DEV3 : gestion de version

loig.jezequel@univ-nantes.fr

Gestion de version

Point de départ

Quand on travaille sur un projet (typiquement du code) il est fréquent de vouloir :

1. revenir à un état passé d'un fichier,
2. voir les modifications d'un fichier au cours du temps,
3. savoir qui est l'auteur d'une modification donnée.

La méthode classique

Faire des copies de fichiers de temps en temps.

- peut gérer 1., éventuellement 2., mais difficilement 3.
- demande beaucoup de rigueur pour bien fonctionner,
- peu efficace en terme d'espace de stockage (redondance).

Gestion de version locale

Principe

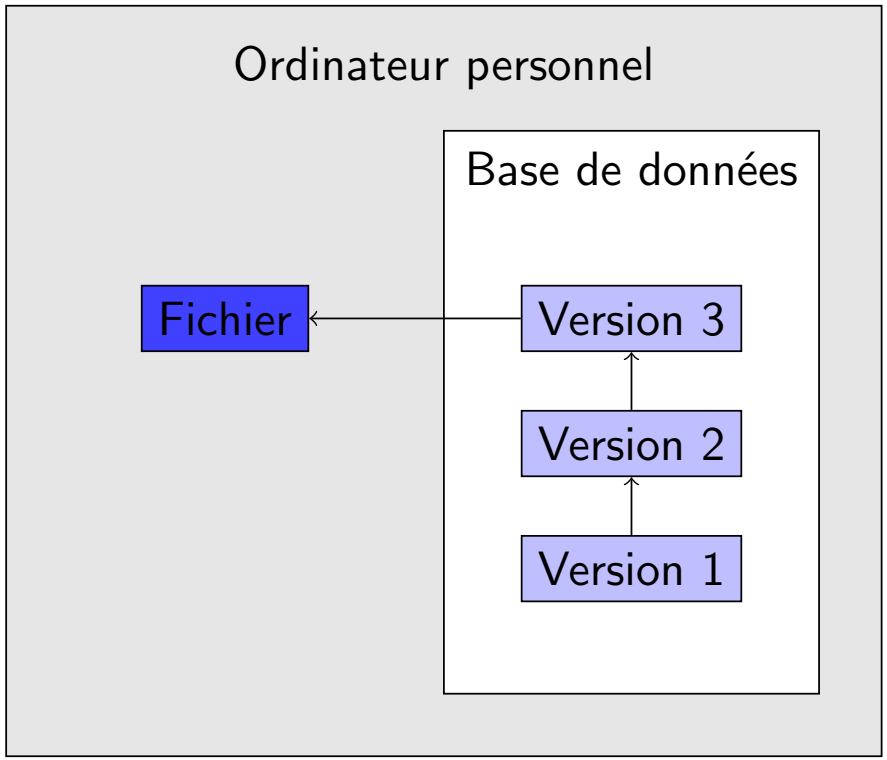
Une base de données stocke les changements faits sur les fichiers au cours du temps.

Avantages

- Gestion simple des versions,
- économie d'espace (versions stockées sous forme de *patchs*).

Limitations

- Travail à plusieurs difficile,
- risque de perte des données.



Gestion de version centralisée

Principe

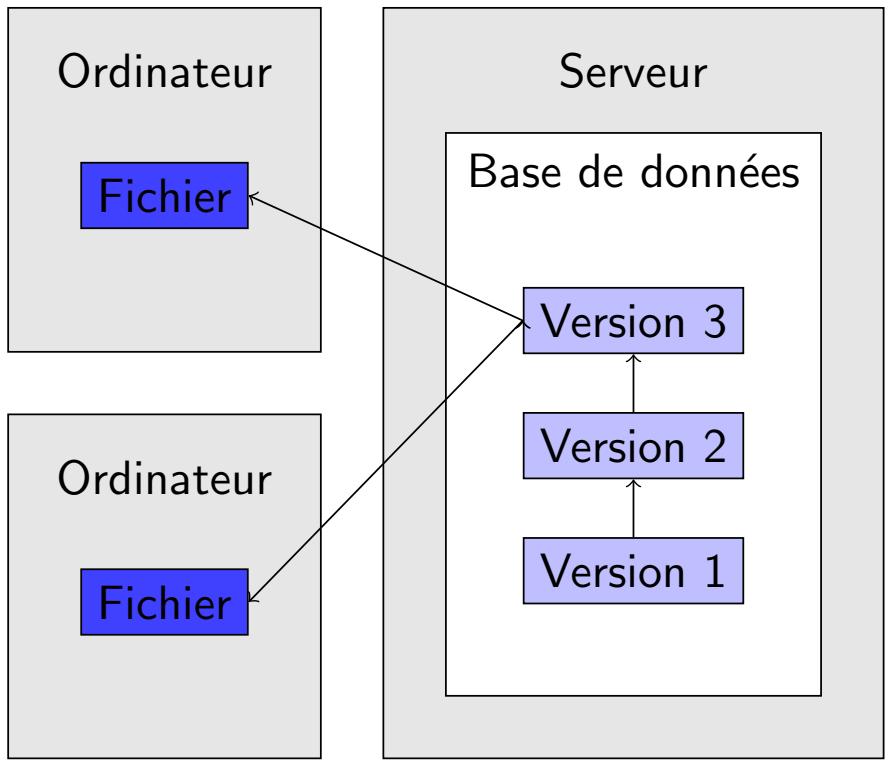
La base de données est stockée sur un serveur distant.

