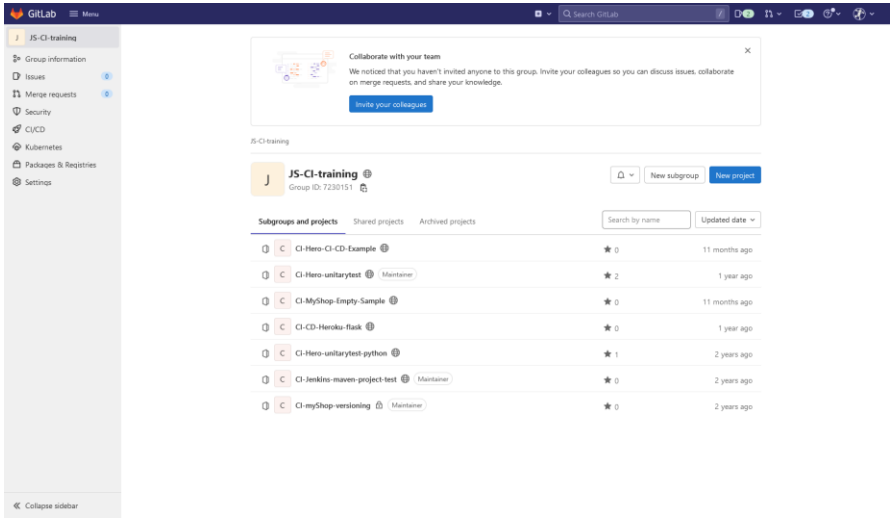
GIT Lab

Remote

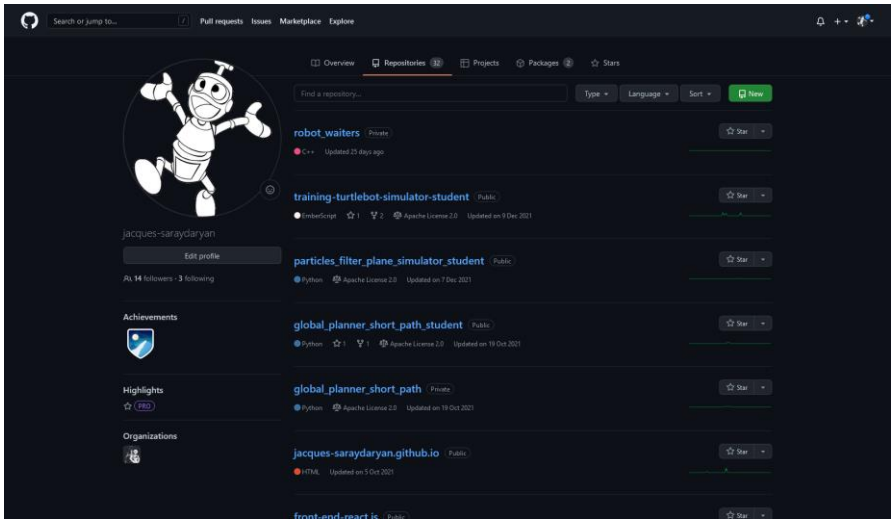
GIT Hub



GIT Lab

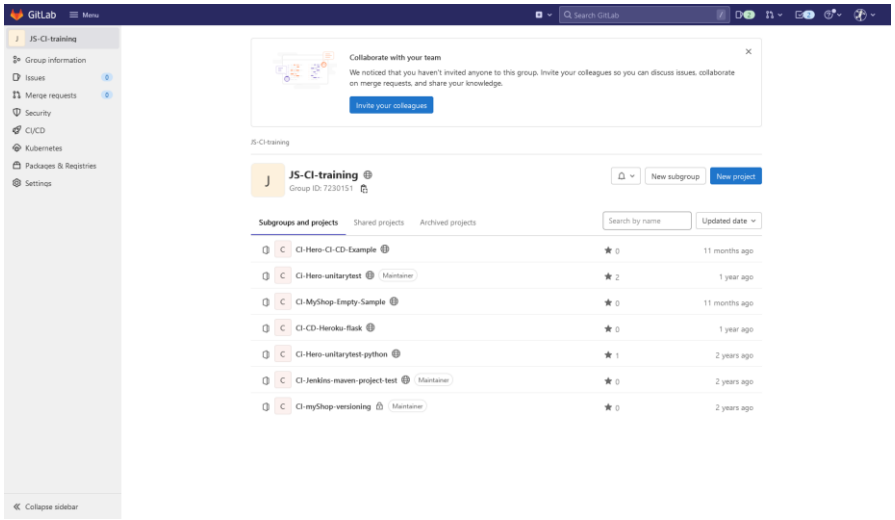


GIT Hub



Introduction Live

GIT Lab



Introduction Live

Remote

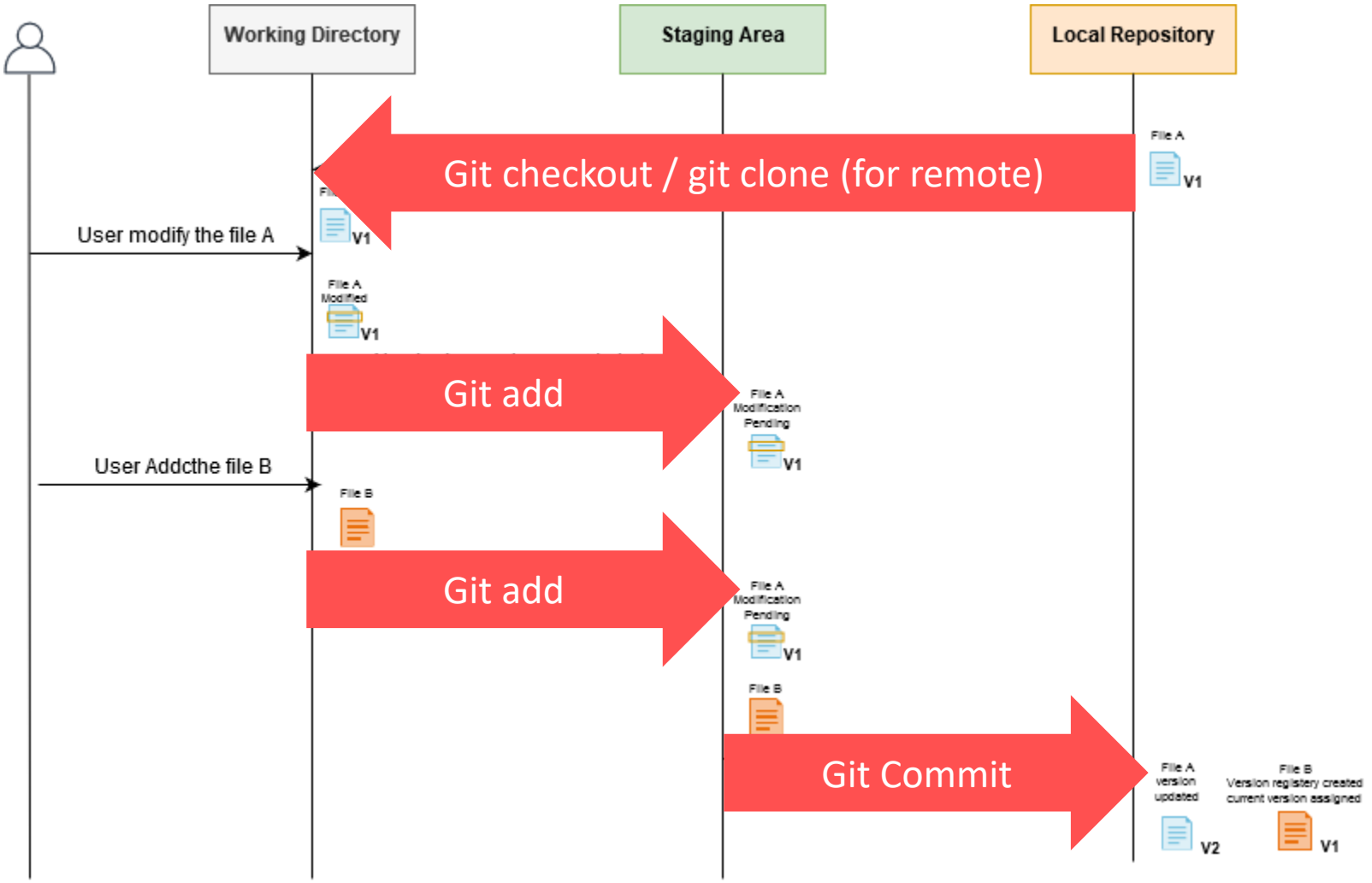
- Récupération des données distantes **`git pull`**

```
$ 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
```

