 Département d’informatique

03/08/2016

Présentation Git



Dyden Ung, Adrian Pinzaru, François Lefebvre

Table des matières

[Problématique 0](#_Toc445213311)

[Qu’est-ce qu’un logiciel de gestion de versions? 0](#_Toc445213312)

[Logiciels centralisés et distribués 0](#_Toc445213313)

[Git est les autres logiciels de gestion de versions 0](#_Toc445213314)

[Quelles sont les particularités de Git ? 0](#_Toc445213315)

# Problématique

Si vous avez déjà travaillé sur un projet informatique, que ce soit un petit projet personnel ou un plus gros projet professionnel, vous avez certainement déjà rencontré un de ces problèmes :

* « Qui a modifié le fichier X, il marchait bien avant et maintenant il provoque des bugs ! » ;
* « Robert, tu peux m’aider en travaillant sur le fichier X pendant que je travaille sur le fichier Y ? Attention à ne pas toucher au fichier Y car si on travaille dessus en même temps je risque d’écraser tes modifications ! » ;
* « Qui a ajouté cette ligne de code dans ce fichier ? Elle ne sert à rien ! » ;
* « À quoi servent ces nouveaux fichiers et qui les a ajoutés au code du projet ? » ;
* « Quelles modifications avions-nous faites pour résoudre le bug de la page qui se ferme toute seule ? »

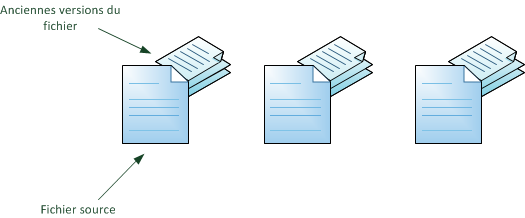
Si ces problèmes-là vous parlent, vous auriez dû utiliser un **logiciel de gestion de versions**. Ce type de logiciel est devenu indispensable lorsqu’on travaille à plusieurs sur un même projet et donc sur le même code source. Même si vous travaillez seuls, vous aurez intérêt à commencer à en utiliser un rapidement car il vous offrira de nombreux avantages, comme la conservation d’un historique de chaque modification des fichiers par exemple.

Il existe de nombreux logiciels de gestion de versions, comme SVN (Subversion), Mercurial et Git. Dans ce tutoriel, je vous présenterai **Git** (prononcez « guite ») qui est un des plus puissants logiciels de ce genre. Nous l’utilisons notamment pour gérer le code source du Site du Zéro !

## Qu’est-ce qu’un logiciel de gestion de versions?

Les logiciels de gestion de versions sont utilisés principalement par les développeurs ; ce sont donc bel et bien des outils pour *geeks*. En effet, ils sont quasi exclusivement utilisés pour gérer des codes sources, car ils sont capables de suivre l’évolution d’un fichier texte ligne de code par ligne de code.    
Ces logiciels sont fortement conseillés pour gérer un projet informatique.

Ces outils suivent l’évolution de vos fichiers source et gardent les anciennes versions de chacun d’eux.



S’ils s’arrêtaient à cela, ce ne seraient que de vulgaires outils de *backup* (sauvegarde). Cependant, ils proposent de nombreuses fonctionnalités qui vont vraiment vous être utiles tout au long de l’évolution de votre projet informatique :

* ils retiennent qui a effectué chaque modification de chaque fichier et *pourquoi*. Ils sont par conséquent capables de dire qui a écrit chaque ligne de code de chaque fichier et dans quel but ;
* si deux personnes travaillent simultanément sur un même fichier, ils sont capables d’assembler (de fusionner) leurs modifications et d’éviter que le travail d’une de ces personnes ne soit écrasé.

Ces logiciels ont donc par conséquent deux utilités principales :

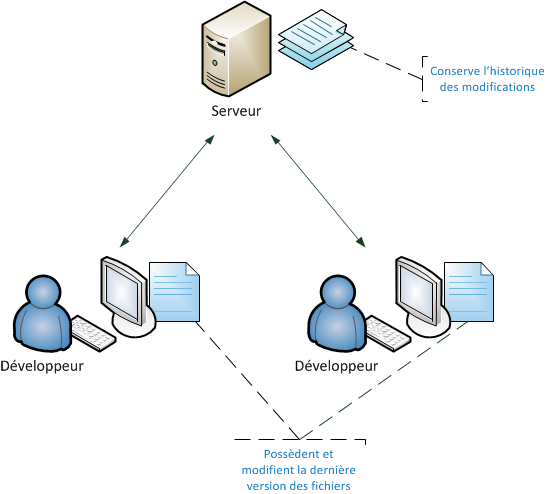
* **suivre l’évolution d’un code source**, pour retenir les modifications effectuées sur chaque fichier et être ainsi capable de *revenir en arrière* en cas de problème ;
* **travailler à plusieurs**, sans risquer de se marcher sur les pieds. Si deux personnes modifient un même fichier en même temps, leurs modifications doivent pouvoir être fusionnées sans perte d’information.