Avantages

- Travail à plusieurs possible,
- avec historique des actions de chacun,
- gestion des droits.

Limitations

- Risque de perte des données.



Remarque importante

Un risque inévitable

À partir du moment où on travaille à plusieurs sur les mêmes fichiers, il y a un risque de conflit.

Exemple de scénario menant à un conflit

1. Les utilisateurs A et B récupèrent la dernière version des fichiers sur le serveur,
2. A modifie des fichiers et enregistre cela sur le serveur,
3. B modifie des fichiers (moins rapidement) puis veut enregistrer ses modifications sur le serveur.

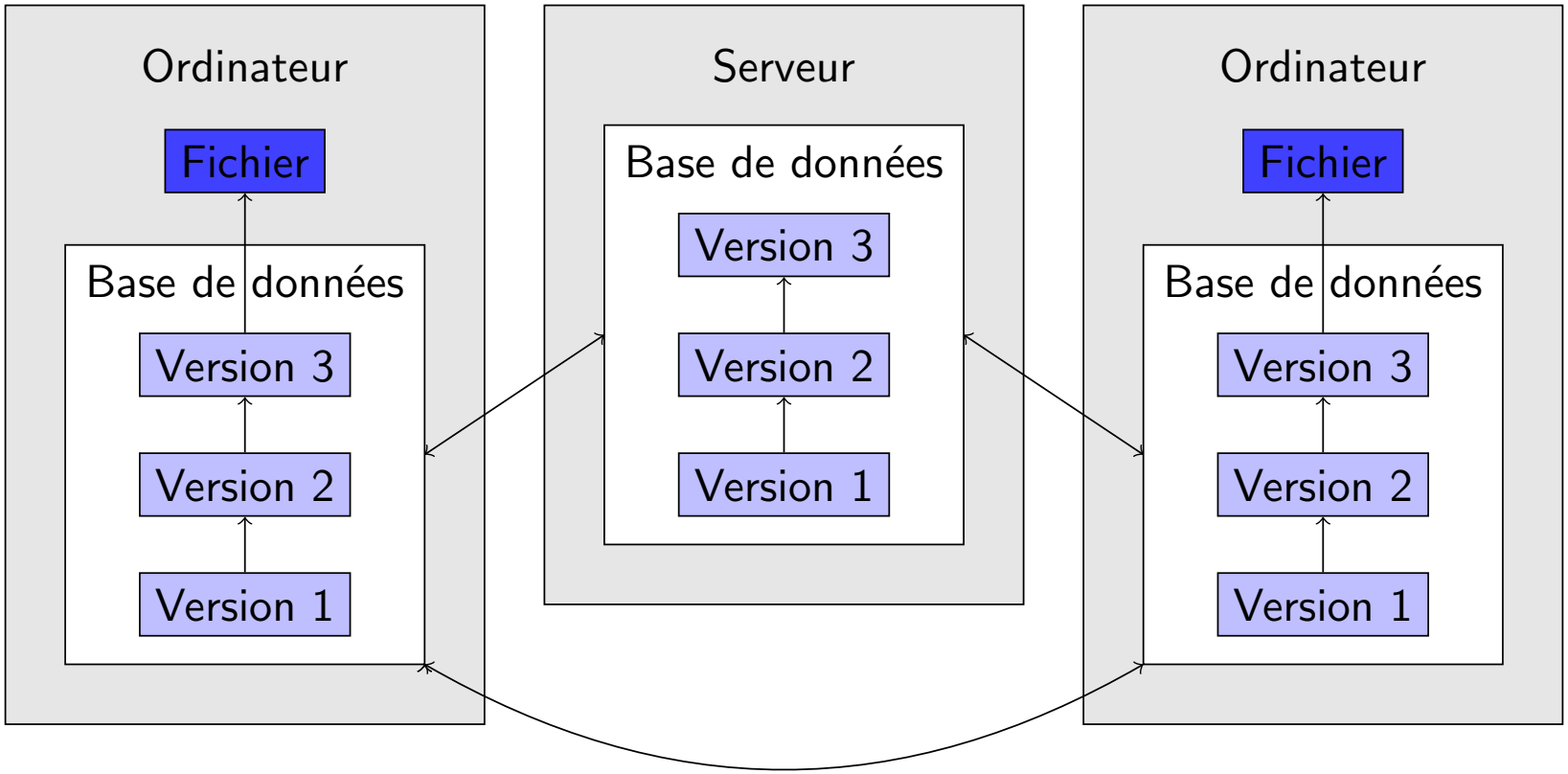
Résoudre les conflits

- Si les modifications en conflit concernent des (parties de) fichiers différents, les conflits sont résolus automatiquement,
- sinon, il n'y a pas d'autre choix que de les résoudre à la main (l'utilisateur qui détecte le conflit choisit quelle version de chaque fichier garder).

Gestion de version répartie

Principe

La base de données est dupliquée chez tous les utilisateurs.



Gestion de version répartie, suite

Avantages

- Tous ceux d'avant (gestion de version, travail en groupe),
- resistance aux pertes de données (duplications),
- possibilités d'organisations complexes (base de données partagée avec différents groupes dans différentes versions).

Inconvénients

- Conflits plus complexes (entres fichiers mais aussi entre versions de l'historique stocké en base de données),
- plus laborieux à utiliser (interactions avec la base de données locale et avec le serveur).

Git (github, gitlab) est un système de gestion de version réparti !

Git : copie locale

Initialisation

Pour commencer la gestion de version (du contenu) d'un répertoire il suffit de se placer dans se répertoire et d'utiliser la commande

git init

Effet de l'initialisation

on regarde un .git

Création d'un répertoire .git dans le dossier courant. Ce répertoire correspond au dépôt git nouvellement créé.

Voir l'état du dépôt

on essaye

On utilise la commande git status

Ajouter des fichiers au dépôt

on essaye

On utilise la commande git add *filename*

Git : états des fichiers

Trois états possibles pour les fichiers versionnés

modified : le fichier a été modifié localement mais les modifications n'ont pas été stockées en base,

staged : le fichier a été marqué comme modifié pour être inclus dans la prochaine mise-à-jour de la base de données

committed : la dernière version du fichier est stockée en base.