Remote


Working Directory
Staging Area
Local Repository
Git checkout / git clone (for remote)
File A
V1
User modify the file A
File A
Modified
V1
Git add
File A
Modification
Pending
V1
User Addcthe file B
File B
Git add
File A
Modification
Pending
V1
File B
Git Commit
File A
version
updated
V2
File B
Version registery created
current version assigned
V1

Rappel sur la sécurité

HTTPS

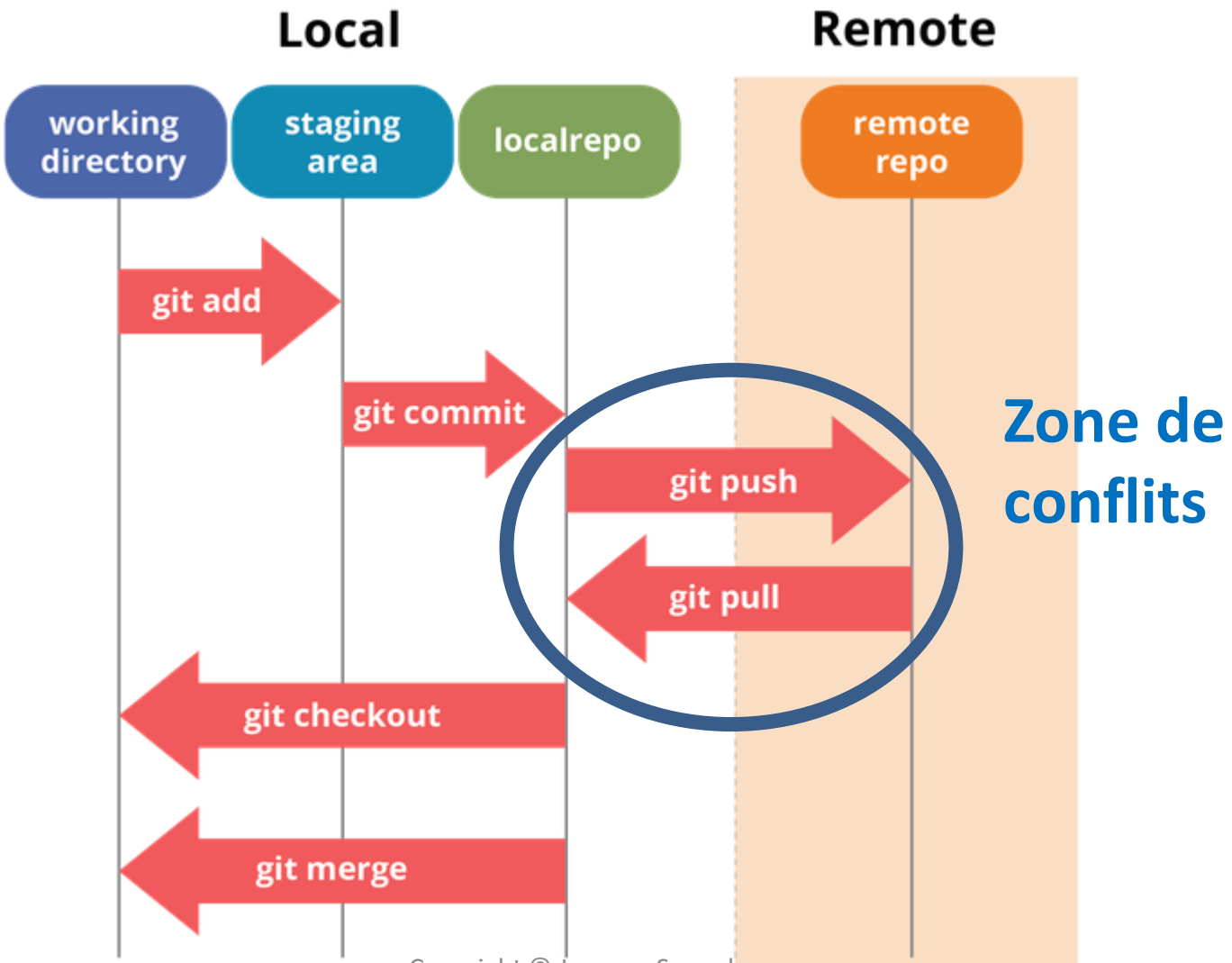
SSH

Certificats

Signature Numérique

Chiffrement

Remote



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

Remote

- Annulation des modifications (remote) **`git revert HEAD`**
  - Annulation d'un commit publié sur le repo remote

```
$ git revert 5478cc1
```

  - Attention permet de remettre votre working directory dans l'état du commit `5478cc1`
  - Il est nécessaire après de valider ce retour en arrière

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

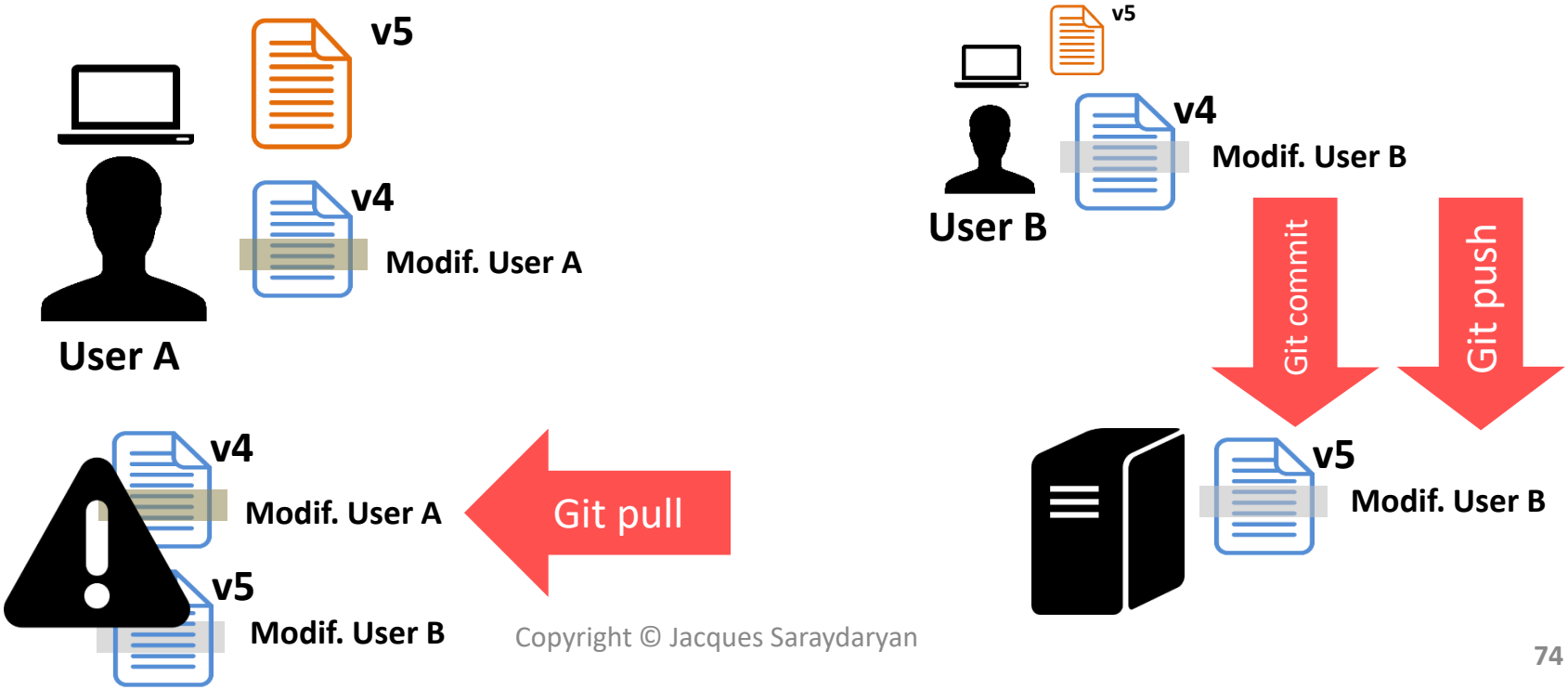
Exercice

- Dans votre repo. myFirstRepo
- **Modifier** votre fichier **A**
- **Valider** votre modification
- **Exporter** votre modification
- **Modifier** votre fichier **B**
- **Annuler** la modification effectuée
- **Modifier** votre fichier **B**
- **Valider** votre modification
- **Annuler** votre modification **validée**
- **Modifier** votre fichier **B**
- **Valider** votre modification
- **Exporter** votre modification
- **Annuler** votre modification exportée