## Logiciels centralisés et distribués

Il existe deux types principaux de logiciels de gestion de versions.

* **Les logiciels centralisés** : un serveur conserve les anciennes versions des fichiers et les développeurs s’y connectent pour prendre connaissance des fichiers qui ont été modifiés par d’autres personnes et pour y envoyer leurs modifications.
* **Les logiciels distribués** : il n’y a pas de serveur, chacun possède l’historique de l’évolution de chacun des fichiers. Les développeurs se transmettent directement entre eux les modifications, à la façon dupeer-to-peer.

Voici, schématiquement, comment fonctionne un logiciel de gestion de versions centralisé :



Un logiciel de gestion de versions centralisé.  
Le serveur retient les anciennes versions des fichiers et communique les changements aux développeurs.

De même, voici le fonctionnement d’un logiciel de gestion de versions distribué :



Un logiciel de gestion de versions distribué.  
Il n’y a pas de serveur. Les développeurs conservent l’historique des modifications et se transmettent les nouveautés.

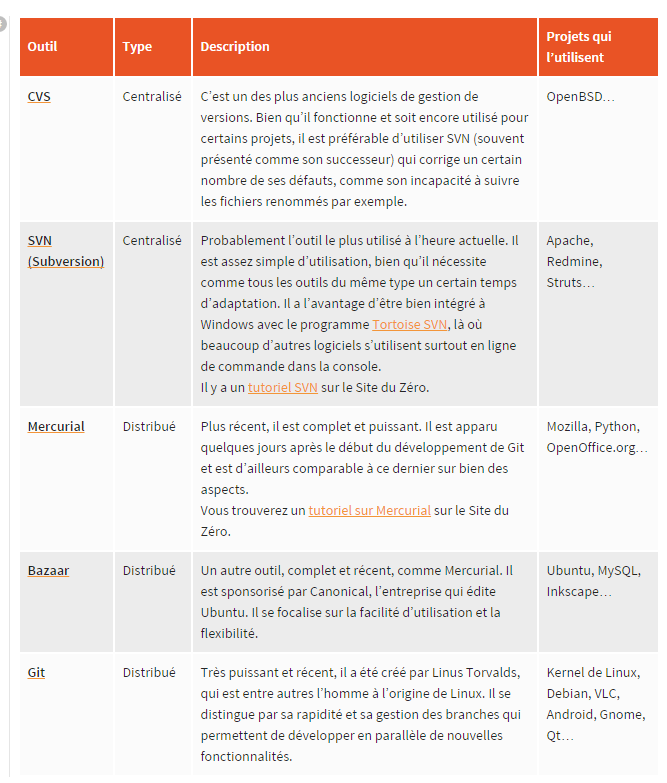
Dans la pratique, les logiciels distribués sont rarement utilisés comme sur le schéma précédent. Même lorsque les logiciels sont capables de fonctionner en mode distribué, on utilise très souvent un serveur qui sert de point de rencontre entre les développeurs. Le serveur connaît l’historique des modifications et permet l’échange d’informations entre les développeurs, qui eux possèdent également l’historique des modifications.



Un logiciel de gestion de versions distribué avec un serveur.  
Le serveur sert de point de rencontre entre les développeurs et possède lui aussi l’historique des versions.

C’est dans ce dernier mode que nous allons fonctionner avec Git.   
Il a l’avantage d’être à la fois flexible et pratique. Pas besoin de faire de sauvegarde du serveur étant donné que tout le monde possède l’historique des fichiers, et le serveur simplifie la transmission des modifications.

