**Bloc 01 – Git**

1. **Introduction**

Un gestionnaire de version est un système qui enregistre l’évolution d’un fichier ou d’un ensemble de fichiers au cours du temps de manière à ce qu’on puisse rappeler une version antérieure d’un fichier à tout moment.

1. **Pourquoi faire de la gestion de version ?**

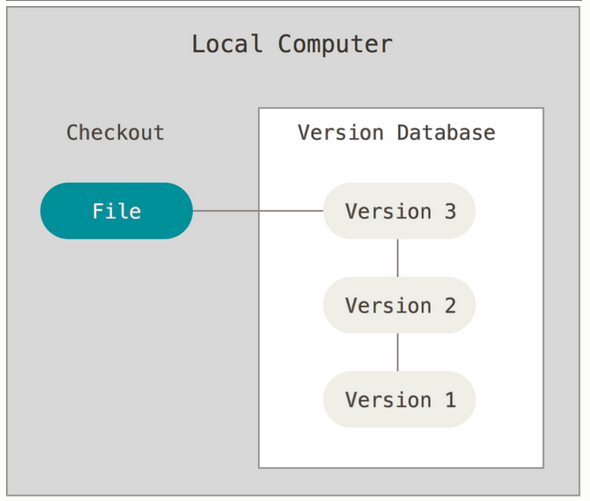
* Vous permet de ramener n’importe quel fichier ou projet à un état antérieur.
* En cas d’erreur ou perte de fichier, vous permet de revenir à un état stable de votre projet.
* Demande peut de travail additionnel.

Il existe plusieurs familles de gestionnaire de versions :

* 1. [**Les systèmes de gestion de version locaux**](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version#Les-syst%C3%A8mes-de-gestion-de-version-locaux)

La méthode à plus simple, ou la plus intuitive pour gérer les versions d’un projet est de recopier les fichiers dans un autre répertoire (peut-être avec un nom incluant la date dans le meilleur des cas), ce qui est peu fiable …

La première solution pour gérer les versions, créé par les développeurs est VCS (Version Control System) locaux qui utilisaient une base de données simple pour conserver les modifications d’un fichier.



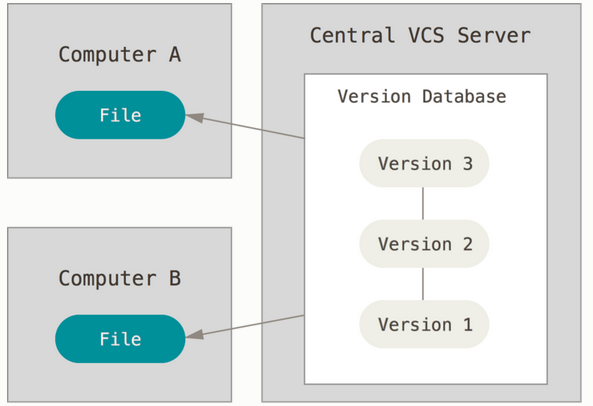
*Gestion de version locale. Source : Démarrage rapide - À propos de la gestion de version. Repéré au :* [*https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version*](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version)

Le schéma ci-dessus représente une solution fonctionnelle localement, d’ailleurs plusieurs systèmes d’exploitation tels que Mac OS inclut des logiciels de ce type tel que RCS

Ceci dit cette solution ne permet une collaboration entre deux développeurs sur deux ordinateurs différents. D’où vient la nécessité de créer un système **centralisé.** C’est ainsi que les CVCS *( Centralized Version Control Systems* ) furent développés, tels que CVS, Subversion (SVN), et Perforce.

* 1. [**Les systèmes de gestion de version centralisés**](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version#Les-syst%C3%A8mes-de-gestion-de-version-centralis%C3%A9s)

Les systèmes de gestion de version centralisés mettent en place un serveur central qui contient tous les fichiers sous gestion de version, et des clients qui peuvent extraire les fichiers de ce dépôt central.



