Gestion des versions - Versioning

Utilitaire *git*

Système de gestion de versions

Définitions

Un gestionnaire de version **est un système qui enregistre l'évolution d'un fichier ou d'un ensemble de fichiers d'un même projet au fil du temps de manière**, à ce qu'on puisse rappeler une version antérieure d'un fichier à tout moment.

- Un **logiciel de gestion de versions** (ou **VCS** en anglais, pour *Version Control System*) est un logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus.

- Il permet notamment de retrouver les différentes versions d'un lot de fichiers connexes.

Source : https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions

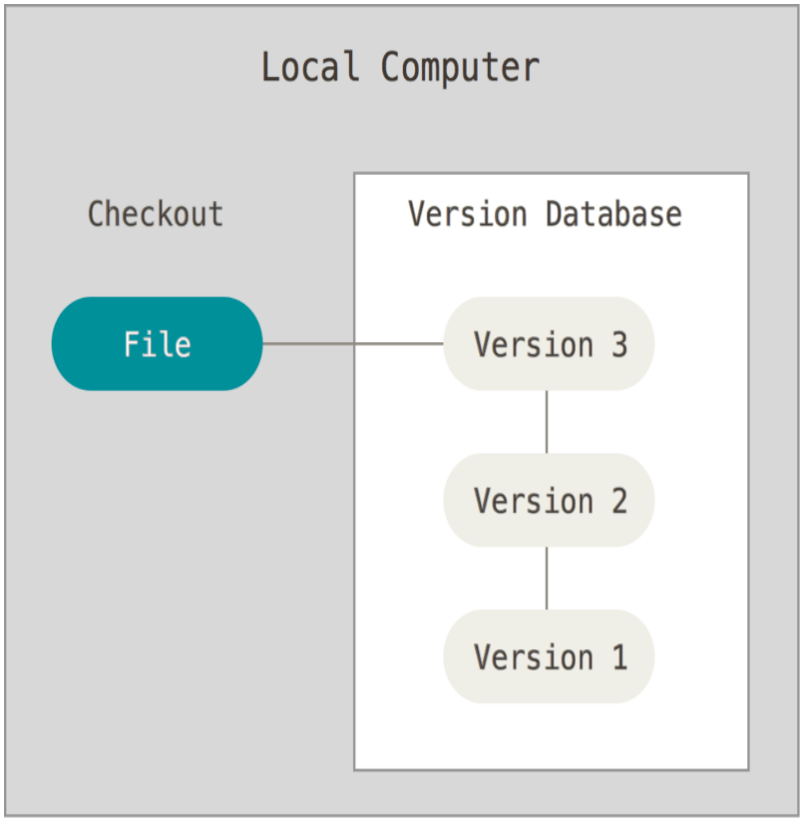
Pourquoi un système de gestion de version ?

- Retour facile aux versions précédentes
- Suivi de l'évolution du projet au cours du temps
- Travail en parallèle sur des parties disjointes du projet et gestion des modifications concurrentes
- Détection et correction des erreurs
- Identification des responsabilités et des modifications apportées : QUI FAIT QUOI ?
- Accès à la version d'un fichier à une date donnée

Catégories de SGV

Système de gestion de version local

Cette méthode est la plus courante pour la gestion de version est généralement de recopier les fichiers dans un autre répertoire (peut-être avec un nom incluant la date dans le meilleur des cas).



