Introduction à la gestion de versions avec *git* et *Gitlab*

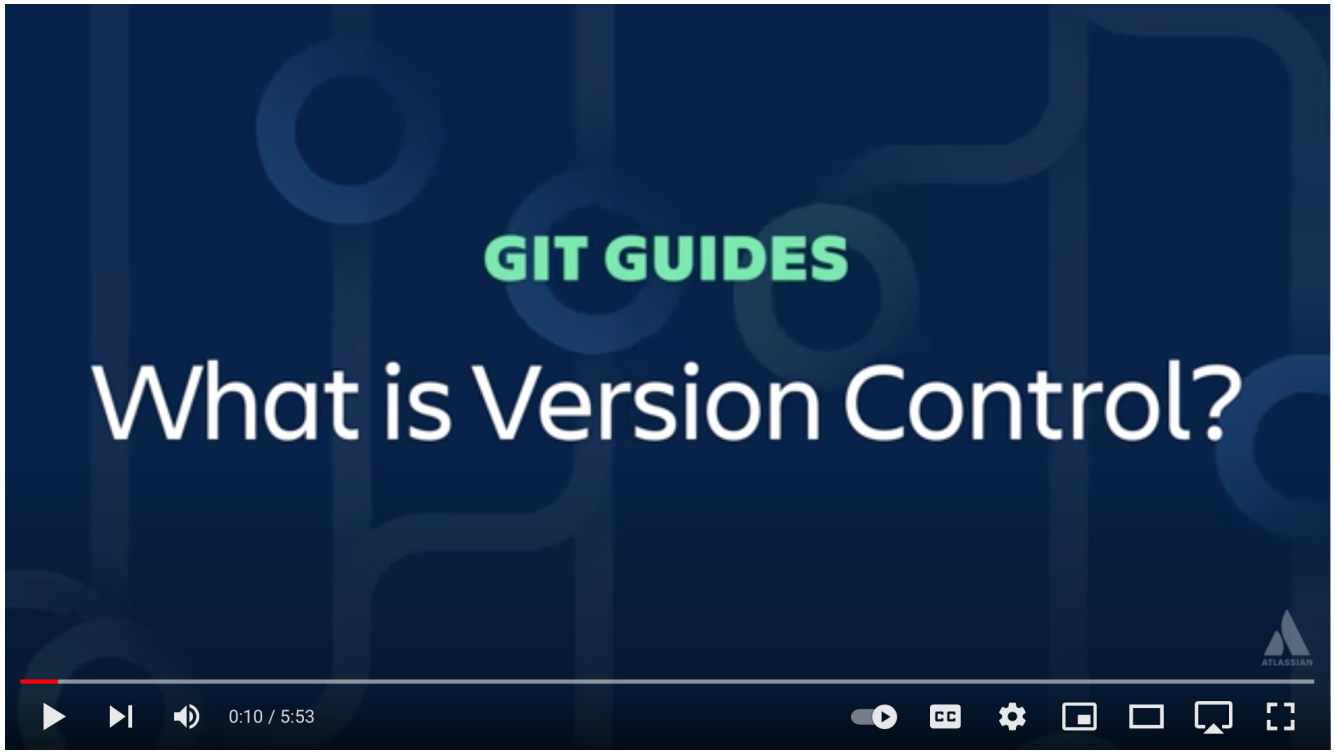
Sébastien Jean

IUT de Valence
Département Informatique

v2.1, 13 septembre 2022

IUT Valence
Université Grenoble Alpes

Zapping



- Source : `https://www.youtube.com/watch_popup?v=xQujH0ElTUg`

IUT Valence Université Grenoble Alpes

Gestion de versions : pourquoi ?

- Sauvegarder (localement, à distance)

IUT A Valence
Université Grenoble Alpes

Gestion de versions : pourquoi ?

- Maintenir un historique des modifications (**versions**)

IUT Valence
Université Grenoble Alpes

Gestion de versions : pourquoi ?

- Pouvoir facilement identifier les modifications, revenir en arrière

IUT Valence
Université Grenoble Alpes

Gestion de versions : pourquoi ?

- Expérimenter sereinement (**branches**)

IUT Valence
Université Grenoble Alpes

Gestion de versions : pourquoi ?

- Collaborer (**dépôts partagés**)

IUT Valence
Université Grenoble Alpes

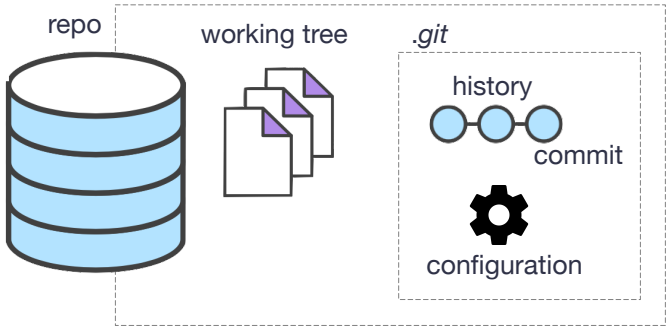
Git *Vs* Gitlab

- **Outil en ligne de commande** (local),
  pour les opérations élémentaires de gestion de versions
  - Alternatives : *Hg* (*Mercurial*), *SVN* (*SubVersion*)

- **Plateforme d'hébergement** de code
  - En mode *SaaS* (`gitlab.com`) ou déployé en interne
  - Alternatives : *Github*, *BitBucket*, *SourceForge*, ...

IUT Valence
Université Grenoble Alpes

Notion de dépôt



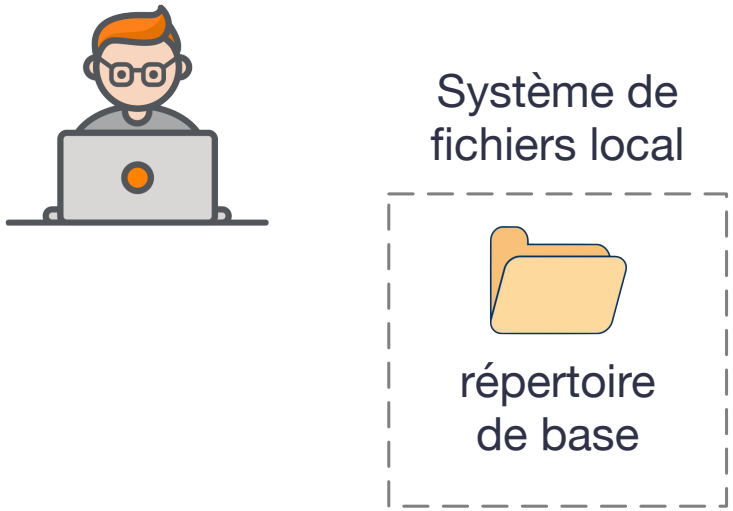
- **Dépôt** (*repository* ou *repo*)

  → **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, constitué d'une ou plusieurs **branches**
    - Succession de points de validation (**commits**), exprimant les modifications entre 2 versions (ajout/suppression de fichier, différence de contenu)
    - En *rejouant* l'historique, il est possible de reconstruire le *Working Tree* de n'importe quelle version

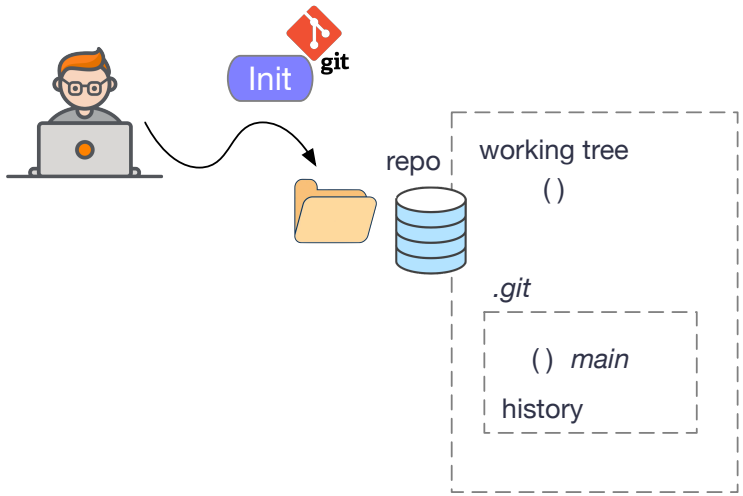
IUT A Valence Université Grenoble Alpes

Commencer une gestion de versions locale (avant)



- Le développeur souhaite commencer un développement, dans un répertoire de base sur son **système de fichiers local**
  - Le répertoire de base est supposé vide

