Introduction à la gestion de versions (locale) avec *git*

Sébastien Jean

IUT de Valence
Département Informatique

v2.1, 10 septembre 2025

IUT Valence
Université Grenoble Alpes

Gestion de versions : pourquoi ?

- Sauvegarder (localement, à distance) une arborescence
  - Fichiers texte (code, ...) mais pas que

IUT Valence
Université Grenoble Alpes

Gestion de versions : pourquoi ?

- Maintenir un historique des modifications (**versions**)
  - Quelles ressources ont changé ?
  - Quelle est la nature du changement ?
  - ...

IUT Valence
Université Grenoble Alpes

Gestion de versions : pourquoi ?

- Identifier les modifications, revenir en arrière

IUT Valence
Université Grenoble Alpes

Gestion de versions : pourquoi ?

- Expérimenter sereinement (**branches**)
  - (on y reviendra plus tard . . .)

IUT Valence
Université Grenoble Alpes

Gestion de versions : pourquoi ?

- Collaborer
  - (on y reviendra plus tard également . . .)

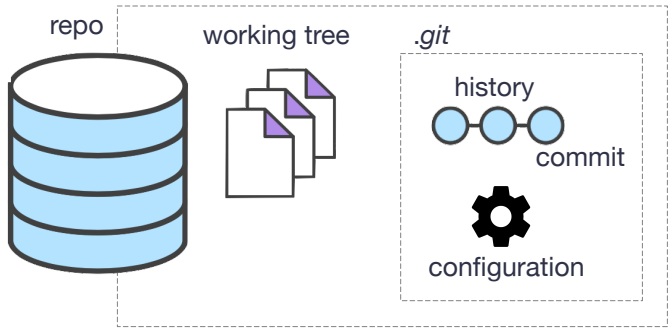
IUT Valence
Université Grenoble Alpes

Git

- Système de gestion de version **décentralisé**, qui permet la gestion de version locale (hors ligne) et distante
- **Suite d'outils** (locale) en ligne de commande, pour les opérations élémentaires de gestion de versions
- Alternatives : *Hg* (*Mercurial*), *SVN* (*SubVersion*)

IUT Valence
Université Grenoble Alpes

Notion de dépôt (local)



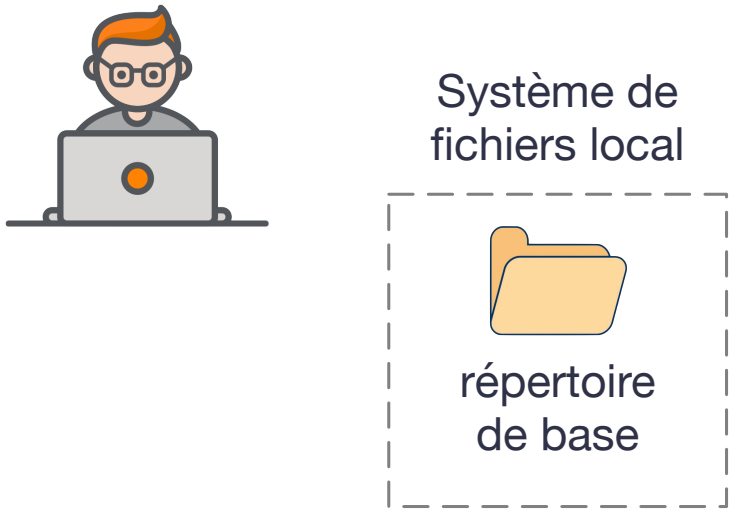
- **Dépôt** (*repository* ou *repo*)

  → **Ensemble de fichiers** 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 évolutions entre 2 versions (ajout/suppr./modif. de fichier)
    - En *rejouant* l'historique, il est possible de reconstruire le *Working Tree* de n'importe quelle version

