Git, rappels et éléments avancés

Sébastien Jean

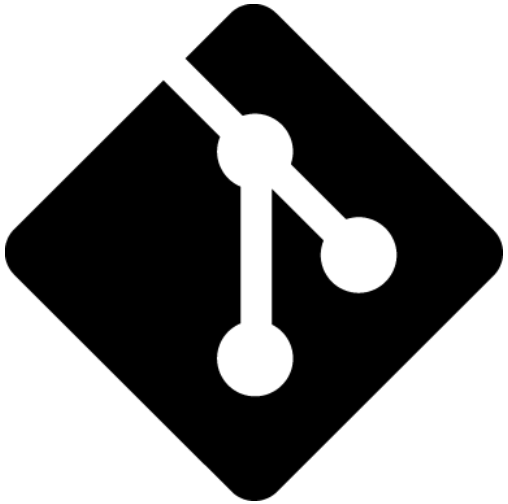
IUT de Valence
Département Informatique

v1.0, 13 septembre 2023

IUT Valence
Université Grenoble Alpes

Intérêts de la gestion de versions ?

- **Sauvegarder le code** (localement, à distance)
- **Maintenir un historique des modifications** (versions)
- Pouvoir facilement **identifier les modifications**, **revenir en arrière**
- **Expérimenter** sereinement et indépendamment (branches)
- **Collaborer** (dépôts partagés)



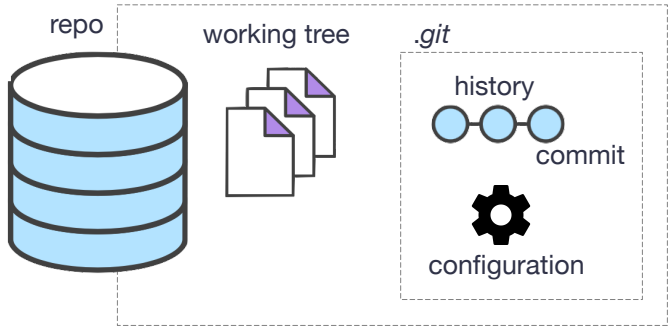
IUT Valence
Université Grenoble Alpes

Comment versionner ?

- Choisir un **système de gestion de versions** (SCM)
  - svn, hg (mercurial), git
- Créer un dépôt local et gérer les versions via les **outils associés**
  - En ligne de commande ou via des assistants d'IDE
- **Héberger** un dépôt distant sur un serveur (communautaire ou privé)
  - BitBucket, pour hg
  - Github ou Gitlab, pour git
  - Sourceforge, pour svn

IUT Valence
Université Grenoble Alpes

Notion de dépôt



- **Dépôt** (*repository* ou *repo*)

  → **ensemble de fichiers** (stockés localement ou sur une plateforme d'hébergement) utiles pour la gestion de versions de code

  - *Working Tree* : arborescence du code **courant**
  - *History* : historique des versions (*commits*), constitué d'une ou plusieurs **branches**
  - *Configuration* : propriétés de configuration du dépôt (identité, ...)

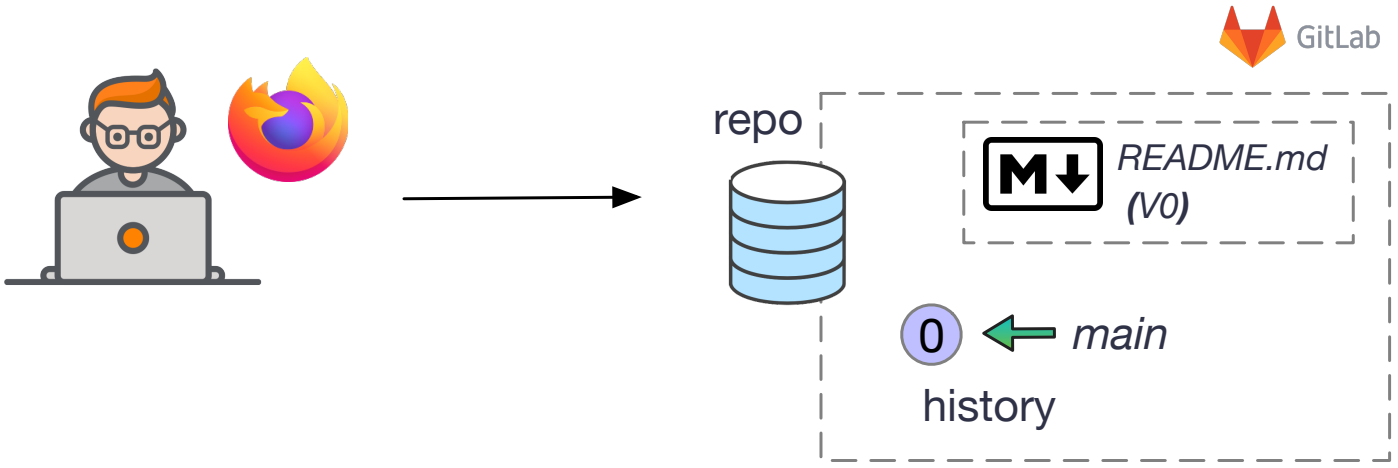
IUT Valence Université Grenoble Alpes

Notion de commit

- Un **commit** permet de répondre à un certain nombre de questions sur la nouvelle version :
  - Qui ? → le développeur (*committer*) est identifié par un **nom/mail**
  - Quand ? → **horodatage**
  - Quoi ? → ensemble des **modifications** intégrées à la nouvelle version
  - Où ? → **référence au(x) commit(s) précédent(s)**
  - Pourquoi ? → **message** destiné aux autres développeurs, résumant ce qui change, éventuellement accompagné de détails.
- A chaque commit est associé un identifiant unique sur 20 octets