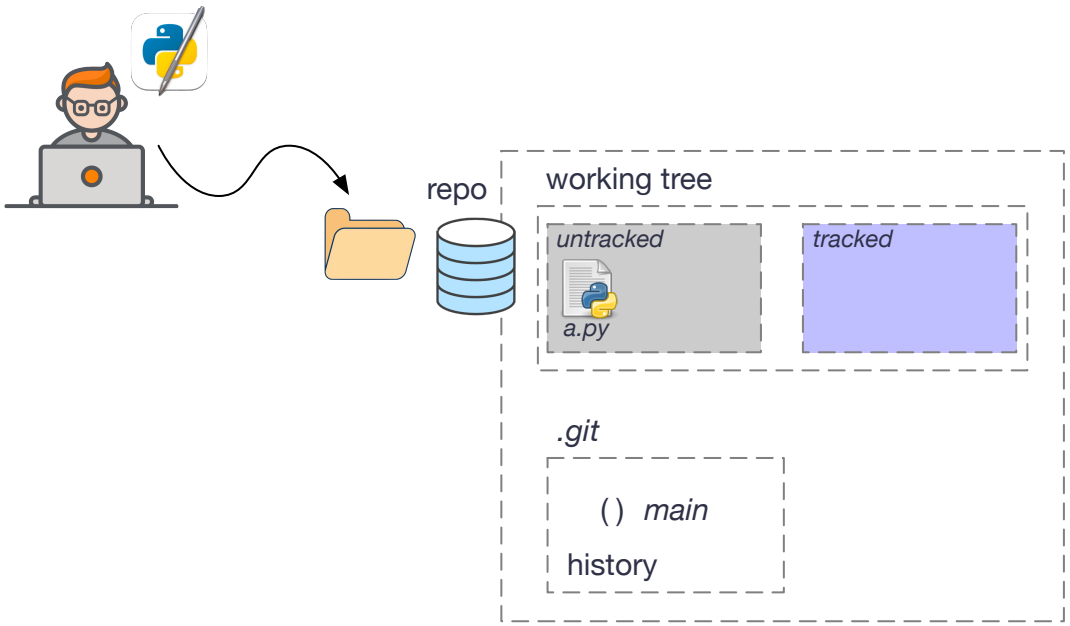
Commencer une gestion de versions locale



- Le développeur initialise explicitement, avec la commande `git init`, un nouveau dépôt dans le répertoire de base
  - Le *working tree* est le répertoire lui-même, son contenu est maintenant surveillé
  - L'historique est constitué d'une seule branche, appelée par défaut `main` (ou parfois `master`), et il est initialement vide (pas de version du code)
    - NB. *L'historique est représenté par un ensemble de fichiers dans un sous-répertoire `.git` qui contient aussi des informations de configuration*

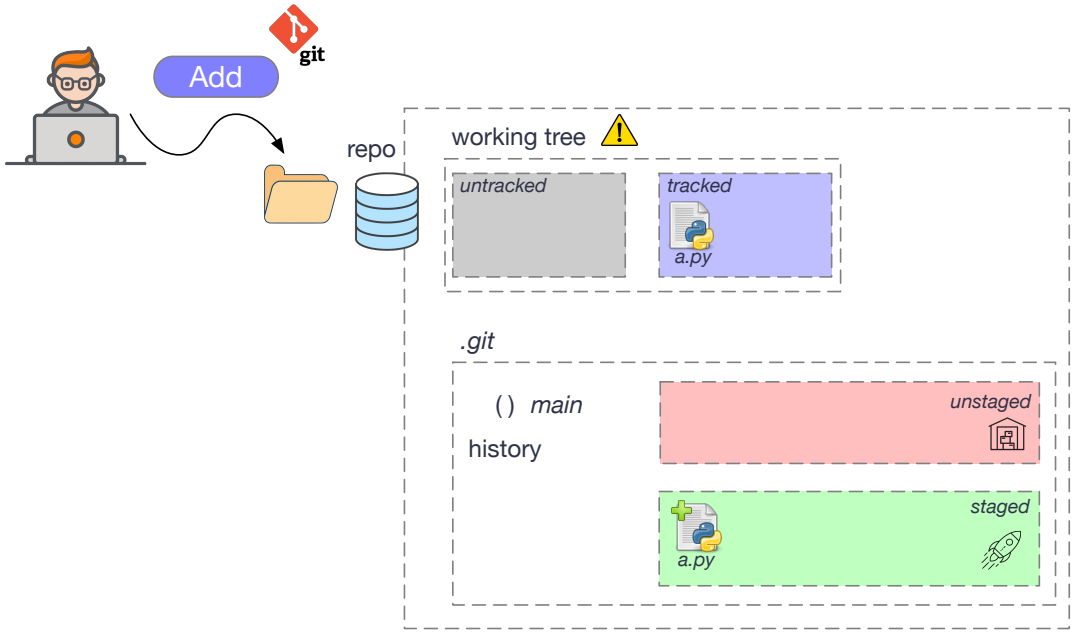
IUT Valence Université Grenoble Alpes

Ajouter du code



- Le développeur produit un **nouveau fichier**, avec un IDE quelconque, dans le *working tree*
  - L'historique est toujours vide (aucune version n'a été produite)
  - Le nouveau fichier est **détecté** et étiqueté *untracked* (pas encore versionné)

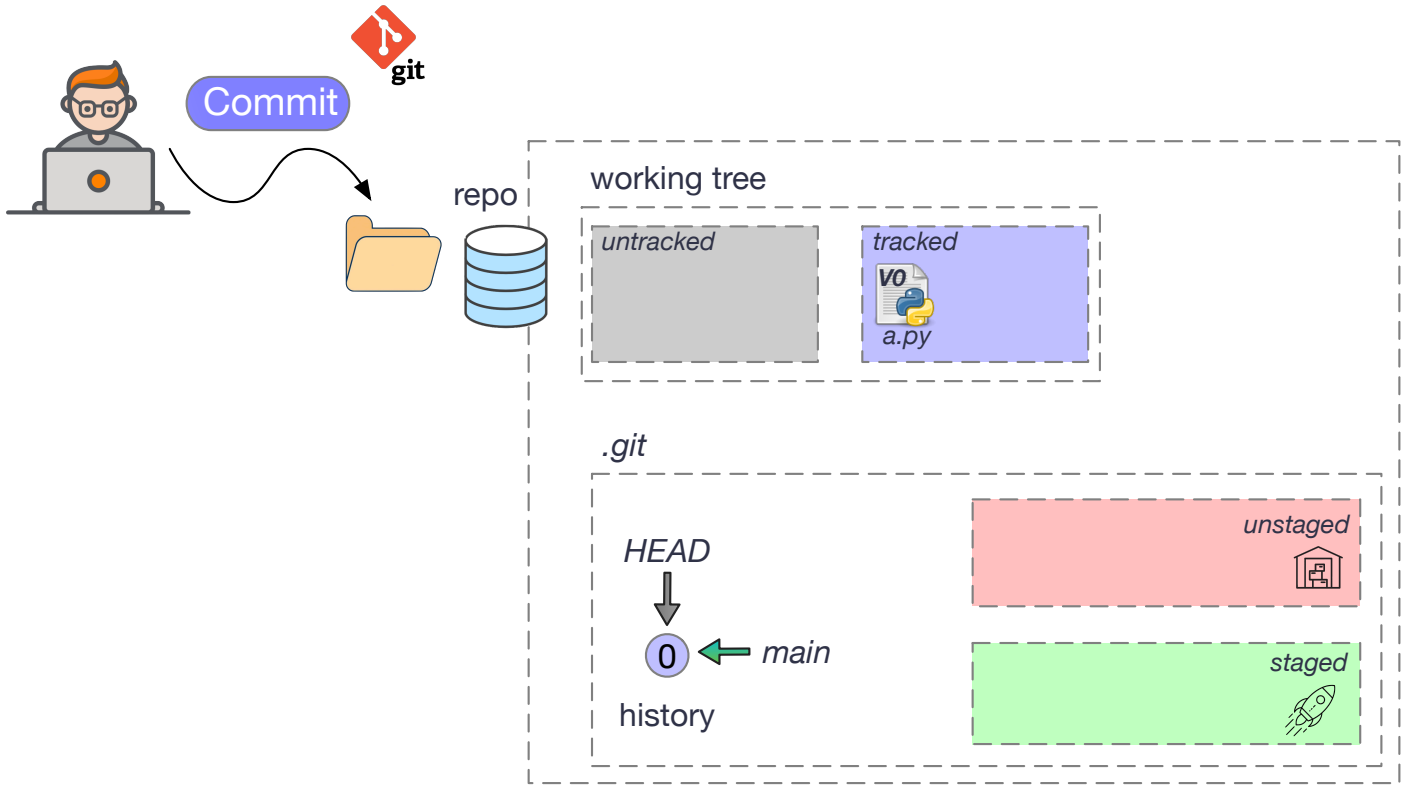
Intégrer des modifications à la (future) prochaine version



- Le développeur demande explicitement, avec la commande `git add`, à ce que les modifications sur le fichier soient intégrées à l'ensemble des modifications qui constitueront la prochaine version
  - La modification est consignée une zone dédiée appelée `staged`
  - Le fichier en lui-même est étiqueté `tracked` (versionné)
  - L'historique est toujours vide (aucune version n'a été produite)
  - Le *Working Directory* est dans un état "entre 2 versions" (*dirty*)