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
  - Merge des informations

User A

User B

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

```
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
  - Merge des informations

```
$ 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
  - Merge des informations

fileToMerge

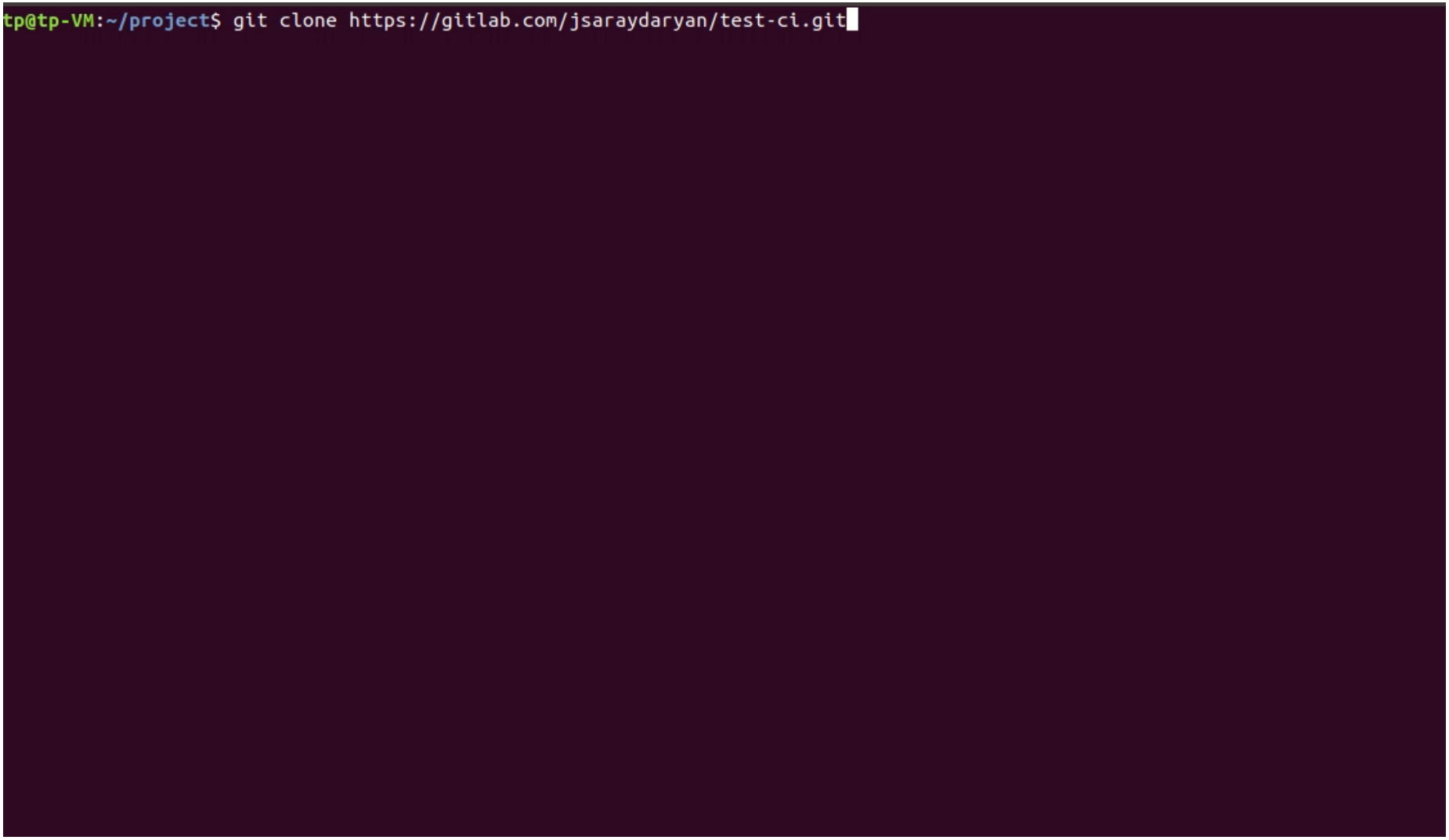
```
Hello the is the fileToMerge
<<<<<<< HEAD
I am A i modified the file here
=======
I am b and i modified the file in
the middle
>>>>>>>
6d7d209bf5efef05ff4d90790be8f56fe6468a00
This is the text of end of the file
```

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



https://youtu.be/FccI-XW-QLM

Exercice

- En binôme
- Créer un repo. Gitlab
- Autoriser votre binôme a y accéder
- Chaque binome (UserA et UserB) créer les fichiers ci-dessous
- Valider vos modifications (UserB puis UserA)
- Résoudre les conflits

fileA

```
Head of userA
Middle of the file
```

fileB

```
Head of the file
Middle of the file
End of the file
```

fileA

```
Middle of the file
End user B
```

fileC

```
Head of the file
Middle of the file
End of the file
```

User A

Utilisateur qui commit et push tjs en second devant résoudre les conflits

User B

Utilisateur qui commit et push tjs en premier

Exercice

- En binôme
- Créer un repo. Gitlab
- Autoriser votre binôme a y accéder
- Créer les fichiers suivants dans votre repo.

