Usages et intérêts des sytèmes de gestion de versions (ou VCS) pour un usage professionnel et personnel

Florian Legendre

Université de Poitiers

Année 2020 - 2021

Table of Contents

1. La notion de version et ses implications
   - Première étude de cas : notions de version/log/reset/diff
   - Seconde étude de cas : notions de branches/merge
   - Dernière étude de cas : notions de conflits/pull/push
2. Les VCS et Git
   - Les services offerts par les VCS
   - Les VCS comparés aux outils collaboratifs en ligne
   - Les services offerts par Git
   - Les services offerts par GitHub
3. Le plan de la formation

Université de Poitiers 1431

Table of Contents

1. La notion de version et ses implications
   - Première étude de cas : notions de version/log/reset/diff
   - Seconde étude de cas : notions de branches/merge
   - Dernière étude de cas : notions de conflits/pull/push
2. Les VCS et Git
   - Les services offerts par les VCS
   - Les VCS comparés aux outils collaboratifs en ligne
   - Les services offerts par Git
   - Les services offerts par GitHub
3. Le plan de la formation

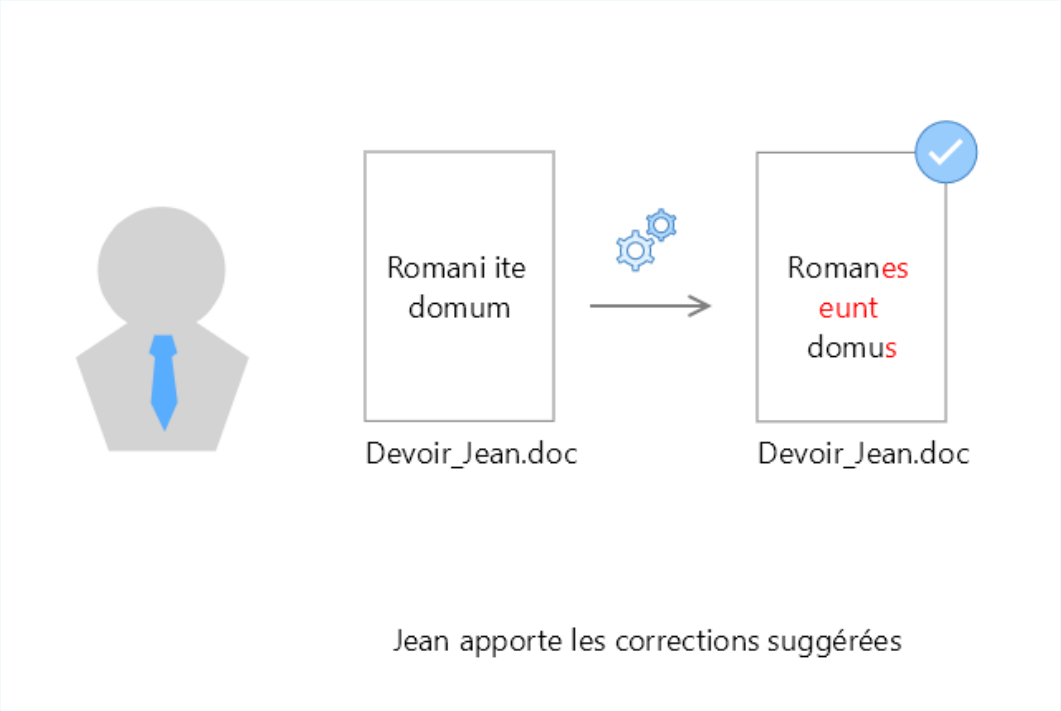
Université de Poitiers

Premier cas : Jean étudiant en lettres

Jean écrit une première version de son devoir qu'il envoie à son enseignante par mail :

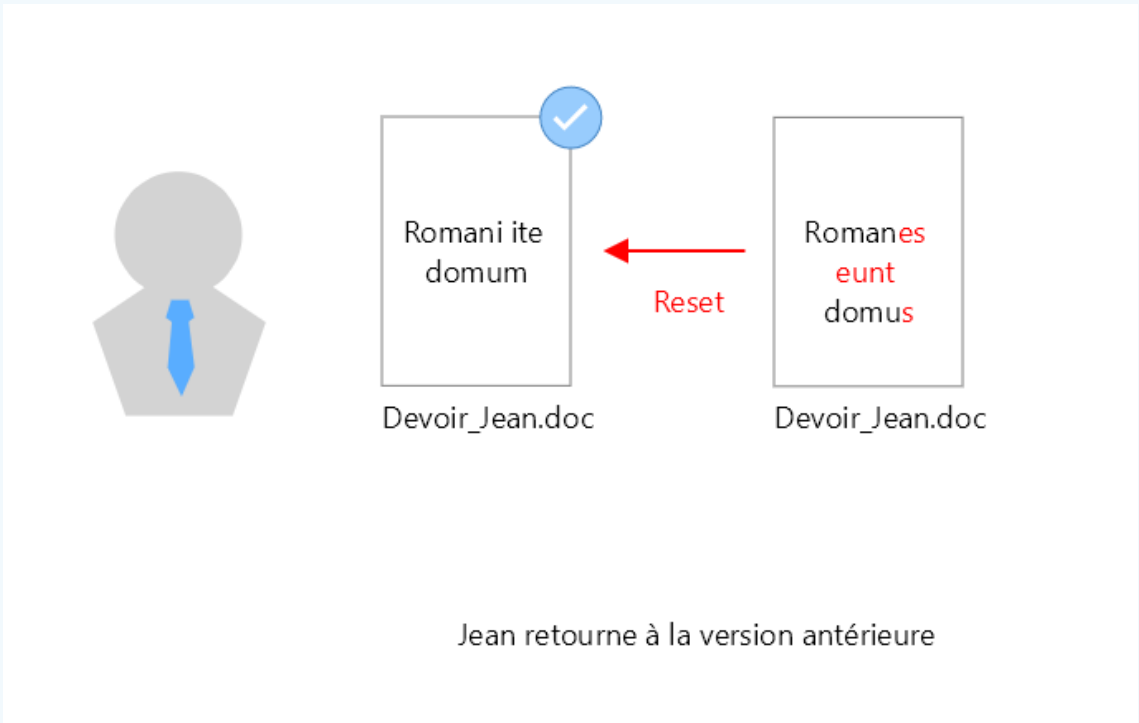
Premier cas : Jean étudiant en lettres

L'enseignante de Jean suggère dans sa réponse des corrections dont Jean tient compte. Il écrit donc dans le document les corrections :



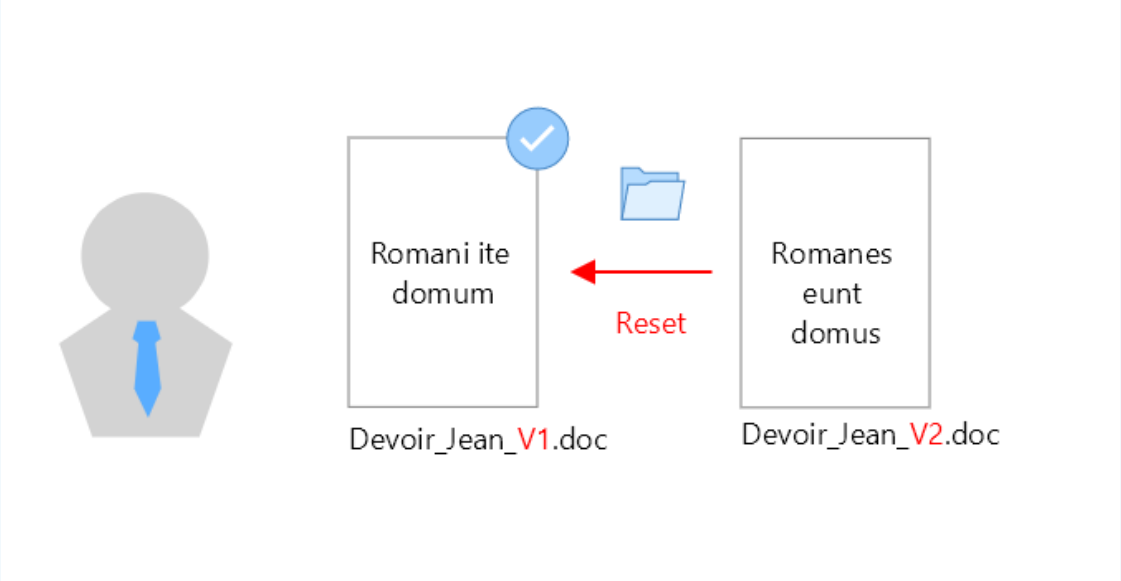
Premier cas : Jean étudiant en lettres

Finalement, l'enseignante de Jean est allée un peu trop vite et dit à Jean qu'elle s'était trompée dans son précédent mail... Jean doit revenir à la première version en se souvenant de toutes les modifications qu'il a pu faire :



Premier cas : Jean étudiant en lettres

Si Jean avait été un étudiant prévoyant il aurait nommé ses fichiers monDevoir_V1 pour désigner la première version qu'il envoie à son enseignante et monDevoir_V2 pour désigner la version tenant compte des corrections de son enseignante :



Résumé du premier cas

Jean envoie sa version à son enseignante :

Jean écrit une nouvelle version tenant compte des corrections de son enseignante :

Jean revient à la première version sous les conseils de son enseignante :

Conclusion du premier cas

Quelques notions :

- **Une version :** un état d'un document à un point de sauvegarde (ou "commit") donné
- **Historique des versions :** fichier contenant toutes les versions passées et présente d'un document
- **Comparaison des versions :** Jean ne peut pas le faire facilement (à moins de lire les différentes versions) mais nous verrons que les VCS permettent de comparer les versions de l'historique entre elles en fournissant de nombreuses informations sur celles-ci (zones qui ont été supprimées ou ajoutées, etc.)