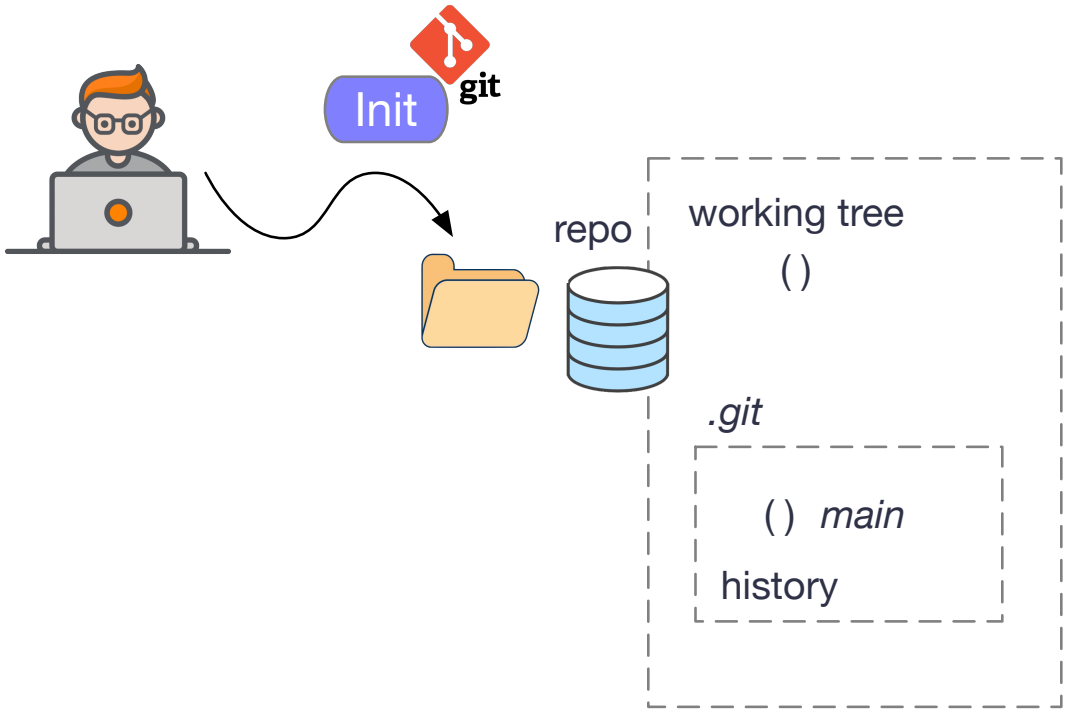
IUT Valence
Université Grenoble Alpes

Commencer une gestion de versions locale (avant)



- On souhaite commencer un développement, dans un répertoire de base sur son **système de fichiers local**
  - Le répertoire de base (celui où l'on veut développer) est supposé vide

Commencer une gestion de versions locale



- La gestion de versions commence par une initialisation explicite
  - à l'aide de la commande `git init` (exécutée depuis le rép. de base)

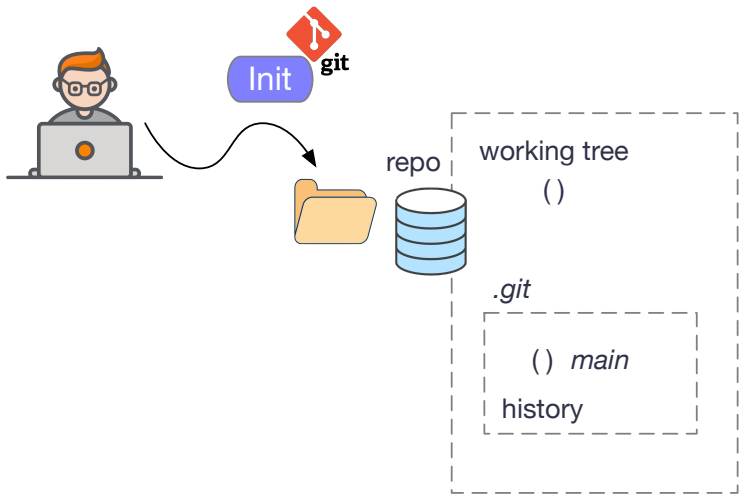
IUT Valence Université Grenoble Alpes

Commencer une gestion de versions locale

- Ouvrir un nouveau terminal
- Créer un répertoire `repo` (commande `mkdir`)
- S'y déplacer (commande `cd`)
- Visualiser le contenu du répertoire (commande `ls`)
- Initialiser un dépôt git (commande `git init`)

IUT Valence
Université Grenoble Alpes

Commencer une gestion de versions locale



- Un **nouveau dépôt** est alors créé dans le répertoire de base
  - Le *working tree* est le répertoire lui-même, il est maintenant surveillé
  - L'historique est constitué d'une seule branche, appelée par défaut `main` ou `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*

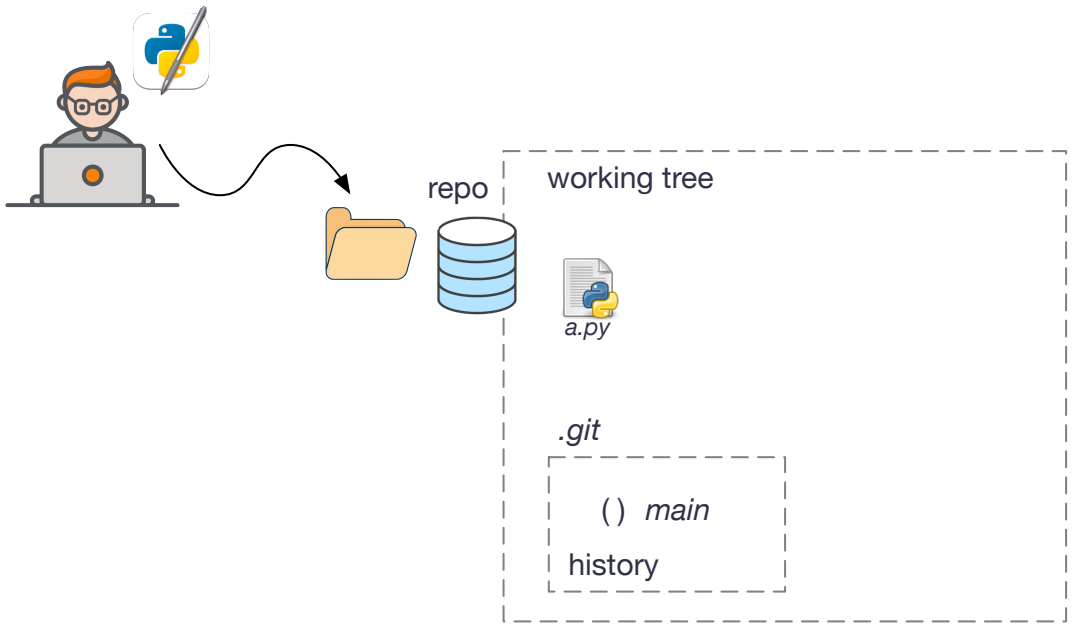
IUT Valence
Université Grenoble Alpes

Commencer une gestion de versions locale

- Afficher le contenu du répertoire `repo` en faisant apparaitre le répertoire `.git`
  - (RTFM `ls` avec `man` ;-))
- Se déplacer dans le répertoire `.git`, visualiser son contenu
- Visualiser le contenu du fichier `config`
- Revenir dans le répertoire de base (`..`)