Avantages

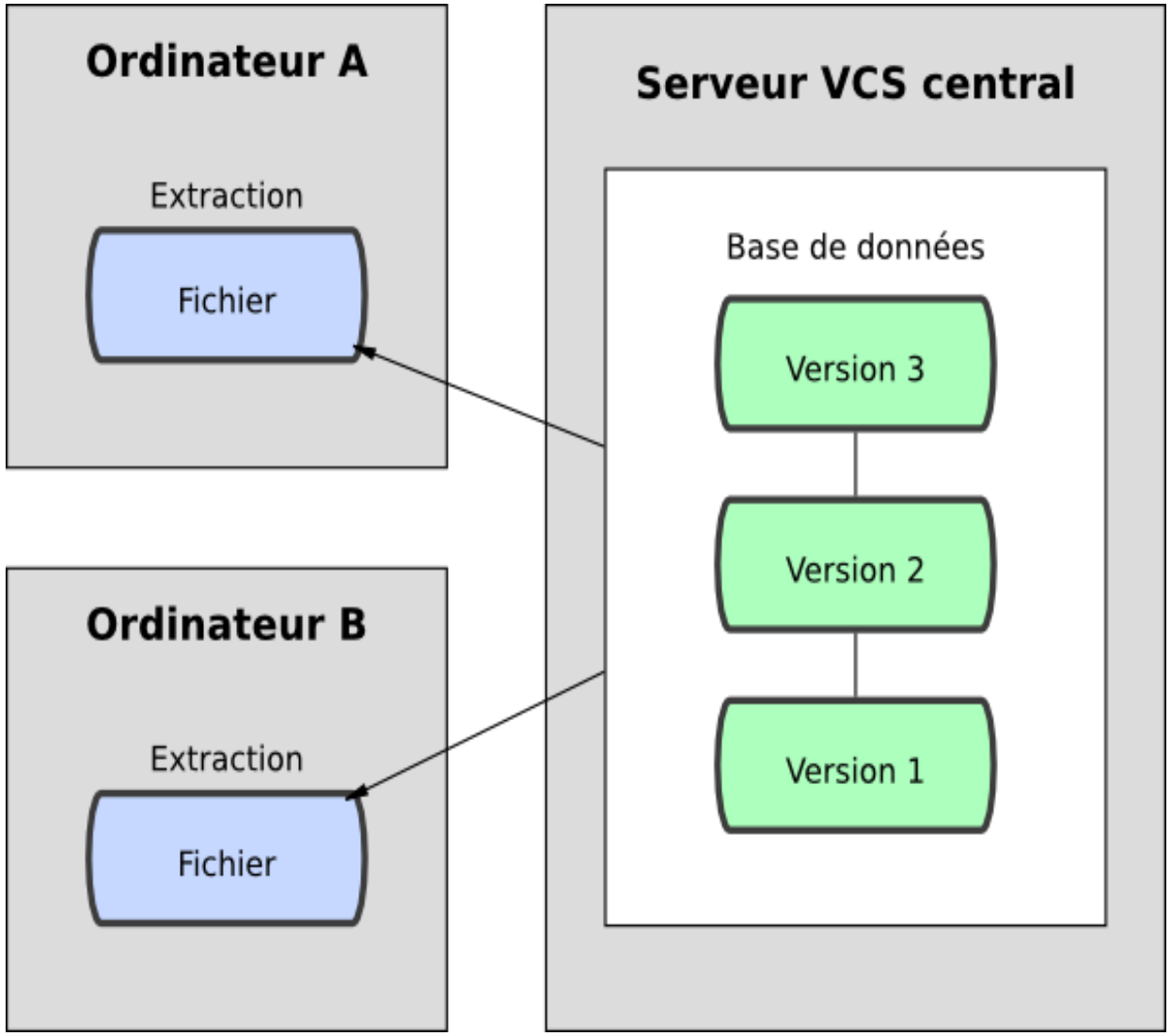
- Gestion et utilisation très simplifiées.

Inconvénients

- Très sensible aux pannes.
- Ne permet pas la collaboration.