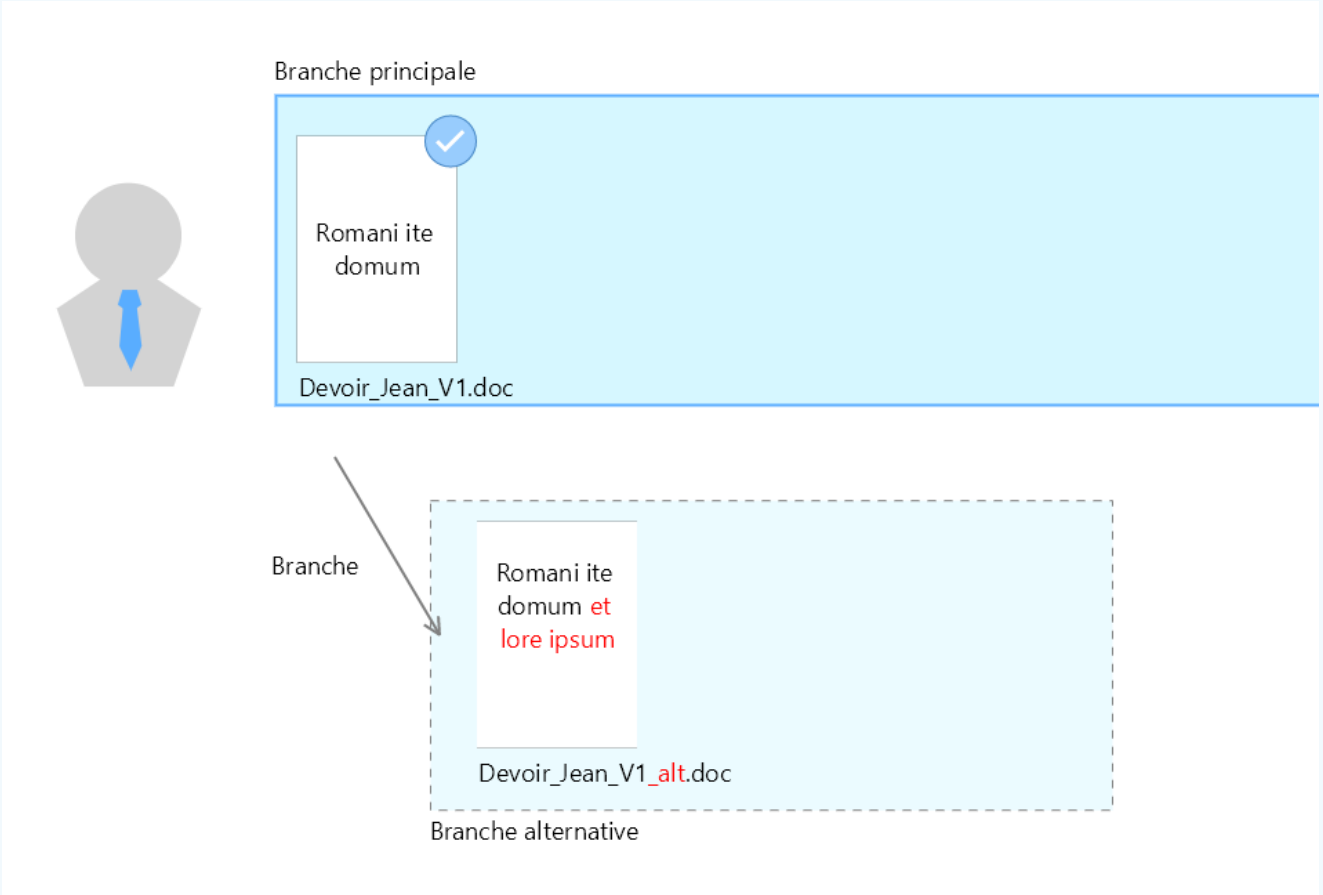
Université de Poitiers 1431

Second cas : Jean bidouille son commentaire de texte

On reprend depuis le début, Jean rédige un commentaire de texte, il envoie une première version à son enseignante qui la valide :

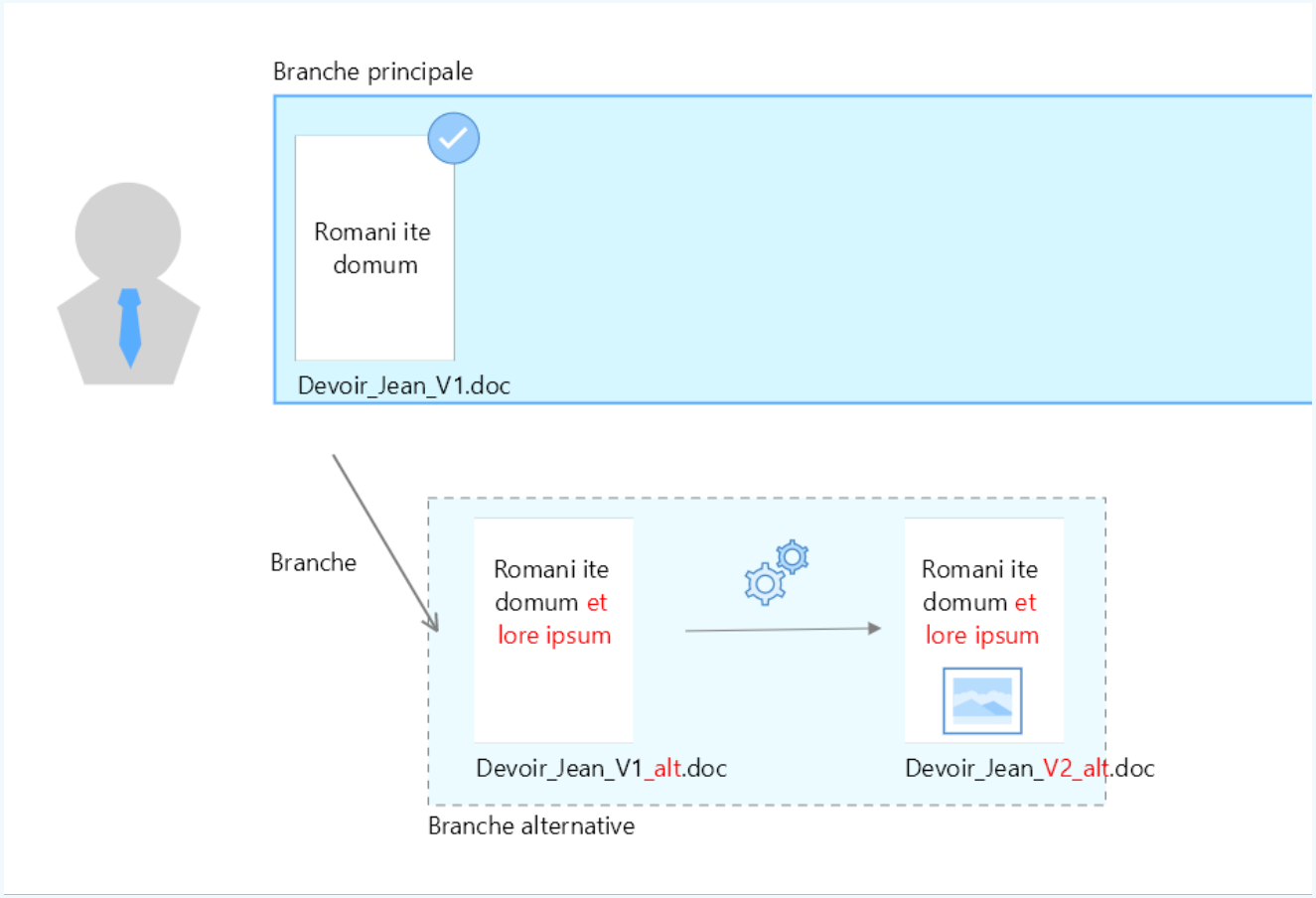
Second cas : Jean bidouille son commentaire de texte

Bien que sa première version soit validée Jean aimerait la perfectionner... Il crée alors une "branche alternative" à son devoir :