IUT A Valence
Université Grenoble Alpes

Produire une nouvelle version : préambule



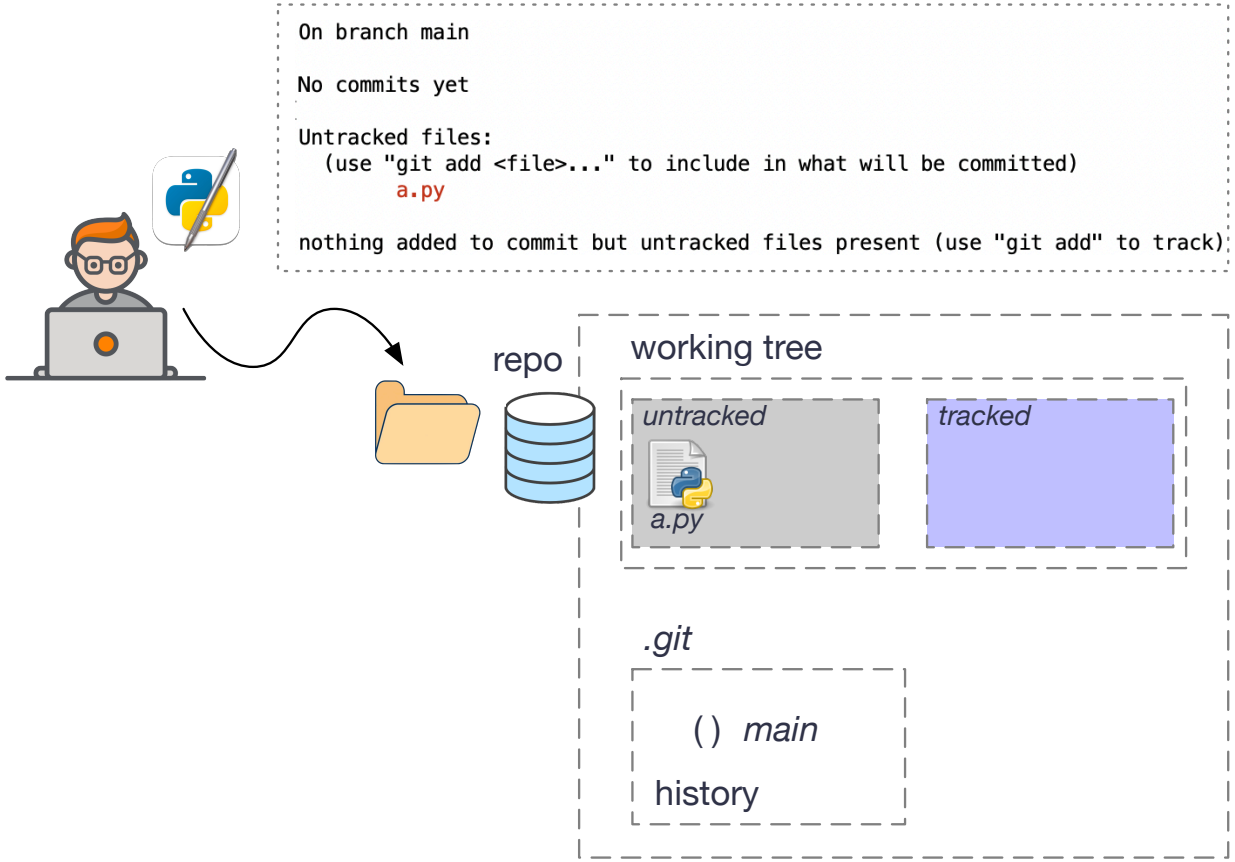
- On produit un **nouveau fichier** (`a.py`), avec un l'éditeur `vi`, dans le repertoire `repo` (*working tree*)
  - le guide de survie `vi` est précieux → Vi *cheat sheet*
  - le contenu du fichier ne doit pas être vide, **ajouter une seule ligne de texte**

IUT Valence Université Grenoble Alpes

Produire une nouvelle version : préambule

- Visualiser l'état du dépôt
  - (commande `git status`)

Produire une nouvelle version : préambule



- L'historique est toujours vide (aucune version n'a été produite)
- Le nouveau fichier est **détecté** et étiqueté *untracked* (pas encore versionné)