## Git est les autres logiciels de gestion de versions



## Quelles sont les particularités de Git ?

Je n’entrerai pas dans les détails de la comparaison de Git avec les autres outils concurrents comme SVN et Mercurial. Retenez simplement que :

* CVS est le plus ancien et il est recommandé de ne plus l’utiliser car il est le moins puissant et n’est plus très bien mis à jour ;
* SVN est le plus connu et le plus utilisé à l’heure actuelle, mais de nombreux projets commencent à passer à des outils plus récents ;
* Mercurial, Bazaar et Git se valent globalement, ils sont récents et puissants, chacun a des avantages et des défauts. Ils sont tous distribués, donc chaque développeur possède l’historique des modifications et ils permettent en théorie de se passer de serveur (bien qu’on utilise toujours un serveur pour des raisons pratiques).

Concernant les avantages de Git sur les autres, certains ont fait des [listes comparatives](http://www.smashingmagazine.com/2008/09/18/the-top-7-open-source-version-control-systems/) intéressantes (bien que toujours criticables).

On retiendra surtout que Git :

* est très rapide ;
* sait travailler par branches (versions parallèles d’un même projet) de façon très flexible ;
* est assez complexe, il faut un certain temps d’adaptation pour bien le comprendre et le manipuler, mais c’est également valable pour les autres outils ;
* est à l’origine prévu pour Linux. Il existe des versions pour Windows mais pas vraiment d’interface graphique simplifiée. Il est donc à réserver aux développeurs ayant un minimum d’expérience et… travaillant de préférence sous Linux.

Une des particularités de Git, c’est l’existence de sites web collaboratifs basés sur Git comme [GitHub](http://github.com/). GitHub, par exemple, est très connu et utilisé par de nombreux projets : jQuery, Symfony, Ruby on Rails…  
C’est une sorte de réseau social pour développeurs : vous pouvez regarder tous les projets évoluer et décider de participer à l’un d’entre eux si cela vous intéresse. Vous pouvez aussi y créer votre propre projet : c’est gratuit pour les projets open source et il existe une version payante pour ceux qui l’utilisent pour des projets propriétaires.

GitHub fournit le serveur où les développeurs qui utilisent Git se rencontrent. C’est un excellent moyen de participer à des projets open source et de publier votre projet !

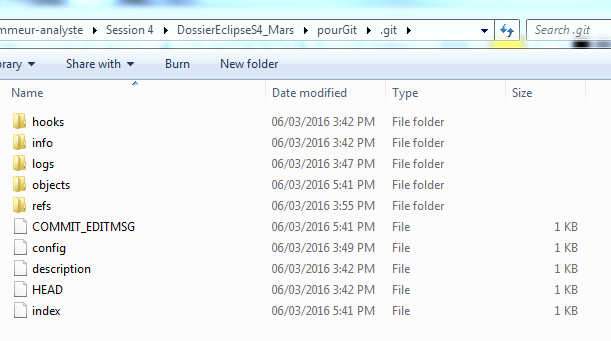
# Le cas de Git

--- Concepts et vocabulaires

- Architecture de git (dyden)

Lorsqu’un projet a été initialisé pour être utiliser avec Git, un répertoire caché nommé .git est ajouté au dossier dans lequel vous travaillez. Si le dossier n’est pas visible, vérifiez les paramètres de dossiers pour s’assurer que l’option d’afficher les dossiers cachés n’est pas décochée. Il est important de différencier le *repository* (dossier .git) et le répertoire de travail comme le dossier d’un projet eclipse qui est communément appelé un *working directory.*

Voici une image représentant la structure de base du dossier .git :



Dans l’image ci-dessus, on voit que le dossier .git a été créé comme un sous-répertoire du dossier de projet nommé « pourGit » (voir dans la barre d’adresses).

Il y a différents types de fichiers qui composent le *repository* et ils se répartissent ainsi :

* Configuration : Des fichiers comme *config* et *description* aident pour la configuration de son répertoire local git.
* Hooks: le dossier *hooks* contient des fichiers scripts qui peuvent faire exécuter une instruction durant des évènements durant le projet. Par exemple, une instruction a exécuté avant un commit.
* Staging Area: le fichier *index* permet d’offrir l’espace tampon (staging area) entre le commit et le fichier modifié. Cette espace offre la possibilité de sélectionner les fichiers qu’on désire, de les placer sur le « staging area » et de faire le commit par la suite, donnant ainsi une flexibilité dans le choix de ce que le commit enregistrera.
* *Object Database* : le dossier objets contient toute une collection d’enregistrement crée par l’opération « commit ».
* *References :* le dossier refs stocke tous les pointeurs qui permettent de référencer les éléments qui composent la structure de git.