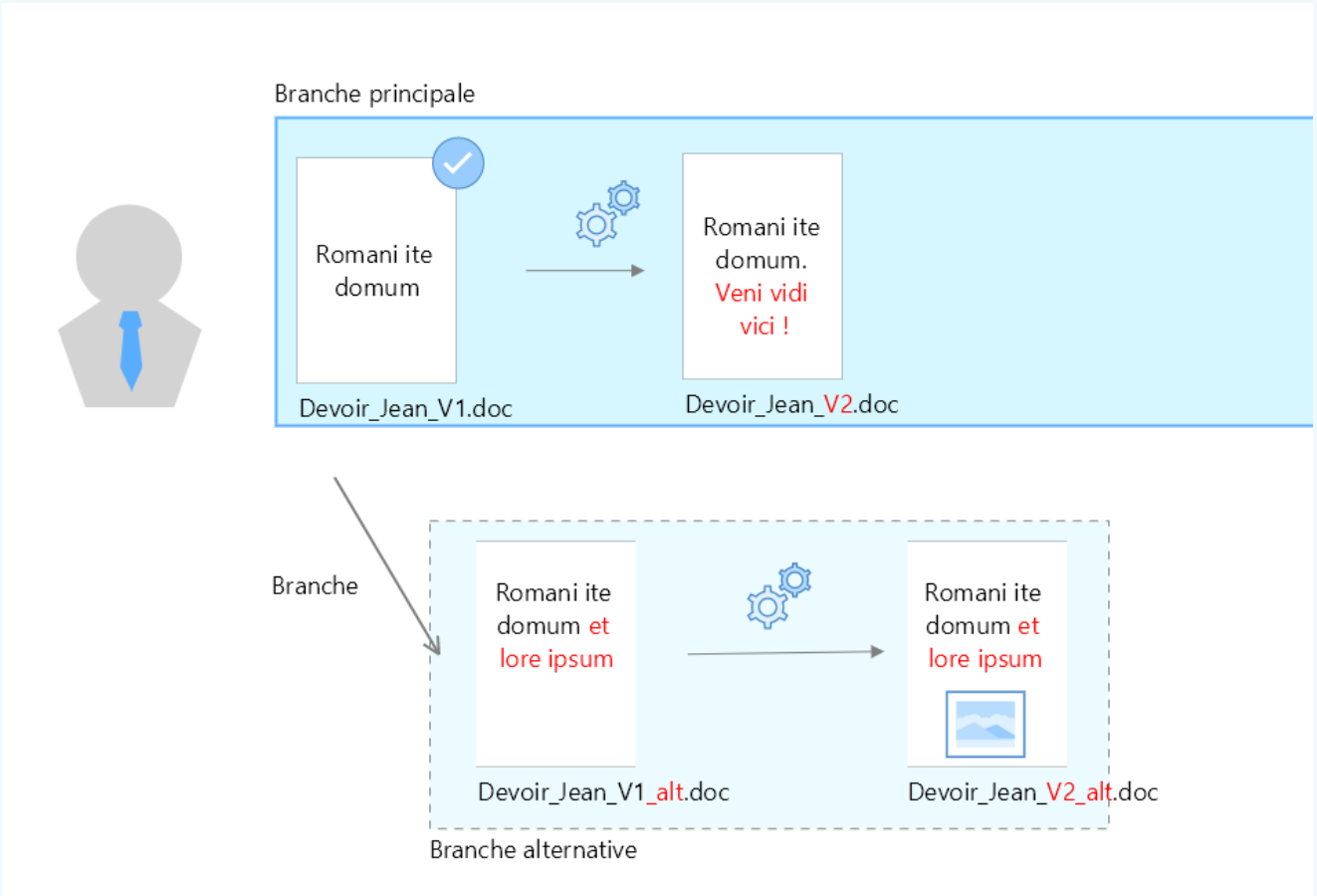
Second cas : Jean bidouille son commentaire de texte

Juste avant d'envoyer sa version alternative, Jean se rend compte qu'il peut encore améliorer cette version en ajoutant une image de Jules César. Sauf que si l'enseignante lui dit que l'image est de trop il aimerait pouvoir revenir en arrière :



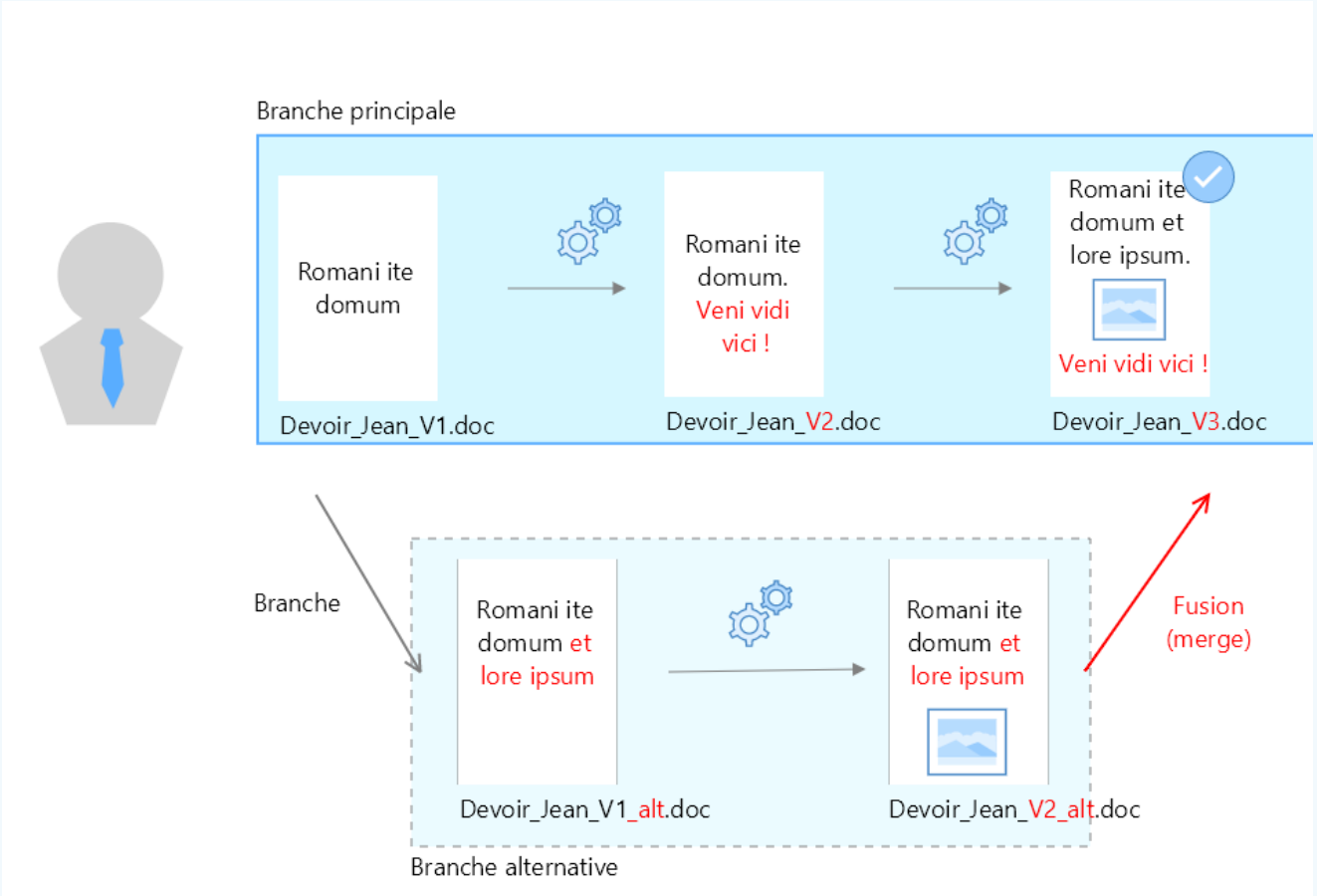
Second cas : Jean bidouille son commentaire de texte

Jean a envoyé sa version alternative... En attendant une réponse de son enseignante il continue d'écrire son devoir dans la version qui a été validée :



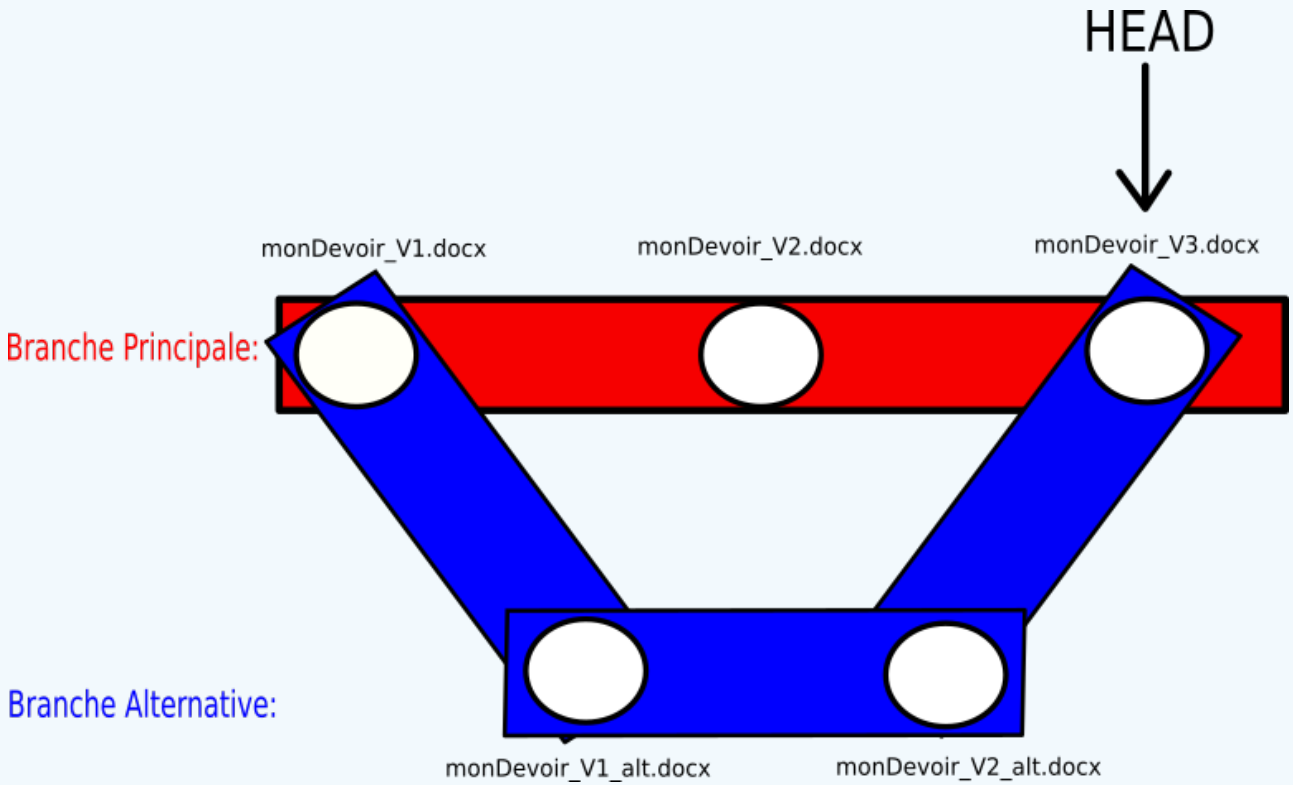
Second cas : Jean bidouille son commentaire de texte