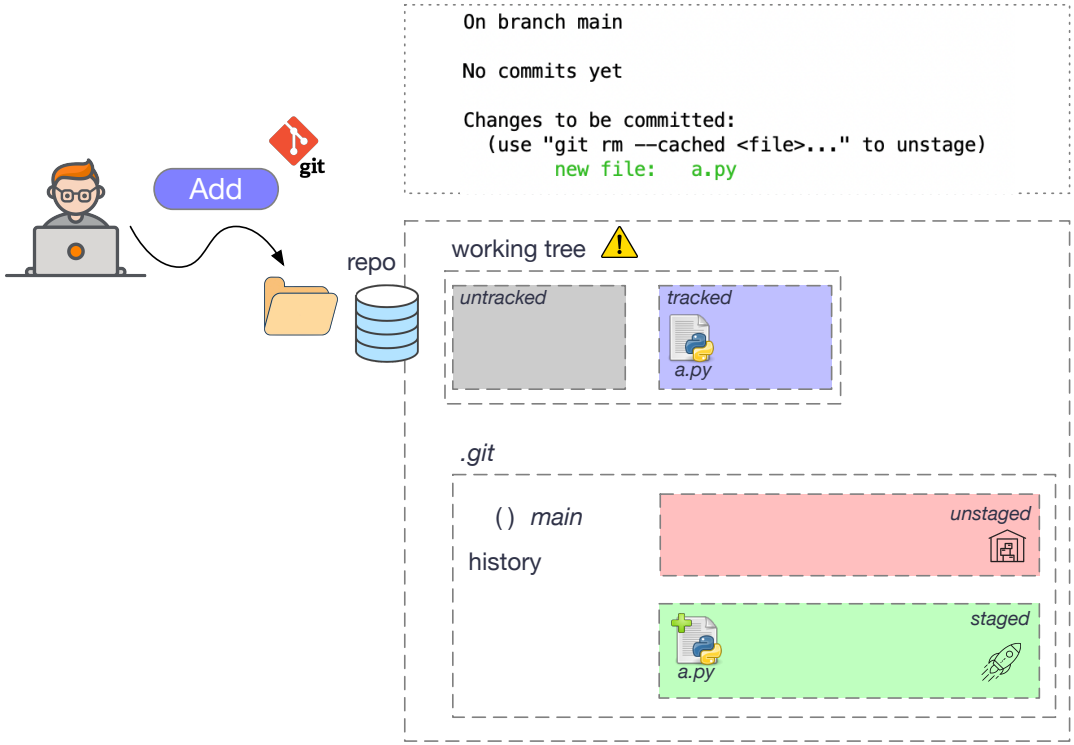
IUT Valence Université Grenoble Alpes

Produire une nouvelle version : ajouter une nouvelle ressource

- L'**intégration d'une nouvelle ressource** à une **prochaine version** est explicite, via la commande `git add`

- Ajouter le fichier `a.py` à la prochaine version
- Visualiser l'état du dépôt

Produire une nouvelle version : ajouter une nouvelle ressource



- L'ajout du fichier est consigné dans une zone appelée `staged`
- Le fichier en lui-même est étiqueté `tracked` (versionné)
- L'historique est toujours vide (aucune version n'a été produite)
- Le *Working Tree* est dans un état "entre 2 versions" (*dirty*)

IUT Valence Université Grenoble Alpes

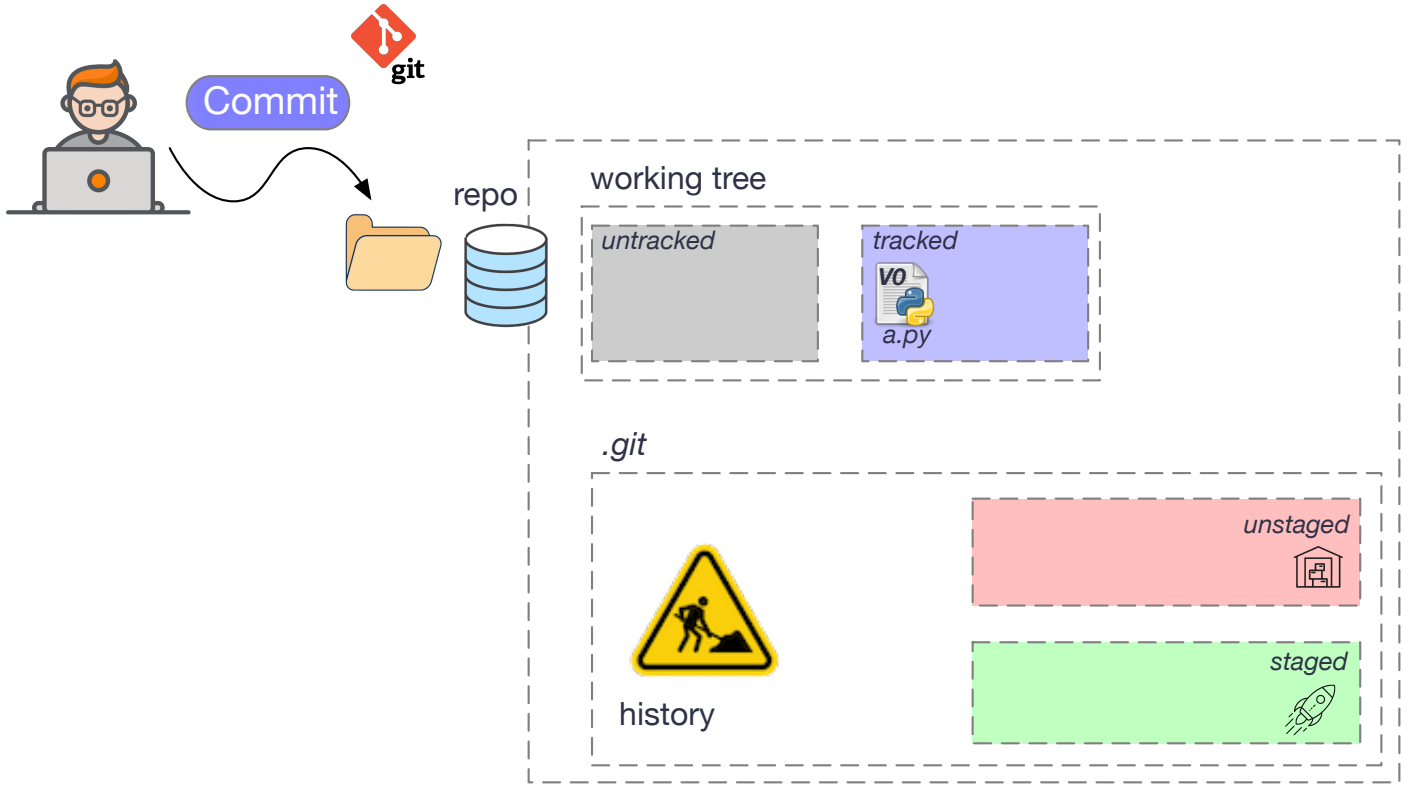
Produire une première version

- La **production d'une nouvelle version** est explicite, via la commande `git commit`