fileA

```
Head of the file
Middle of the file
End of the file
```

fileB

```
Head of the file
Middle of the file
End of the file
```

fileC

```
Head of the file
Middle of the file
End of the file
```

- Apporter les modifications suivantes

fileA

```
Head userA
Middle of the file
End user B
```

fileB

```
Head of the file
Middle A
Middle B
End of the file
```

fileC

```
Head of the file
Middle of the file
End A
End B
```

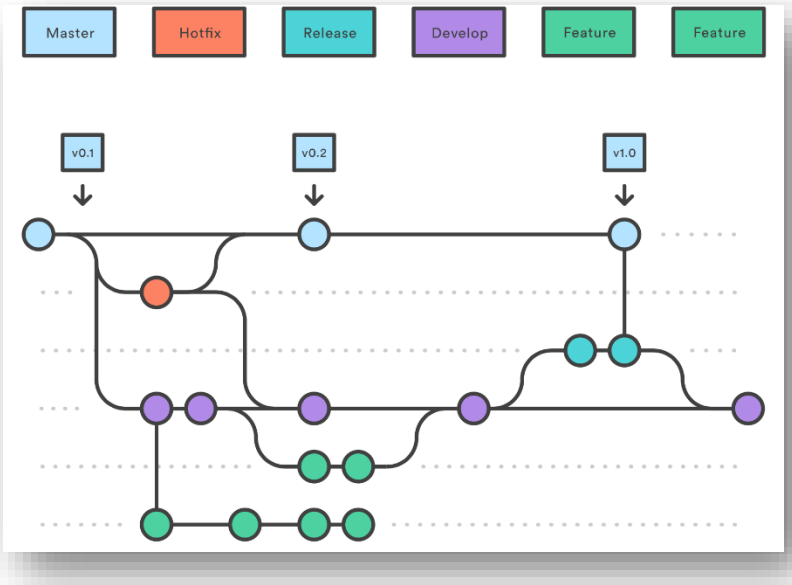
User A

Utilisateur qui commit et push tjs en second devant résoudre les conflits

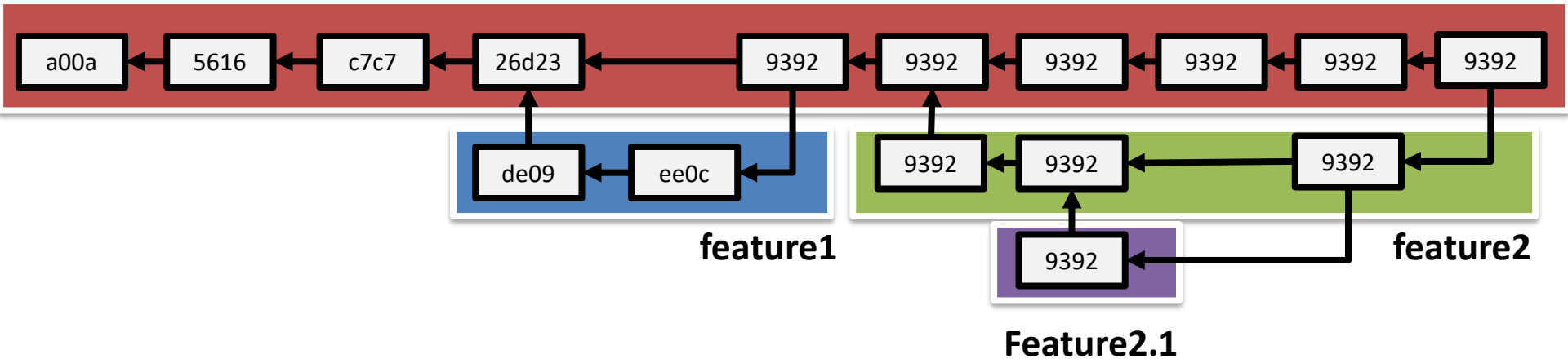
User B

Utilisateur qui commit et push tjs en premier

Les branches

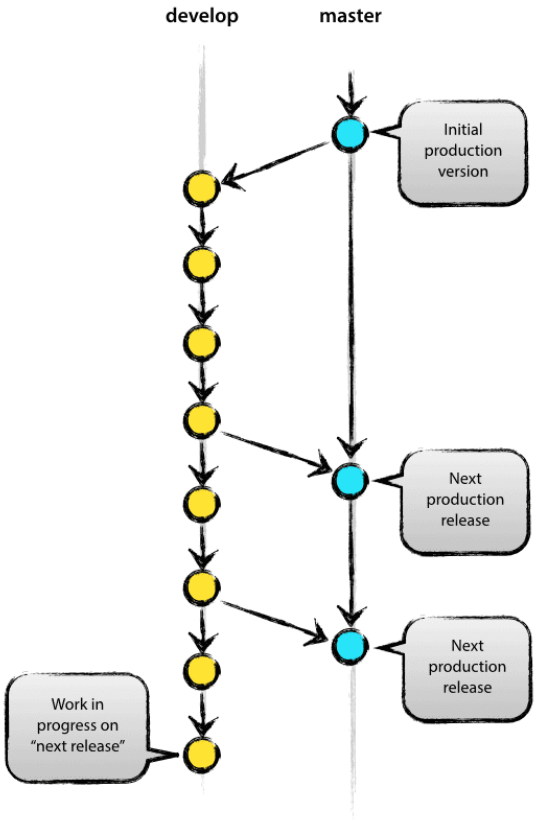


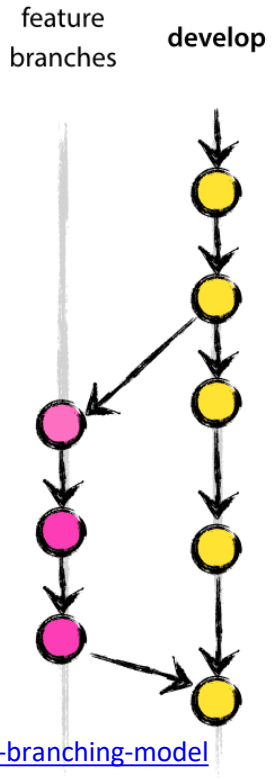
https://blog.xebia.fr/2018/03/28/gitflow-est-il-le-workflow-dont-jai-besoin/



Les branches

- Bonne pratique pour le travail avec les branches
  - Tout développement non finalisé doit être fait sur une branche
  - Toutes nouvelles versions doivent être effectuées sur une branche
  - Toutes fonctionnalités d'une version doivent être effectuées sur une branche

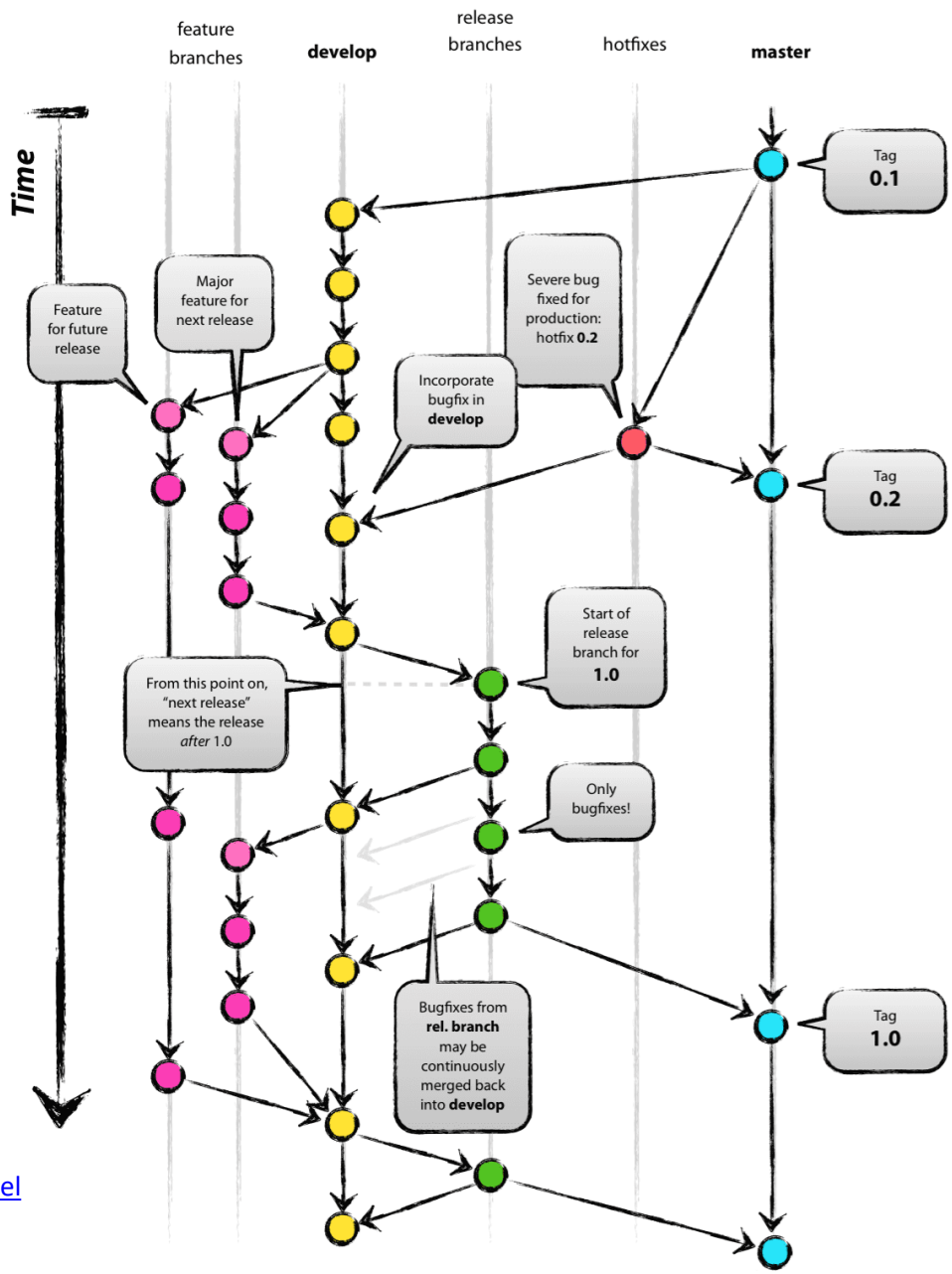




Author: Vincent Driessen
Original blog post: http://nvie.com/posts/a-succesful-git-branching-model
License: Creative Commons BY-SA

