

Formation Git

Devenir autonome dans l'utilisation de l'outil de gestion de version **Git**



Arnaud MERCIER
arnaud.mercier@hexotech.fr



Bienvenu dans cette formation Git qui a pour objectif de vous rendre autonome dans l'utilisation de Git. L'objectif est de ne plus dupliquer son environnement par peur de perdre des données lors de la manipulation de commandes GIT.

Présentation du formateur



Codeur-pro
45k abonnés



Arnaud Mercier
20k élèves



Arnaud MERCIER
Président d'HexoTech

En bref:

- Passionné de développement logiciel et de nouvelles technologies.
- Un "touche à tout".
- Formateur en ligne via des plateformes telles que Youtube et Udemy.
- Devops pour plusieurs clients (CI/CD, GitLab, Docker, ...).
- Architecte et développeur logiciel.

Je suis Arnaud MERCIER, Expert logiciel et président d'HexoTech. J'utilise Git depuis plus de 10 ans. Au début, je détestais Git car je ne comprenais pas comment il fonctionne et j'avais la sensation que l'outil compliquait mon quotidien de développeur. J'avais également peur de perdre mes modifications lors de commandes comme le merge ou le rebase.

J'ai alors pris la décision de me former plus sérieusement à cet outil. Lorsque j'ai découvert son fonctionnement et son potentiel, cela a totalement changé ma vision sur l'outil et surtout, cela a clairement amélioré ma productivité.

HexoTech



HexoTech est une entreprise d'**expertise en développement logiciel**, qui offre de la conception, du conseil et du développement sur mesure pour ses clients. HexoTech est notamment engagé auprès de projets innovants via son agrément CII

HexoTech fournit également des **formations continues** éligibles aux OPCO via son agrément QUALIOP1.



Architecture système et logicielle



Développement logiciel



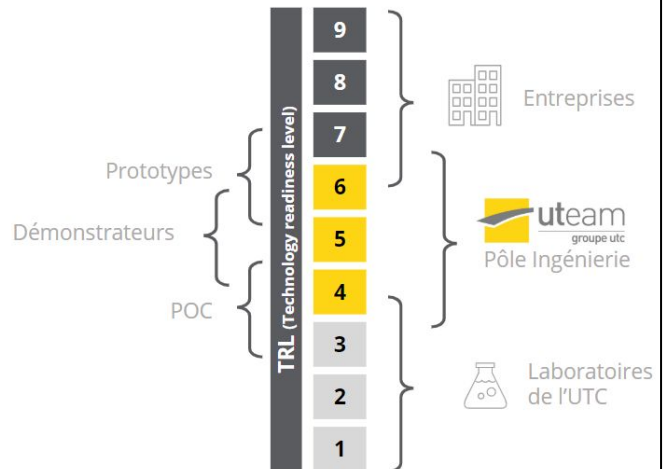
Usine logicielle (CI/CD)

Chez HexoTech, nous faisons de l'expertise en développement logiciel (Architecture, choix technique, Conception, DevOps, ...) mais aussi du développement logiciel.

Nous faisons également de la formation, et c'est justement pour cela que nous sommes ici :)

Pôle ingénierie UTeam

- ✓ Accompagner les entreprises dans leurs projets d'innovation technologique : prestations court terme de développement de produits, prototypes, banc d'essai...
- ✓ Les PME/ETI qui n'ont pas de R&D cherchent à se développer en s'appuyant sur un pôle de compétences pluridisciplinaires.
- ✓ Combler l'écart entre l'offre de recherche des laboratoires et les besoins des entreprises en se positionnant sur les TRL intermédiaires.



Nous travaillons également en collaboration avec l'UTC via sa filiale de valorisation: UTeam

Nous faisons parti du pôle ingénierie de UTeam. Nous faisons alors l'interface entre les laboratoires de l'UTC et les entreprises. Nous réalisons des prototypes, démonstrateurs et POC pour les entreprises en nous basant sur les connaissances et les recherches de l'UTC.

Philosophie de nos formations



Apprendre en pratiquant :

Comme le dit Richard Branson « La meilleure façon d'apprendre c'est de faire ! » Eh oui, il n'y a pas de secret, plus vous pratiquez, plus vous progressez ! Vous trouverez alors environ 20% de théorie pour 80% de pratique dans nos formations.