Passer de l'état *staged* à l'état *committed* on essaye

On utilise la commande git commit

Passer de l'état *committed* à l'état *modified*

On modifie simplement le fichier

Passer de l'état *modified* à l'état *staged* on essaye

On utilise la commande git add *filename*

Git : autres commandes utiles

Supprimer des fichiers — on essaye

On utilise la commande git rm *filename*

Voir l'historique des versions — on essaye

On utilise la commande git log

Revenir à une version précédente — on essaye

On utilise la commande git checkout *numcommit* .

- le point à la fin est important, sans lui le retour est temporaire (on voit un état précédent mais on ne peut pas le changer),
- git peut être beaucoup plus subtile sur les retours en arrière (fichier par fichier),
- en pratique on fait rarement ça, on crée des branches (mais ça sort du spectre de ce cours).

Git : copie distante

Créer une copie d'un dépôt distant — on essaye

On utilise la commande git clone *repositoryaddress*

Envoyer ses modifications au serveur — on essaye

On utilise la commande git push

- pour des raisons pratiques, on ne le fait que sur un dépôt nu : qui n'a qu'une base de données et pas de copie locale des fichiers (pas de répertoire de travail)

Mettre à jour sa copie locale à partir du serveur — on essaye

On utilise la commande git pull

Git : gestion des conflits

Détection et correction des conflits

- Détection lors d'un git push,
- pour résoudre, trois étapes :
  1. git pull pour récupérer la version en conflit,
  2. résolution locale du conflit,
  3. enregistrement en base de la solution.

Conflits sans incidence on essaye

Si on modifie des fichiers différents dans deux copies d'un dépôt, un conflit est détecté, mais il est corrigé automatiquement.

Véritables conflits on essaye

- Les fichiers avec conflits sont signalés,
- les deux versions sont visibles dans ces fichiers,
- on choisit une version à la main.

Dernières remarques

Fichiers non versionnés

On peut lister dans un fichier .gitignore les fichiers qui ne doivent pas être versionnés.

- On ne versionne pas tous les fichiers,
- les fichiers binaires ne devraient par l'être,
- comme n'importe quel fichier pour lequel on ne sait pas gérer les conflits à la main.

Git fait beaucoup plus que ce qu'on a vu aujourd'hui

Branches (pour versions alternatives d'un projet), tags de versions (pour identifier les versions significatives), etc

Pour aller plus loin

Un bon livre disponible gratuitement :
`https://git-scm.com/book`