IUT Valence Université Grenoble Alpes

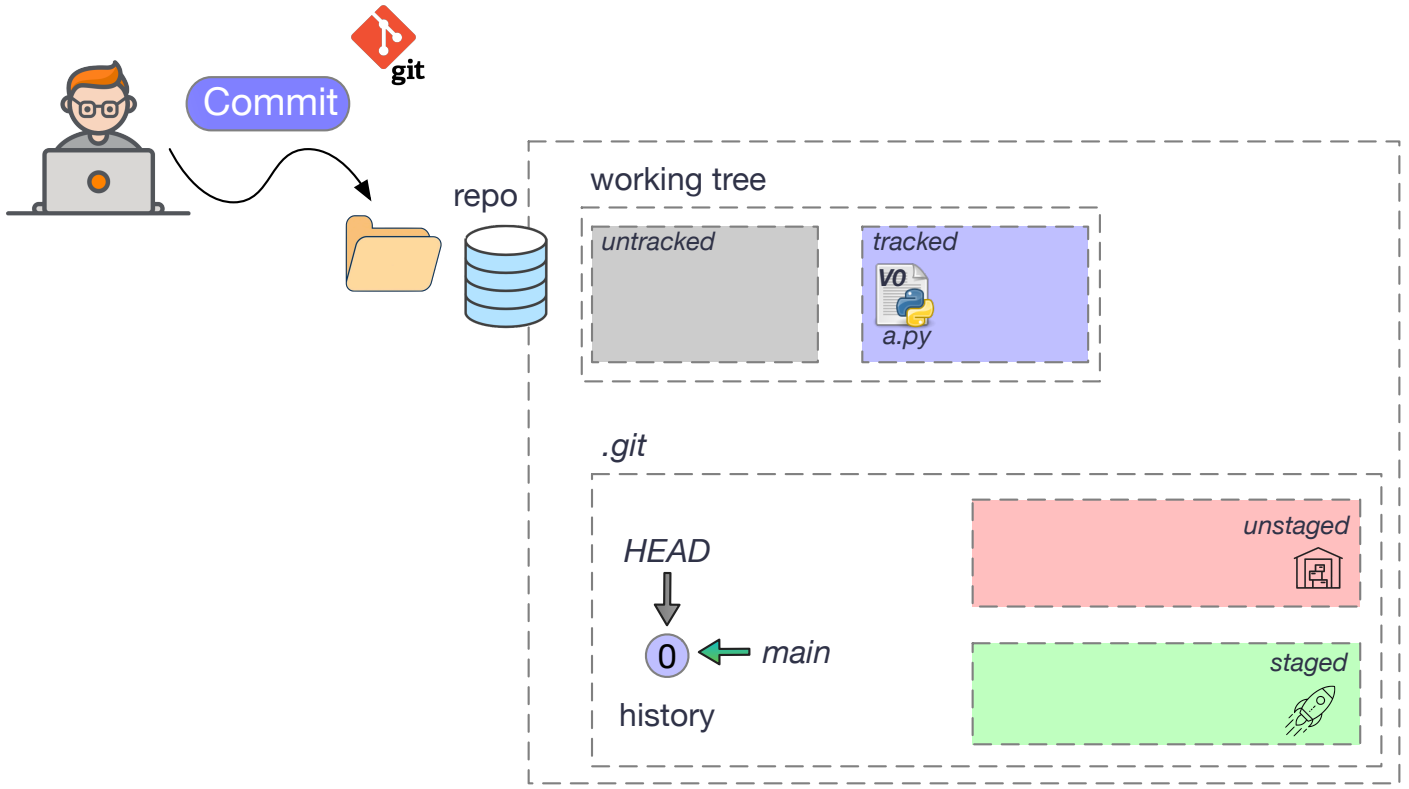
Produire une première version



- Le développeur valide explicitement les modifications sélectionnées, avec la commande `git commit`, et produit une nouvelle version
  - La zone *staged* est vidée, les modifications qu'elle contient forment la nouvelle version (commit)
    - A chaque commit est associé un identifiant unique sur 20 octets

IUT A Valence Université Grenoble Alpes

Produire une première version (suite)



- La référence `main` désigne le dernier commit de la branche `main`
- La référence `HEAD` désigne la version courante

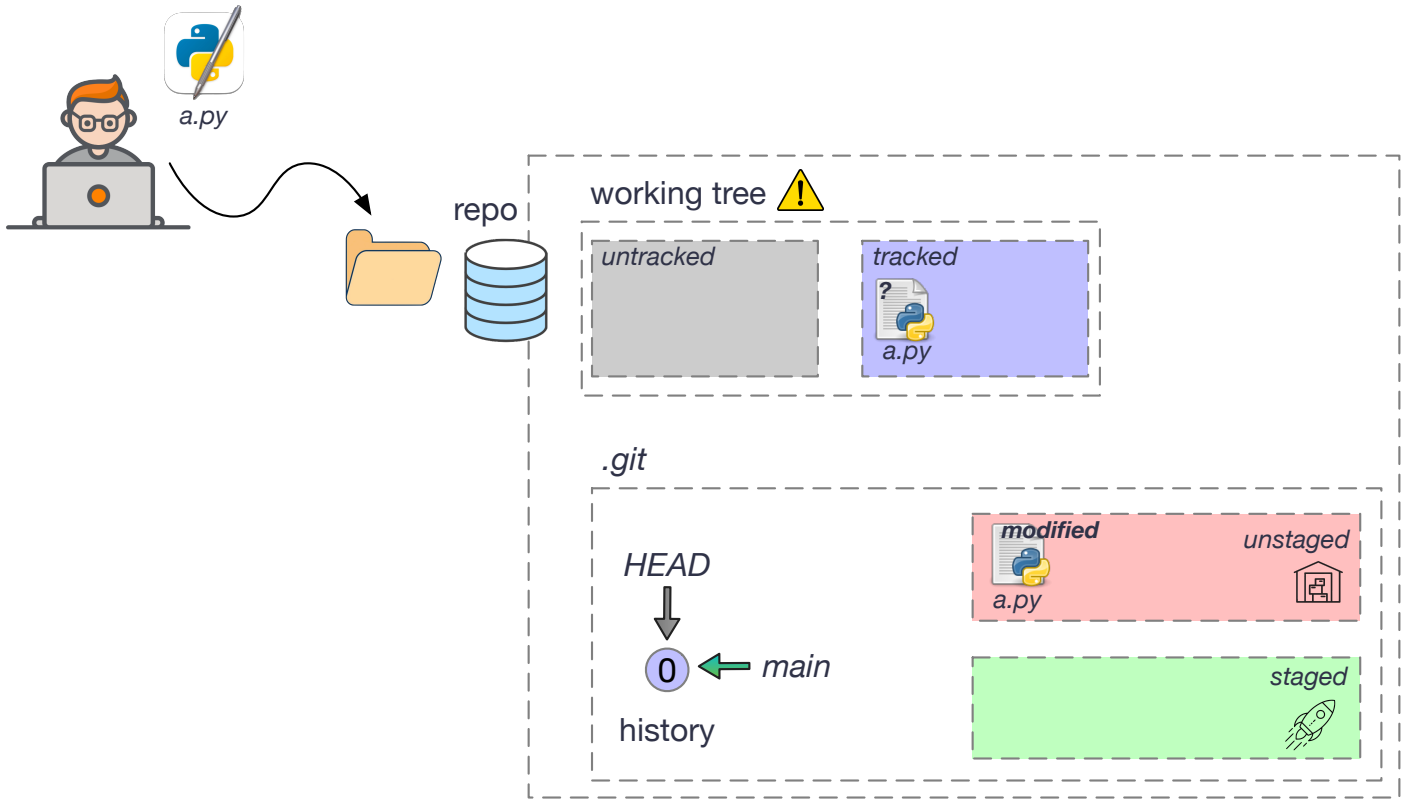
IUT Valence Université Grenoble Alpes

Produire une première version (fin)