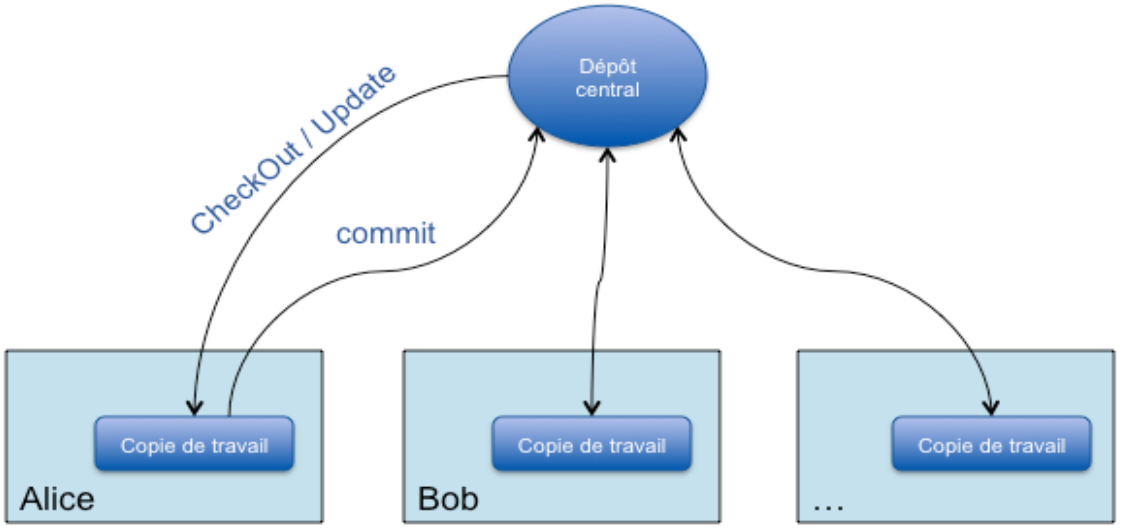
Catégories de SGV

Système de gestion de version centralisé



Catégories de SGV

Système de gestion de version centralisé (SVN)



