Versionning

1.0

Cyrille François © Ateno Tech

Table des matières

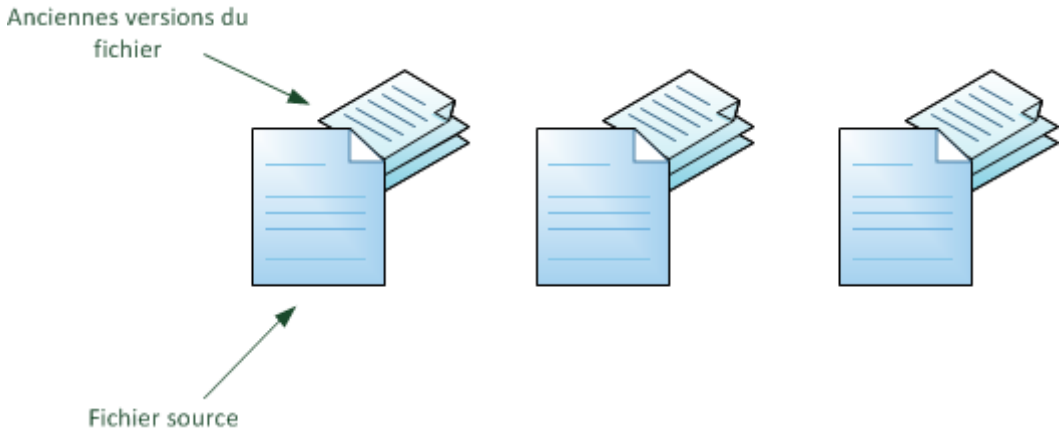
Généralités

I

A. Gestion des sources - Introduction

La gestion des fichiers textes en révision

Un système de gestion des sources permet de conserver l'ensemble des modifications de chaque fichier texte



Gestion en configuration des révisions des fichiers

Chaque fichier est accompagné de l'ensemble de ses révisions.

Une révision représente toutes les différences entre 2 modifications d'un fichier

Attention : Ne pas confondre révision et version

Une version regroupe plusieurs fichiers, alors qu'une révision est un ''paragraphe'' supplémentaire dans l'historique d'un fichier.

Définition : Système de gestion des sources

Un système de gestion des sources permet de **conserver l'ensemble des modifications de chaque fichier texte** qui lui est confié.

C'est une base de données fichiers dans laquelle est stockée :

- Chaque modification d'un fichier est une nouvelle **révision** du fichier
- Les révisions d'un ensemble de fichiers constitue une **version**

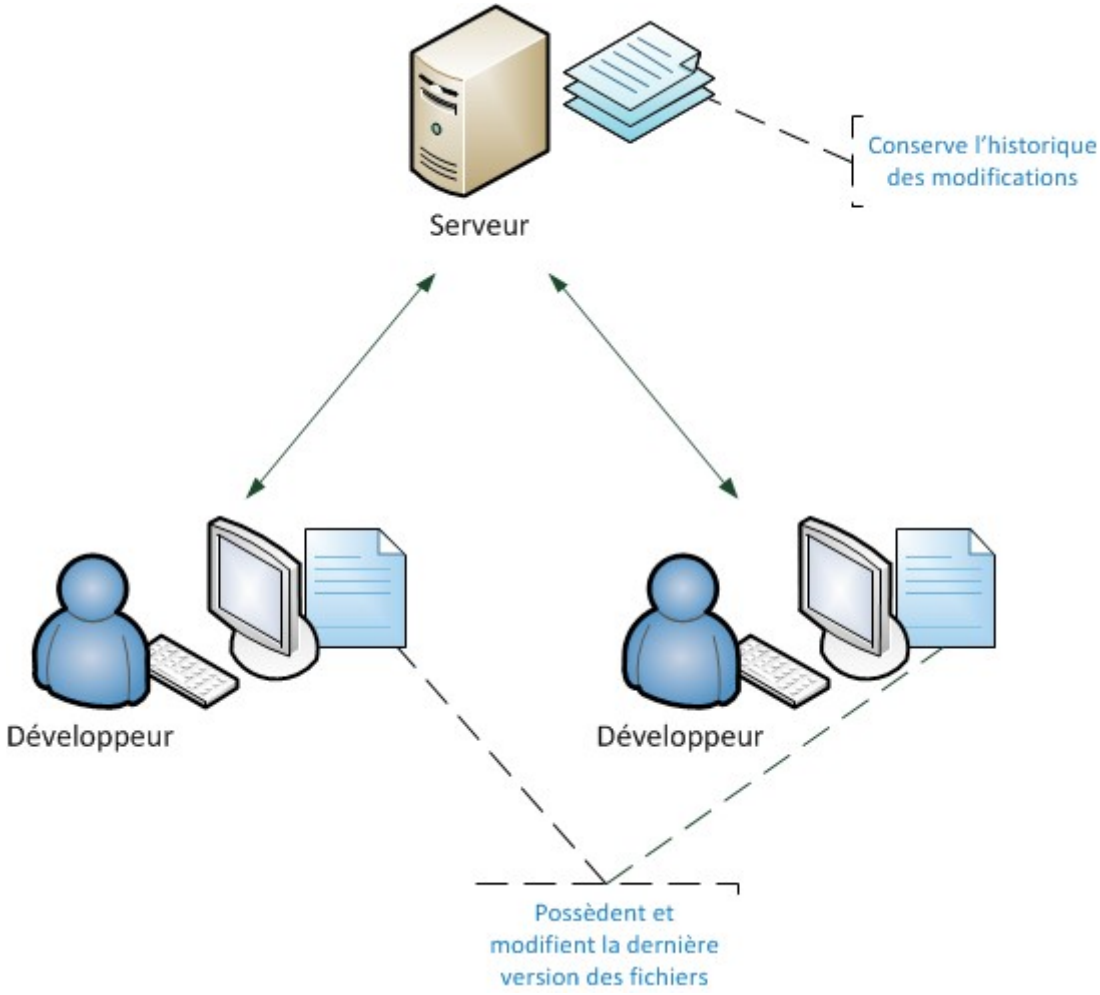
Différentes architectures des systèmes de gestion des sources