Dans le dossier .git, on peut aussi voir le fichier *HEAD.* Ce fichier contient la référence (head) qui pointe vers la position actuelle du projet. La position est en fait un commit qui a été enregistré dans la base de donnée du répertoire git. C’est ainsi qu’en pointant le *head*, vers un autre commit, on peut se repositionner à un moment antérieur du projet.

- Concepts de développement (adrian)

- branches

- commit

- checkout

- Concepts de synchronisation (francois)

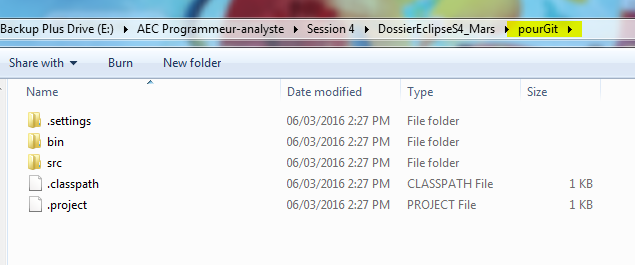
- merge, rebase

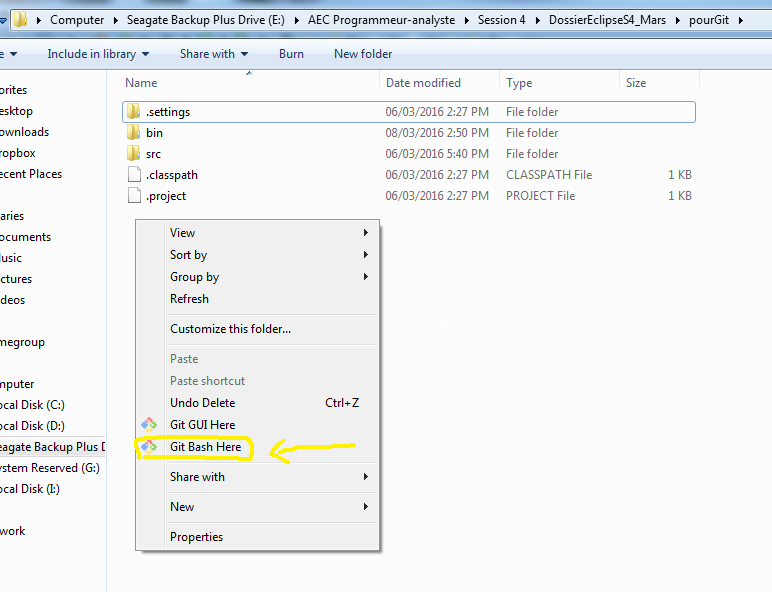
- conficts

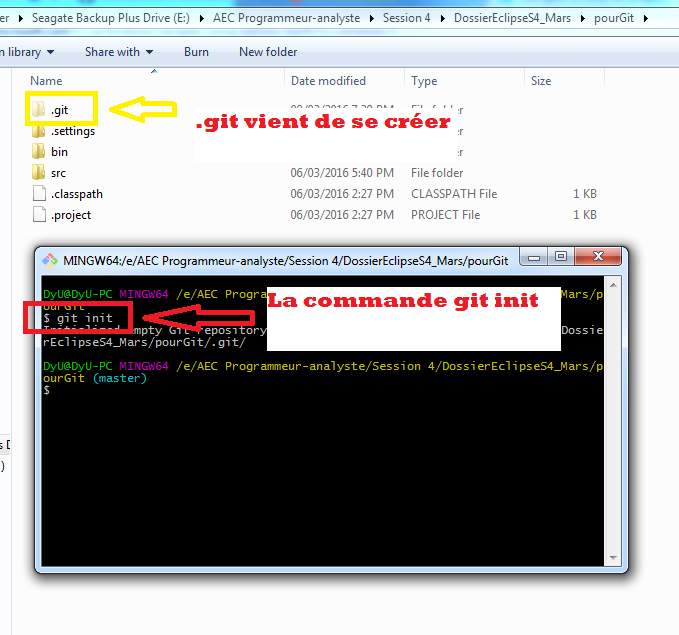
--- En ligne de commande (dyden)

Pour créer un projet git, il est possible d’ouvrir la console et d’utiliser les lignes de commande. Voici la procédure simple pour initialiser git dans un projet local :