Les branches

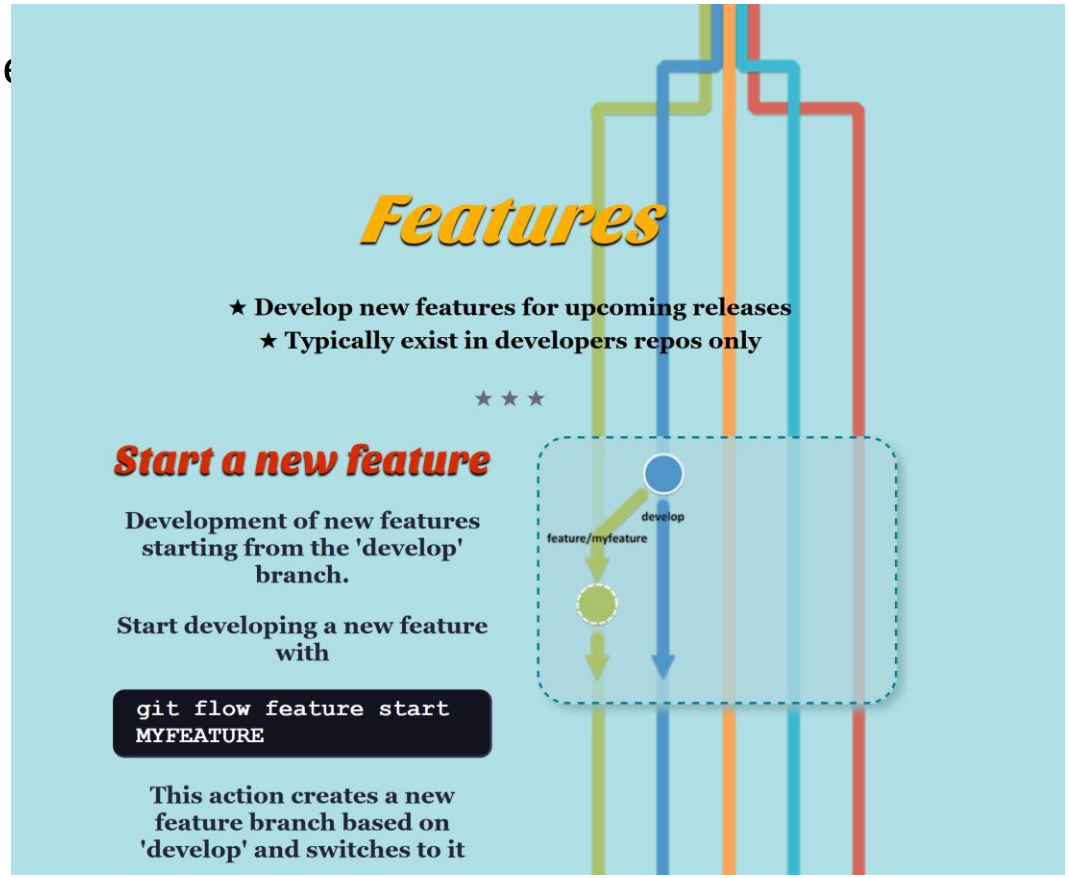
- Bonnes pratiques pour le travail avec les branches



Author: Vincent Driessen
Original blog post: http://nvie.com/posts/a-succesful-git-branching-model
License: Creative Commons BY-SA

Les branches

- Bonnes pratiques pour le travail avec les branche

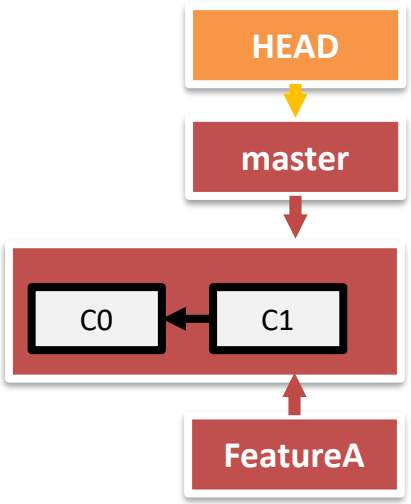


https://danielkummer.github.io/git-flow-cheatsheet/

Les branches

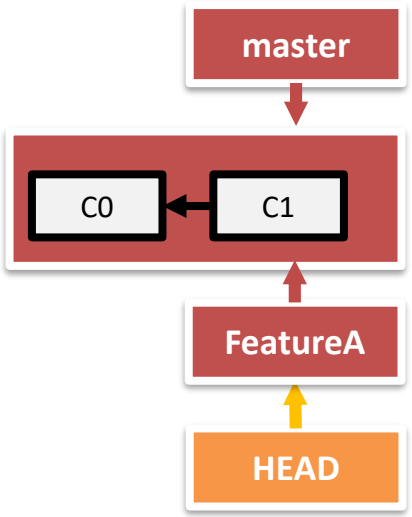
- Création d'une branche

```
$ git branch FeatureA
```



- Sélection de la branche

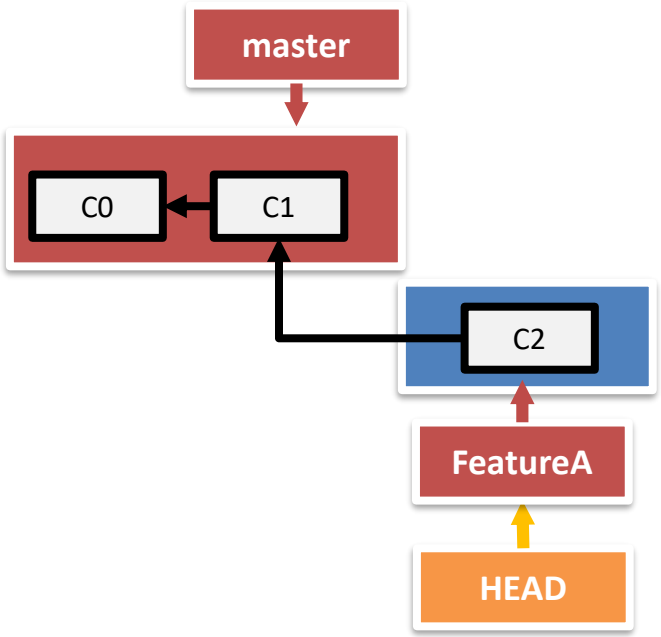
```
$ git checkout FeatureA
```



Les branches

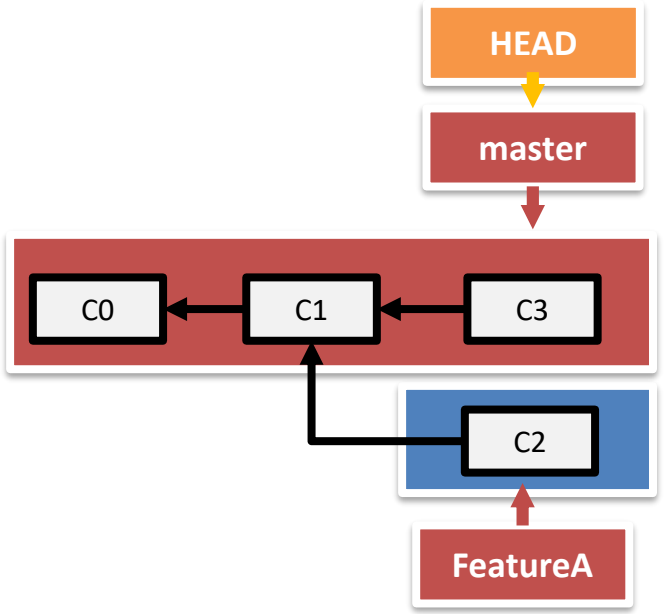
- Modification de la branche

```
$ git checkout FeatureA
$ nano index.html
$ git commit –a –m "new feature added"
$ git push
```



- Retour sur la branche master et modif.

```
$ git checkout master
$ nano custom.css
$ git commit –a –m "small hotfix"
$ git push
```



Les branches

- Fusion branche / master en local

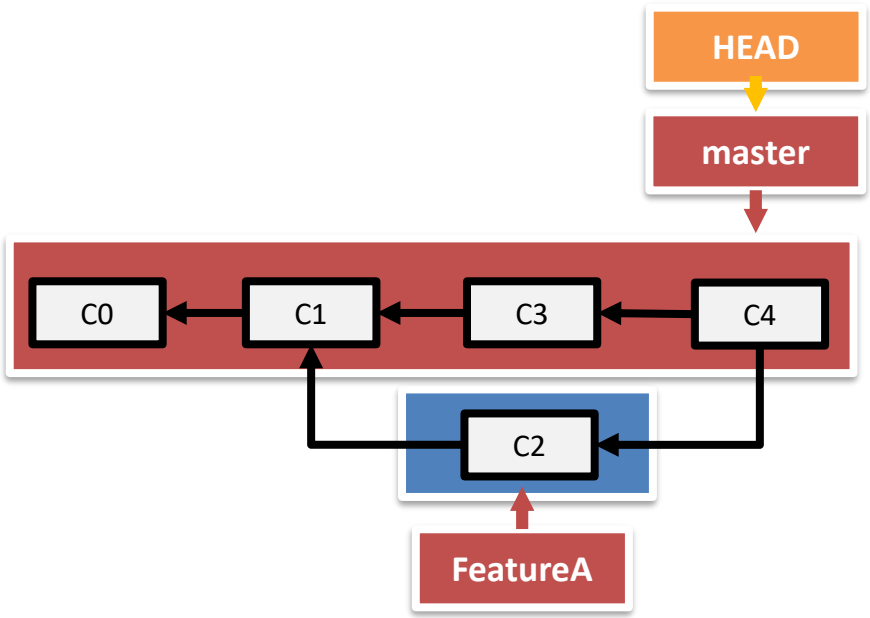
```
$ git checkout master
$ git merge FeatureA
Auto-merging README
Merge made by the 'recursive' strategy.
 README | 1 +
 1 file changed, 1 insertion(+)
```