Plusieurs solutions existe, les principales se distinguent par :

- Une gestion centralisée des versions -> un seul système centralise les versions
- Une gestion distribuée des versions -> les bases sont répliquées sur chaque client

Architecture de gestion en configuration de type Centralisée

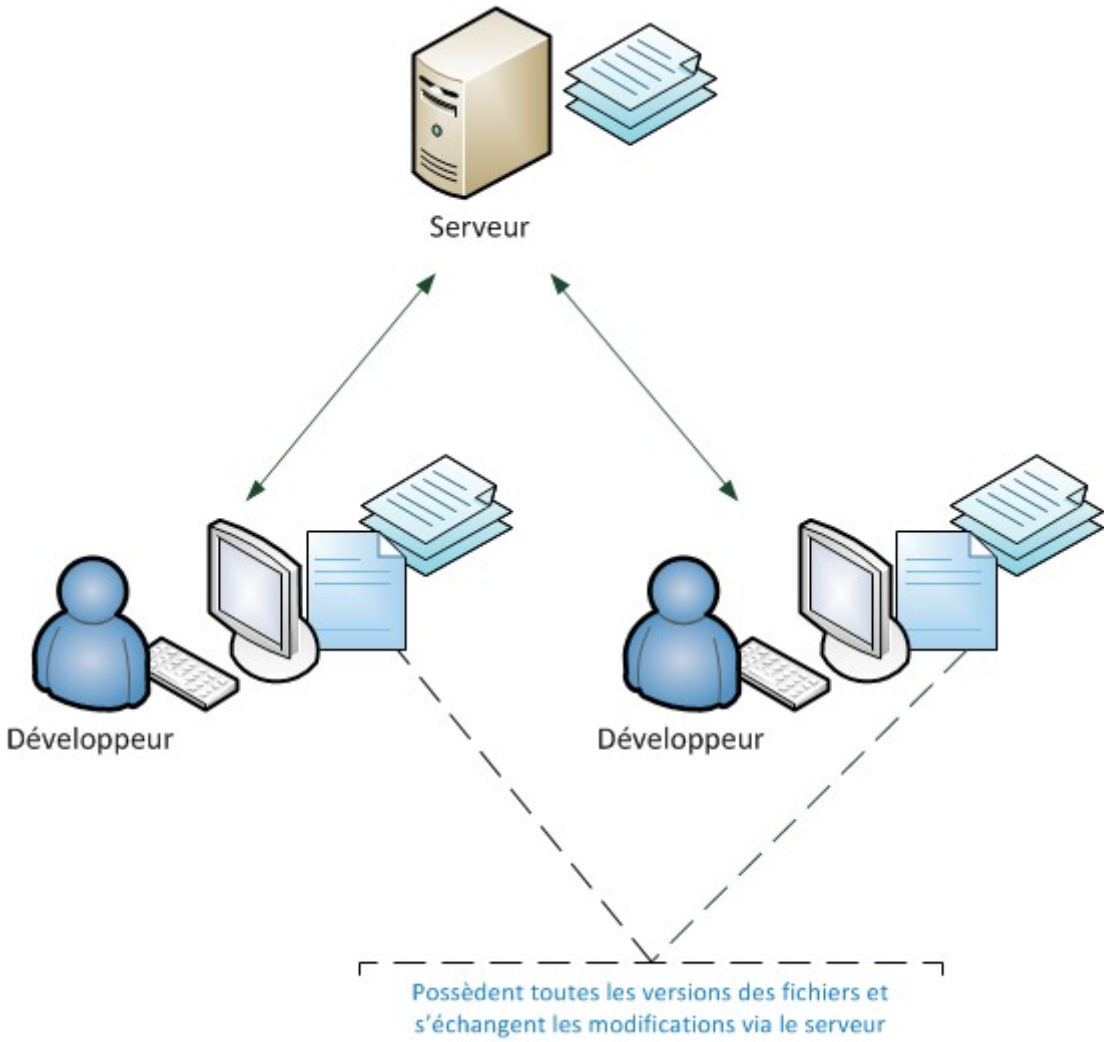
Chaque client duplique en local une version d'un projet (**checkout**)
Ses dernières modifications sont conservées en local dans le projet
Ses modifications sont rendues visibles aux autres clients que lors de la publication
Chaque client publie sur le serveur ses modifications du projet (**commit**)



Gestion en configuration centralisée

Architecture de gestion en configuration de type Distribuée

Chaque client duplique en local la gestion en configuration d'un projet (**pull**)
Ses modifications sont gérées en local
Ses modifications sont rendues visibles aux autres clients que lors de la publication
Chaque client publie sur le serveur ses modifications dans sa gestion en configuration d'un projet (**push**)



Gestion distribuée des versions

Fondamental : Différentes architectures des systèmes de gestion des sources

SVN : Gestion centralisée (ne conserve pas l'historique de modification local de tout fichier)

Git : Gestion distribuée (conserve l'historique de modification local de tout fichier)

B. Gestion des sources - Init, Add et remove

Initialisation d'une base de gestion des sources

Cette opération (**init**) permet de créer et d'initialiser l'ensemble des fichiers de gestion de la base de fichiers.