Jean a reçu une réponse, l'enseignante adore la nouvelle version !
Jean fusionne alors ses changements avec la branche en cours :



Résumé du second cas

Si on schématise le flux des versions ("workflow") du travail de Jean on obtient le schéma suivant :



Remarque avant de conclure

Vous remarquerez sur le schéma précédent un pointeur appelé "HEAD". Ce pointeur indique la version courante dans laquelle se trouve le projet de Jean.

Je l'ai représenté ici pour illustrer où en était Jean à la fin de notre exemple. Mais nous verrons que Git offre des fonctionnalités pour manipuler ce pointeur (comme la commande "git checkout"...)

Conclusion du second cas

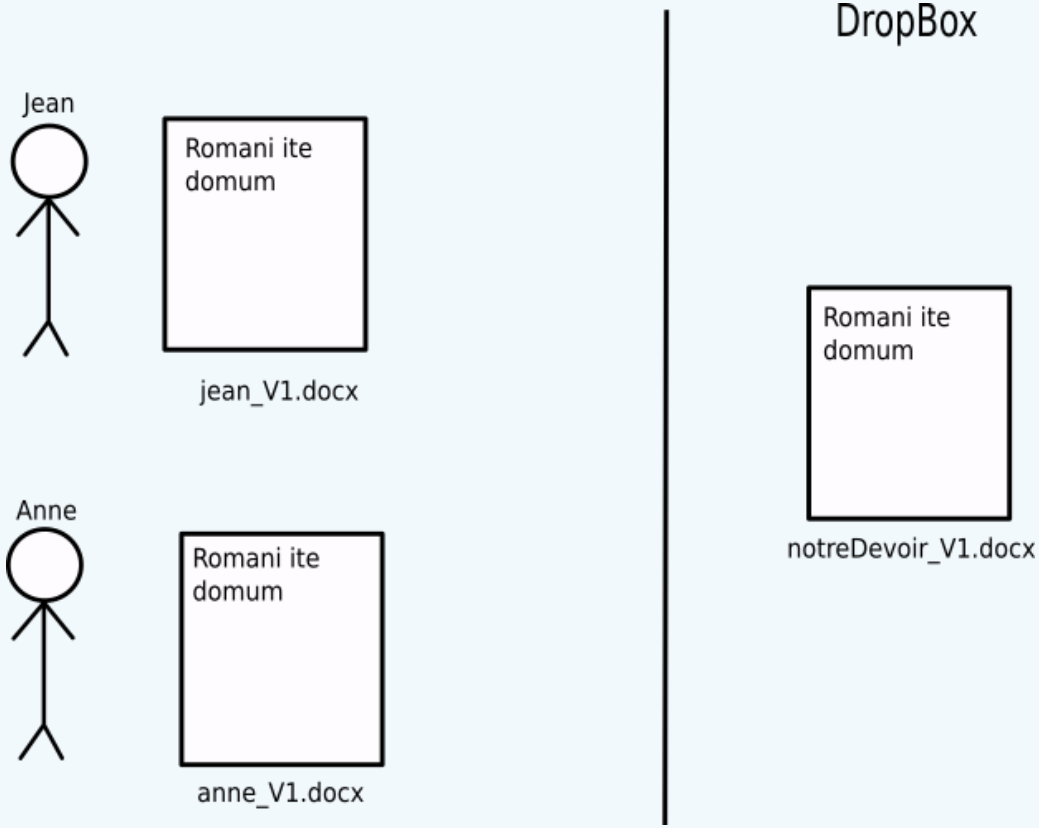
Deux nouvelles notions :

- **Une branche :** une séquence de versions prenant racine dans une version donnée.
- **Une fusion (ou merge) de branches :** Modification d'une version d'une branche en fonction des modifications apportées à une version d'une autre branche (**ce qui inclut aussi les suppressions !**)

Remarque : avec un système de gestion des versions comme Git on peut donner des noms à nos branches pour les identifier et passer d'une branche à une autre aisément !

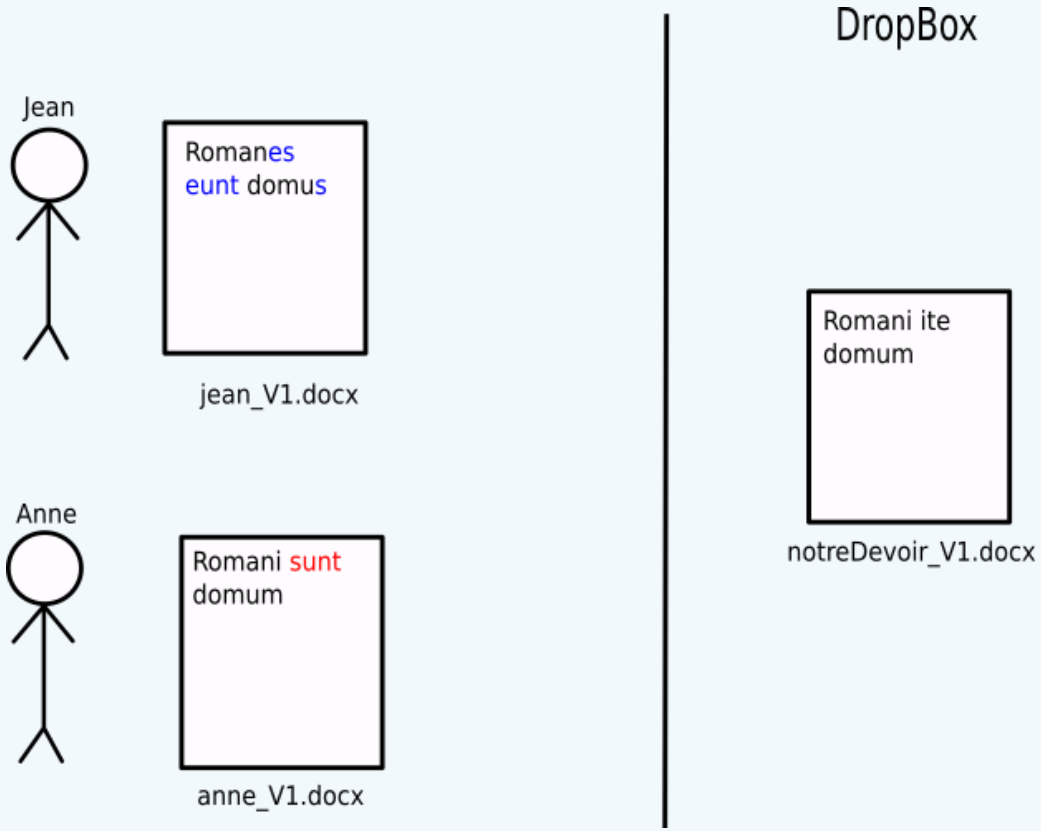
Dernier cas : Jean travaille avec Anne

Au début tout va bien. Jean et Anne ont le même document. Pour se partager le devoir ils ont un serveur type "dropbox" où ils mettent la dernière version commune du devoir :