IUT Valence
Université Grenoble Alpes

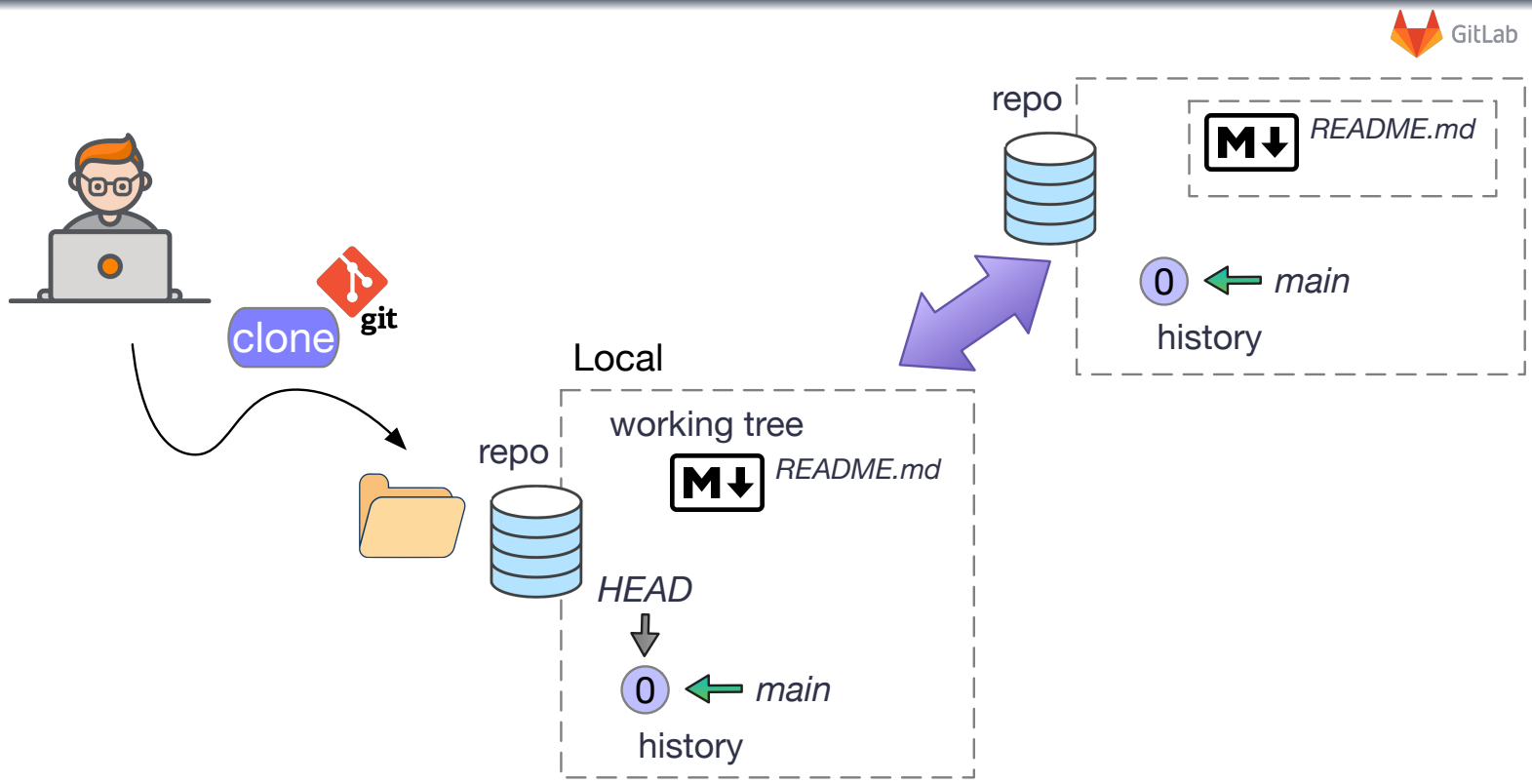
Héberger un dépôt distant



- Un **dépôt distant** est hébergé sur un serveur (ici un serveur *gitlab*)
  - En général, ce dépôt est initialisé avec un fichier `README.md` (qui décrit à quoi sert ce dépôt) qui constitue le premier commit
  - Un dépôt distant est dit ***bare***, il ne contient pas de *working tree* mais simplement l'historique
    - Le développement est local, il ne peut pas s'effectuer sur le serveur

IUT Valence Université Grenoble Alpes

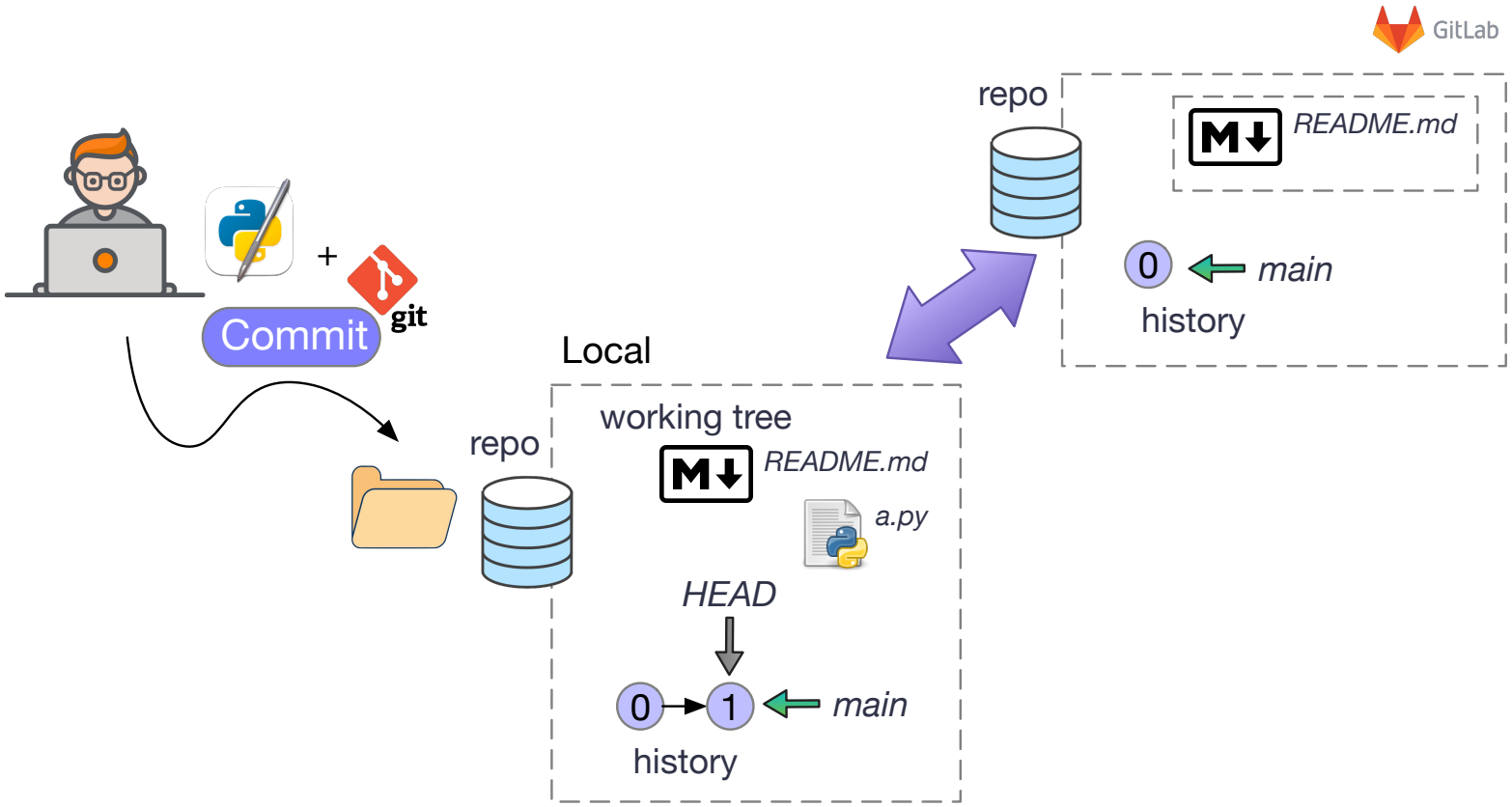
Travailler localement avec un clone



- Un **clone** est une reconstruction locale d'un dépôt distant (le développement est toujours local)
  - L'historique est reproduit intégralement, le *working tree* est reconstruit en rejouant intégralement l'historique
  - Le clone local et son dépôt origine restent **liés**