Connexion à la base de fichiers
Préparation des indexes de révision des fichiers
Préparation des indexes de version des projets
Prêt pour les premières révisions

Image 1 Initialisation

Remarque : Importation/Exportation

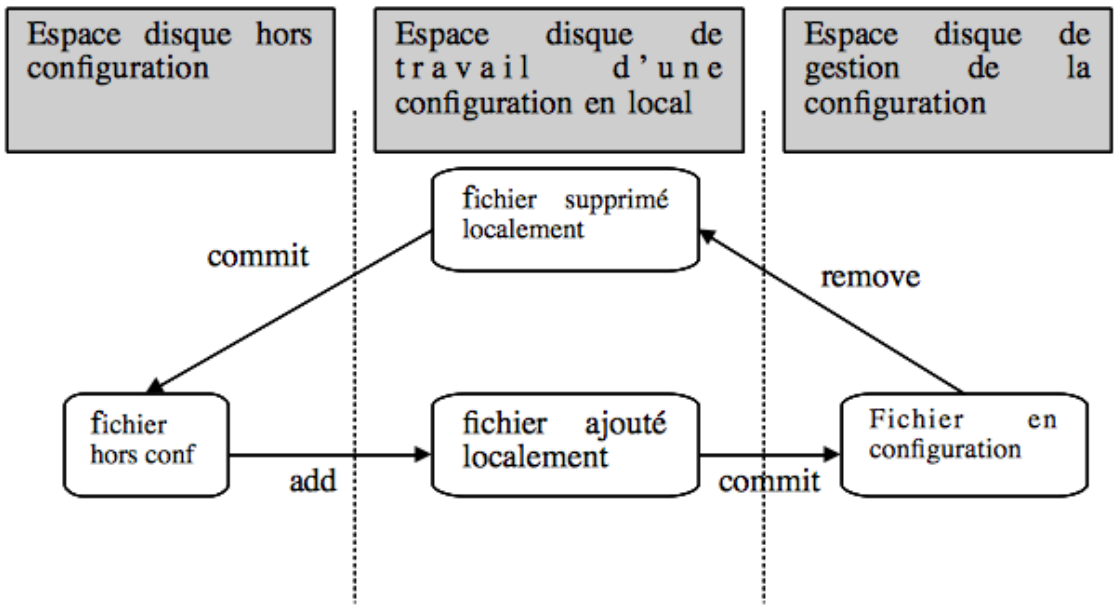
Exportation : sortir une version des fichiers sans les informations de gestion en configuration

Importation : ajouter un ensemble de fichiers dans une première version

Fondamental : Cycle d'archivage

Le cycle d'archivage d'un fichier correspond à un processus itératif de mise à jour successives de ses révisions (commit) à partir de son insertion (add) jusqu'à sa suppression de la base (remove).

Exemple



Ajout et suppression d'un fichier en base

Remarque

L'espace local est le point de passage de tous fichiers de l'anonymat à sa gestion en configuration et vice versa.

Cet espace local assure la connexion à la base, il permet la mise à jour avec cette base (**update**) mais n'assure aucune mise en configuration tant qu'un acquittement (**commit**) n'est pas effectué.

Fondamental

Un fichier supprimé de la base reste toujours présent dans les versions antérieur des projets dont il fait partie

C. Gestion des sources - Checkout et Commit

Retrouver une révision d'un fichier, une version d'un projet

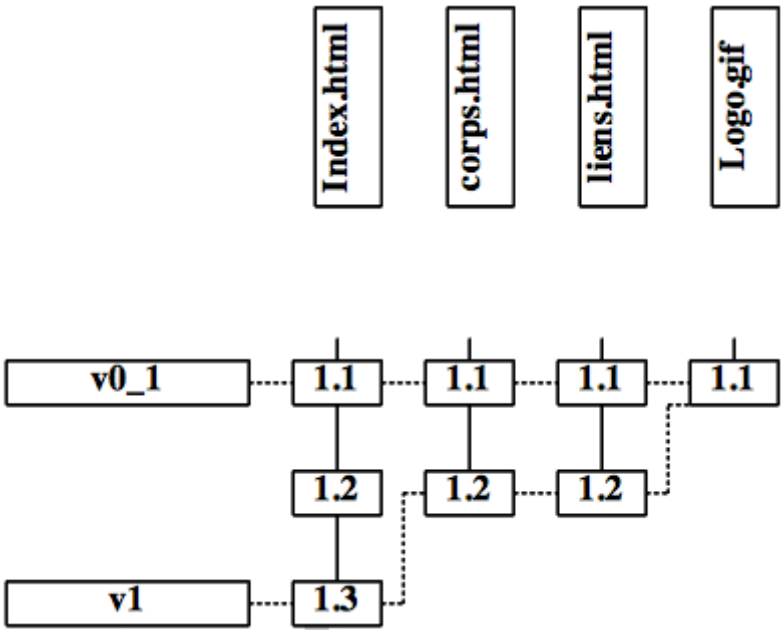
La commande **Checkout** permet la restitution d'une version d'un projet ou d'une révision d'un fichier

L'ensemble des fichiers du projet sont à la révision associée à la version du projet

Exemple : Checkout

Checkout de V0_1 : Index.html, corps.html, liens.html et Logo.gif en révision 1.1

Checkout de Index.html en révision 1.2 : Index.html dans sa révision 1.2
Checkout de v1 : ...