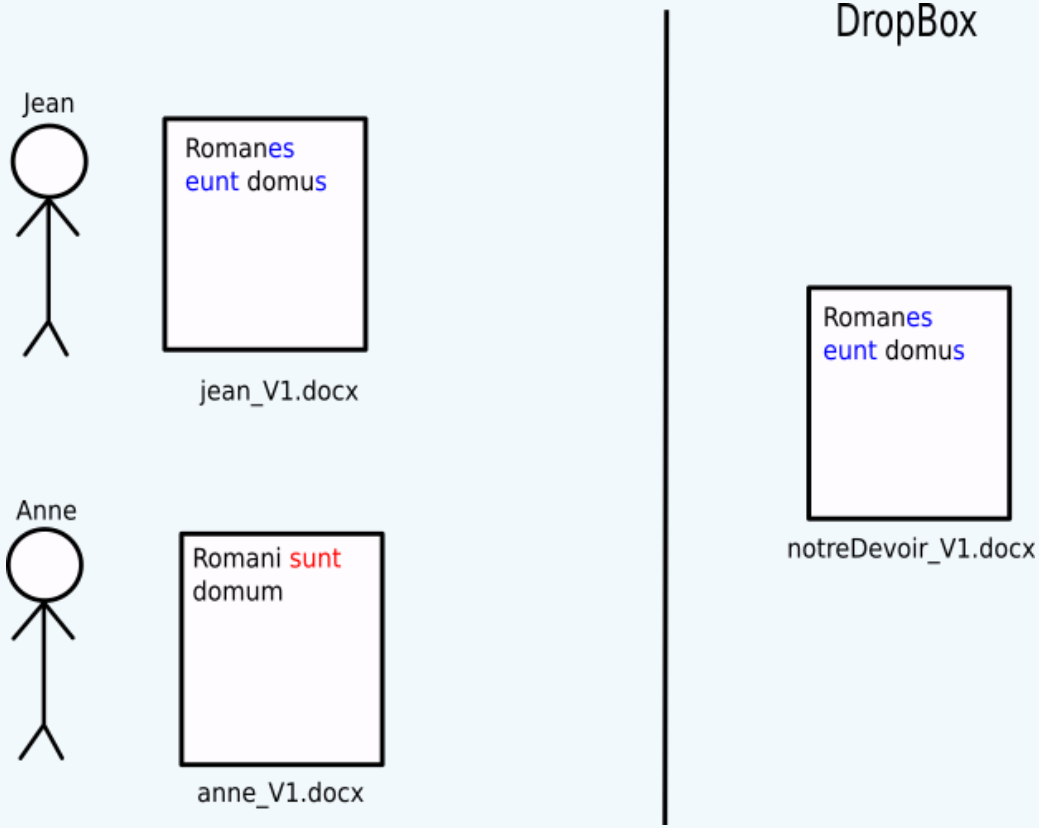
Dernier cas : Jean travaille avec Anne

Jean et Anne se sont bien partagé les tâches... Mais à un moment donné Jean a besoin de modifier une partie "commune". Il pense qu'Anne n'aura pas besoin de la modifier... Sauf qu'Anne pense la même chose de son côté :



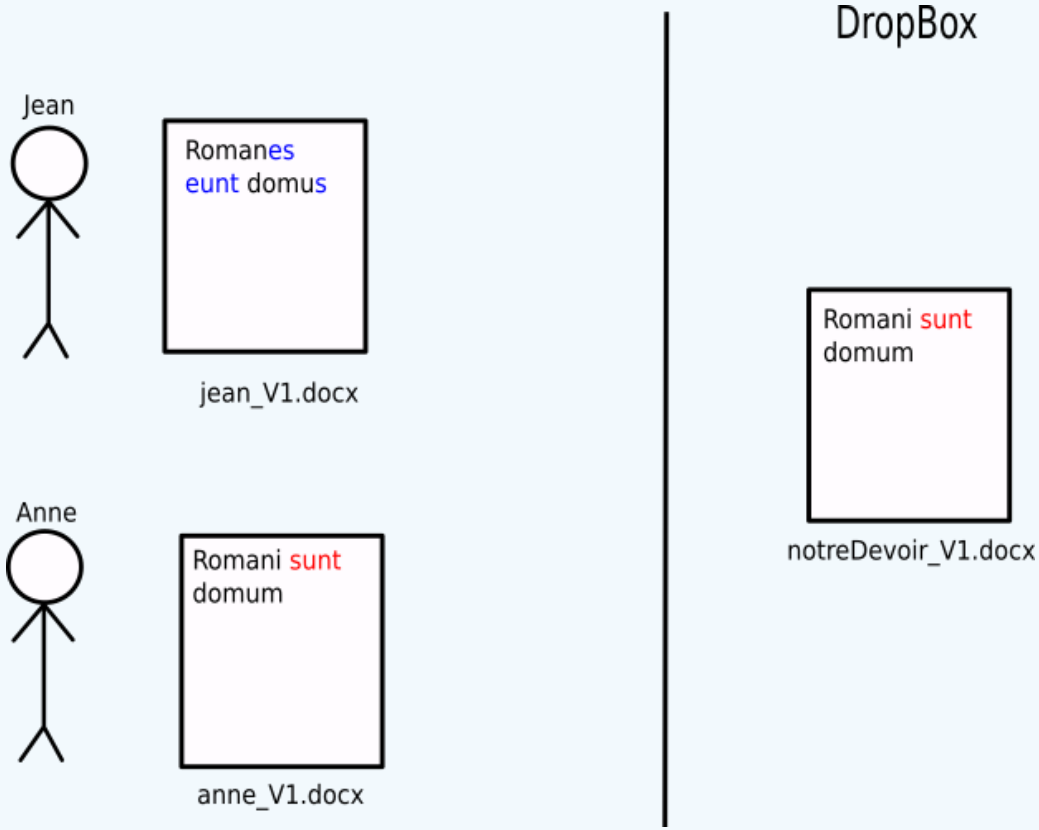
Dernier cas : Jean travaille avec Anne

Jean met alors à jour la version commune :



Dernier cas : Jean travaille avec Anne

Puis c'est au tour d'Anne :



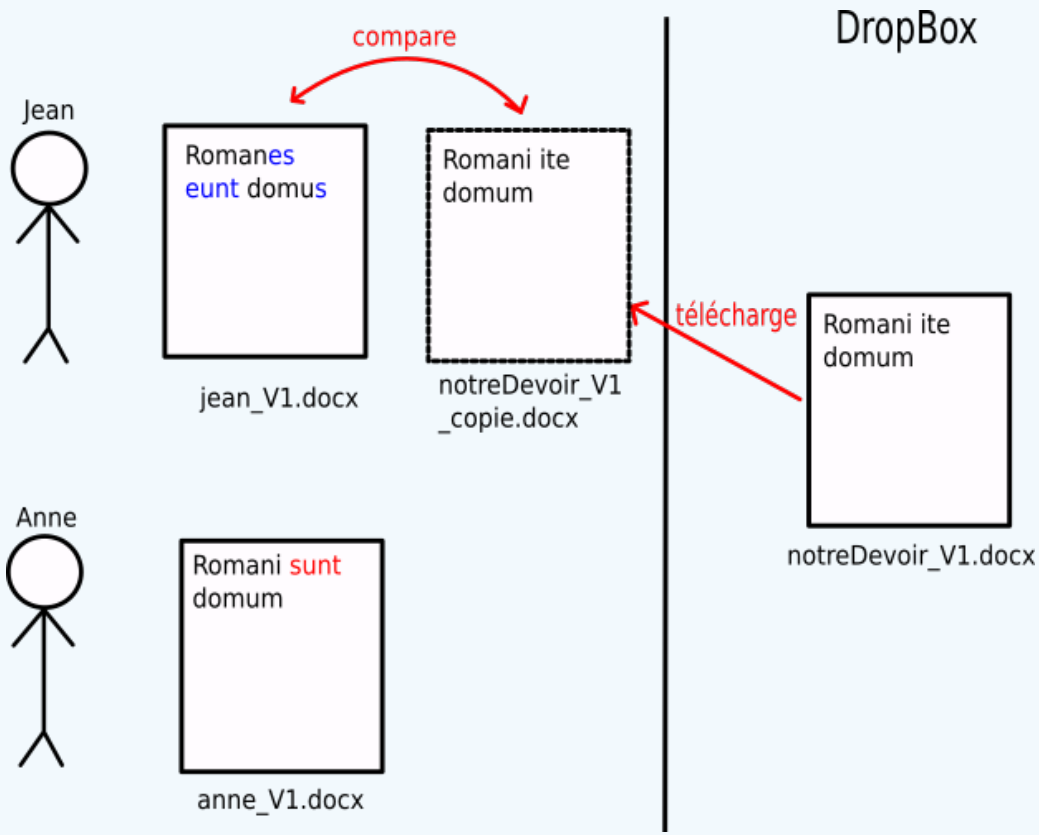
Dernier cas : Jean travaille avec Anne

On comprend alors que de nombreux problèmes peuvent survenir après cette situation :