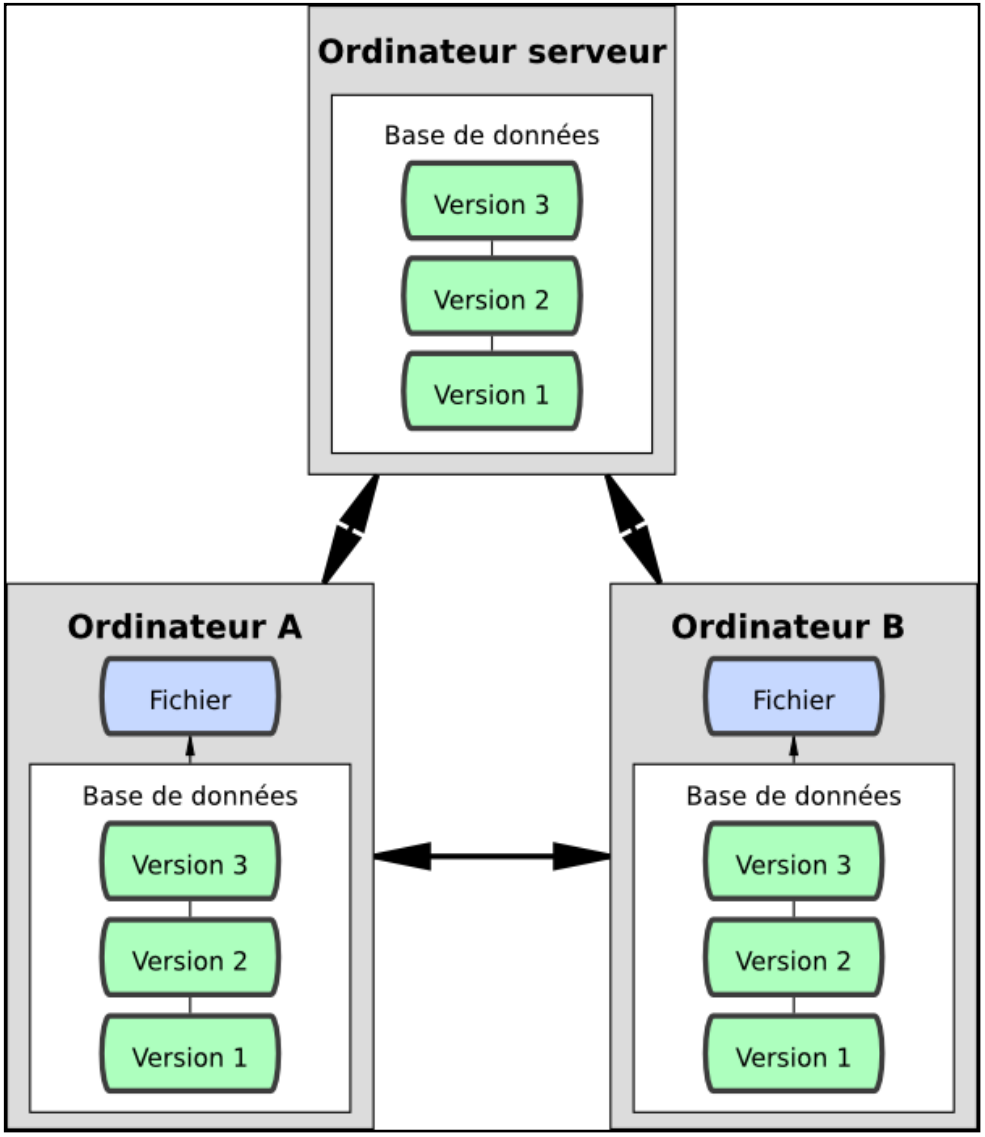
Avantages

- Gestion et utilisation simples.
- Contrôle fin des permissions
- Chaque utilisateur sait ce que font les autres

Inconvénients

- Méthode très sensible aux pannes.
- Inadaptée aux très grands projets et/ou avec une forte structure hiérarchique.

Système de gestion de version distribué



Catégories de SGV

Dans un DVCS (tel que Git, Mercurial, Bazaar or Darcs), les clients dupliquent complètement le dépôt.
Si le serveur disparaît, n'importe quel dépôt d'un des clients peut être copié sur le serveur pour le restaurer.
Chaque extraction devient une sauvegarde complète de toutes les données (voir figure 1-3).

Avantages

- Moins sensible aux pannes.
- Adapté aux grands projets
- Pouvoir utiliser ce système hors connexion
- Ne pas être dépendant d'un dépôt centralisé (panne, temps, . . .)
- Pouvoir échanger ses fichiers avec une partie des développeurs.

Inconvénients

- Gestion et utilisation plus compliquées.
- Peut devenir très complexe structurellement.