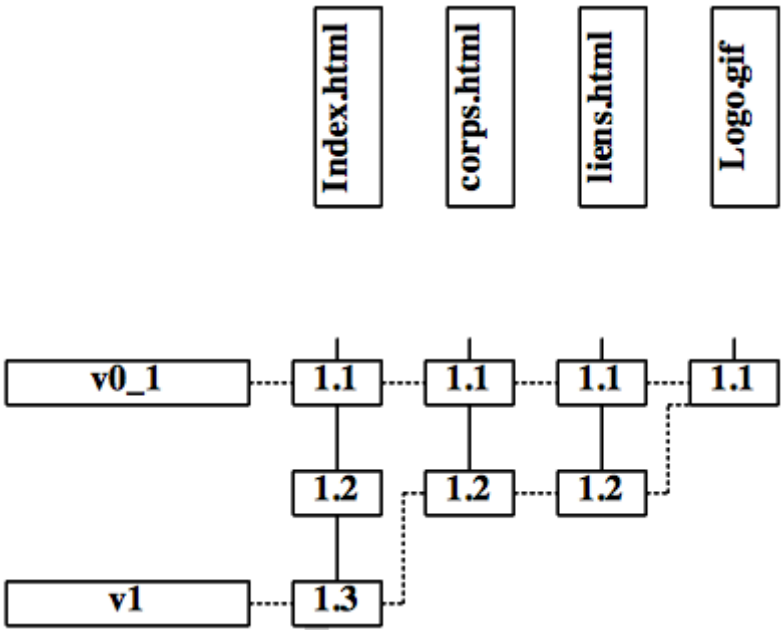
Mise en version d'un ensemble de révisions

Créer une nouvelle révision d'un fichier

La commande **commit** permet de créer une nouvelle révision d'un ensemble de fichiers d'un projet ou bien d'un fichier

Exemple : Commit

Commit de V1 : Dernière version des fichiers Index.html, corps.html, liens.html. Le fichier Logo.gif reste inchangé en révision 1.1



Mise en version d'un ensemble de révisions

Attention : Commenter toute modification

A chaque commit vous devez commenter les modifications

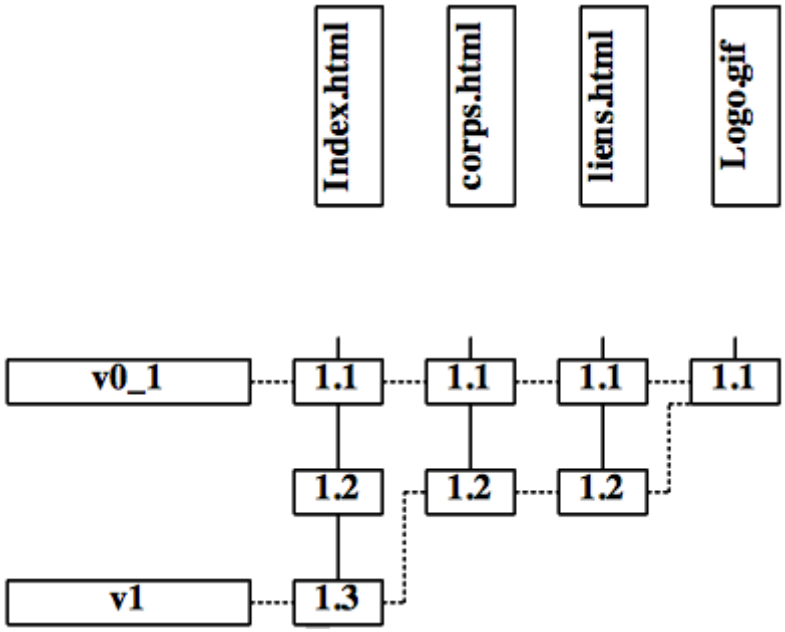
Exemple : correction bug n°382993

D. Gestion des sources - Branches et merge des branches

Rappel : Rappel des révisions et versions

Il est possible de gérer des révisions sur les fichiers

Et des versions regroupant un ensemble de révision des fichiers du projet



Mise en version d'un ensemble de révisions

La gestion des versions à plusieurs

Les modifications à plusieurs sur une version ne posent généralement pas de problème...

Tant qu'un même fichier en version 1.x n'est pas modifié par 2 personnes !

Exemple : Exemple d'une gestion des versions à plusieurs

David et Roland récupèrent la version 1.1 de *index.html* et chacun en local la modifie...

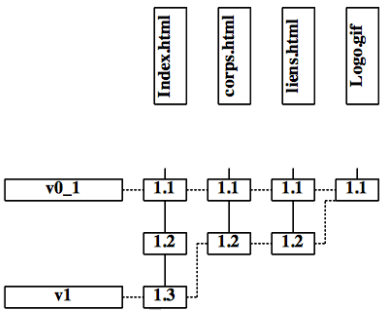


Image 2 Mise en version d'un ensemble de révisions

David est le plus rapide et partage sa version sur le serveur (révision 1.2 de index.html)

Quand Roland veut partager sa version sur le serveur il faut qu'il lève les éventuels conflits...

... c'est donc à Roland de comparer ses modifications avec celles de David et de les fusionner dans une nouvelle révision de index.html