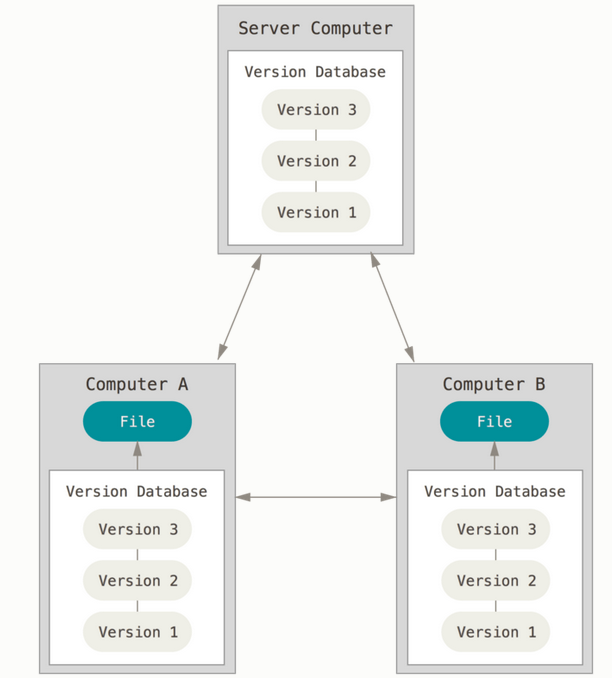
*Gestion de version centralisée. Source : Démarrage rapide - À propos de la gestion de version. Repéré au :* [*https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version*](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version)

Ce type de système a plusieurs avantages :

* Chacun sait jusqu’à un certain point ce que tous les autres sont en train de faire sur le projet
* Les administrateurs ont un contrôle fin des permissions
* Il est beaucoup plus facile d’administrer un CVCS que de gérer des bases de données locales.

Ce type de système a aussi plusieurs défauts :

* Le fait que le serveur soit au centre de toutes les utilisations rend l’utilisation fragile. En effet si le serveur tombe en panne aucun client ne peut collaborer ou enregistrer les modifications issues de son travail
* Si le disque du serveur tombe en panne, sans aucune sauvegarde vous perdez absolument tout de l’historique d’un projet
* Ceci dit les systèmes de gestion de version locaux souffrent du même problème.
  1. [**Les systèmes de gestion de version distribués**](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version#Les-syst%C3%A8mes-de-gestion-de-version-distribu%C3%A9s)
* Les systèmes de gestion de version distribués DVCS *Distributed Version Control Systems* représente une altérnative au CVCS.
* Il existe plusieurs DVCS tel que Git, Mercurial, Bazaar ou Darcs,
* Dans les DVCS, les clients n’extraient plus seulement la dernière version d’un fichier, mais ils dupliquent complètement le dépôt.
* Dans ce type de système si le serveur disparaît et si les systèmes collaboraient via ce serveur, 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.



*Gestion de versions distribuées. Source : Démarrage rapide - À propos de la gestion de version. Repéré au :* [*https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version*](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version)

* Un grand nombre de ces systèmes gère particulièrement bien le fait d’avoir plusieurs dépôts avec lesquels travailler, vous permettant de collaborer avec différents groupes de personnes de manières différentes simultanément dans le même projet.
* Cela permet la mise en place de différentes chaînes de traitement tels que les modèles hiérarchiques.

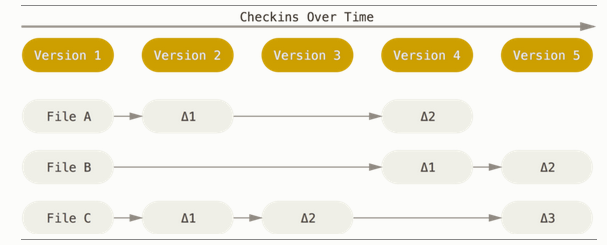
1. **Une rapide histoire de Git**

* Git est née avec le travail des différents collaborateurs sur le projet portant sur le noyau de Linux.
* les modifications étaient transmises sous forme de patchs et d’archives de fichiers.
* En 2002, le projet du noyau Linux commença à utiliser un DVCS propriétaire appelé BitKeeper.
* En résultat de problèmes entre la communauté de développement du noyau Linux et la société en charge du développement de BitKeeper, la communauté Linux décida de développer son propre produits (en se basant sur les leçons apprises lors de l’utilisation de BitKeeper).
* Certains des objectifs du nouveau système étaient les suivants :
  + vitesse ;
  + conception simple ;
  + support pour les développements non linéaires (milliers de branches parallèles) ;
  + complètement distribué ;
  + capacité à gérer efficacement des projets d’envergure tels que le noyau Linux (vitesse et compacité des données).