- Produire une nouvelle version
  - Un **message** doit accompagner le commit, il peut être spécifié entre " avec l'option `-m`
- Visualiser l'état du dépôt

IUT Valence
Université Grenoble Alpes

Produire une première version



- La zone *staged* est vidée, les modifications qu'elle contient forment la nouvelle version (**commit**)

Produire une première version

- Des informations sur la dernière version peuvent être visualisées à l'aide de la commande `git show`
- L'historique peut être visualisé à l'aide de la commande `git log`

- Visualiser les informations sur la version courante
- Visualiser l'historique

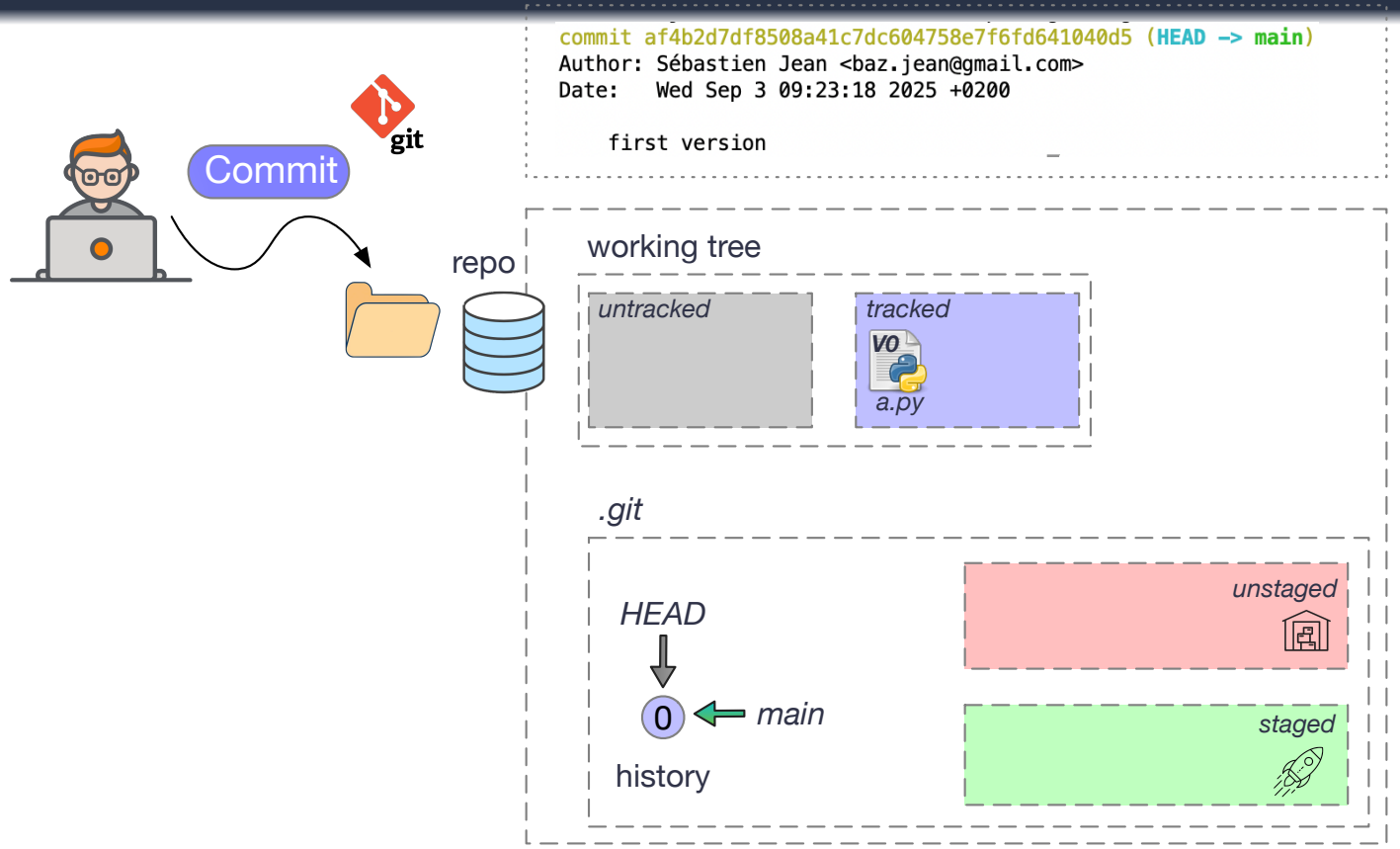
IUT Valence
Université Grenoble Alpes

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.
- Un commit est **unique** et ne peut **plus être modifié**

IUT Valence
Université Grenoble Alpes

Produire une première version



- A chaque commit est associé un identifiant unique sur 20 octets
- La référence `main` désigne le dernier commit de la branche `main`
- La référence `HEAD` désigne la version courante
  - Celle sur laquelle se base la reconstruction du *Working Tree*