Attention : Gestion des versions/révisions : évolutions en parallèle

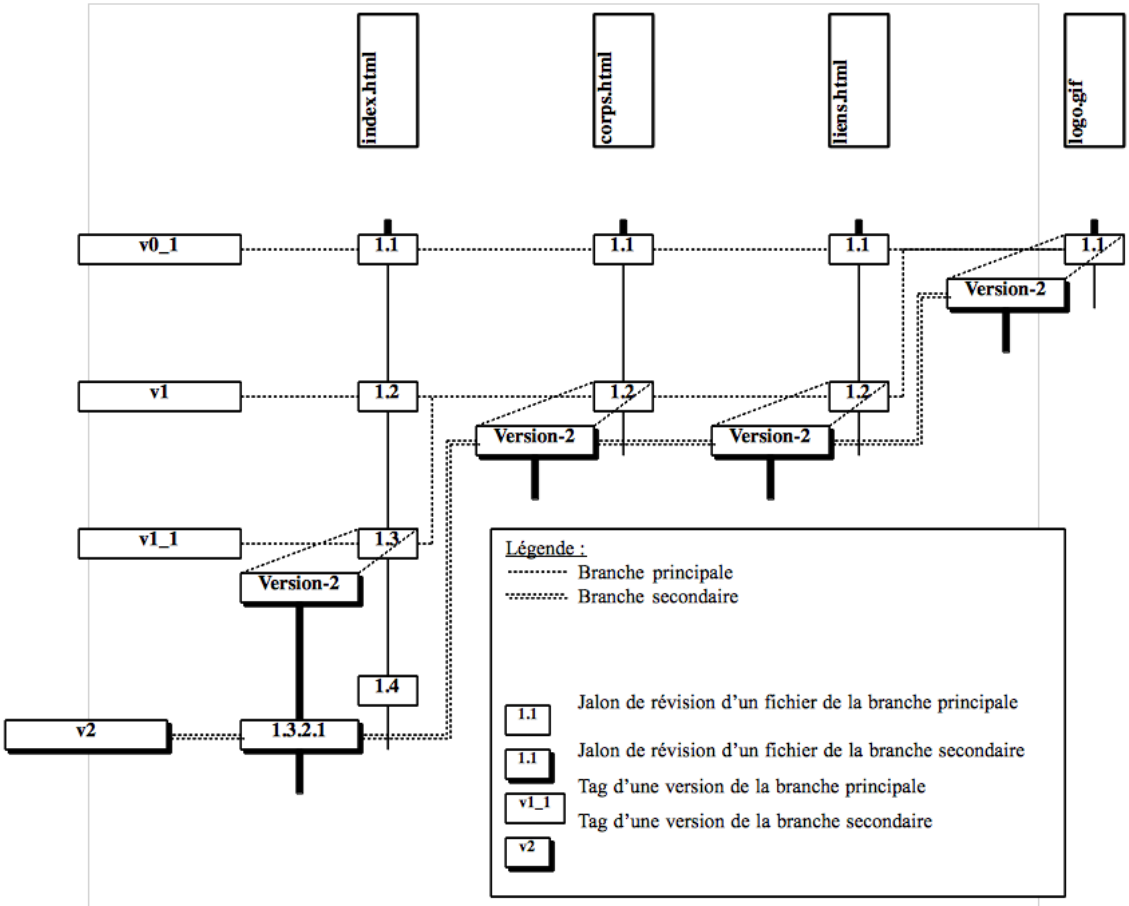
Comment limiter les conflits ?

=> faire évoluer le code en parallèle

Comment faire évoluer en parallèle plusieurs versions ?

Gestion des versions/révisions

Les branches : ensembles de révisions de fichiers issue d'une même version du projet et continuant à évoluer **indépendamment** d'une branche à l'autre



Versions et branches

Exemple

La version de Roland en 1.3.2.1 est différente de celle de David en 1.4

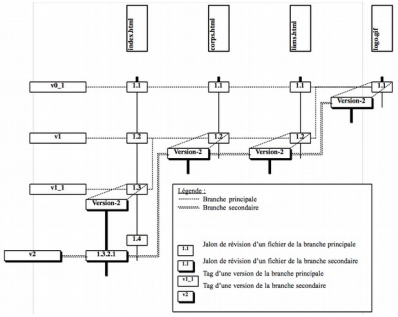


Image 3 Versions et branches

Attention : Gestion des versions/révisions

Comment regrouper des branches ?

Gestion des versions/révisions : le merge

L'opération de merge facilite le regroupement de branches.

L'outil de gestion en configuration identifie le code en conflit sur les 2 branches

L'opérateur du merge lève les conflits

L'opérateur du merge crée une nouvelle version de merge

Fondamental : Gestion des versions/révisions

Travailler avec des branches permet de ne pas laisser au développeur la responsabilité de lever les conflits entre branches !

Git

II

A. Gestion des sources - EGit

Texte légal

Les illustrations et la découpe de ce support suivent celle de l'excellente présentation faite de EGit à l'EclipseCon Toulouse 2013

Git & EGit beginner Workshop

B. Gestion des sources - EGit : Staging area

Fondamental : Vue des ressources dans Eclipse

Eclipse ne restitue pas directement le contenu du disque mais une image

Modification des ressources

Modification des ressources du workspace :

- Depuis Eclipse : l'image est rafraîchie automatiquement par Eclipse
- Hors Eclipse : rafraîchir Eclipse avec F5

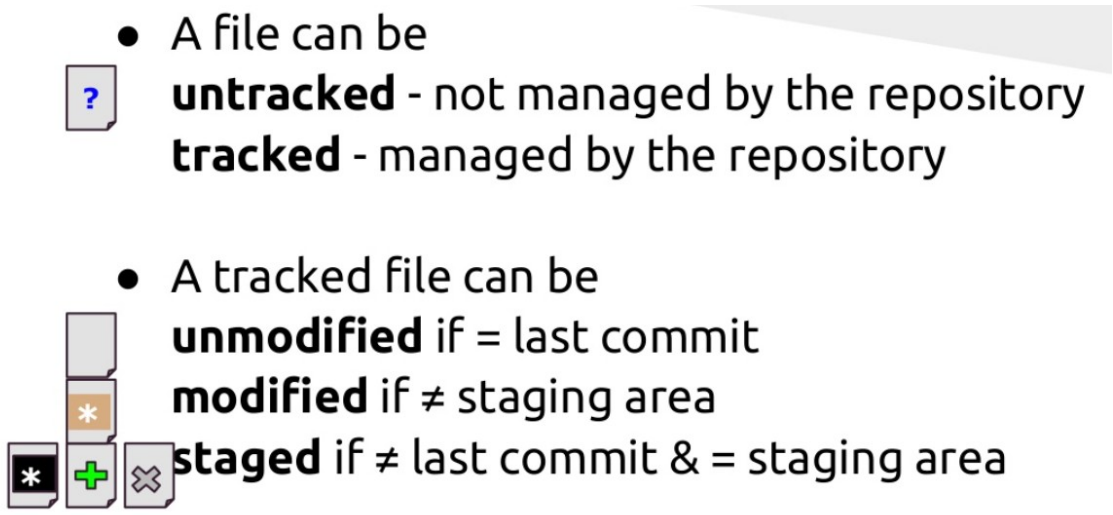
Dans Eclipse, les ressources ont toutes un état dans le workspace :