- 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

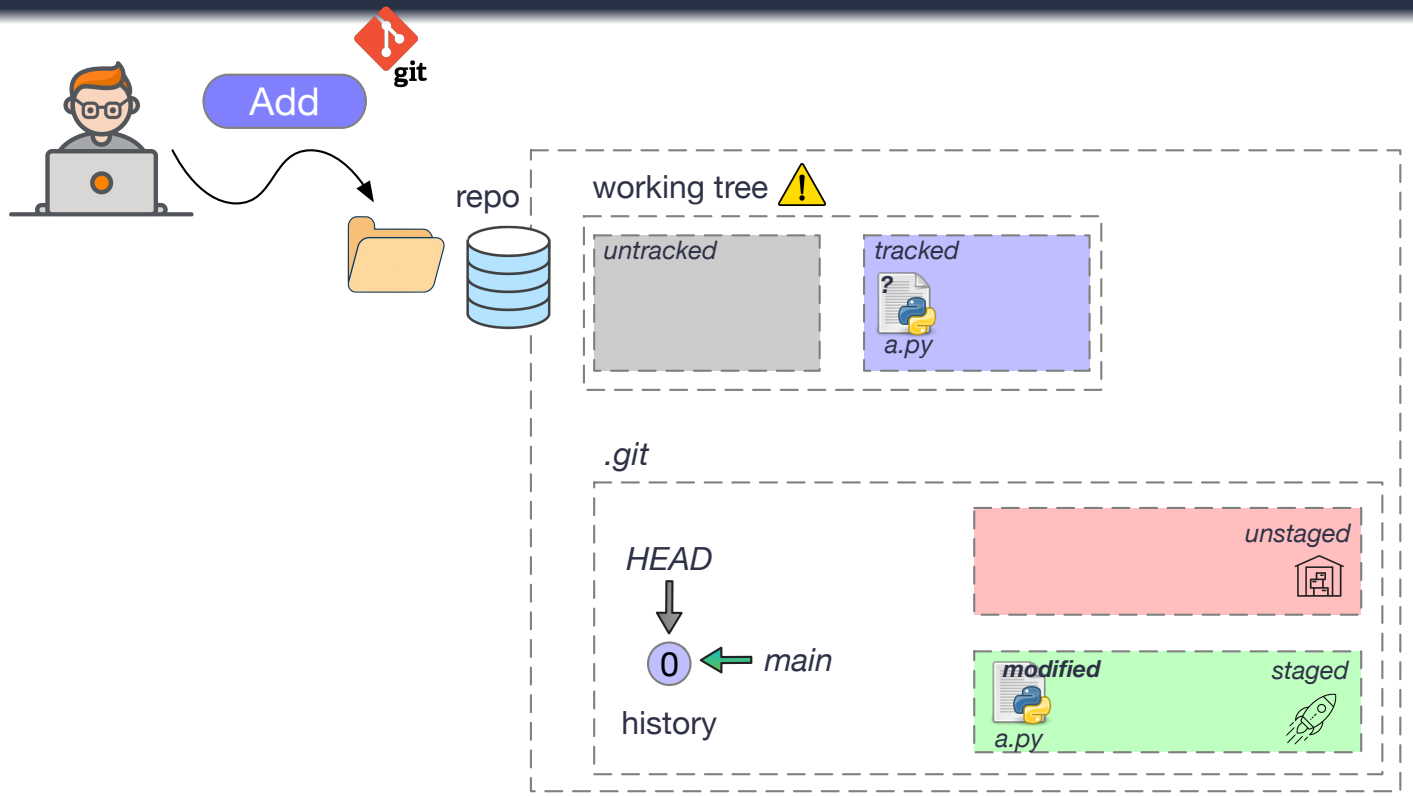
Produire une nouvelle version



- Le développeur fait des **modifications sur le fichier** déjà suivi
  - Les modifications sont consignées dans la zone `unstaged`, contenant l'ensemble des modifications détectées qui n'ont **pas encore été intégrées** à la *future* nouvelle version

IUT Valence Université Grenoble Alpes

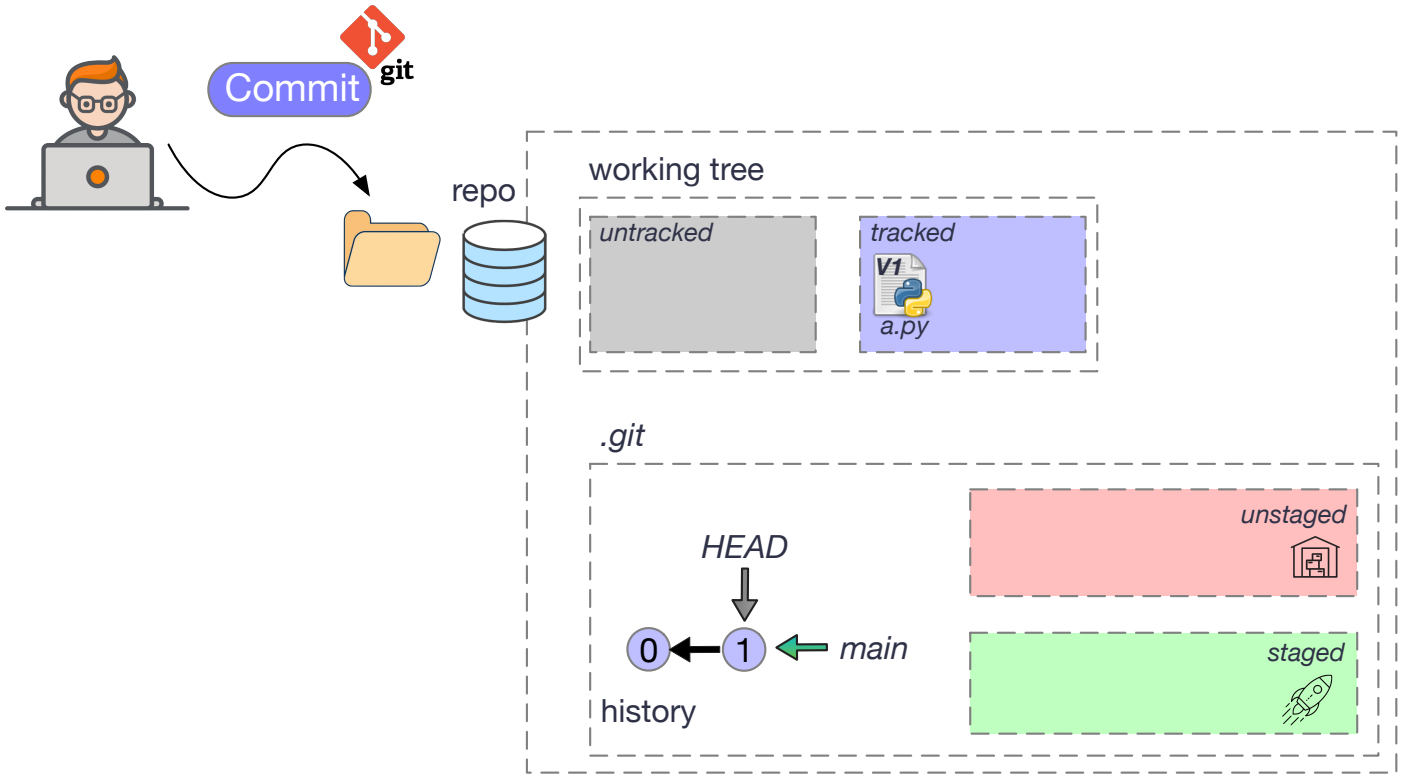
Produire une nouvelle version (suite)



- Le développeur demande explicitement, avec la commande `git add`, à ce que les modifications sur le fichier soient intégrées à l'ensemble des modifications qui constitueront la prochaine version
  - Les modifications passent de la zone `unstaged` à la zone `staged`
    - N.B Le développeur peut choisir de déplacer tout ou seulement une partie des modifications

IUT Valence Université Grenoble Alpes

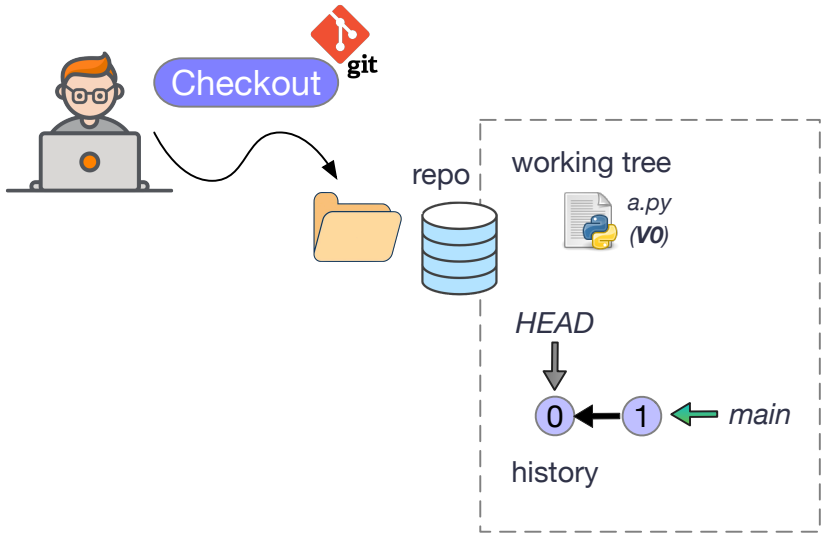
Produire une nouvelle version (fin)



- Le développeur valide explicitement les modifications sélectionnées, avec la commande `git commit`, et produit une nouvelle version
  - La nouvelle version vient **s'accrocher à la suite de la précédente**
  - Les références `main` et `HEAD` sont mises à jour