Apprendre avec des experts :

Les formations sont développées et présentées par des experts dans le domaine enseigné. Nous ne nous contentons pas seulement d'enseigner, mais nous appliquons nos connaissances lors des collaborations avec nos clients et partenaires.



Apprendre en s'amusant :

Vous l'avez sûrement remarqué, mais nous apprenons bien mieux quand les choses sont présentées de manière ludique. Vous trouverez alors dans nos formations des fils rouges sous forme de projets concrets que nous ferons évoluer tout au long des cours. Cela permettra non seulement d'illustrer les notions vues ensemble mais aussi de vous exercer.

La formation se base sur 3 piliers

Vos attentes et votre niveau ?



- Quel est votre niveau actuel en Git ?
- Pourquoi suivre cette formation ?
- Quelles sont vos attentes ?

Faisons un tour de table

Plan de la formation **jour 1**

1 - Introduction

- Présentation de la formation et du formateur
- Pourquoi et comment versionner son code
- Présentation Git

2 - Installation et configuration de Git

- Télécharger et installer Git sous Windows
- Installation de Git sous Linux
- Les invites de commandes
- Configuration de Git
- Trouver de l'aide

3 - Les commits

- Comprendre le processus de commit
- Initialiser un dépôt Git en local
- Indexer ses modifications
- Faire son premier commit
- Voir les modifications en cours (diff)
- Corriger son dernier commit
- Faire un commit sélectif
- Filtrer les fichiers gérés par Git

4 - Voyager dans l'historique des commits

- Comprendre l'archivage des commits
- Voir l'historique
- Voir les détails d'un commit
- Naviguer dans l'historique
- Utiliser les tags
- Annuler un commit

5 - Les interfaces graphiques pour Git

- Différences entre ligne de commande et IHM
- Git Gui
- Gitk
- Autres interfaces

Le jour 1 se concentre sur les bases de Git en locale

Plan de la formation **jour 2**

6- Les branches

- Comprendre le système de branche
- Créer une branche
- Récupérer une branche
- Copier un commit
- Faire un merge entre deux branches
- Faire un rebase entre deux branches
- Merge VS Rebase
- Branches locales et distantes

7 - Réécrire l'histoire

- Modifier son dernier commit
- Corriger des erreurs
- Modifier des commits plus anciens
- Diviser ou rassembler des commits

8 et 9 - Dépôt distant

- Comprendre le modèle distribué
- Cloner un dépôt
- Gérer les remotes
- Pousser ses modifications sur le serveur
- Récupérer les modifications depuis le serveur
- github et gitlab

Le jour 2 se concentre sur le travail en équipe et via un serveur

Plan de la formation **jour 3**

10- Travailler en équipe

- Les organisations d'équipes autour de Git
- Gérer les conflits lors des pull et push
- Utiliser des branches de travail
- Forcer les historiques

11 - Debugger son code

- Maîtriser les modifications
- Retrouver le commit qui pose problème
- Utiliser Gerrit

12 - Personnaliser Git

- Configurer git
- les attributs
- Les hooks

13 - Gestion projets multi dépôt

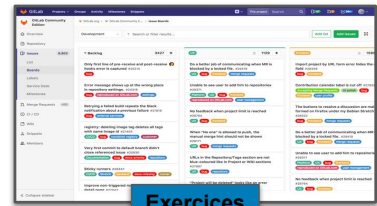
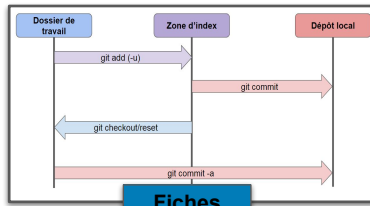
- Présentation des enjeux et difficultés
- Sans gestion particulière
- Via des scripts
- Via les submodules

14 - git LFS

- mise en place
- configuration
- utilisation

Le jour 3 se concentre sur des notions avancées et sur des plugins et outils annexes

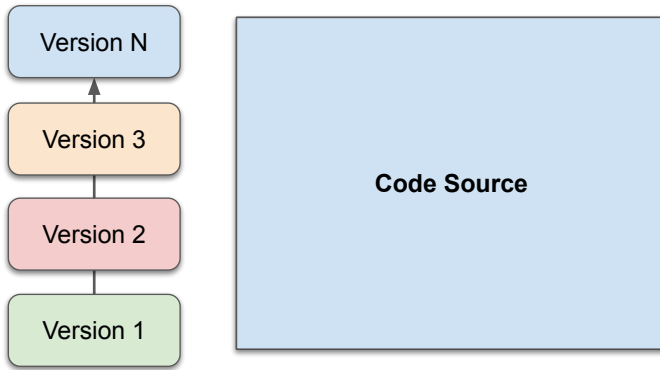
Vos supports !