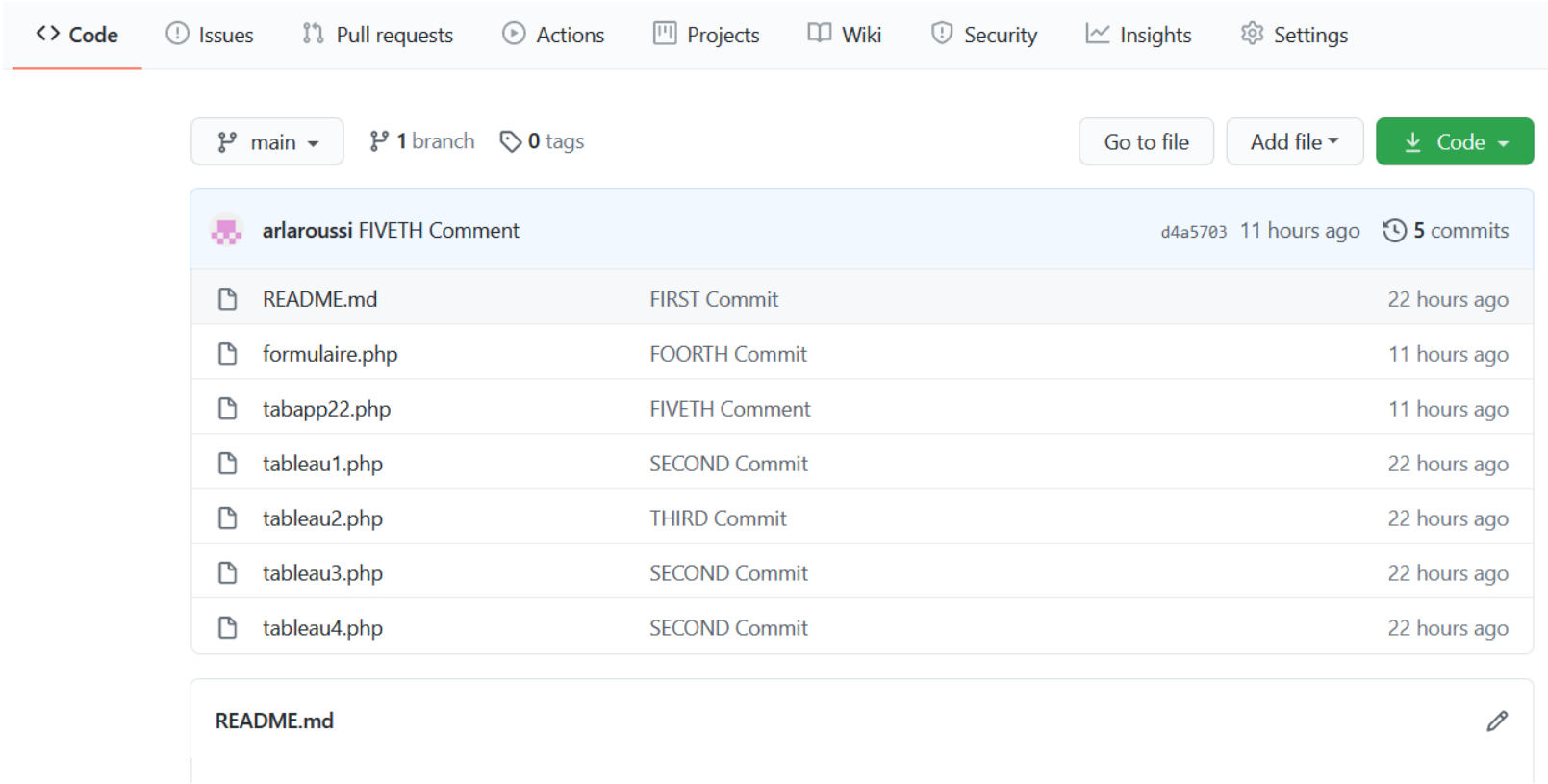
- SGV décentralisé DVCS (Distributed Version Control System)
- Très répandu (surtout dans le monde du logiciel libre)
- Développement actif
- Ressources croissantes (documentations, outils graphiques, ...).

Bref historique (Wikipedia)

- De 1991 à 2002, le noyau Linux était développé sans utiliser de système de gestion de version.
- A partir de 2002, la communauté a commencé à utiliser BitKeeper, un DVCS propriétaire.
- En 2005, suite à un contentieux, BitKeeper retire la possibilité d'utiliser gratuitement son produit.
- Linus Torvalds lance le développement de Git et après seulement quelques mois de développement, Git héberge le développement du noyau Linux.

Git - Les principes de base

Un dépôt (Repository) Git est une sorte de système de fichiers (base de données), enregistrant les versions de fichiers d'un projet à des moments précis au cours du temps sous forme d'instantanés.



Git - Les principes de base

Le répertoire Git/dépôt /Repository / Référentiel

Un dépôt Git est un entrepôt virtuel de votre projet. Il permet d'enregistrer les versions de votre code et d'y accéder au besoin.