IUT Valence Université Grenoble Alpes

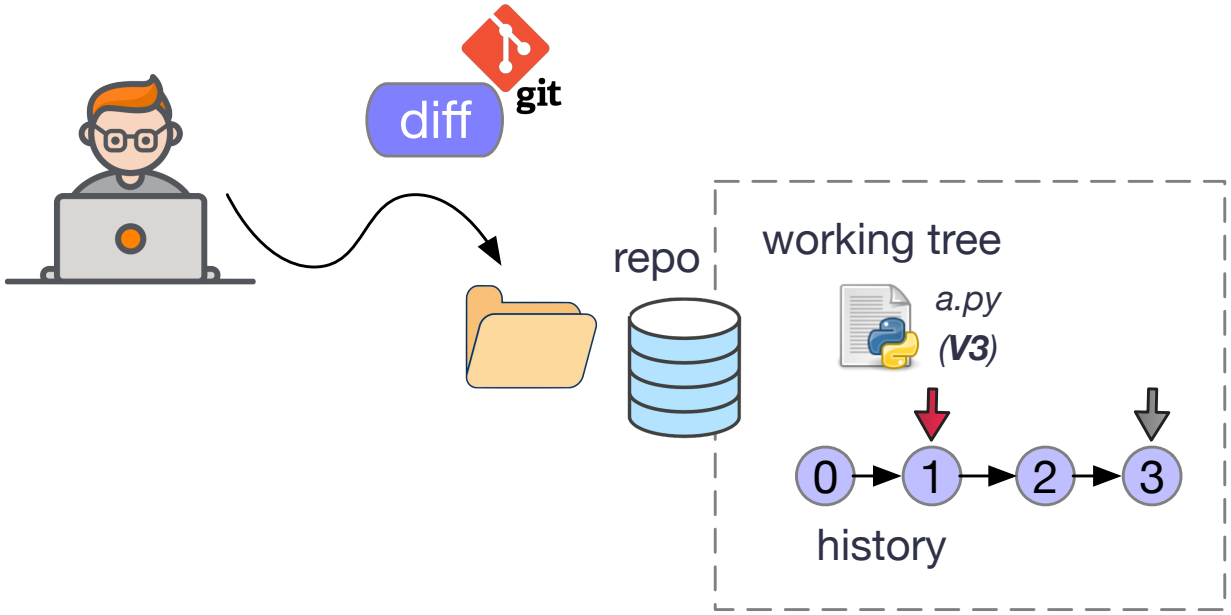
Voyager dans le temps



- Le contenu du *working tree* est basé sur une version courante appelée `HEAD` (en général le dernier commit)
- Le développeur peut à n'importe quel moment déplacer la référence `HEAD`, avec la commande `git checkout`, pour se replacer sur une version quelconque de l'historique
  - Pour observer, poursuivre sur une branche parallèle, ...
  - N.B. la commande `git checkout` peut aussi s'appliquer à des fichiers et dans ce cas elle permet de restaurer une version antérieure dans le *Working Tree*

IUT Valence Université Grenoble Alpes

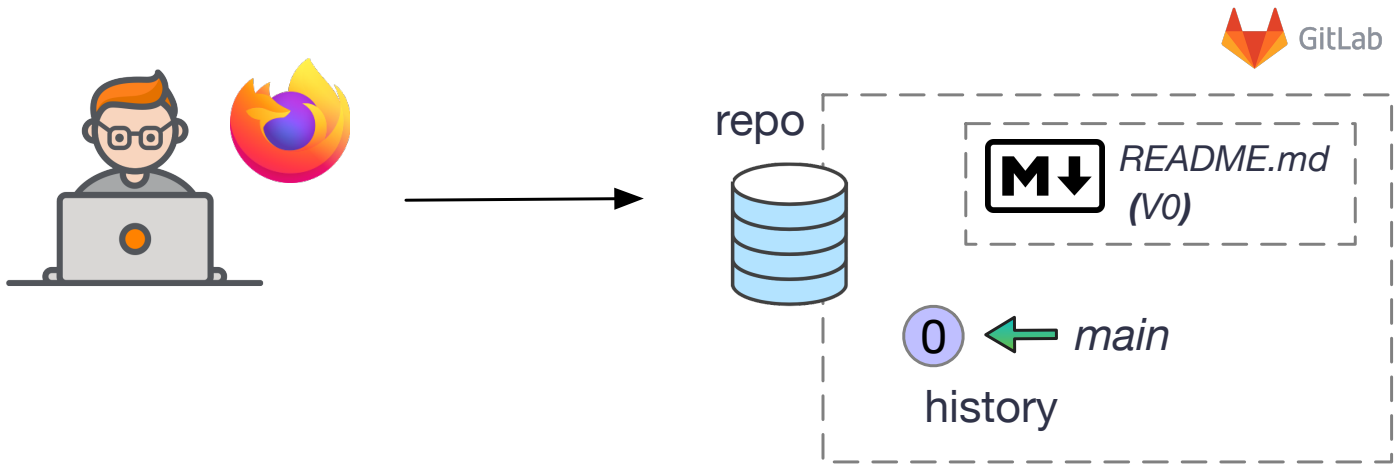
Comparer



- Le développeur peut,à n'importe quel moment, avec la commande `git diff`, **comparer deux versions** d'un fichier
  - Affichage des différences avec une syntaxe particulière (diff Unix)

IUT Valence Université Grenoble Alpes

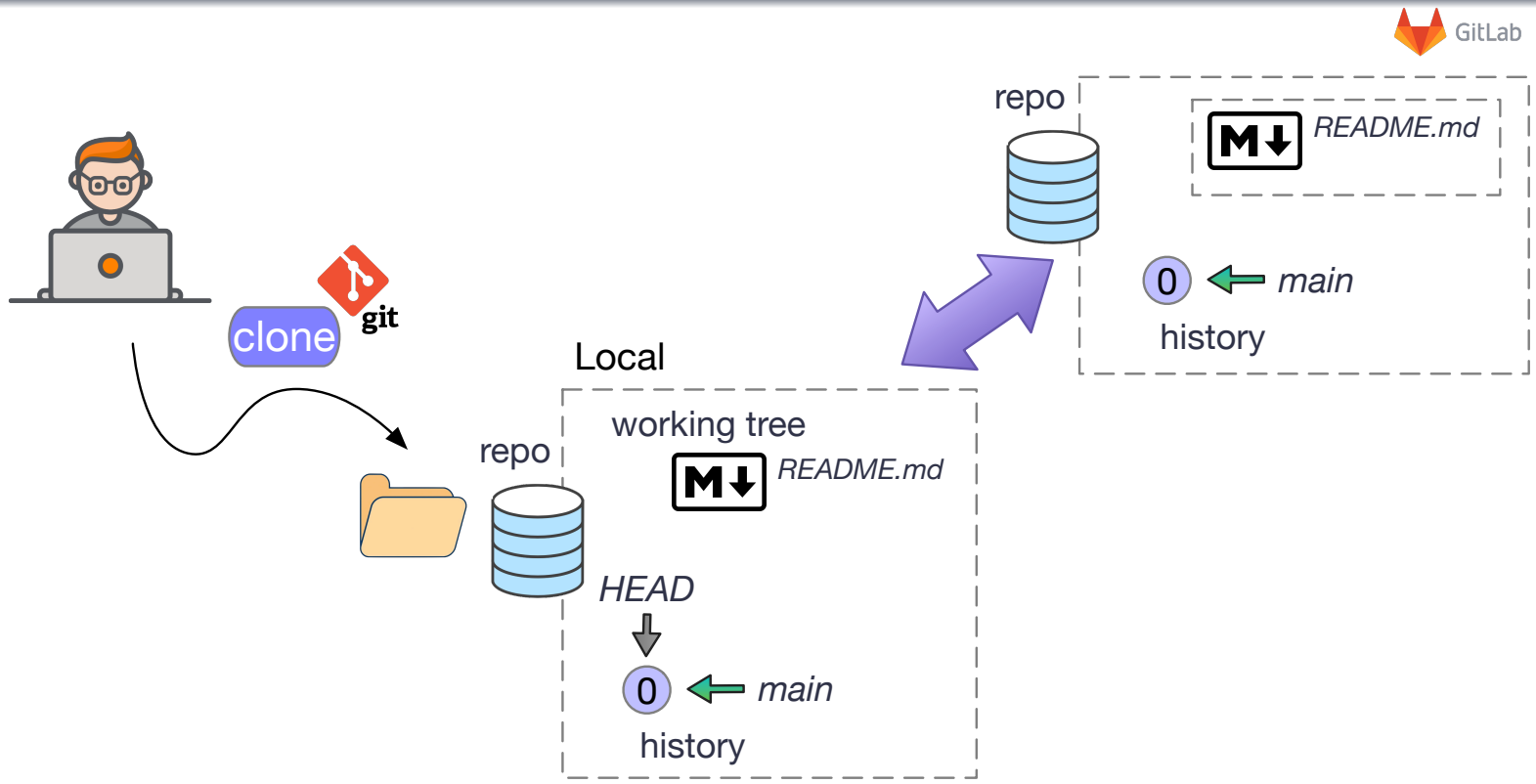
Héberger un dépôt distant avec Gitlab



- Le développeur, qui dispose d'un compte sur un **serveur Gitlab**, créé un nouveau dépôt distant
  - 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
    - La référence `HEAD` n'a pas de sens, elle n'existe pas