<https://hexotech.fr/formation-continue/git-github/>

Récupérez vos ressources

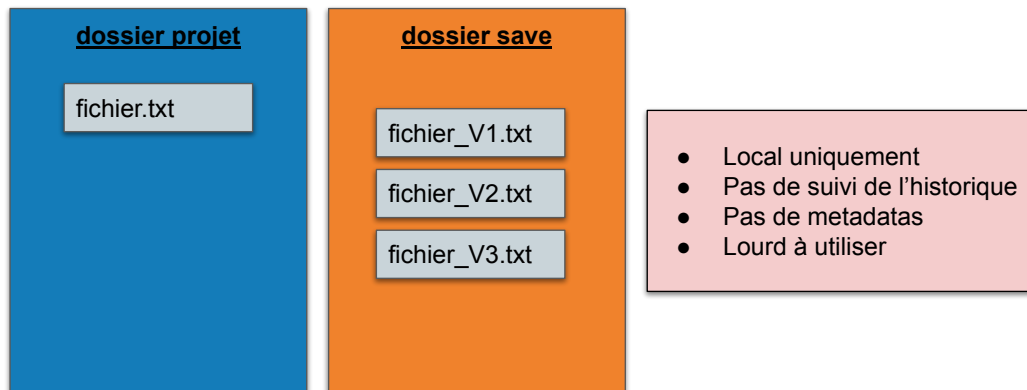
La gestion de version c'est quoi?



- Sauvegarder l'évolution du code
- Pouvoir revenir en arrière
- Comparer des versions
- Comprendre l'évolution du code
- Travailler à plusieurs sur un projet
- En réseau ou en local
- Pouvoir se tromper

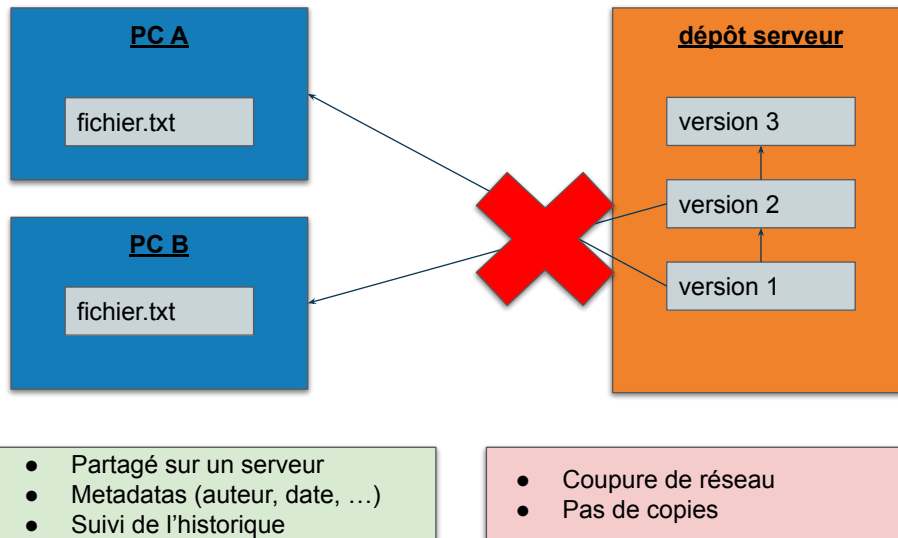
Qu'est-ce que la gestion de version et pourquoi devriez-vous vous en soucier ? Un gestionnaire de version est un système qui enregistre l'évolution d'un fichier ou d'un ensemble de fichiers au cours du temps de manière à ce qu'on puisse rappeler une version antérieure d'un fichier à tout moment. La gestion de version est un peu comme votre album photo (l'historique ou dépôt) qui regroupe des instantanés de votre vie (commit, versions)

Un monde sans gestion de version



Sans gestion de version, le moyen le plus utilisé, est de créer des copies de nos fichiers et de modifier leur nom pour y ajouter une notion de version. Ce système est très lourd, peu fiable et ne permet pas un suivi de l'historique correcte.