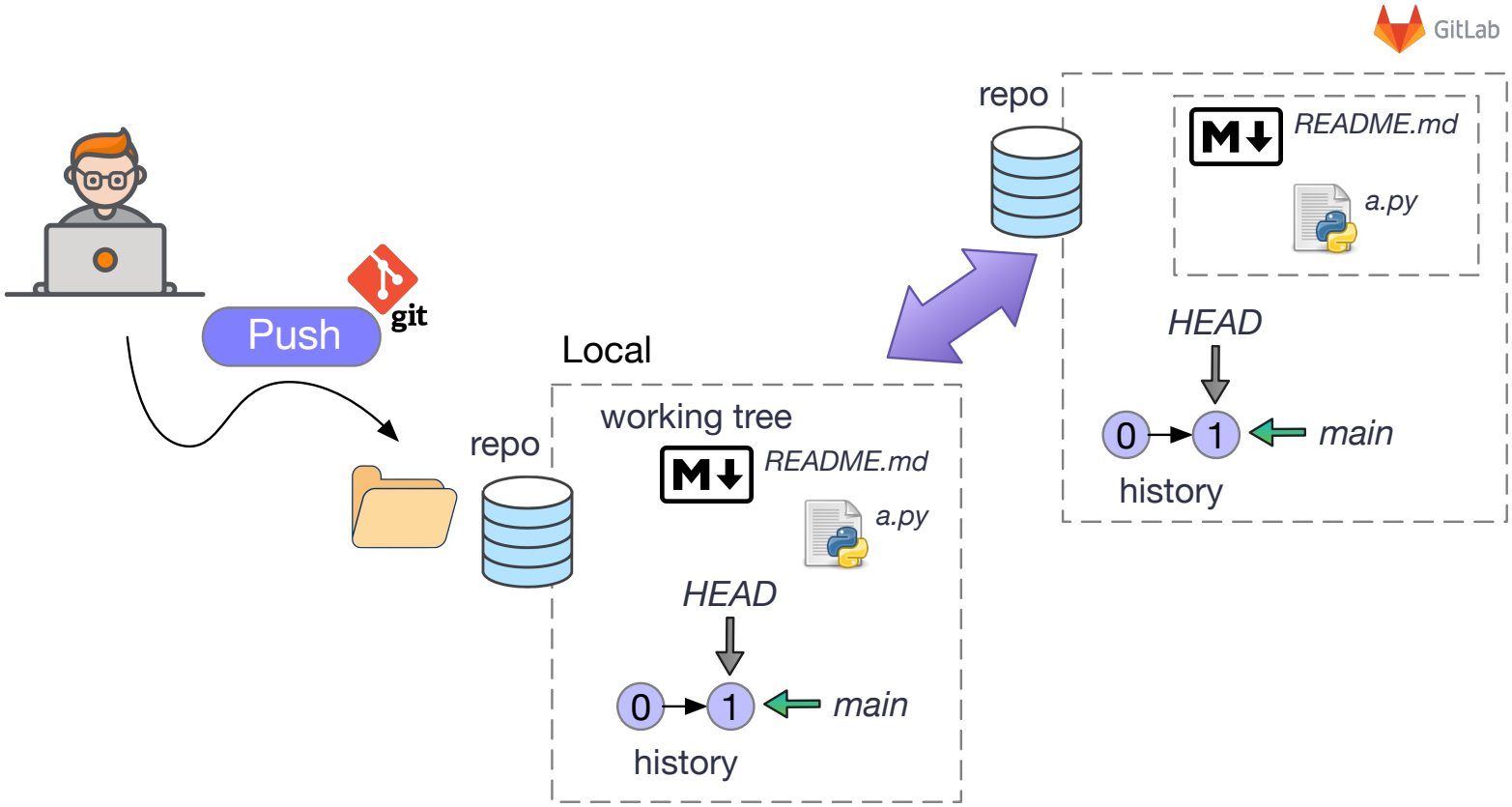
IUT Valence Université Grenoble Alpes

Travailler localement avec un clone (suite)



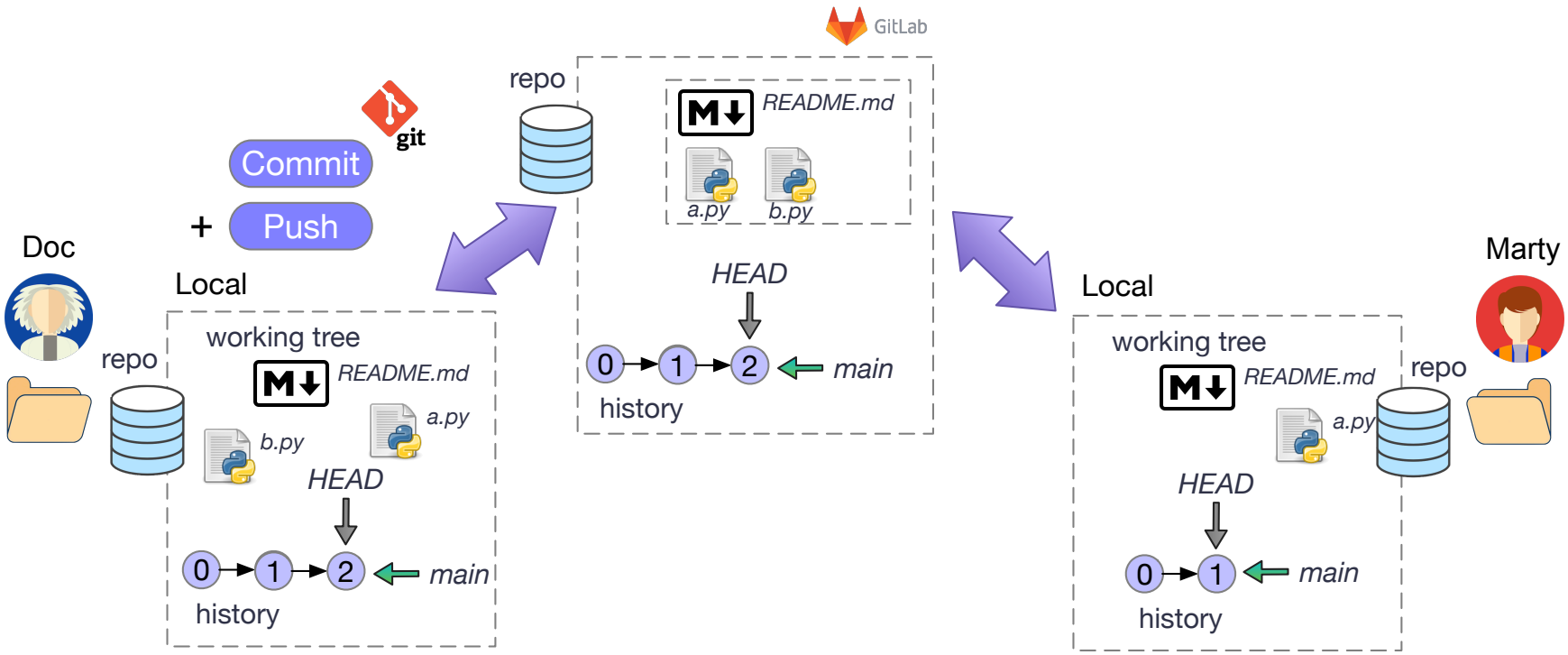
- Les commits sont toujours locaux, **sans communication avec le serveur**

Travailler localement avec un clone (suite)



- Les commits locaux sont explicitement envoyés sur le serveur (Push)