- Modifié
- Non modifié

Zone de préparation : Staging area

En gestion en configuration, de nouveaux états :

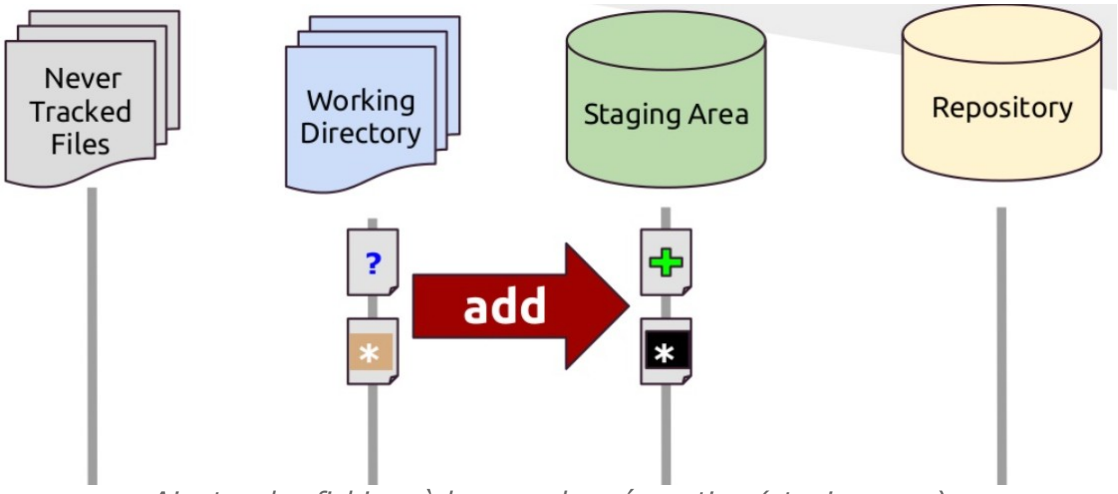
- untracked/tracked : ressources gérées en configuration
- unmodified : identique au dernier commit
- modified : différent de la zone de préparation
- staged : différent du commit mais dans la zone de préparation



Statut des fichiers dans EGit

Ajout d'une ressource en gestion en configuration

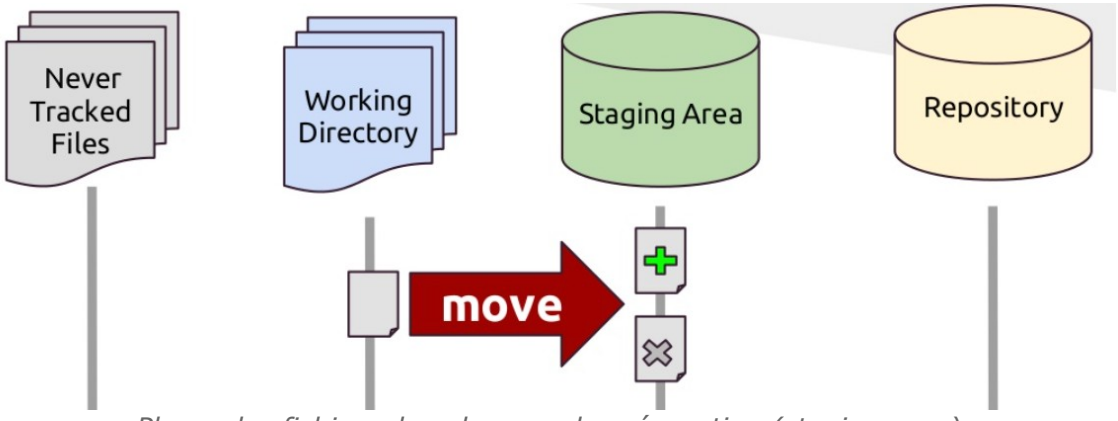
Ajouter un fichier qui était dans le workspace



Ajouter des fichiers à la zone de préparation (staging area)