Gestion de version centralisée [cvs, svn, p4v]

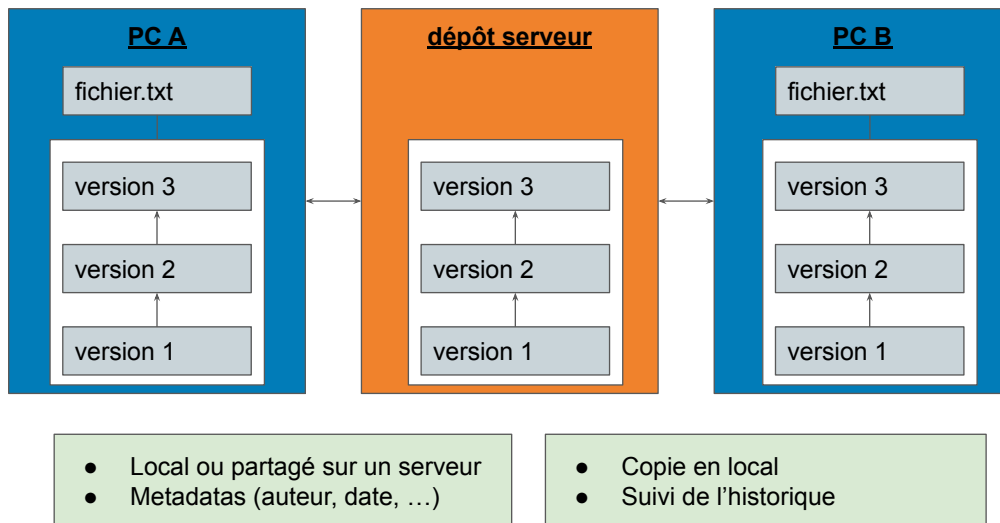


L'un des besoins des développeurs est de pouvoir collaborer sur un projet. Pour y répondre, des systèmes de gestion de version centralisés (CVCS en anglais pour Centralized Version Control Systems) furent développés. Ces systèmes tels que CVS, Subversion, et Perforce, mettent en place un serveur central qui contient tous les fichiers sous gestion de version, et des clients qui peuvent extraire les fichiers de ce dépôt central. Pendant de nombreuses années, cela a été le standard pour la gestion de version.

Cette gestion offre de nombreux avantages par rapport à la gestion de version local. Notamment le fait de pouvoir facilement partager son travail avec le reste de son équipe.

Mais cette gestion comporte également une part d'ombre. Premièrement si un problème réseau survient, les développeurs ne peuvent plus partager leur travail et ne peuvent plus réaliser des sauvegarde ce celui-ci. De plus, les développeurs se retrouvent bloqué sur une version donné de l'historique sans pouvoir en bouger. Pour finir, en cas de corruption du disque du serveur, les données seront perdus si aucun système de backup n'est en place.

Gestion de version distribuée [Git, Mercurial]



C'est à ce moment que les systèmes de gestion de version distribués entrent en jeu (DVCS en anglais pour Distributed Version Control Systems). Dans un DVCS (tel que Git, Mercurial, Bazaar ou Darcs), les clients n'extraient plus seulement la dernière version d'un fichier, mais ils dupliquent complètement le dépôt. Ainsi, 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.

De plus, 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 qui ne sont pas réalisables avec les systèmes centralisés, tels que les modèles hiérarchiques.

Et Git dans tout ça?

version 2.x.x



Linus Torvalds



Linux



1991

Patches

Création de Linux, utilisation de patch pour faire évoluer le code



2002

BitKeeper

Passage sous BitKeeper pour la gestion de version du code source



2005

Git

Création de Git par Linus Torvald pour remplacer BitKeeper pour la gestion du code de Linux



today

12 000 000 d'utilisateurs

Git a plus de 12 Millions d'utilisateurs dans le monde. Il est présent dans les grandes entreprises et projets open source.

Git a été créé par Linus Torvald, le papa du noyau Linux. Il a justement été conçu pour faire la gestion de version de ce projet d'envergure que représente Linux.

Aujourd'hui, Git est utilisé par plus de 12 millions de personnes dans le monde entier. On le retrouve bien sûr dans le monde de l'open source mais aussi de plus en plus dans les entreprises privées.