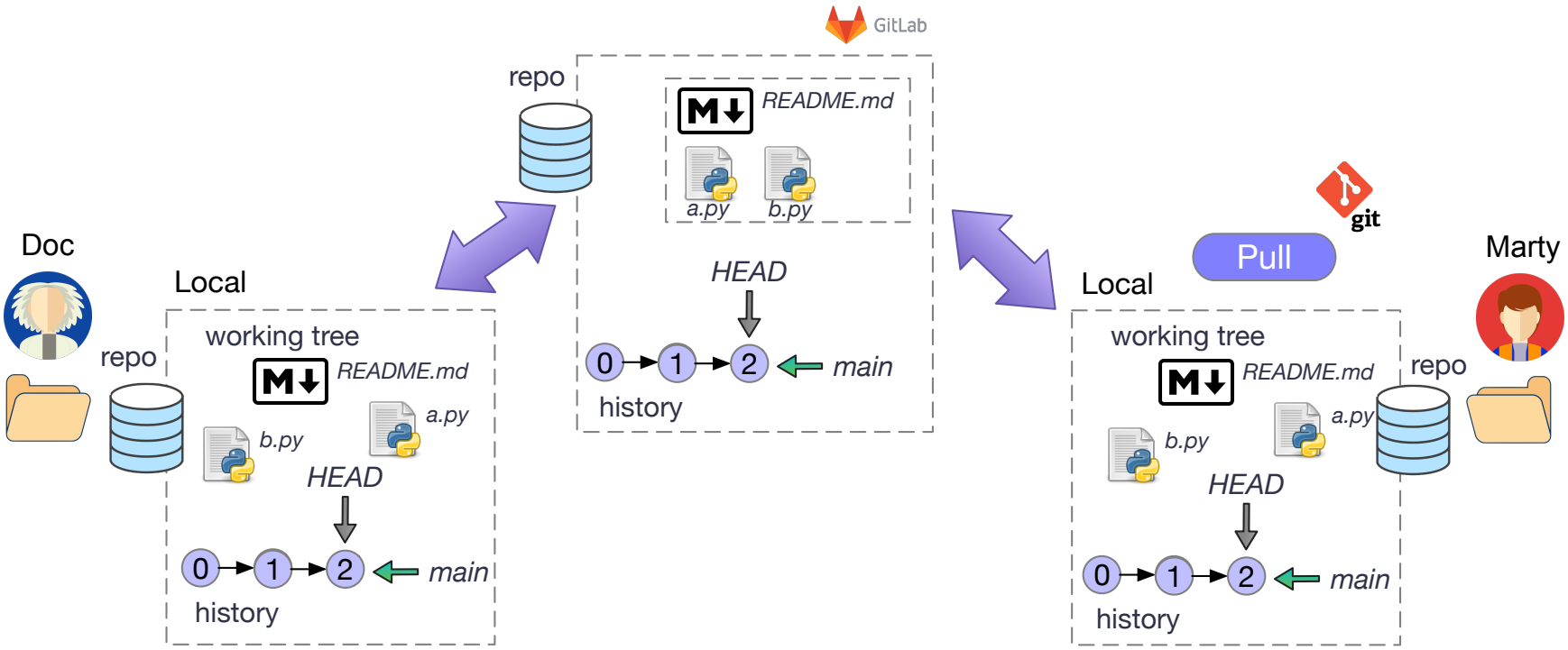
Se remettre à jour avec le dépôt distant



- *Doc* produit une nouvelle version locale qu'il envoie au serveur
- *Marty* n'est pas averti de l'existence de cette nouvelle version

IUT Valence Université Grenoble Alpes

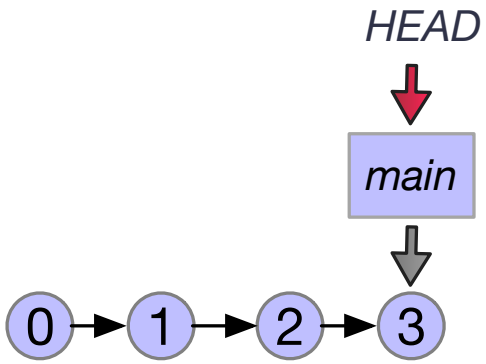
Se remettre à jour avec le dépôt distant



- La synchronisation descendante (`pull`), permet à *Marty* de se mettre localement à jour
- La mise à jour peut se faire de manière simple et autonome, ou faire apparaitre des situations de conflits à résoudre

IUT Valence Université Grenoble Alpes

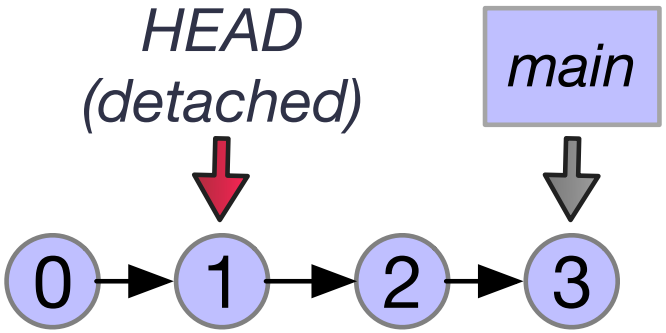
Notion de branche



- Une **branche** est une succession nommée de commits
  - ici, la branche s'appelle `main`, c'est aussi le nom de la référence sur le dernier commit
- Rappel :
  - La référence `HEAD` identifie la version courante, version contenue dans le *working tree*, sur laquelle on travaille actuellement
  - C'est cette version qui sert de parent au prochain commit
  - Par défaut elle suit la référence de la branche

IUT Valence Université Grenoble Alpes

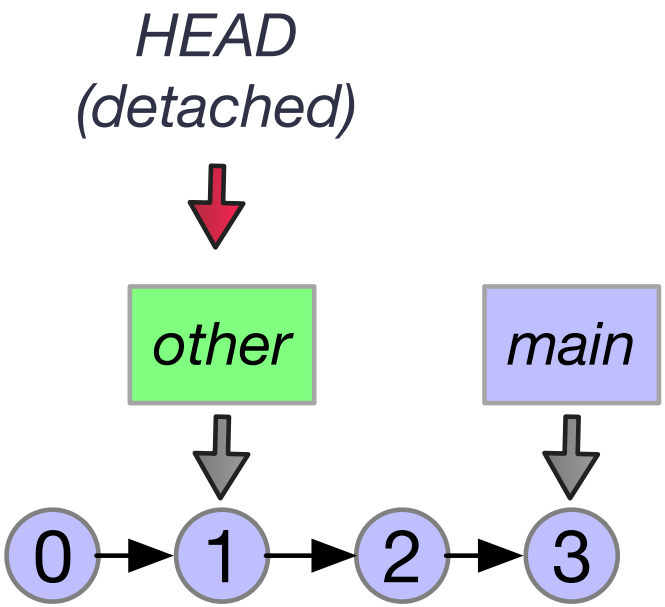
Déplacement de HEAD



- La référence `HEAD` peut être **déplacée** via la commande `git checkout`
- Le déplacement crée une situation dite d'état détaché (*detached*)
  - Il n'est pas possible de créer des nouveaux commits à partir de cet endroit puisqu'il y a déjà des commits en aval

IUT Valence Université Grenoble Alpes

Création d'une branche

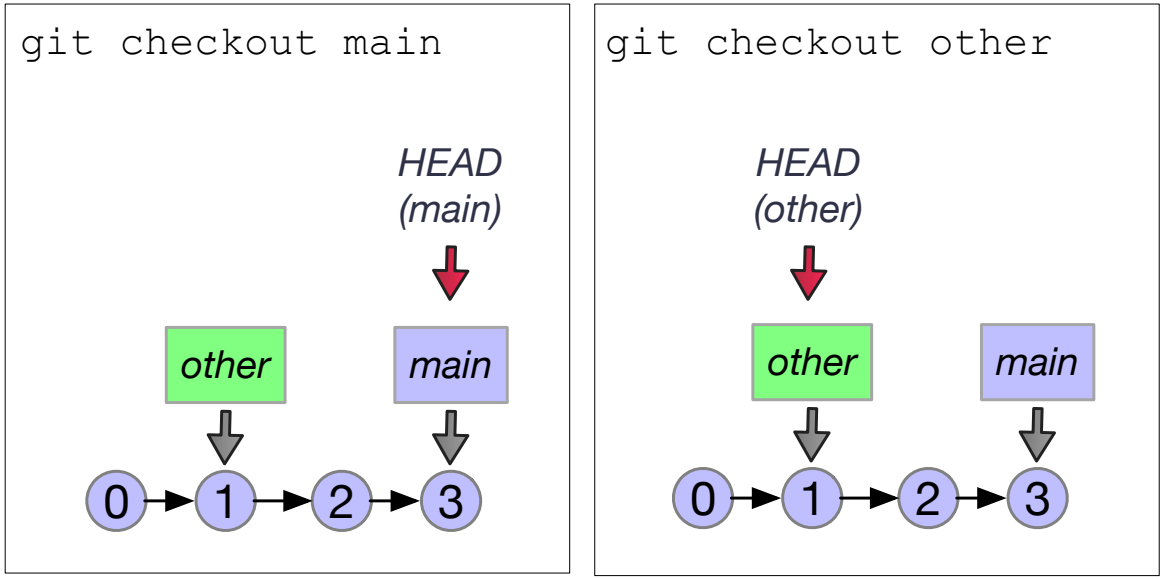


- la commande `git branch` permet de **créer une nouvelle branche**, qui permet de poursuivre un *historique parallèle*
  - Ici, la branche s'appelle `other`
  - Elle contient les commits `0` et `1` uniquement, l'historique est commun sur ces 2 commits pour les 2 branches