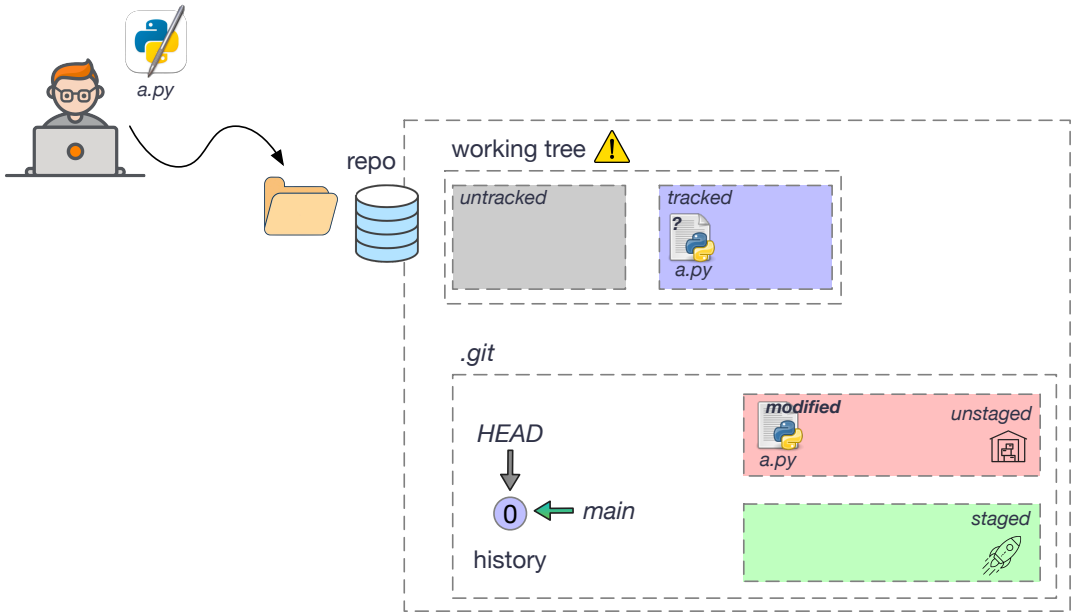
IUT Valence Université Grenoble Alpes

Produire une nouvelle version : Cas de modifications de ressources déjà suivies

- Modifier le fichier `a.py`
  - En modifiant la ligne de texte, ...
- Visualiser l'état du dépôt, l'historique et les informations sur la dernière version
- Exécuter la commande `git diff` **sans** option **puis avec** l'option `--staged`
- Intégrer les modifications dans la version à venir
- Exécuter la commande `git diff` **sans** option **puis avec** l'option `--staged`
- Visualiser (tout)
- Produire la nouvelle version
- Visualiser (tout), exécuter `git diff` (sans puis avec option)

IUT Valence Université Grenoble Alpes

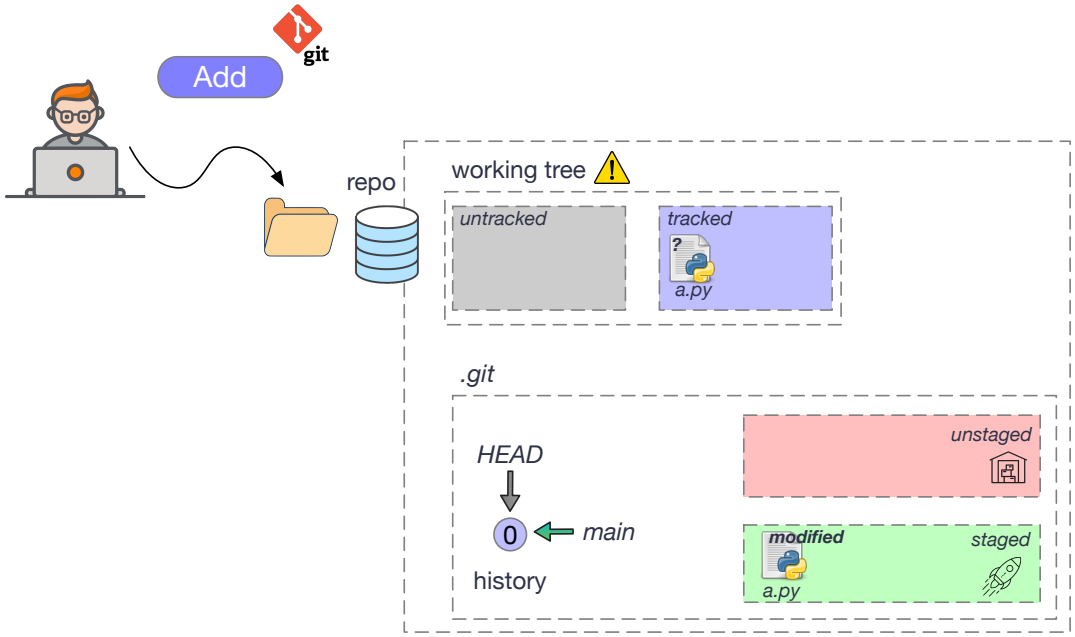
Produire une nouvelle version : Cas de modifications de ressources déjà suivies



- Les modifications sont **automatiquement** consignées dans la zone `unstaged`, contenant l'ensemble des modifications détectées qui n'ont **pas encore été intégrées** à la *future* nouvelle version
- La commande `git diff` (sans option) permet de visualiser les modifications de la zone *unstaged* par rapport à la version courante (`HEAD`)
  - Affichage des différences avec la syntaxe `diff` Unix