IUT Valence Université Grenoble Alpes

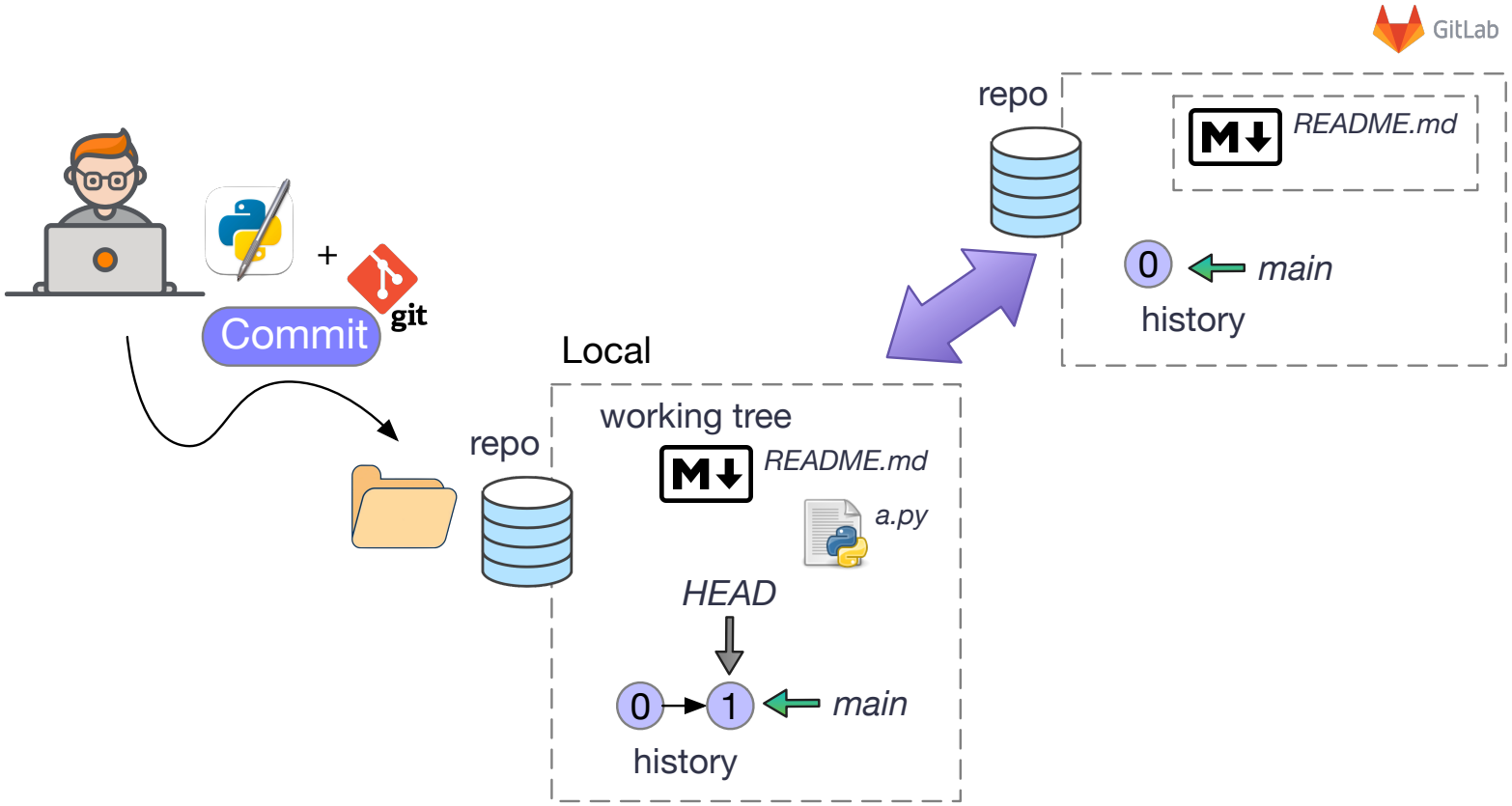
Travailler localement avec un clone



- Le développeur reconstruit localement, avec la commande `git clone`, un dépôt local **clone** du dépôt distant
  - 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

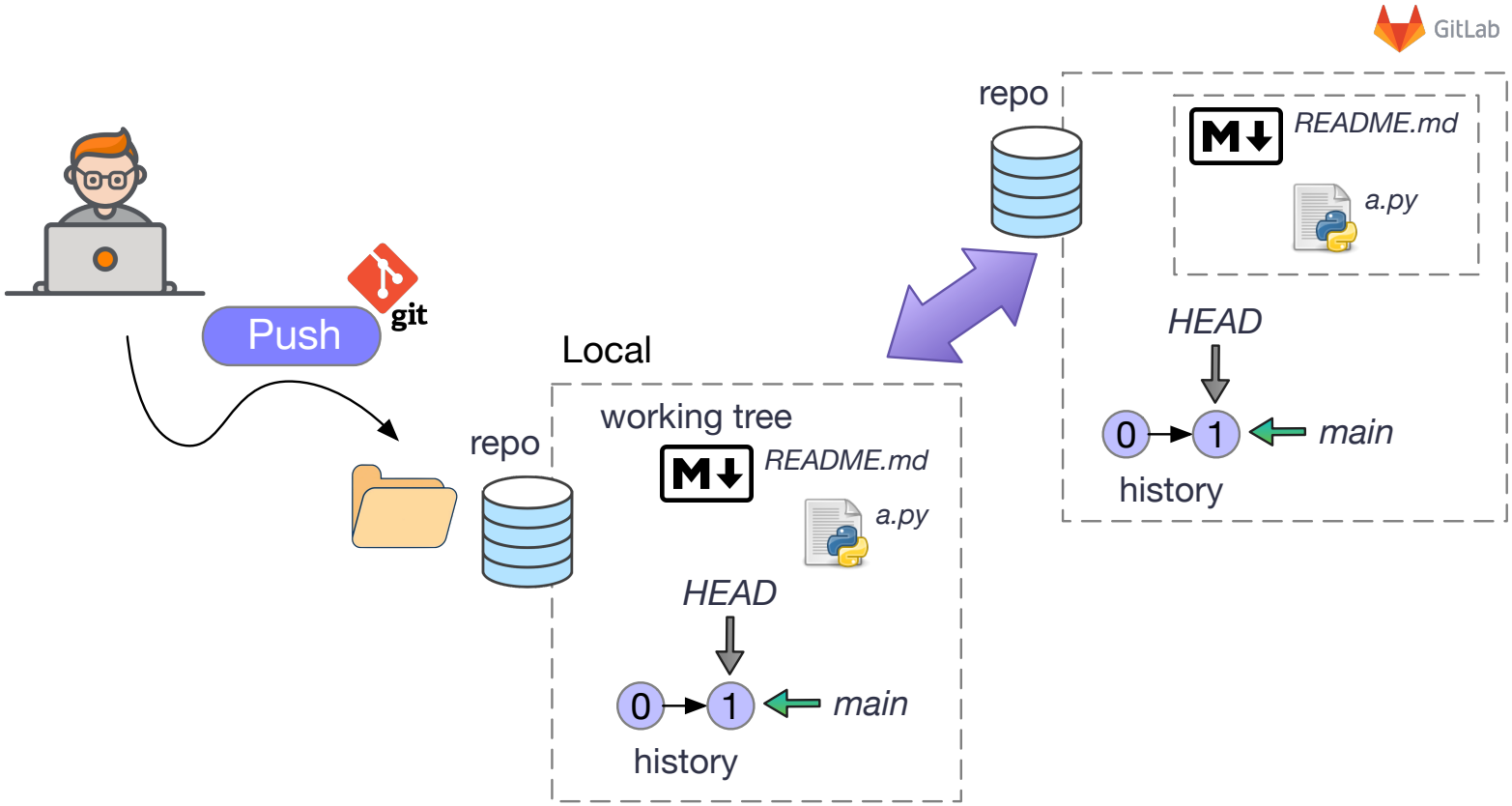
Travailler localement avec un clone (suite)



- Le développeur produit localement une nouvelle version
  - NB. **opération toujours locale**, **sans communication avec le serveur**
  - L'historique local change, **l'historique distant ne change pas**