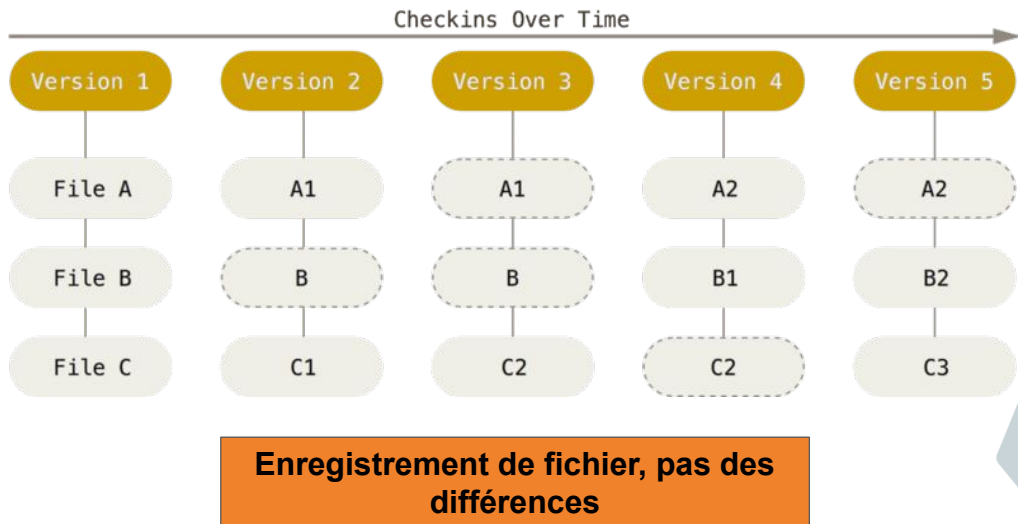
On peut même noter le rachat de GitHub (plateforme d'hébergement de dépôt Git) pour plus de 7.5 milliards de dollars en 2018.

Git est aujourd'hui un incontournable de la gestion de version.

Et pour Git, quel gestion de version? et bien oui Git est un outil qui a été généré à partir de code source. Et bien c'est Git qui est utilisé pour faire la gestion de version de Git!

Info: Toutes les versions 2.x.x sont compatibles entre elles

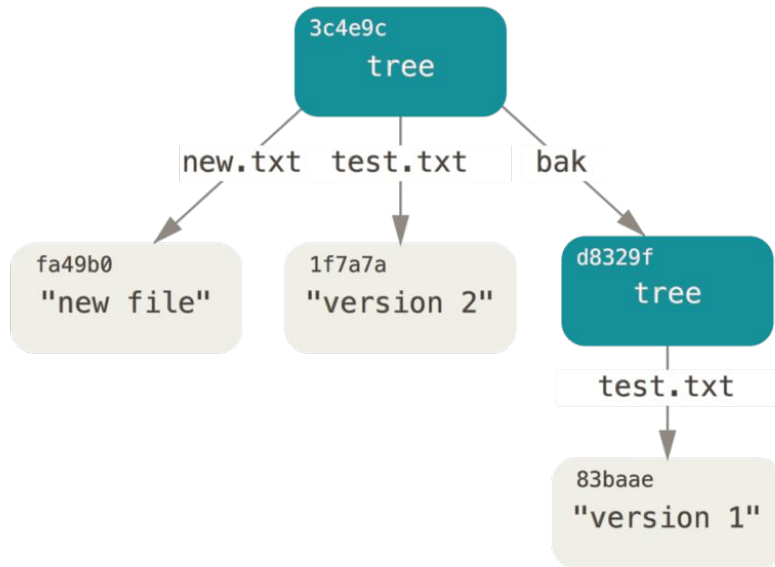
Git et la gestion de version



La différence majeure entre Git et les autres VCS (Subversion et autres) 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.

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.

Les objets Git



tout objet git possède un identifiant unique (sha1)
on retrouve deux grand types d'objets de base:

- tree: dossiers
- blob: fichiers

QUIZ

1 - Utiliser un logiciel de gestion de version me permet :

- A. De travailler plus facilement en équipe sur un même code source
- B. D'avoir un historique des modifications de mon code source
- C. Compiler mon programme
- D. De sauvegarder mon code source

2 - Git enregistre en base :

- A. Uniquement les différences
- B. L'intégralité des fichiers même sans modifications
- C. Les instantanés des fichiers après modification

3 - Git est un système :

- A. Centralisé
- B. Déconnecté
- C. Distribué
- D. Cloné

Petit quiz qui est plus là pour faire le bilan que vous tester