IUT Valence Université Grenoble Alpes

Déplacement sur une branche

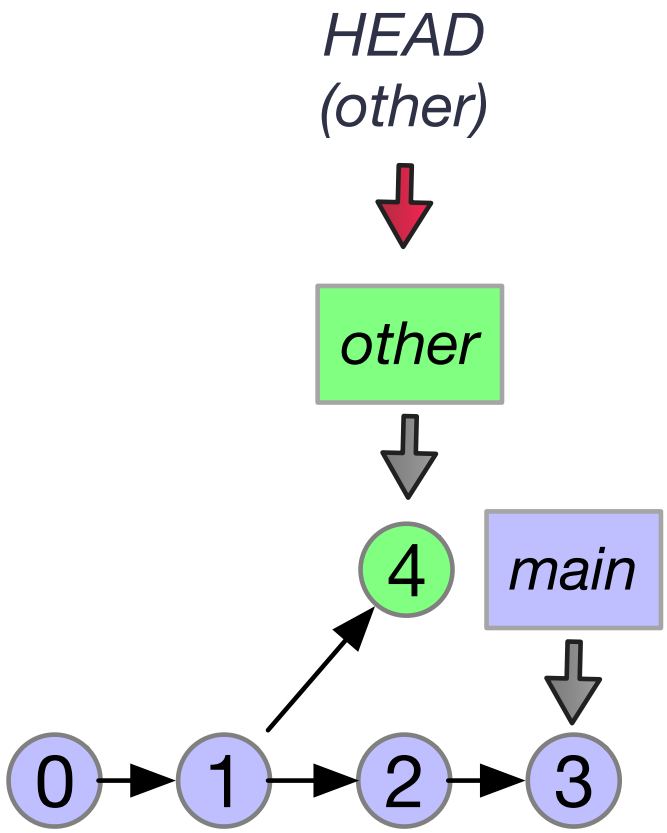
- Pour poursuivre l'historique d'une branche, il faut s'y déplacer explicitement
  - Le déplacement sur une branche s'effectue via la commande `git checkout`
  - La branche devient active et la référence `HEAD` se replace sur le dernier commit.



- Remarque : il existe un raccourci pour créer une nouvelle branche et directement s'y déplacer

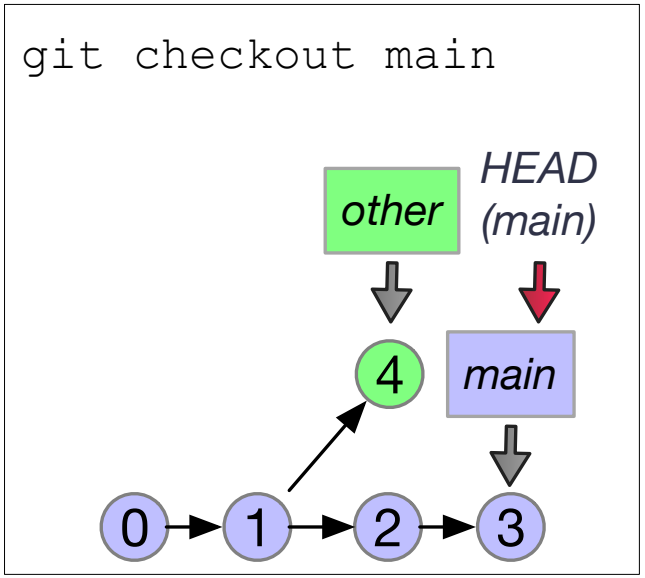
Divergence

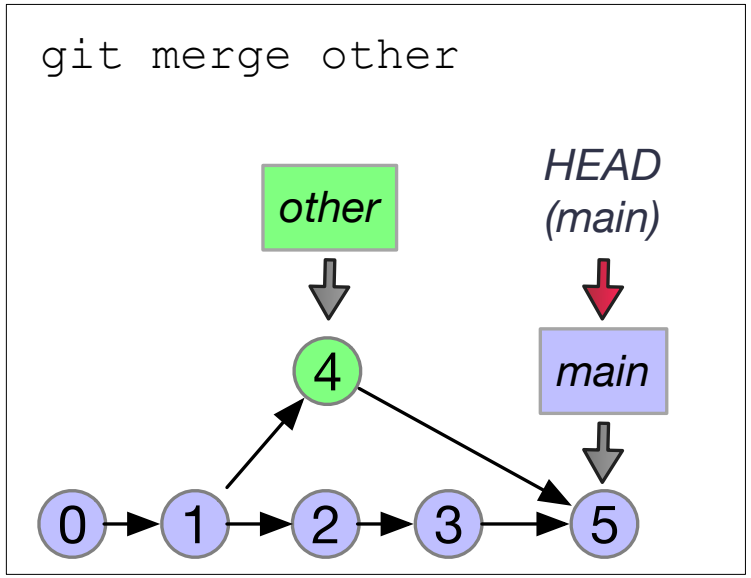
- Si un commit est effectué sur la branche `other`, il vient se placer derrière le commit `1`
- Il y a **divergence**, deux historiques évoluent en parallèle



Fusion

- Une **fusion** est une fermeture de la divergence entre 2 branches
- Elle s'opère depuis la branche vers laquelle on veut intégrer les commits de la branche à fusionner, via la commande `git merge`