IUT Valence Université Grenoble Alpes

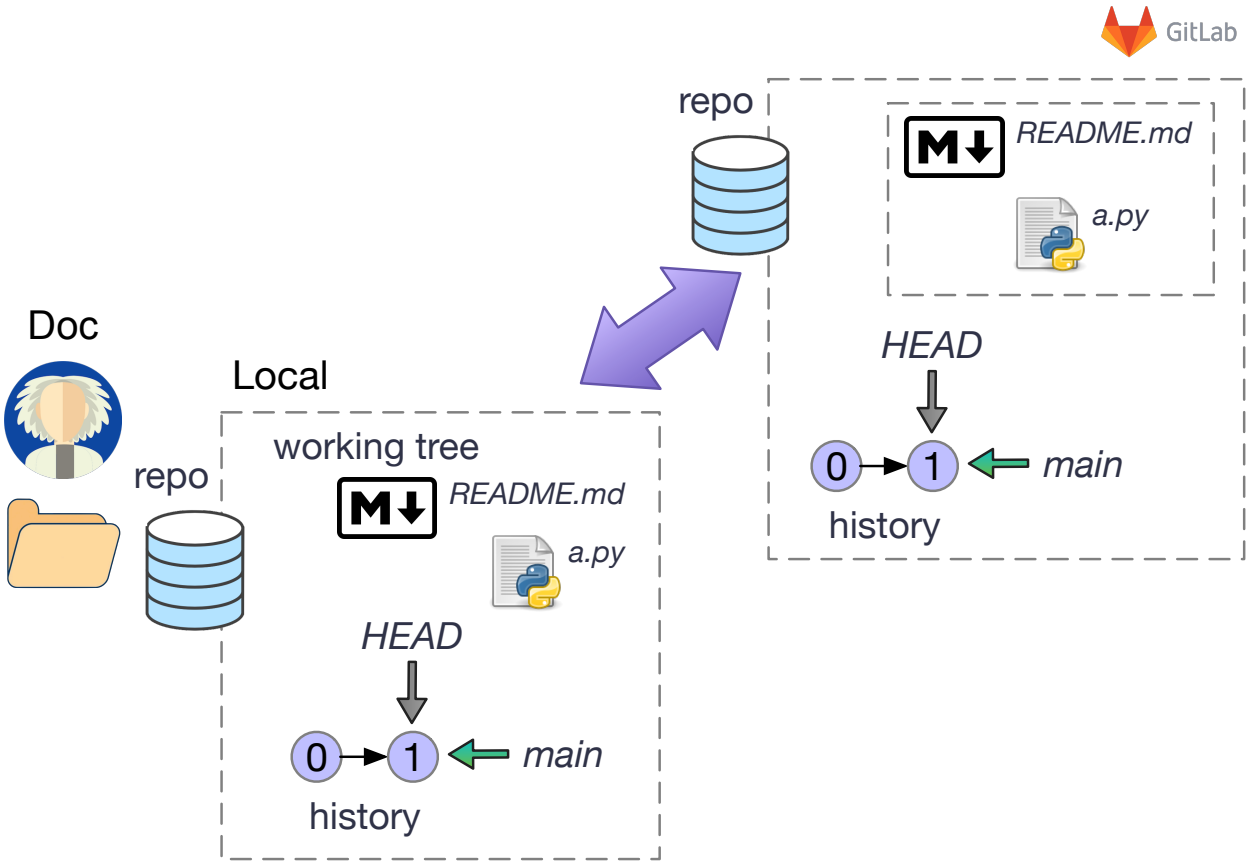
Mettre à jour le dépôt distant



- Le développeur envoie ses commits locaux sur le serveur, explicitement, avec la commande `git push`

IUT Valence Université Grenoble Alpes

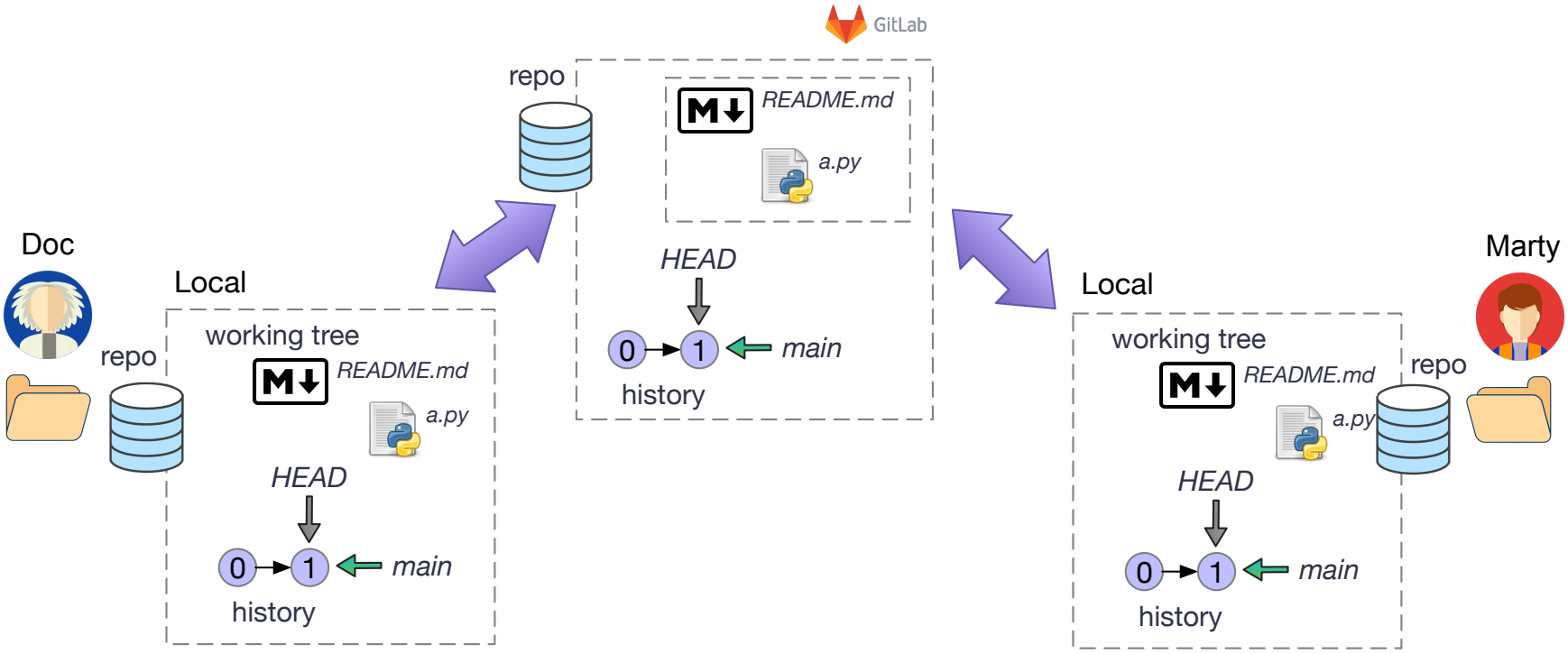
Se remettre à jour avec le dépôt distant



- *Doc* a créé un dépôt sur *gitlab*, l'a cloné localement (`clone`), a produit 2 nouvelles versions localement (`commit`), les a envoyées sur le serveur (`push`)