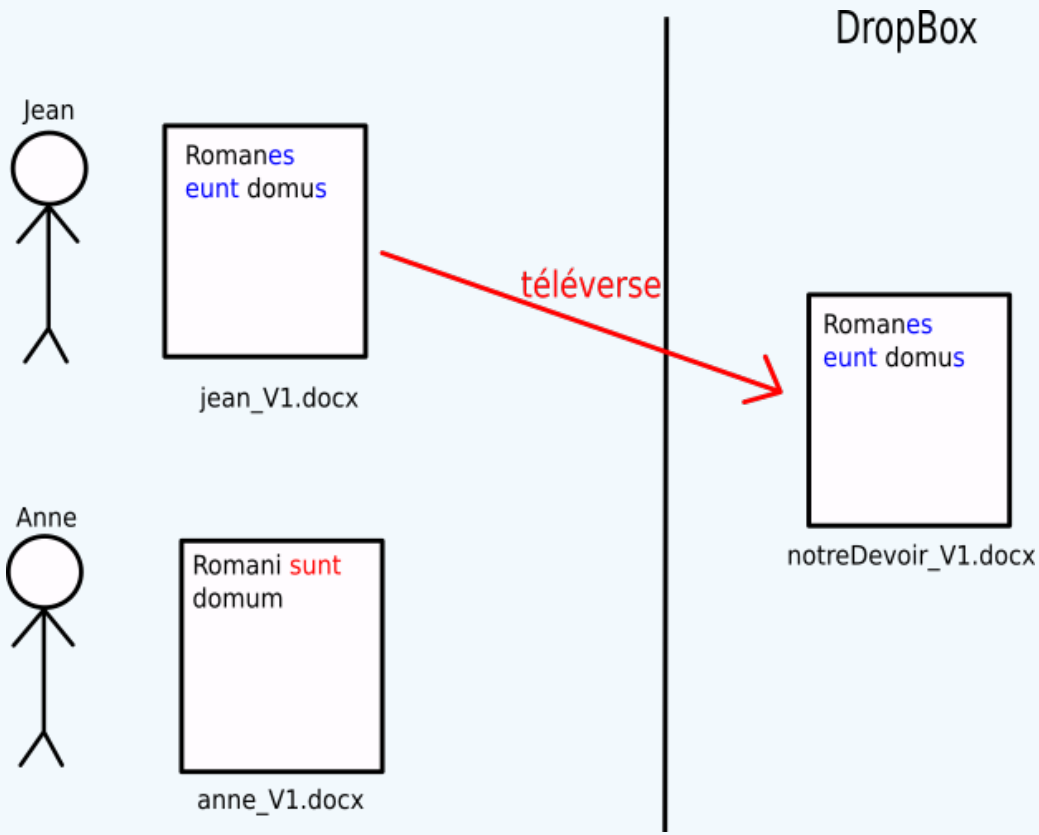
1. Si Jean ne se rend pas compte du problème suffisamment tôt ses modifications seront perdues.
2. S'il se rend compte du problème il devra de toute façon relire toutes les parties "communes" qu'ils ont tous les deux modifiés pour revenir sur un document "propre".
3. Si le nombre de conflits est conséquent un retour à un document "propre" peut devenir tout simplement impossible...

Comment Anne et Jean auraient-ils alors pu prévenir ce problème ?

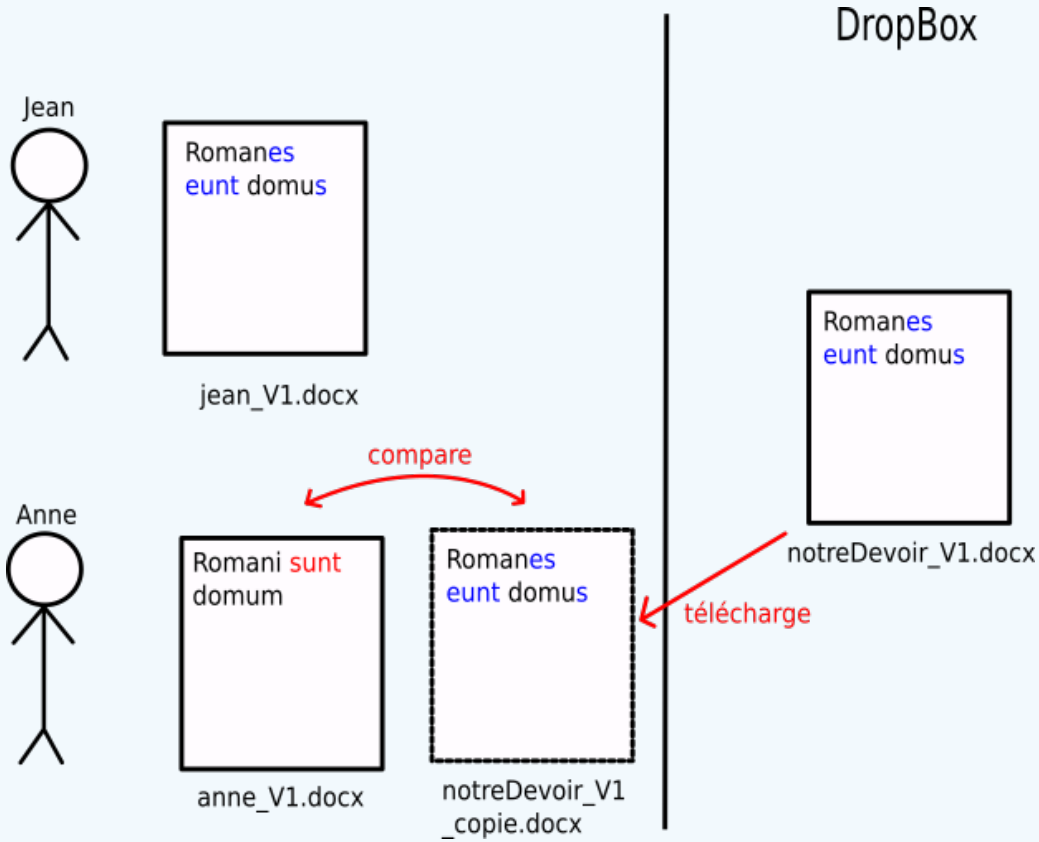
Dernier cas : Jean travaille avec Anne



Dernier cas : Jean travaille avec Anne



Dernier cas : Jean travaille avec Anne



Résumé du dernier cas

- **Un conflit** survient lorsque deux collaborateurs sur un même document éditent la même zone de ce document.
- Une manière de gérer ce conflit est de vérifier que le document distant n'a pas été modifié avant de le mettre à jour. Ce processus se fait en deux étape : 1) on télécharge le document (cf. "fetch") 2) on compare/fusionne les documents si besoin. Sous Git, ces deux opérations combinées s'appellent un "**pull**".
- Une fois que ce "pull" a été fait on peut alors mettre à jour le document distant. Ce processus de téléversement (ou "upload") est appelé sous Git un "**push**".

Université de Poitiers 1431

Conclusion du dernier cas

La gestion "à la main" des conflits et des mises à jour d'un document peut être sérieusement laborieuse à mesure que le projet grandit.

Ajouté à cela la notion de branches à gérer, voire de branches sur le document distant (et/ou sur le document local) et le tout devient au mieux très chronophage, au pire menace le développement futur du projet.

Table of Contents

1 La notion de version et ses implications
- Première étude de cas : notions de version/log/reset/diff
- Seconde étude de cas : notions de branches/merge
- Dernière étude de cas : notions de conflits/pull/push

2 Les VCS et Git
- Les services offerts par les VCS
- Les VCS comparés aux outils collaboratifs en ligne
- Les services offerts par Git
- Les services offerts par GitHub

3 Le plan de la formation

Ingéniérie logicielle = VCS

Les VCS ("Version Control System") offrent généralement les services suivants :

1. Sauvegarde de versions
2. Historique de versions
3. Comparaison de versions
4. Branches de versions
5. Partage de versions
6. Identification de conflits

Remarque : Tout peut être versionné !

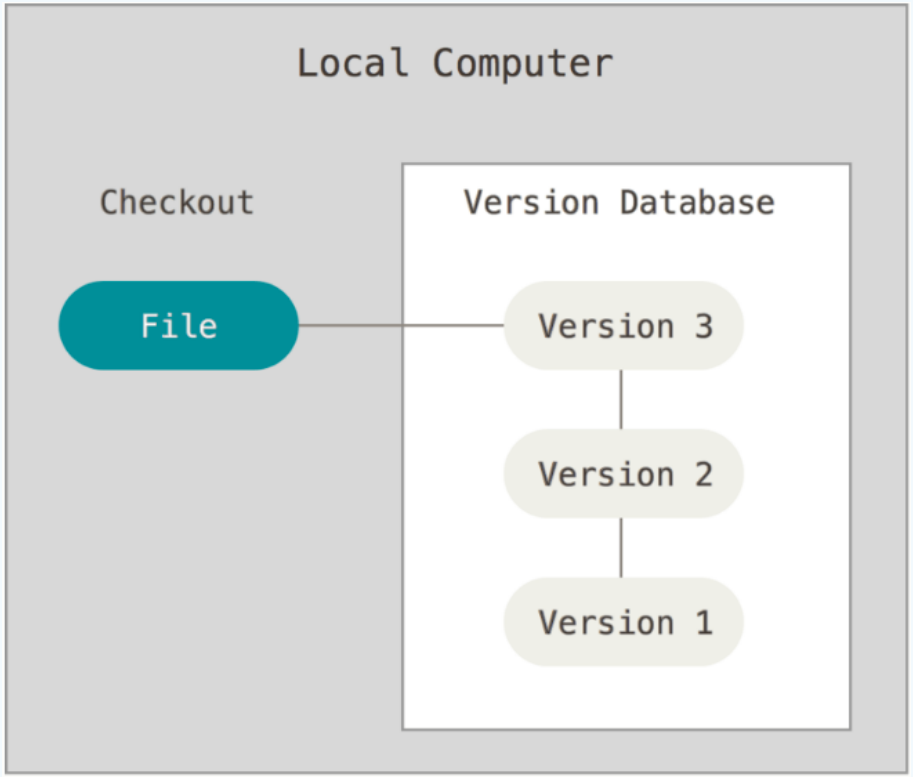
VCS versus Google Docs

VCS	Google Docs
1. Sauvegarde de versions 2. Historique de versions 3. Comparaison de versions 4. Branches de versions 5. Partage de versions 6. Identification de conflits	1. Sauvegarde de versions 2. Historique de versions 3. Comparaison de versions

Remarque : On ne peut pas exécuter du code dans google docs non plus...