IUT Valence Université Grenoble Alpes

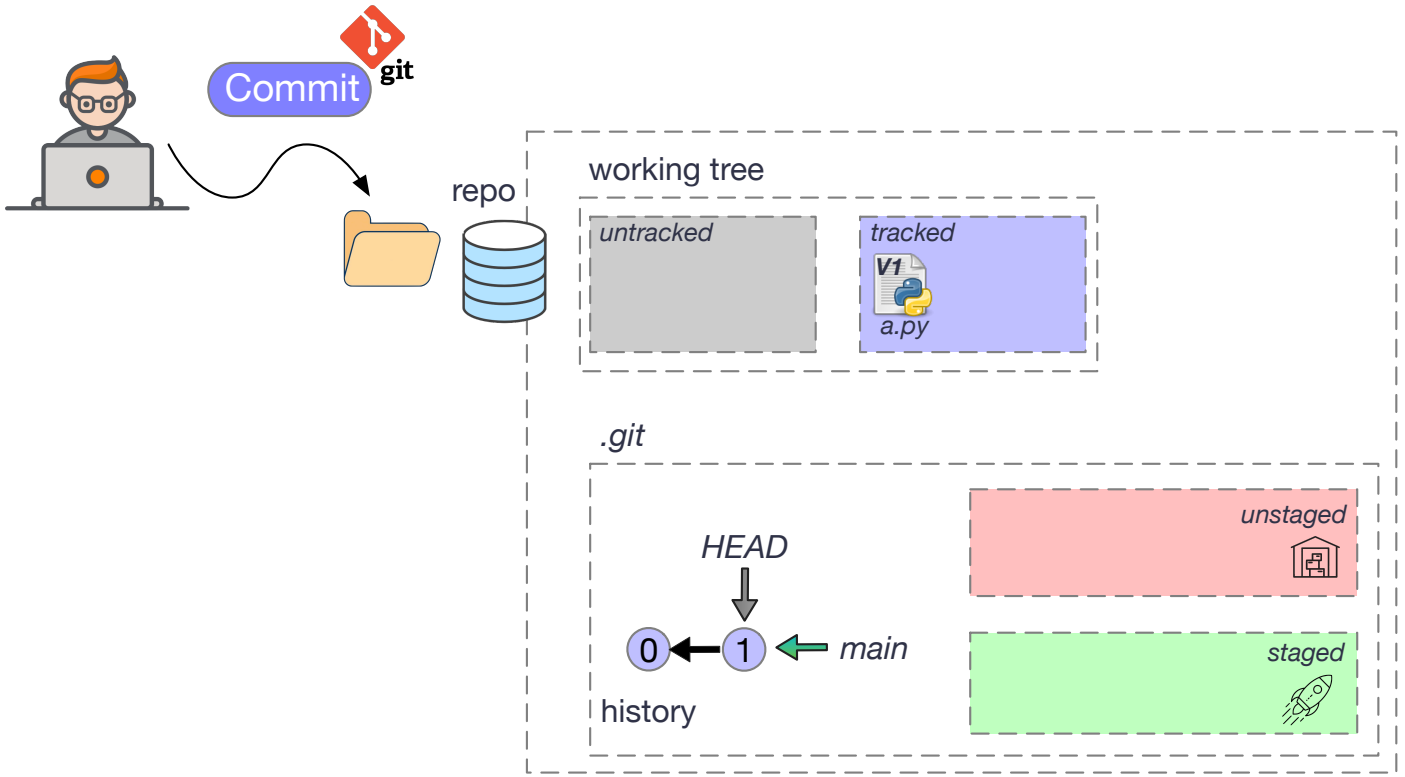
Produire une nouvelle version : Cas de modifications de ressources déjà suivies



- On déplace tout ou seulement une partie des modifications vers la zone `staged` à l'aide de la commande `git add` (réversible)
- La commande `git diff` (avec option `--staged`) permet de visualiser les modifications de la zone *staged* par rapport à la version courante (`HEAD`)

IUT Valence Université Grenoble Alpes

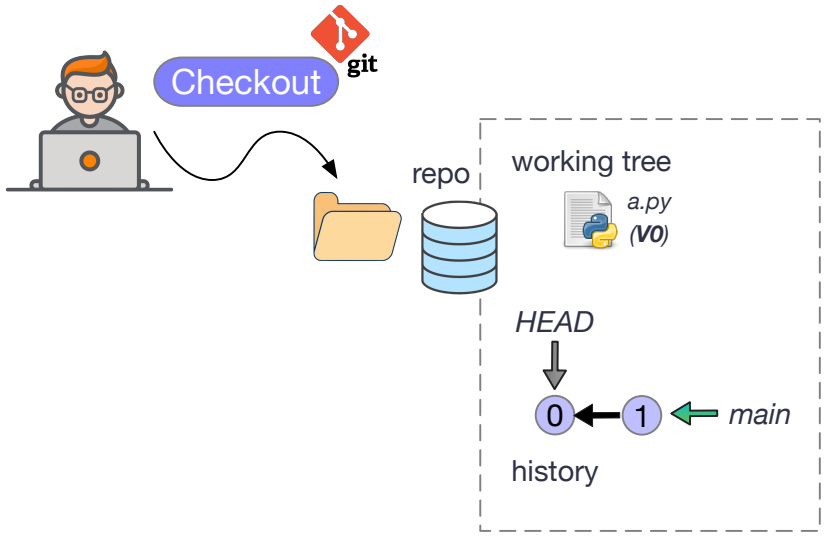
Produire une nouvelle version (fin)



- 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)
- On 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**

IUT Valence
Université Grenoble Alpes

Voyager dans le temps

- Il faut pouvoir **désigner le commit destination**, soit :
  - par une référence absolue :
    - son id. complet (ex : `af4b2d7df8508a41c7dc604758e7f6fd641040d5`)
    - son id. abrégé (ex : `af4b2d7`) → (premiers digits uniques)
  - par une référence relative :
    - `main` (commit **le plus récent**)
    - `HEAD` (commit **courant**)
    - `HEAD^` (le **parent direct du commit courant**)
    - `HEAD~2` (le **grand-parent du commit courant**)
    - `HEAD~n` (le **nième commit parent du commit courant**)
    - ...
  - ...