Ajout d'une ressource en gestion en configuration

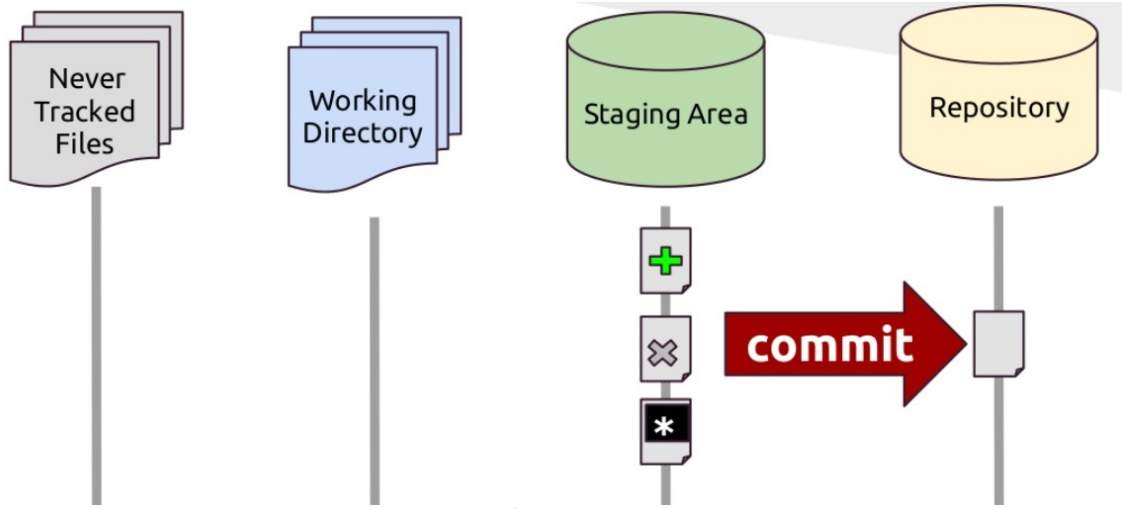
Déplacer un fichier dans la gestion en configuration : il est prêt à être ajouté...



Placer des fichiers dans la zone de préparation (staging area)

Enregistrement en gestion en configuration

Dans la zone de préparation, enregistrer en configuration avec un commit

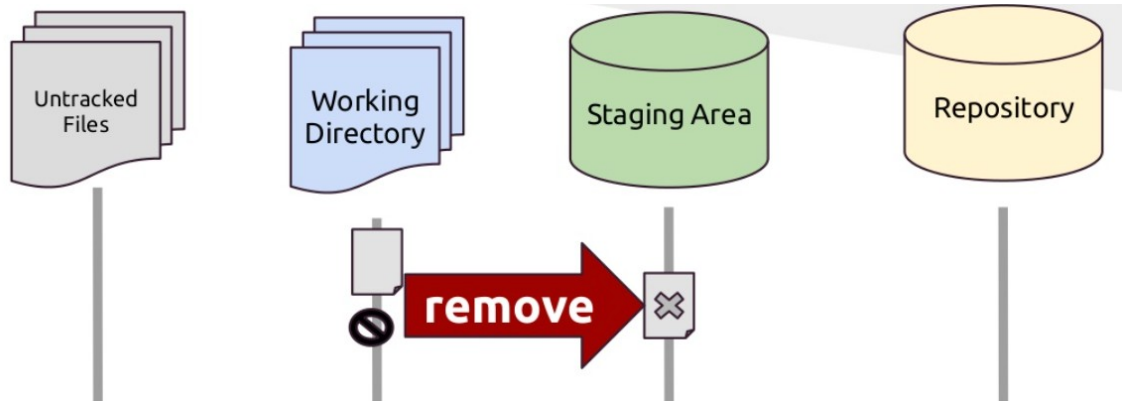


Commit : passer de la zone de préparation (staging area) -> au repo local

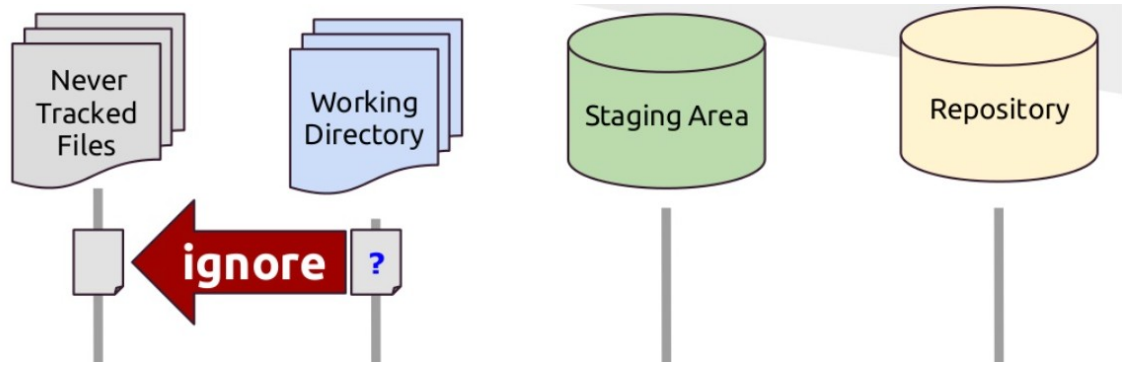
Complément : Autres opérations sur les différents espaces de EGit

Supprimer de la zone de préparation

Ignorer de la gestion en configuration



Supprimer des fichiers de la zone de préparation (staging area)