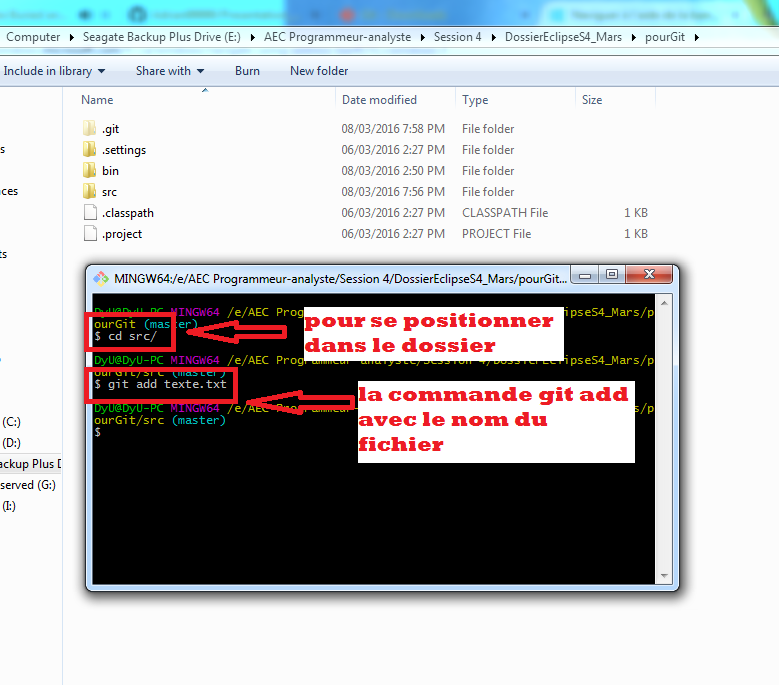
* 1. Si vous n’avez pas encore installé Git, allez à cette adresse <https://git-scm.com/downloads> pour effectuer l’installation.
  2. Si tout est installé, dirigez-vous maintenant à la racine du dossier de votre projet.



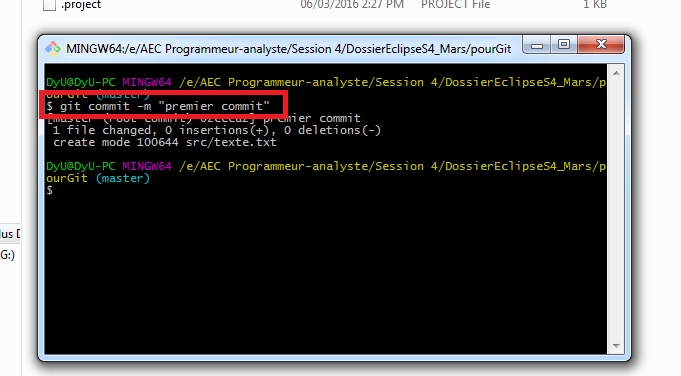
* 1. Clic-droit dans le répertoire et choisir l’option « git bash here ».
  2. Lorsque la console est ouverte, entrez la commande *git init* . Cette commande crée votre dossier .git qui sera placé dans le répertoire courant.



* 1. Bien que le dossier .git ait été créé, le projet n’est pas tout à fait initialisé. Aucun suivi sur les fichiers n’a été encore fait. En effet, pour que Git puisse faire un commit sur les fichiers, ceux-ci doivent être marqués comme « tracked ». Pour s’y prendre, créez un nouveau fichier dans le projet et entrez la commande *git add <nom du fichier>*.



* 1. Finalement, il ne vous reste plus qu’à faire la commande *git commit –m « mettre un message significatif pour ce commit. »* pour terminer l’initialisation. Le commit étant ainsi créé, le pointeur *head* se positionne sur ce commit, formant ainsi le point de départ de votre branche principale.



* 1. Voilà, l’initialisation d’un projet git local est terminée.

-- Demo Eclipse

- Créer un projet Git (dyden)

- Installation de EGIT sur Eclipse

- Share a project

- Bonnes pratiques pour le dossier .git

- commit, new branch, commit, history, checkout

- Synchronsiser avec Github (francois)

- créer projet sur github

- foncitonnement et tarifs

- push branch

- import project from github (adrian)

- Commit projet 2 sur master, Commit projet 1 sur master, merge, conflit, résoudre conflits

--- Discussions et monitoring sur github (dyden)

# Conclusion (adrian)

- Retour sur les point avec questions (francois, dyden, adrian)

- Prendre les questions du public