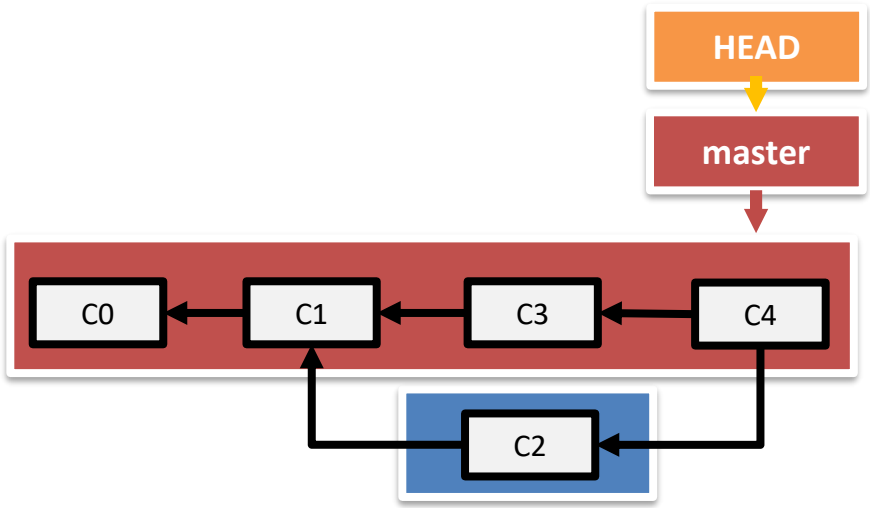
- Suppression d'une branche localement

```
$ git branch -d FeatureA
```

- Suppression d'une branche remote

```
git push origin --delete FeatureA
```





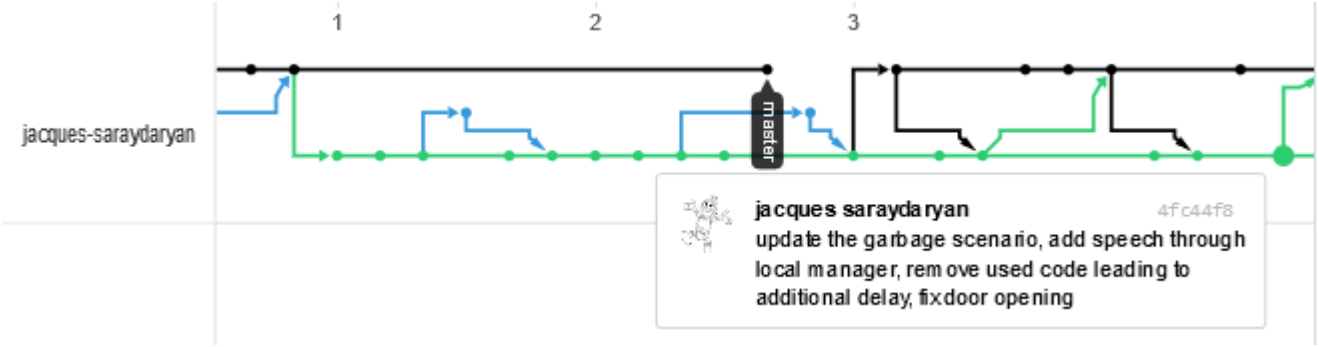
Les branches

- Visualisation des branches du repo.