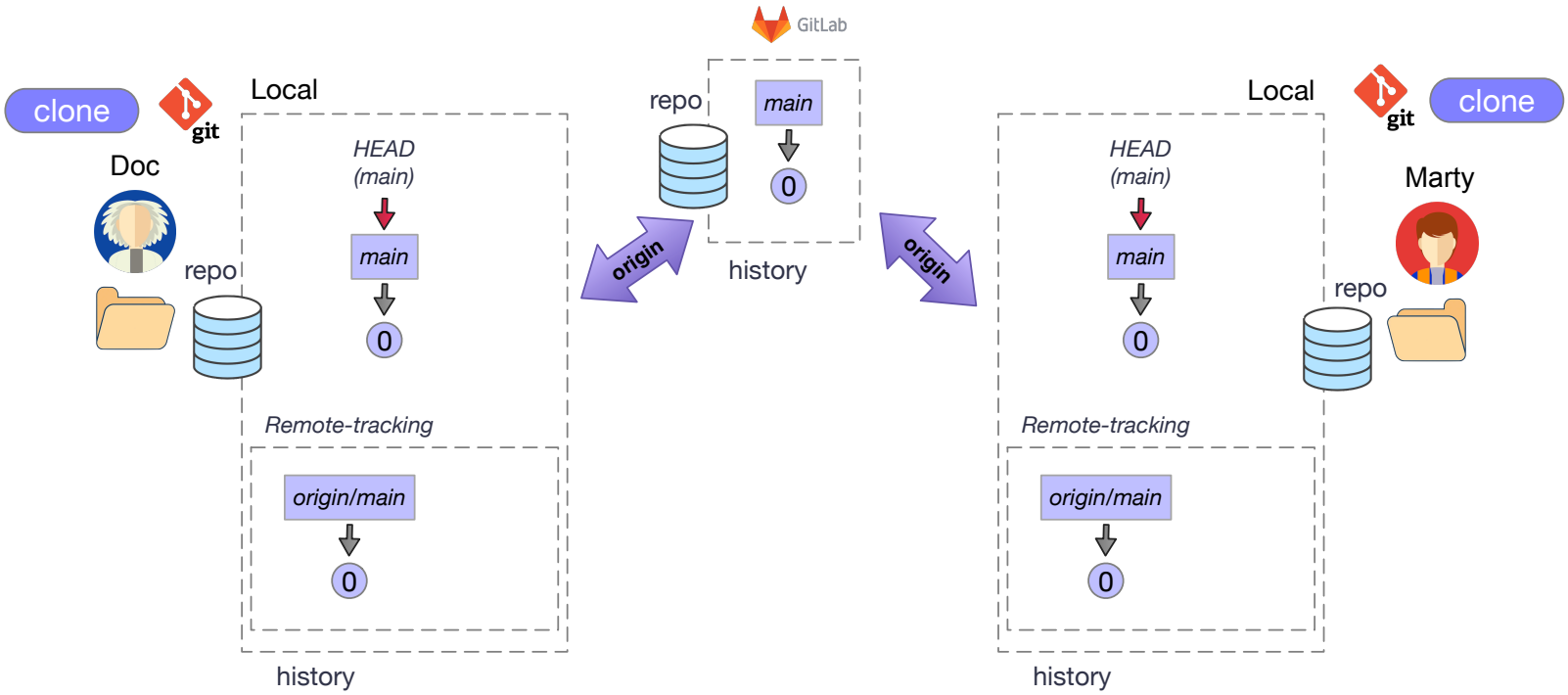
Fusion (suite)

- Pour réaliser la fusion, *git* :
  1. Identifie le point de divergence (ici le commit 1)
  2. Identifie les changements en concurrence (**conflits**) entre les commits des 2 branches
  3. S'il n'y a pas de conflit, la fusion est automatique
     - Le commit de fusion ne contient pas de nouvelle modification, il ferme simplement la divergence en déclarant 2 parents, on peut lui associer un message
  4. S'il n'y a des conflits, ils doivent être **résolus par le développeur**
     - Chaque portion de fichier en conflit présente un marquage particulier qui permet d'intégrer et d'identifier les états des 2 branches
     - Les fichiers en conflit doivent alors être corrigés (le marquage doit disparaitre) et intégrés dans un nouveau commit, qui fermera automatiquement la divergence

IUT A Valence
Université Grenoble Alpes

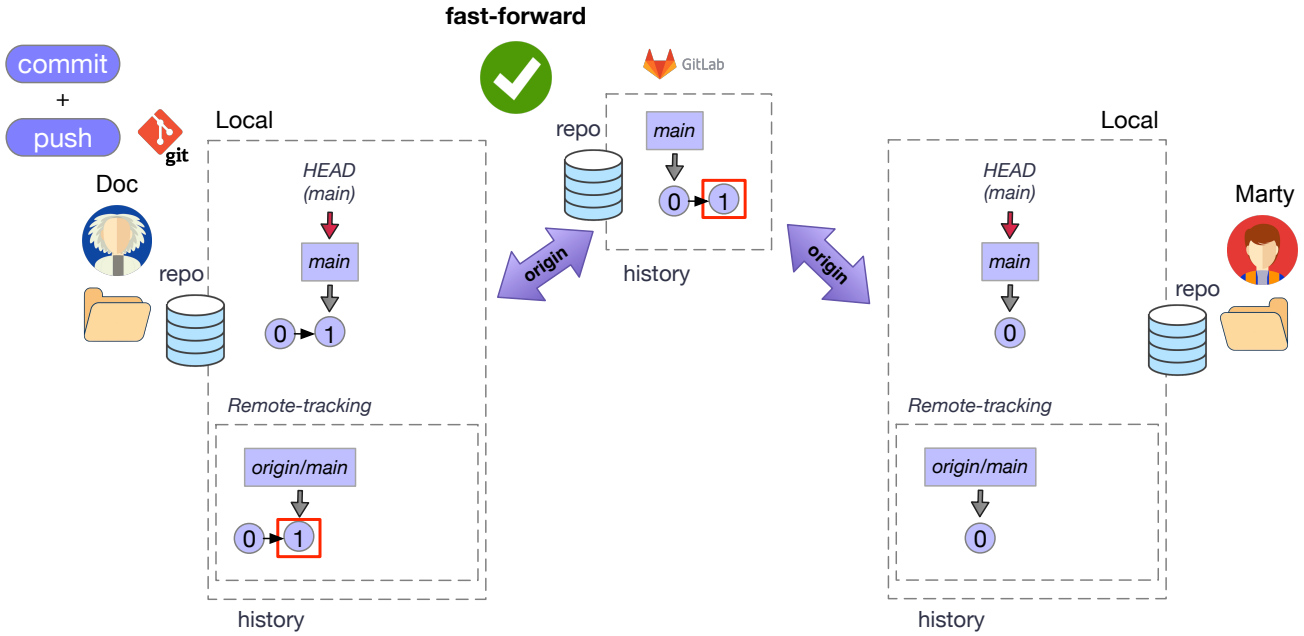
Branches de suivi à distance

- Le clone d'un dépôt local reste lié à son dépôt distant (*remote*)
  - Le remote par défaut est appelé localement `origin`
  - Une copie locale de la branche distante `main` est créée pour la synchronisation
    - Cette branche est dite **de surveillance** (*remote-tracking*), et s'appelle par convention `origin/main`



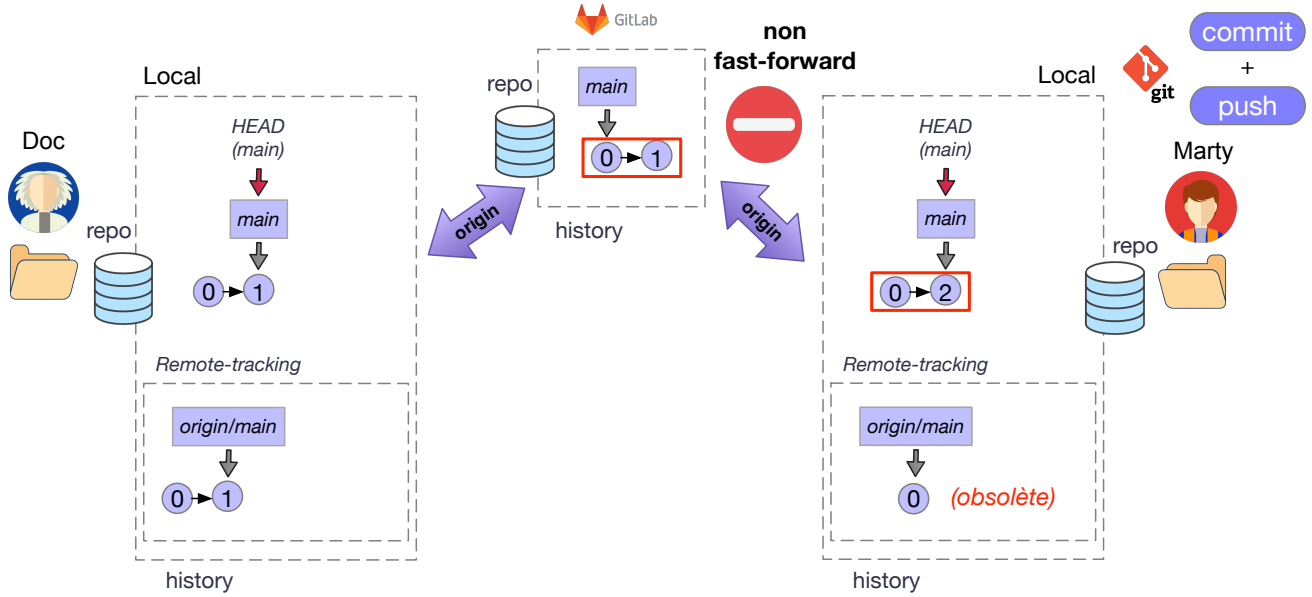
Synchronisation montante (*Push*)