1. **Démarrage rapide - Rudiments de Git**

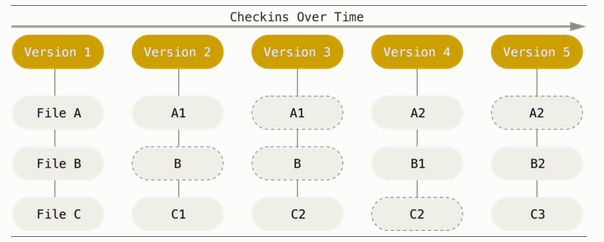
**4.1** [**Des instantanés, pas des différences**](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Rudiments-de-Git#Des-instantan%C3%A9s,-pas-des-diff%C3%A9rences)

* La différence majeure entre Git et les autres VCS réside dans la manière dont Git considère les données.
* Au niveau conceptuel, la plupart des autres systèmes gèrent l’information comme une liste de modifications de fichiers. Ces systèmes (CVS, Subversion, Perforce, Bazaar et autres) considèrent l’information qu’ils gèrent comme une liste de fichiers et les modifications effectuées sur chaque fichier dans le temps.



*D’autres systèmes sauvent l’information comme des modifications sur des fichiers. Source : Démarrage rapide - À propos de la gestion de version. Repéré au :* [*https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version*](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version)

* Git ne gère pas et ne stocke pas les informations de cette manière. À la place, Git pense ses données plus comme un instantané d’un mini système de fichiers.
* À chaque fois que vous validez ou enregistrez l’état du projet dans Git, il prend effectivement un instantané du contenu de votre espace de travail à ce moment et enregistre une référence à cet instantané.
* Pour être efficace, si les fichiers n’ont pas changé, Git ne stocke pas le fichier à nouveau, juste une référence vers le fichier original qu’il a déjà enregistré.
* Git pense ses données plus à la manière d’un **flux d’instantanés**.



*Git stocke les données comme des instantanés du projet au cours du temps. Source : Démarrage rapide - À propos de la gestion de version. Repéré au :* [*https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version*](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version)

* Git ressemble beaucoup plus à un mini système de fichiers avec des outils incroyablement puissants construits dessus, plutôt qu’à un simple VCS.
* Ceci dit la gestion des branches reste un grand avantage à l’utilisation de Git.

**4.2** [**Presque toutes les opérations sont locales**](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Rudiments-de-Git#Presque-toutes-les-op%C3%A9rations-sont-locales)

* La plupart des opérations de Git ne nécessitent que des fichiers et ressources locaux — généralement aucune information venant d’un autre ordinateur du réseau n’est nécessaire.
* Comme vous disposez de l’historique complet du projet localement sur votre disque dur, la plupart des opérations semblent instantanées.
* Par exemple, pour parcourir l’historique d’un projet, Git n’a pas besoin d’aller le chercher sur un serveur pour vous l’afficher ; il n’a qu’à simplement le lire directement dans votre base de données locale.
* Si vous souhaitez connaître les modifications introduites entre la version actuelle d’un fichier et son état un mois auparavant, Git peut rechercher l’état du fichier un mois auparavant et réaliser le calcul de différence, au lieu d’avoir à demander cette différence à un serveur ou de devoir récupérer l’ancienne version sur le serveur pour calculer la différence localement.
* Cela signifie aussi qu’il y a très peu de choses que vous ne puissiez réaliser si vous n’êtes pas connecté ou hors VPN.