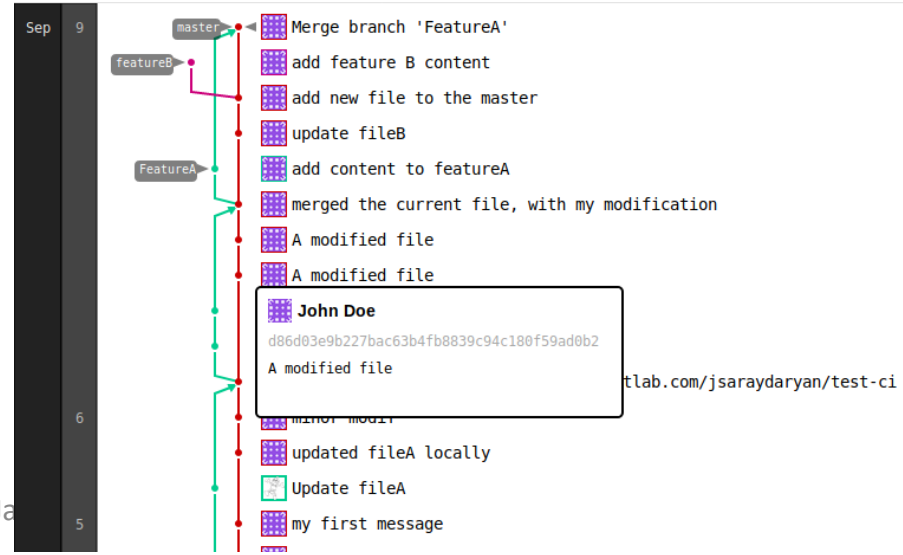
```
$ git log --graph --oneline --all
*   3308e57 (origin/dev, dev) merge featureB on dev and minor correction
|\
| * 284c683 (featureB) add buy feature and Launcher
| * 9499ab5 add buy function
* | b03b041 (featureA) add features and launch file
|/
* 740b5dd (origin/master) add shop class
* 6ddea2c (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



Git CheatSheet

https://ndpsoftware.com/git-cheatsheet.html

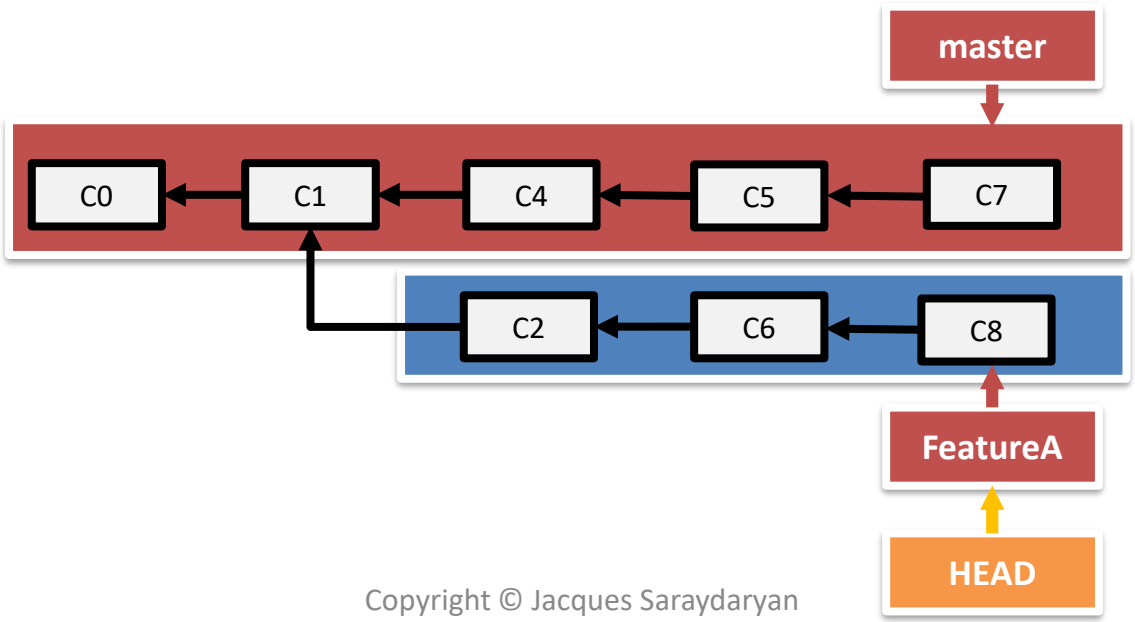
Exercice

- En binôme
- Créer un nouveau repo. Gitlab – **test_branch**
- Ajouter 3 fichiers **fileA**, **fileB**, **fileC**
- Modifier le **fileA** et valider
  - **UserA**: Créer une branche **featureA**
  - **UserA**: Modifier le fileA et valider
  - **UserA**: ajouter le fileA1 et valider
  - **UserB:** modifier le fileC et valider (master)
  - **UserB:** Créer une branche **featureB**
  - **UserB:** Modifier fileB et valider
  - **UserB:** Ajouter le fileB1 et valider
  - **UserA**: fusionner le master et la **featureA**
  - **UserB:** modifier le fileC au même endroit que précédemment
  - **UserB:** fusionner le **master** avec la **featureB**

Les branches: Rebase

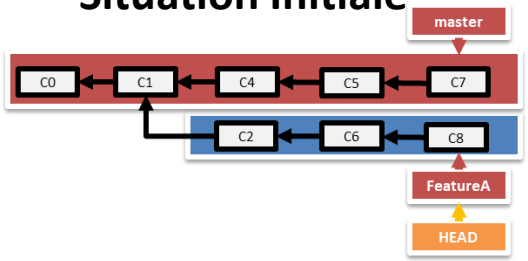
- Repositionner et réecrire/condenser les commits d'une branche
  - Permet de mettre la pointe de la branche sur master
  - Rejoue l'ensemble de opérations de commit d'un seul
    → Nettoyage des commits de la branche courante avant un merge

Situation Initiale

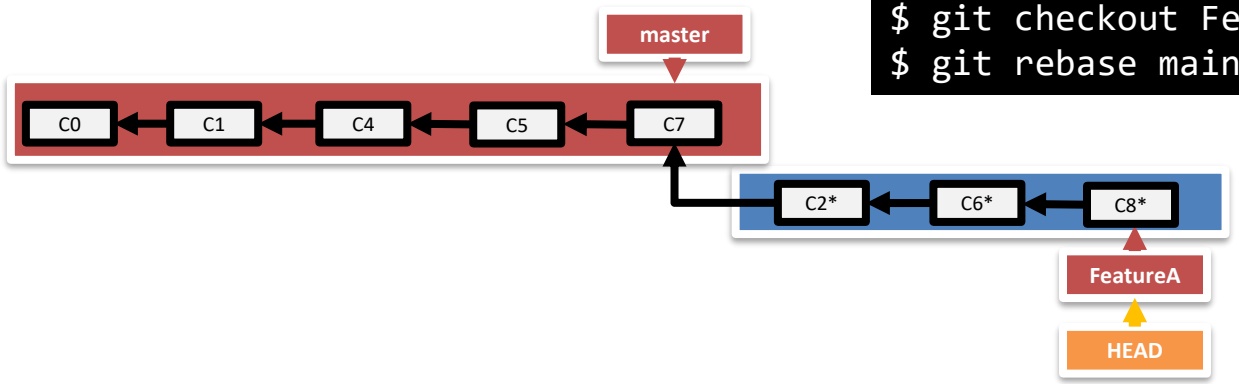


Les branches: Rebase

- Repositionner et réecrire/condenser les commits d'une branche
  - **Situation Initiale**



  - **Situation après un rebase simple**



- Rebase brut

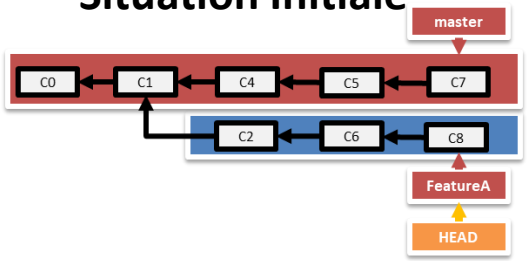
```
$ git checkout FeatureA
$ git rebase main
```

Gestion de versions

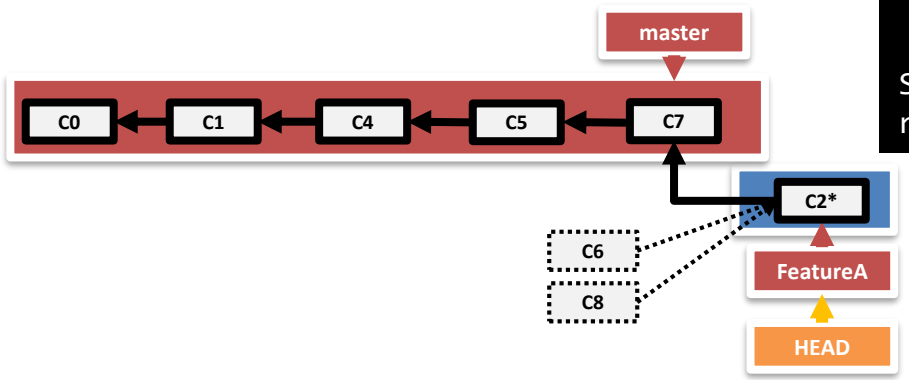
Les branches: Rebase

- Repositionner et réecrire/condenser les commits d'une branche

  - **Situation Initiale**



  - **Situation après un rebase interactif**



- Rebase interactif

```
$ git checkout FeatureA
$ git rebase -i main
```

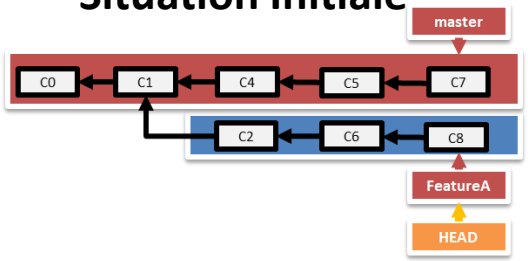
```
pick        C2 First Commit on branch
fixup       C6 Second Commit on branch
fixup       C8 Thrid Commit on branch
```

```
[detached HEAD 464a134] First Commit on branch
 Date: Fri May 20 14:01:58 2022 +0200
 3 files changed, 3 insertions(+)
 …
Successfully rebased and updated
refs/heads/FeatureA.
```

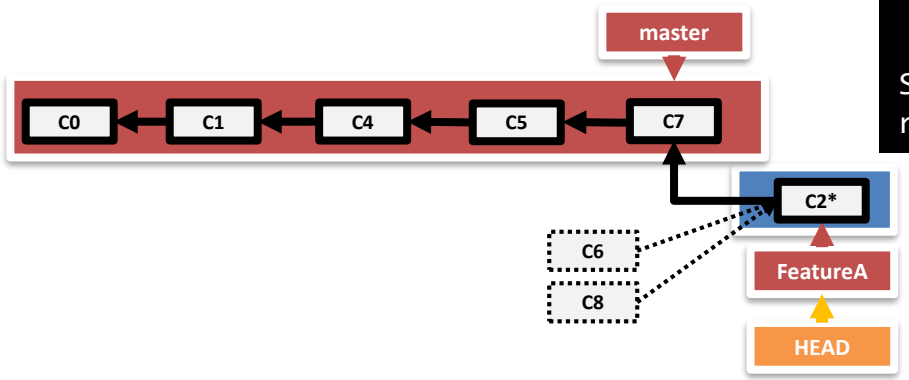
Les branches: Rebase

- Repositionner et réecrire/condenser les commits d'une branche

  - **Situation Initiale**



  - **Situation après un rebase interactif**



- Rebase interactif

```
$ git checkout FeatureA
$ git rebase -i main
```

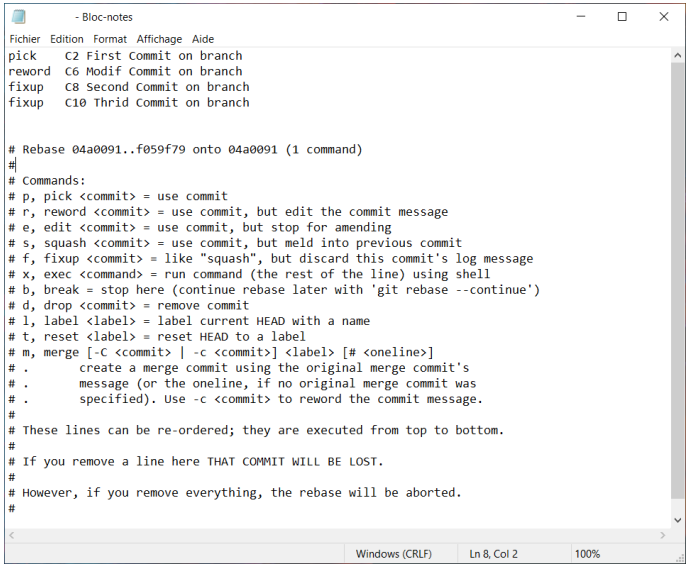
```
pick        C2 First Commit on branch
fixup       C6 Second Commit on branch
fixup       C8 Thrid Commit on branch
```