- La commande `git push`, exécutée localement par Doc :
  1. Compare les branches `main` et `origin/main` et identifie les manques (ici le commit `1`)
  2. Exprime au serveur les manques (ici il faut ajouter `1` après `0`)
     - L'intégration est réalisée par le serveur car elle ne créé pas de divergence non refermée (*fast-forward*)
  3. Met à jour la branche `origin/main` (car succès)



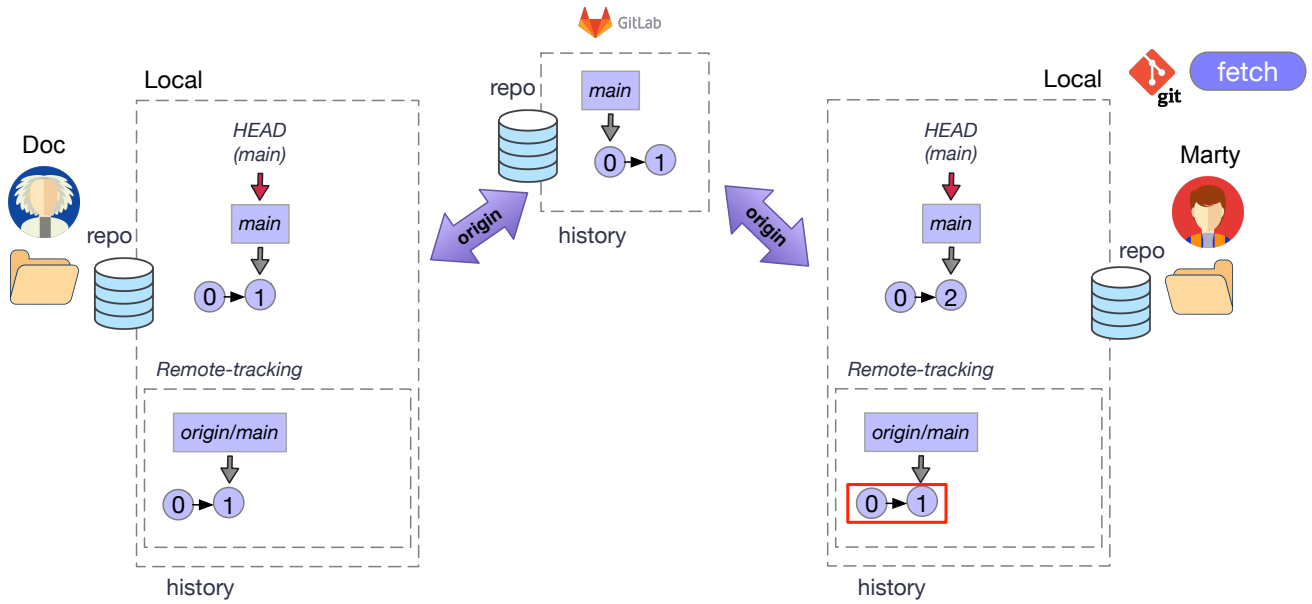
Synchronisation montante (*Push*), suite

- La commande `git push`, exécutée localement par Marty :
  1. Compare les branches `main` et `origin/main` et identifie les manques (ici le commit 2)
  2. Exprime au serveur les manques (ici il faut ajouter 2 après 0)
     - L'intégration n'est pas réalisée par le serveur car elle créé une divergence non refermée (*non fast-forward*)



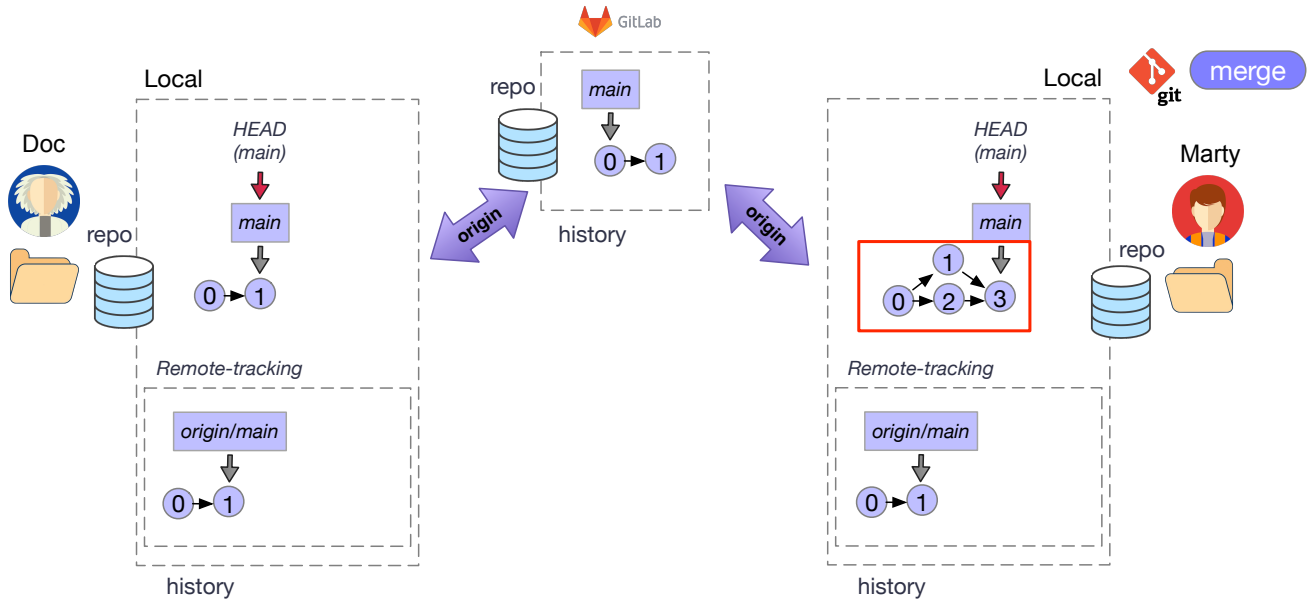
Synchronisation descendante (Pull, Fetch/Merge)

- La commande `git fetch`, exécutée localement par Marty :
  1. Communique au serveur la référence du dernier commit connu sur la branche `origin/main`
  2. Reçoit du serveur les manques (ici il faut ajouter `1` après `0`)
     - L'intégration est réalisée, l'opération `fetch` n'échoue jamais