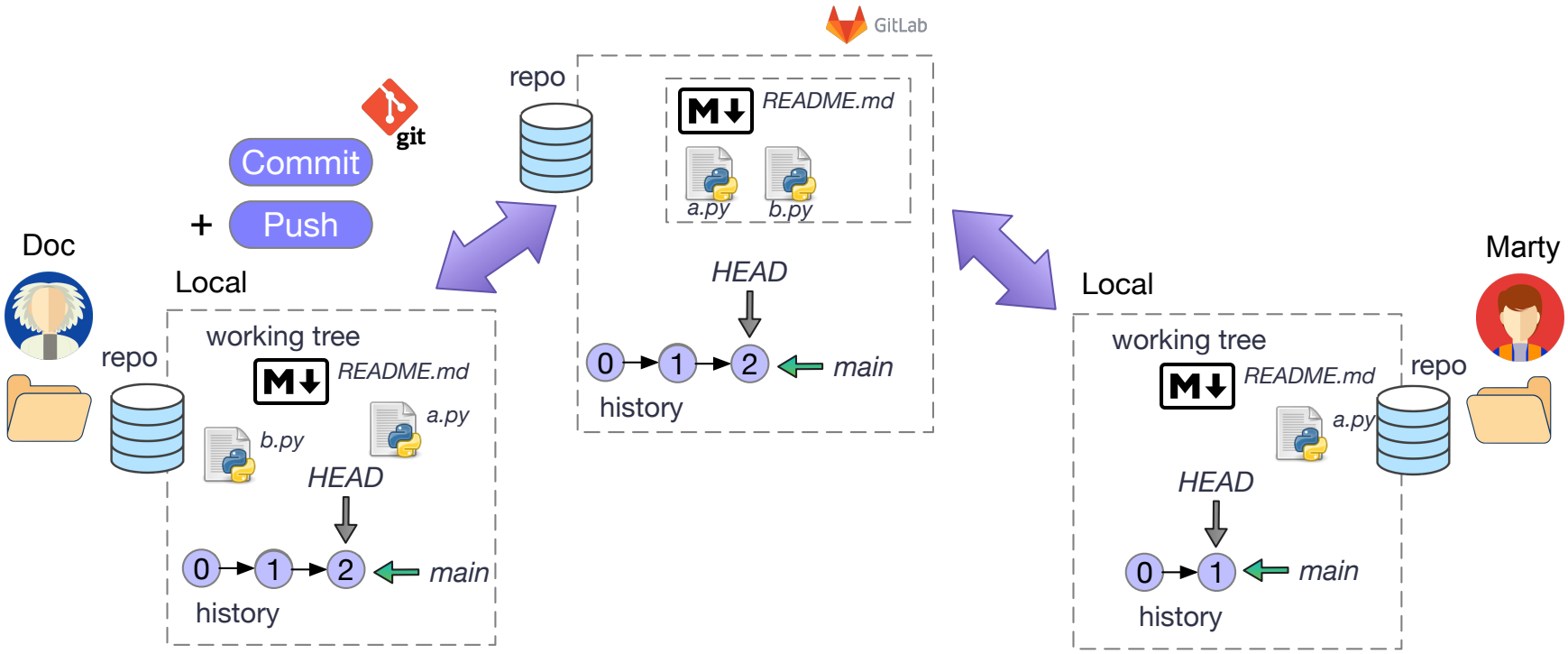
IUT Valence Université Grenoble Alpes

Se remettre à jour avec le dépôt distant



- *Marty* rejoint le développement, et clone a son tour le dépôt

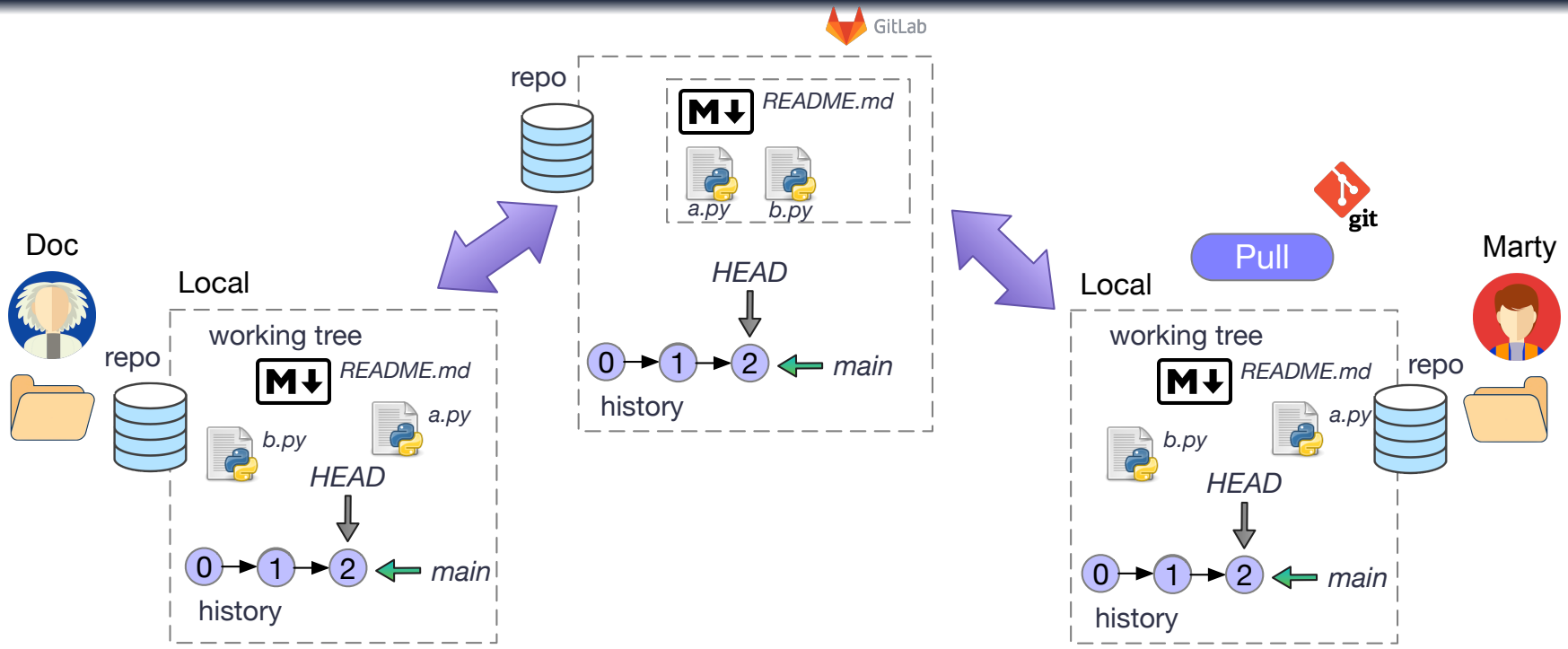
Se remettre à jour avec le dépôt distant



- *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

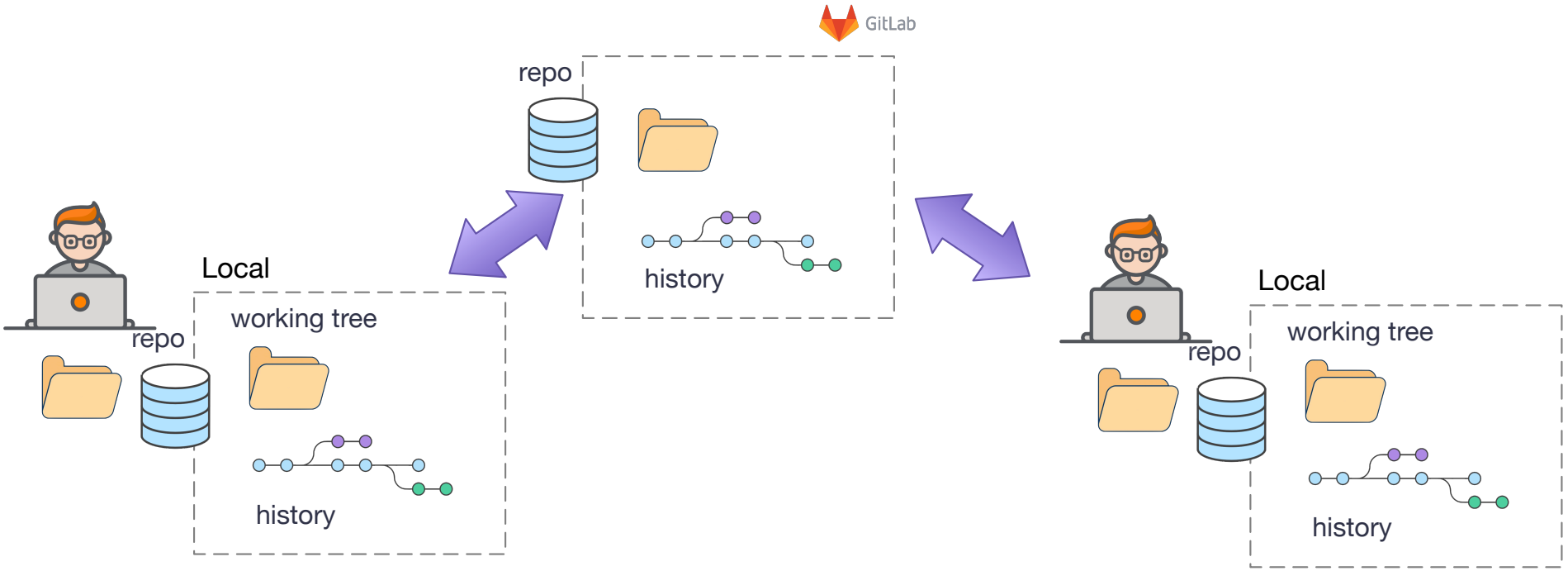
Se remettre à jour avec le dépôt distant



- Pour récupérer les éventuels commits qui lui manquent, *Marty* doit explicitement se synchroniser avec le serveur (`pull`)
- N.B. *ici, c'est un cas simple (`fast-forward`), qui se résout automatiquement et de manière autonome*
  - *Quelquefois, des situations de conflits apparaissent et nécessitent une intervention du développeur*

IUT Valence Université Grenoble Alpes

Et après ? ?



- **Branches** (création, fusion, . . .)
- **Bonnes pratiques** (*git-flow*, *pull/merge requests*, . . .)
- **Intégration continue** (CI/CD)

IUT Valence
Université Grenoble Alpes

Fin !

IUT Valence
Université Grenoble Alpes