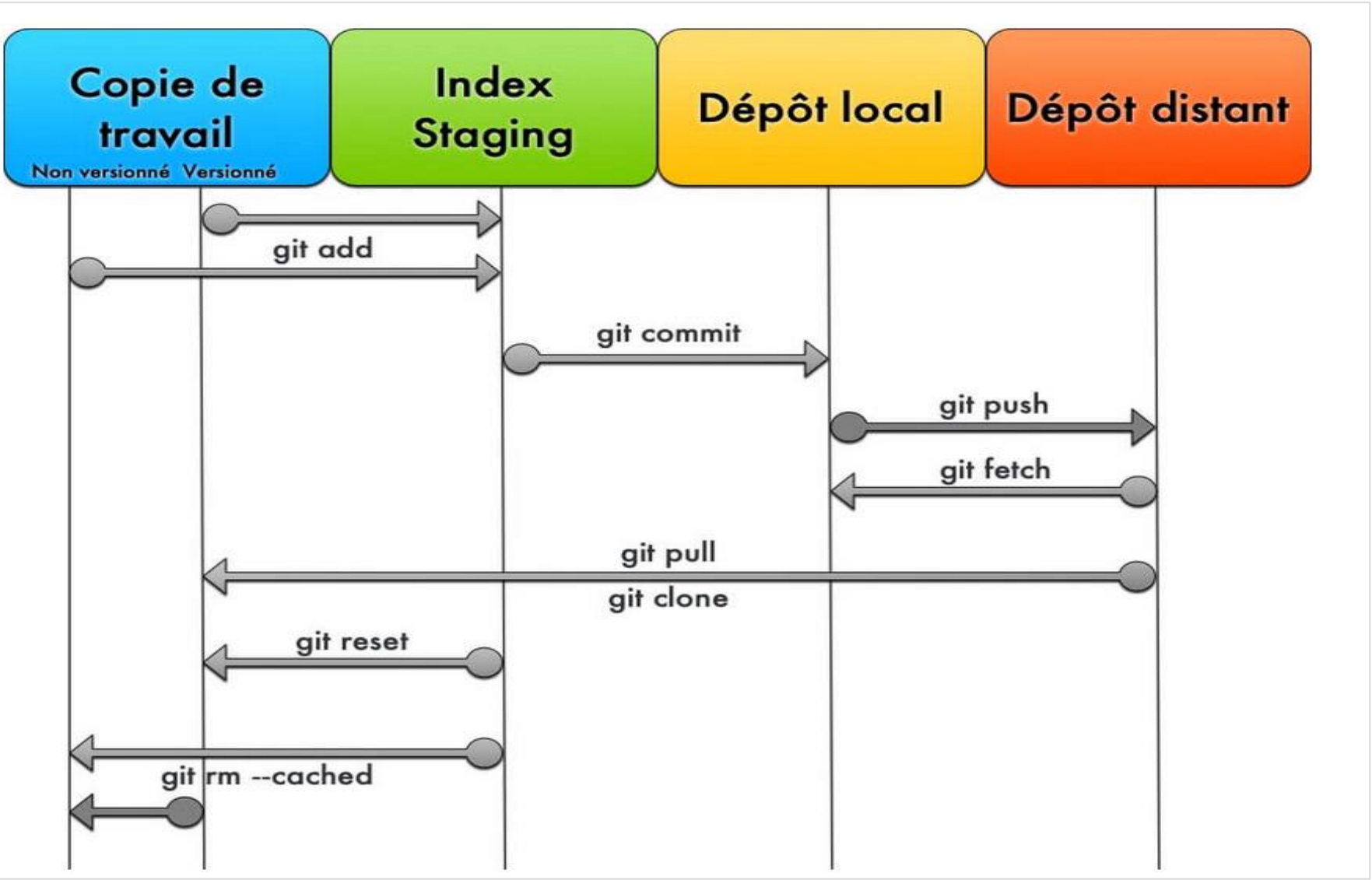
Le répertoire de travail

Extraction unique d'une version du projet depuis la base de données du dépôt.

La zone de transit/d'index

Simple fichier contenant des informations à propos de ce qui sera pris en compte lors de la prochaine soumission.