IUT Valence Université Grenoble Alpes

Voyager dans le temps

- Faire des allers-retours entre les 2 commits actuels de l'historique en utilisant le plus de désignations différentes possible
    - En observant l'historique **avec ou sans** l'option `--all` de `git log`

IUT Valence
Université Grenoble Alpes

Voyager dans le temps

- La commande `git checkout` peut aussi s'appliquer à des ressources et dans ce cas elle permet de **restaurer une version antérieure** dans le *Working Tree*
  - Les modifications induites par la restauration de la ressource sont automatiquement incluses dans la prochaine version (`git add` implicite)

- **A partir du commit le plus récent**, restaurer la version antérieure du fichier `a.py`
- Visualiser l'état du dépôt

Faire marche arrière

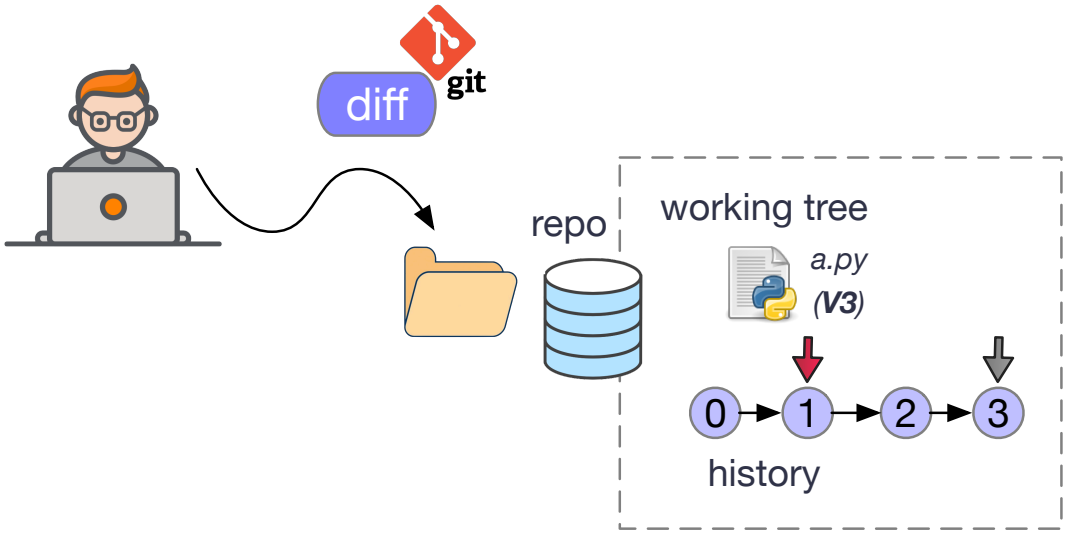
- On peut **refaire passer une ressource de la zone *staged* vers *unstaged*** à l'aide la commande `git restore` (option `--staged`)
- On peut **remplacer une ressource de la zone *unstaged* par sa version courante** (`HEAD`) à l'aide la commande `git restore`

- **A partir du commit le plus récent**, annuler la prise en compte des modifications du fichier `a.py`
- Visualiser l'état du dépôt
- Annuler les modifications du fichier `a.py`
- Visualiser l'état du dépôt

IUT Valence
Université Grenoble Alpes

Comparer

- On peut, à n'importe quel moment, avec la commande `git diff`, **comparer deux versions** d'un fichier



- Produire plusieurs autres versions avec des modifications diverses du fichier `a.py`
- Visualiser les différences (sur ce fichier) entre :
  - La **version courante** et la **première version**
  - **deux versions intermédiaires**

Annuler des modifications

- On peut **annuler les modifications introduites par un commit**, avec la commande `git revert`
  - Cela produit un nouveau commit, avec un **message généré automatiquement** (mais que l'on peut modifier)
- C'est **souvent le commit le plus récent** que l'on souhaite annuler
  - C'est sans risque
  - Annuler un **commit plus ancien** peut avoir un **impact sur le reste de l'historique**

- Annuler le commit le plus récent
- Visualiser l'historique et le contenu du fichier `a.py`
- Visualiser les différences (sur ce fichier) entre :
  - La **version courante** et la **version parente**
  - La **version courante** et la **version grand-parente**

IUT Valence Université Grenoble Alpes

Amender un commit

- On peut **modifier un commit**, avec l'option `--amend` de la commande `git commit`
  - Modifier le message
  - intégrer de nouvelles modification
- Cela remplace le commit par un nouveau commit
  - Amender le **commit le plus récent** est sans risque (en local)
  - Amender un **commit plus ancien** peut avoir un **impact sur le reste de l'historique**

IUT Valence Université Grenoble Alpes

Amender un commit

- Produire une nouvelle version (modification de `a.py`)
- Visualiser l'historique (bien noter l'id. de commit)
- Amender le message du commit le plus récent
- Visualiser l'historique (bien noter l'id. de commit)
- Modifier le fichier `a.py` et intégrer les modifications à la prochaine version (**sans la produire**)
- Amender le commit le plus récent
- Visualiser l'historique (bien noter l'id. de commit)

IUT Valence
Université Grenoble Alpes

Fin !

IUT Valence
Université Grenoble Alpes