```
[detached HEAD 464a134] First Commit on branch
 Date: Fri May 20 14:01:58 2022 +0200
 3 files changed, 3 insertions(+)
 …
Successfully rebased and updated
refs/heads/FeatureA.
```

Les branches: Rebase

- Repositionner et réecrire/condenser les commits d'une branche

```
pick        C2 First Commit on branch
reword      C6 Modif Commit on branch
fixup       C8 Second Commit on branch
fixup       C10 Thrid Commit on branch
```

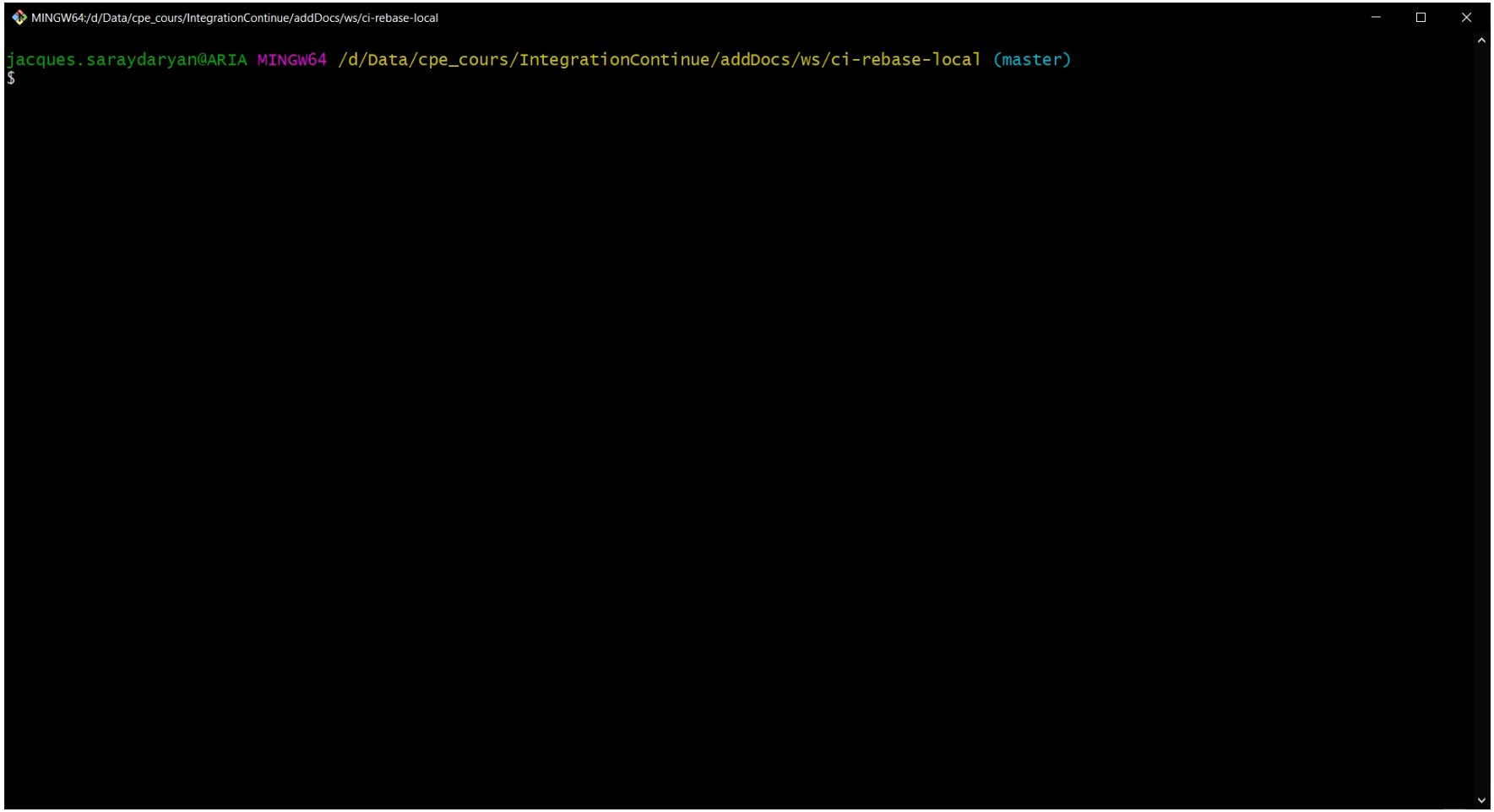



```
# Commands:

# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous
                       commit
# f, fixup <commit> = like "squash", but discard this commit's
                       log message
# x, exec <command> = run command (the rest of the line) using
                       shell
# b, break = stop here (continue rebase later with 'git rebase
                       --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
#           .create a merge commit using the original merge
           commit's
#           .message (or the oneline, if no original merge
           commit was
#           .specified). Use -c <commit> to reword the commit
           message.


# These lines can be re-ordered; they are executed from top to bottom.
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted
```

Les branches: Rebase



Exercice

- Créer un repo. local
- Sur la branche principale
  - 1 Fichier fileA.txt avec du contenu (stage et commit)

- Créer une branche feature1
  - 1 Fichier fileB.txt avec du contenu (stage et commit)
  - 1 Fichier fileC.txt avec du contenu (stage et commit)
  - 1 Fichier fileD.txt avec du contenu (stage et commit)
  - 1 Fichier fileE.txt avec du contenu (stage et commit)

- Sur la branche principale
  - 1 Fichier fileF.txt avec du contenu (stage et commit)

- Sur une branche feature1 → rebase:
  - Garer uniquement les deux premiers commit de la branche

Utilitaires

- Mise en attende des modifications courantes **`git Stash`**

```
$ git status
On branch FeatureA
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:   A.txt
        modified:   C.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

```
$ git stash
Saved working directory and index state WIP on FeatureA: f059f79 FeatureA: All Commits Merged
```

```
$ git status
On branch FeatureA
nothing to commit, working tree clean
```

```
$ git stash pop
On branch FeatureA
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:   A.txt
        modified:   C.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

Utilitaires

- Identifier dans un fichier qui à fait quoi **`git Blame`**

```
$ git blame A.txt
a9702c69 (bip   2022-05-20 11:18:58 +0200 1) sdqdsqdsqd
3a6305cd (Jdoe  2022-05-20 13:45:52 +0200 2) new content at the end
1d3eabbb (Smith 2022-05-20 13:54:40 +0200 3) Feature A new content after the end
1d3eabbb (bip   2022-05-20 13:54:40 +0200 4) Feature A thrid commit after rebase
```

- Lister toutes les opérations git effectuées **`git reflog`**

```
$ git reflog
1d3eabb (HEAD -> feature_A) HEAD@{1}: commit: merge operation during rebase
4dd4cb8 HEAD@{2}: rebase (fixup): # This is a combination of 5 commits.
b70c5be HEAD@{3}: rebase (fixup): # This is a combination of 4 commits.
cc7ed2b HEAD@{4}: rebase (fixup): # This is a combination of 3 commits.
053be27 HEAD@{6}: rebase (pick): Feature A first Commit
3a6305c (master) HEAD@{7}: rebase (start): checkout master
c0ae5f7 HEAD@{8}: rebase (abort): updating HEAD
3a6305c (master) HEAD@{9}: rebase (start): checkout master
c0ae5f7 HEAD@{10}: commit: Feature A thrid commit after rebase
435ab6e HEAD@{11}: commit: Feature A second commit after rebase
…
```

Utilitaires

- Mettre à jour les information depuis remote **`git Fetch`**

```
$ git fetch
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 262 bytes | 131.00 KiB/s, done.
From https://gitlab.com/js-ci-training/ci-hero-unitarytest-python
   77e8b6d..470796d  master     -> origin/master
```

- Permet de récupérer les modifications effectuées sur remote
  - **Commits**
  - **Branches**
  - **Fichiers**
  - **Références**

Questions ?