Git : un VCS distribué, gratuit et open-source

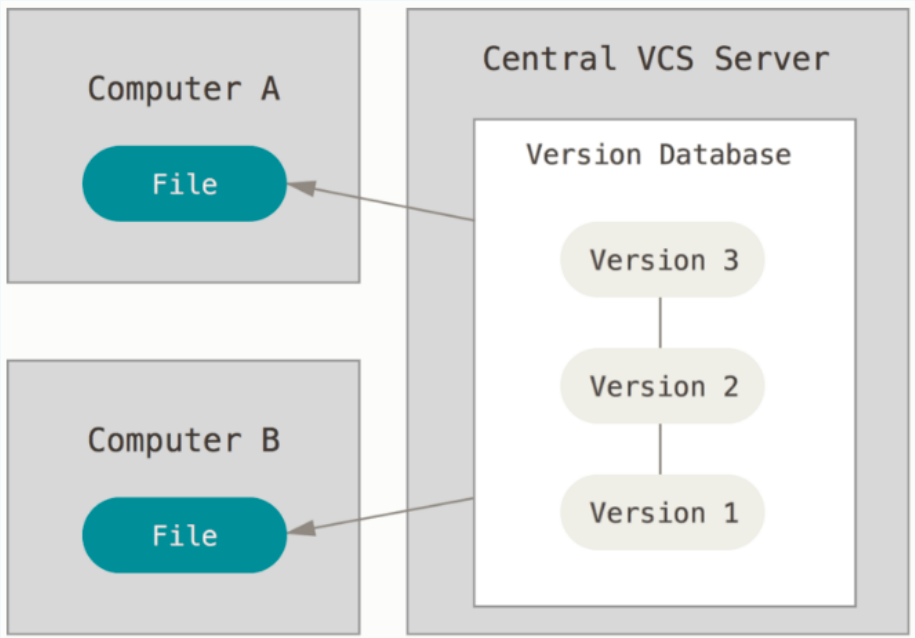
Les VCS locaux :



Remarque : Checkout indique ici que le projet "File" est dans l'état correspondant à la "Version 3"

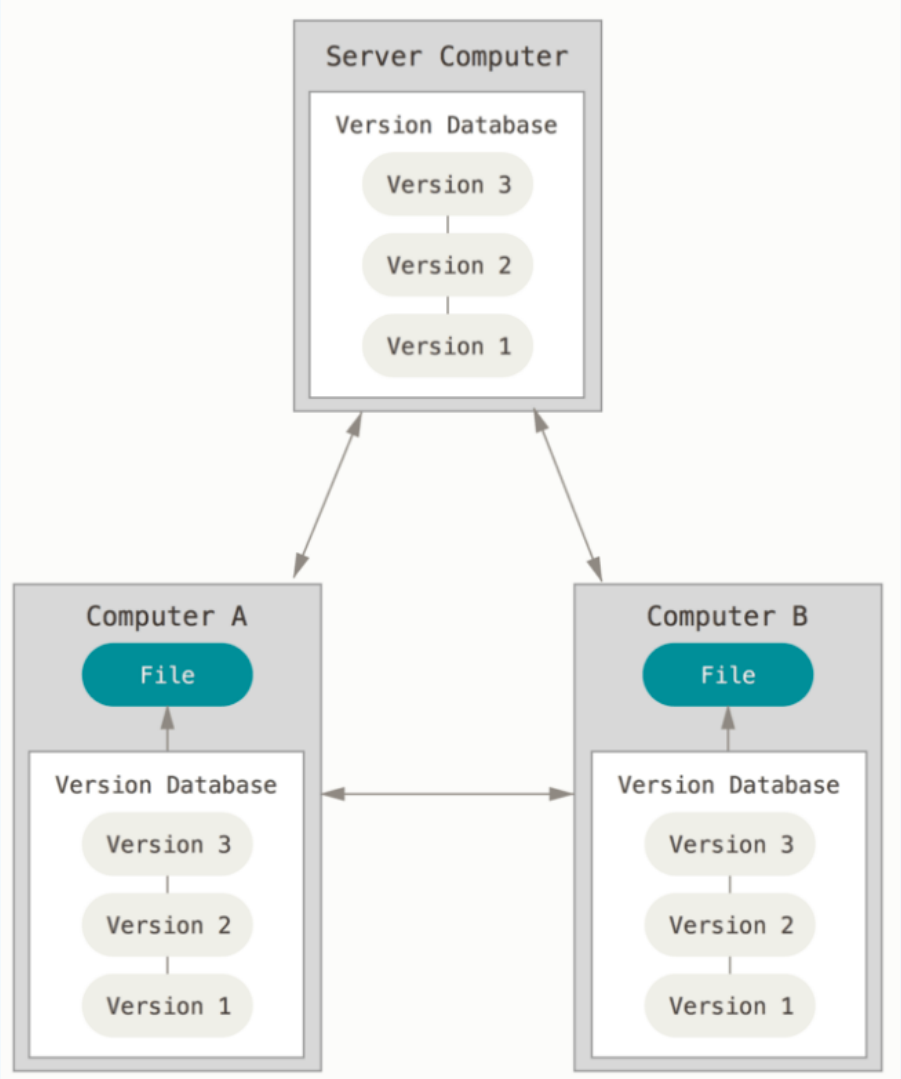
Git : un VCS distribué, gratuit et open-source

Les VCS centralisés :



Git : un VCS distribué, gratuit et open-source

Les VCS distribués :



Git : un VCS distribué, gratuit et open-source

Les avantages des VCS distribués :

1. Si on n'a pas de connexion internet on peut continuer à travailler
2. Si le serveur central est défaillant quelqu'un aura l'historique
3. On peut facilement changer de dépôt distant "de référence" (cf. les "fork")

Git : un VCS distribué, gratuit et open-source

VCS centralisés	VCS distribués
• SVN	• Git • Mercurial • Bazaar

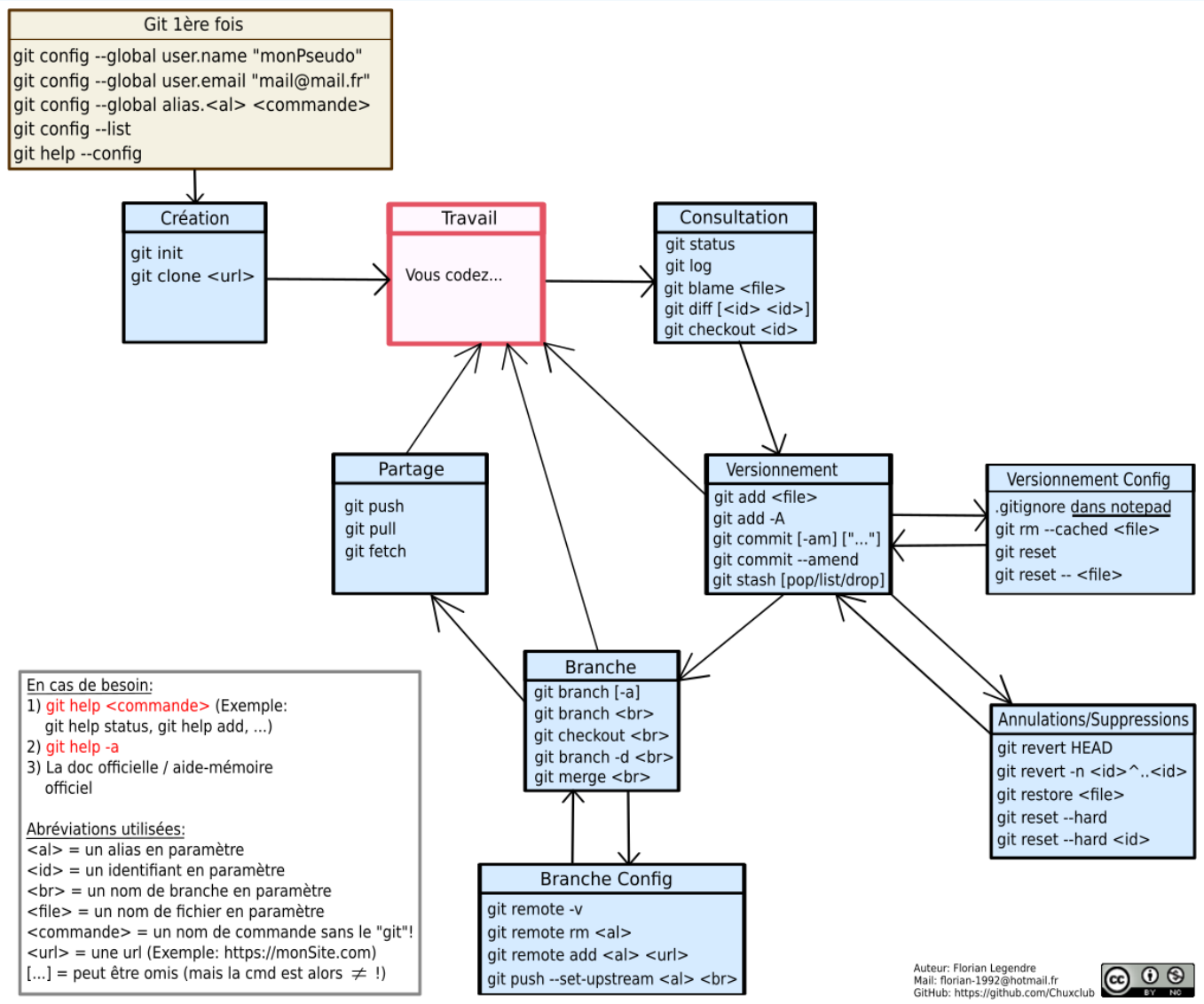
Git n'est pas GitHub et réciproquement

Git	GitHub
1. Git est un VCS 2. Git a une fonctionnalité cliente lui permettant de communiquer avec un dépôt 3. Git a été écrit par Linus Torvald (l'auteur du noyau Linux) 4. Les compétiteurs de Git sont Mercury, Bazaar, SVN...	1. GitHub est l'équivalent de DropBox avec des fonctionnalités spécialisées pour les VCS 2. GitHub est un serveur web d'hébergement (aussi appelé "dépôt") 3. GitHub appartient à Microsoft 4. Les compétiteurs de GitHub sont GitLab, GitBucket...