Etats d'un fichier dans Git



Commandes de base

1) Création d'un nouveau Repository

A partir de votre espace Github, créer un nouveau Repository (Référencement) et récupérer l'URL de votre dépôt.

Cette URL sera votre principal moyen d'accès au dépôt.

2) Installer git sur votre espace de travail local

```
#apt update & apt install git-all      //Linux
```

Télécharger et installer le package à partir du site de git (voir TP) `//Windows`

3. Initialiser un dépôt sur votre machine locale

```
git init
```

4) Associer votre espace de travail local à votre espace distant grâce aux commandes suivantes :

```
git branch -M main
git remote add origin https://github.com/arlaroussi/CPROG.git
git push -u origin main
```

Commandes de configuration globale

- La première chose à faire après l'installation de Git est de renseigner votre nom et votre adresse de courriel. C'est une information importante car toutes les validations dans Git utilisent cette information :

```
git config --global user.name "Sophie LAMBERT"
git config --global user.email slambert@infos.com
```

- Lister tous les paramètres de la configuration :

```
git config -list
```

- Définir un éditeur par défaut :

```
git config --global core.editor nano
```

- Paramétrer le serveur proxy :

```
git config --global http.proxy http://USER:PASSWORD@PROXYSERVER:PORT
git config -global unset http.proxy
```

- Exclure des fichiers lors de l'envoi vers le serveur distant

```
git config --global core.excludefiles **/*.log
```

```
echo "# phpjob" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin
https://github.com/arlaroussi/phpjob.git

git push -u origin main
```

Commandes de base

- Indexer l'ajout ou les changements d'un fichier

```
git add fichier
```

- Annuler les modifications indexées d'un fichier

```
git reset  fichier
```

- Afficher les dépôts distants

```
git remote -v
    origin   https://github.com/schacon/ticgit (fetch)
    origin   https://github.com/schacon/ticgit (push)
```

- Retirer le fichier de l'index (Déversionner), et le supprimer du cache

```
git  rm fichier
```

- Retier le fichier de l'index (Déversionner), mais le garder dans le cache

```
git  rm  --cached fichier
```

- Annuler un commit et revenir à l'état initial

```
git  checkout <num_commit>
```

Commandes de base

- Afficher le détail des modifications non indexées

```
git diff
```

- Afficher le détail des modifications indexées

```
git diff --staged
```

- Visualiser ce qui a été indéxé

```
git diff --cached
```

- Soumettre les modifications indexées en zone de transit

```
git commit -m "Message"
```

- Voir l'historique des soumissions

```
git log
```

→ on pourra utiliser l'option *--oneline* pour moins de details

- Récupérer les données modifiées depuis le serveur distant et les fusionner dans votre branche de travail actuel.

```
git pull ServeurDist branch
```

Cette commande équivaut aux deux commandes suivantes :

```
git fetch
git merge origin
```

Commandes de base

- Cloner un dépôt distant.

```
git clone git://git.kernel.org/pub/scm/.../linux.git rep-local
cd rep-local
```

- Annuler des commit et revenir aux états précédents :

```
git revert HEAD (dernier commit)
git revert HEAD~ (2 derniers commit )
git revert HEAD~2 (3 derniers commit)
```

  - HEAD : dernier commit ;
  - HEAD^ : avant-dernier commit ;
  - HEAD^^ : avant-avant-dernier commit ;
  - HEAD~2 : avant-avant-dernier commit (notation équivalente) ;
  - d6d98923868578a7f38dea79833b56d0326fcba1 : indique un numéro de commit ;
  - d6d9892 : indique un numéro de commit version courte.

- Afficher l'état des fichiers du répertoire courant

```
git status
```

- *Untracked files* : fichiers non versionnés.
- *Changes to be committed* : modifications (ajout, suppression, changements) chargées en zone de transit (staging area), ou indexées.
- *Changes not staged for commit* : modifications n'ayant pas été chargées en zone de transit (ou indexées).