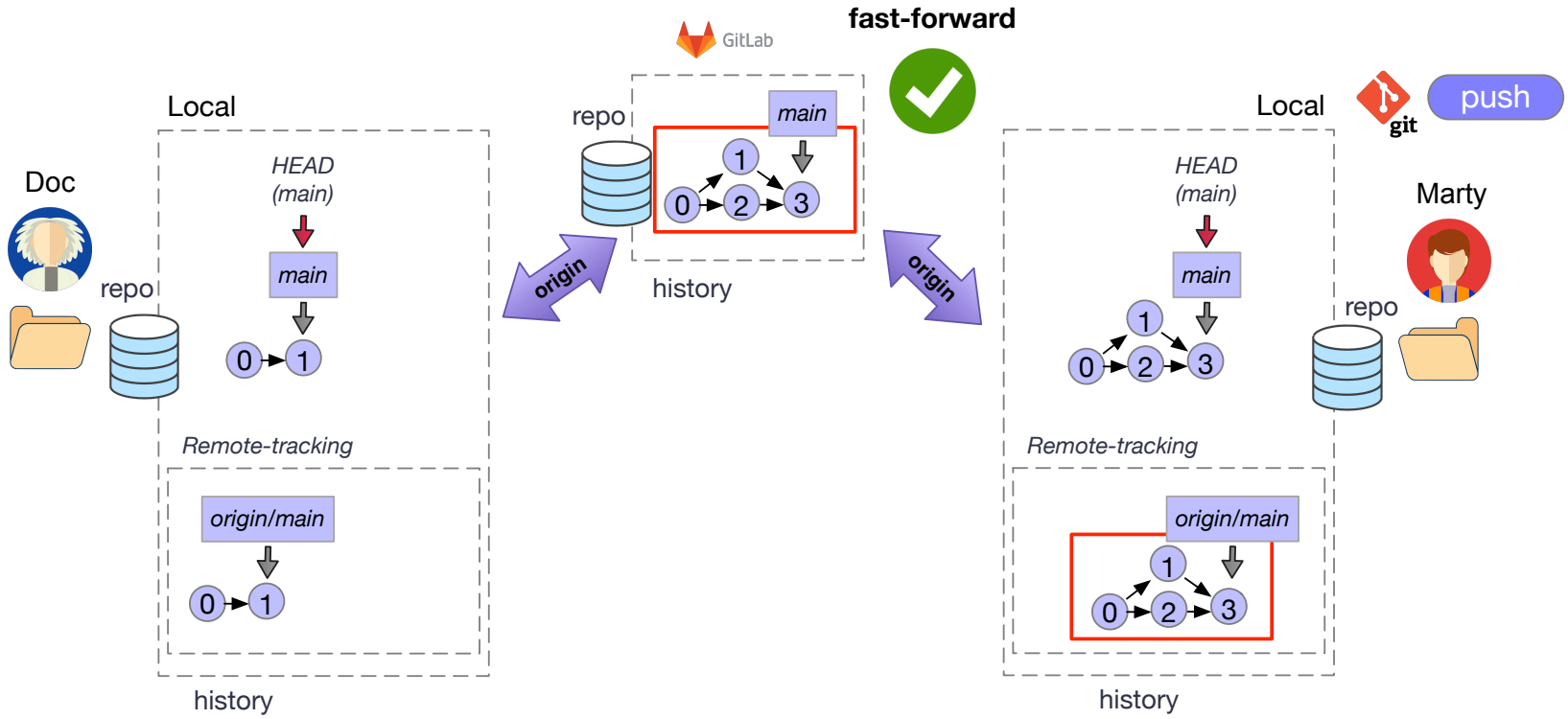
Synchronisation descendante (Pull, Fetch/Merge), suite

- La commande `git merge`, exécutée localement par Marty :
  1. Fusionne la branche `origin/main` dans la branche `main`
     - Avec ou sans intervention du développeur, en fonction des conflits éventuels (ici aucun)



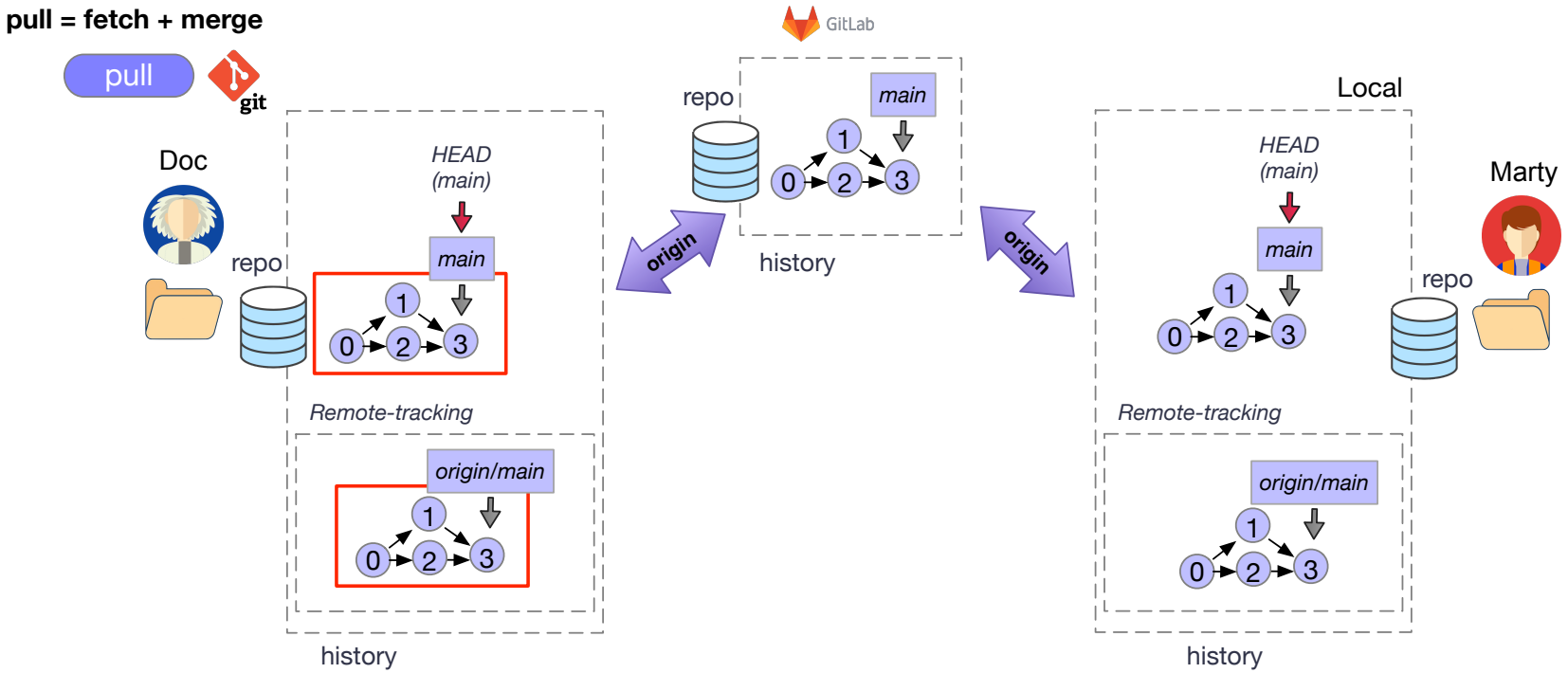
Synchronisation, suite

- La commande `git push`, exécutée localement par Marty, intègre les commits au remote
  - Les manques expriment une divergence, mais résolue par une fusion donc fermée



Synchronisation, suite

- La commande `git pull`, exécutée localement par Doc, enchaine les commandes `fetch` et `merge`



Publication des branches, suivi, remotes multiples

- La commande `git push` permet aussi de **publier sur un remote les branches créés localement** et de **mettre en place des branches de surveillance** (dans le cas où d'autres développeurs intègrent des commits à la branche)
- La commande `git pull` permet aussi d'**obtenir les branches publiées sur un remote** et de **mettre en place des branches de surveillance**
- Il est possible de configurer **plusieurs remotes** pour un même dépôt local et de **synchroniser des branches sur les uns ou les autres**

IUT Valence Université Grenoble Alpes

Et après ?

- Modèles de développement local ou distant
  - `git-flow`, dépôt distant partagé, dépot distant / forks / merge requests, ...)
- Automatisation de tâches
  - Hooks, CI/CD, ...

IUT Valence
Université Grenoble Alpes

Fin !

IUT Valence
Université Grenoble Alpes