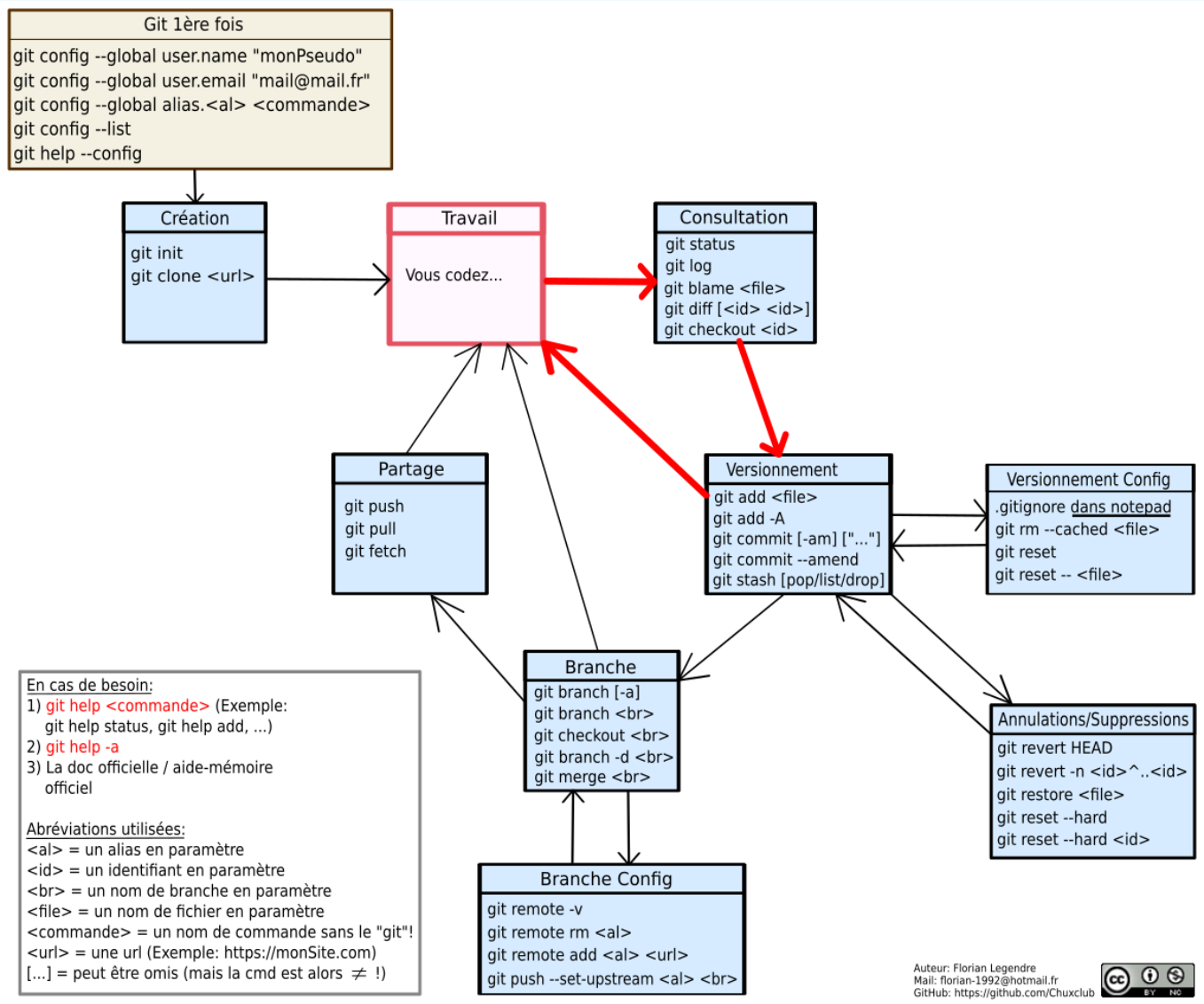
Table of Contents

1. La notion de version et ses implications
   - Première étude de cas : notions de version/log/reset/diff
   - Seconde étude de cas : notions de branches/merge
   - Dernière étude de cas : notions de conflits/pull/push
2. Les VCS et Git
   - Les services offerts par les VCS
   - Les VCS comparés aux outils collaboratifs en ligne
   - Les services offerts par Git
   - Les services offerts par GitHub
3. Le plan de la formation

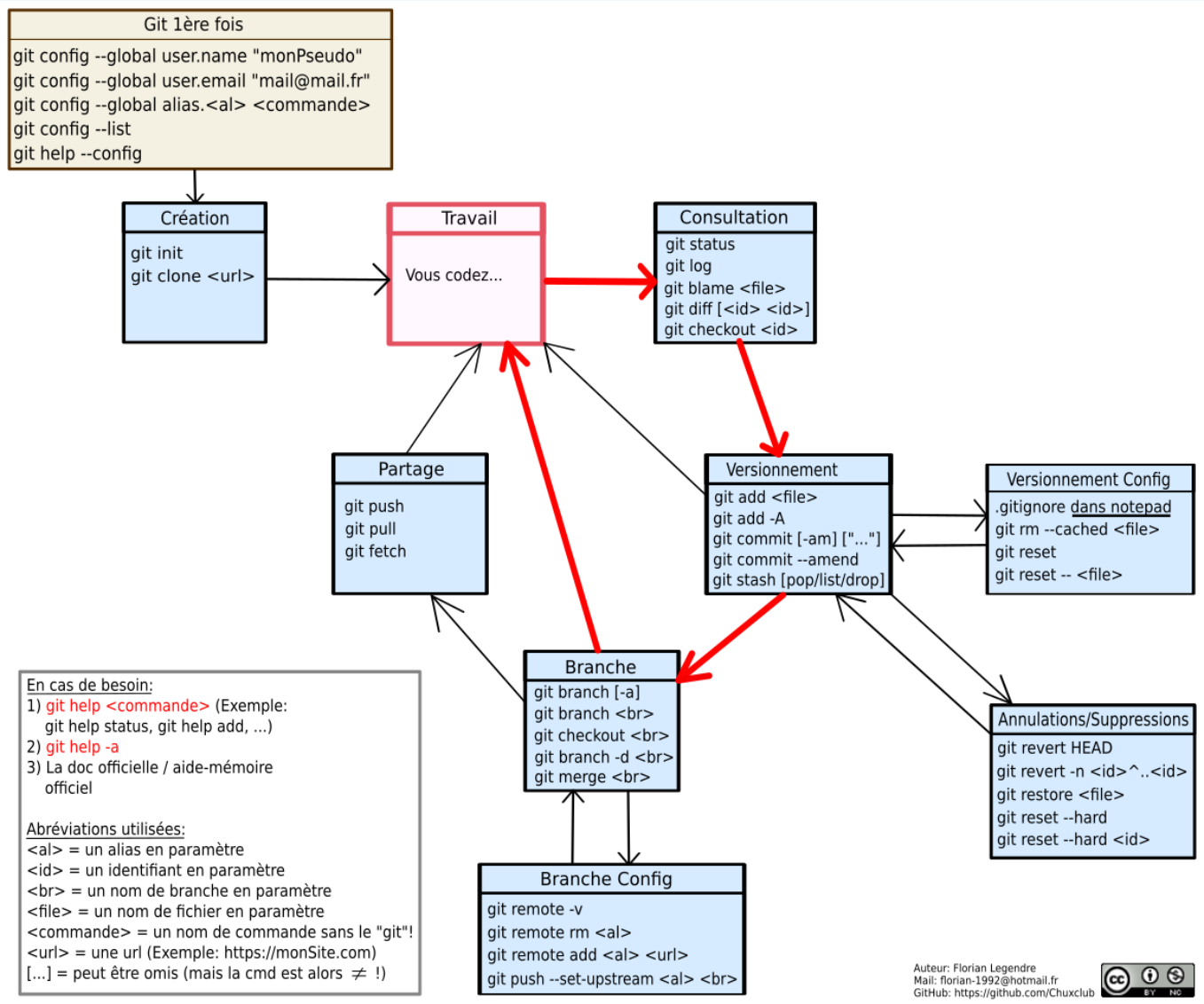
Le flot de commande Git



La boucle locale mono-branche



La boucle locale multibranche



Git collaboratif