**4.3** [**Git gère l’intégrité**](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Rudiments-de-Git#Git-g%C3%A8re-l%E2%80%99int%C3%A9grit%C3%A9)

* Dans Git, tout est vérifié par une somme de contrôle avant d’être stocké.
* Par la suite cette somme de contrôle, signature unique, sert de référence.
* Cela signifie qu’il est impossible de modifier le contenu d’un fichier ou d’un répertoire sans que Git ne s’en aperçoive.
* Vous ne pouvez pas perdre des données en cours de transfert ou corrompre un fichier sans que Git ne puisse le détecter.
* Le mécanisme que Git utilise pour réaliser les sommes de contrôle est appelé une empreinte SHA-1. C’est une chaîne de caractères composée de 40 caractères hexadécimaux (de 0 à 9 et de a à f) calculée en fonction du contenu du fichier ou de la structure du répertoire considéré. Une empreinte SHA-1 ressemble à ceci :

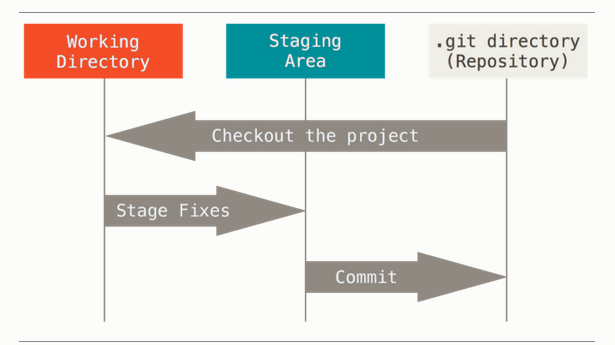
24b9da6552252987aa493b52f8696cd6d3b00373

**4.4** [**Généralement, Git ne fait qu’ajouter des données**](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Rudiments-de-Git#G%C3%A9n%C3%A9ralement,-Git-ne-fait-qu%E2%80%99ajouter-des-donn%C3%A9es)

* Quand vous réalisez des actions dans Git, la quasi-totalité d’entre elles ne font qu’ajouter des données dans la base de données de Git.
* Il est très difficile de faire réaliser au système des actions qui ne soient pas réversibles ou de lui faire effacer des données d’une quelconque manière.
* Par contre, comme dans la plupart des systèmes de gestion de version, vous pouvez perdre ou corrompre des modifications qui n’ont pas encore été entrées en base ; mais dès que vous avez validé un instantané dans Git, il est très difficile de le perdre, spécialement si en plus vous synchronisez votre base de données locale avec un dépôt distant.

**4.5** [**Les trois états**](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Rudiments-de-Git#Les-trois-%C3%A9tats)

* Git gère trois états dans lesquels les fichiers peuvent résider : **validé**, **modifié** et **indexé**.
  + **Validé** signifie que les données sont stockées en sécurité dans votre base de données locale.
  + **Modifié** signifie que vous avez modifié le fichier mais qu’il n’a pas encore été validé en base.
  + **Indexé** signifie que vous avez marqué un fichier modifié dans sa version actuelle pour qu’il fasse partie du prochain instantané du projet.
* Ceci nous mène aux trois sections principales d’un projet Git :
  + Le répertoire Git,
  + Le répertoire de travail
  + La zone d’index.



*Répertoire de travail, zone d’index et répertoire Git. Source : Démarrage rapide - À propos de la gestion de version. Repéré au :* [*https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version*](https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version)

* **Le répertoire Git** est l’endroit où Git stocke les méta-données et la base de données des objets de votre projet. C’est la partie la plus importante de Git, et c’est ce qui est copié lorsque vous clonez un dépôt depuis un autre ordinateur.
* **Le répertoire de travail** est une extraction unique d’une version du projet. Ces fichiers sont extraits depuis la base de données compressée dans le répertoire Git et placés sur le disque pour pouvoir être utilisés ou modifiés.
* **La zone d’index** est un simple fichier, généralement situé dans le répertoire Git, qui stocke les informations concernant ce qui fera partie du prochain instantané. On l’appelle aussi des fois la zone de préparation.

L’utilisation standard de Git se passe comme suit :

1. vous modifiez des fichiers dans votre répertoire de travail ;
2. vous indexez les fichiers modifiés, ce qui ajoute des instantanés de ces fichiers dans la zone d’index ;
3. vous validez, ce qui a pour effet de basculer les instantanés des fichiers de l’index dans la base de données du répertoire Git.

**Référence :**

Démarrage rapide - À propos de la gestion de version

Repéré au : <https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version>