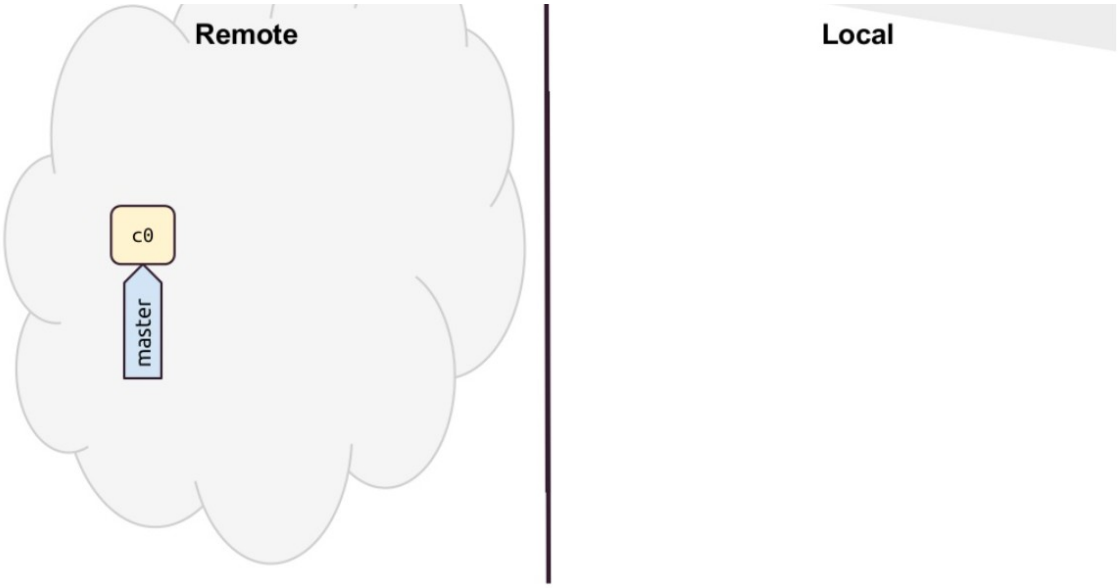


Ignorer des fichiers pour la gestion en configuration

C. Gestion des sources - EGit : Init/Fetch/Pull/Push

Travailler en local avec un repository Git

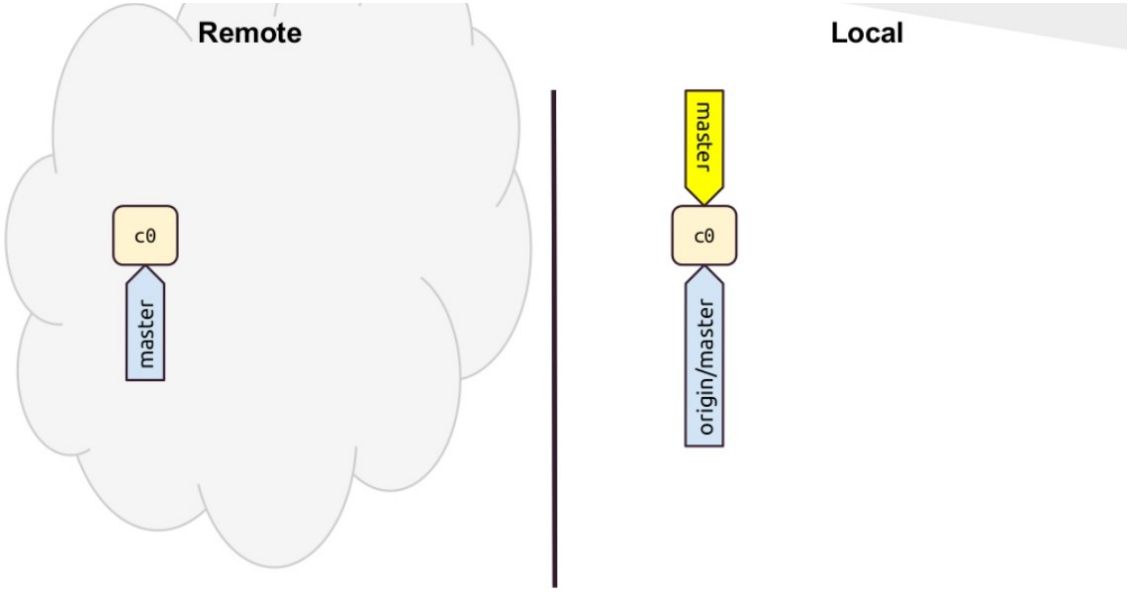
Initialisation du repository



Init

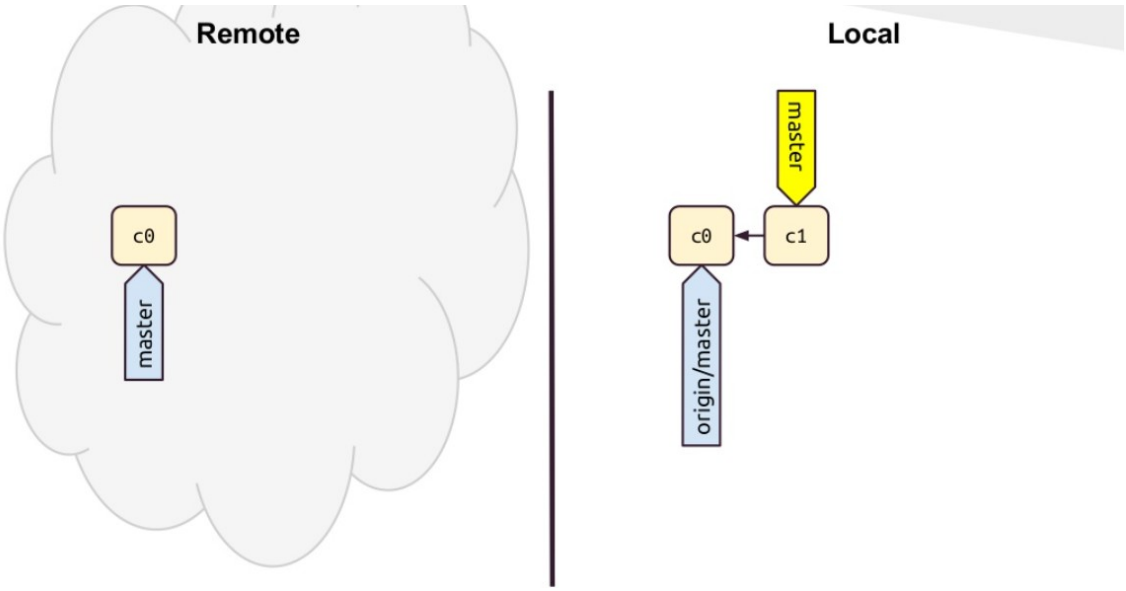
Cloner le repository en local (EGit : clone)

Recopier en local le repository distant dans notre branche *master*



Cloner en local

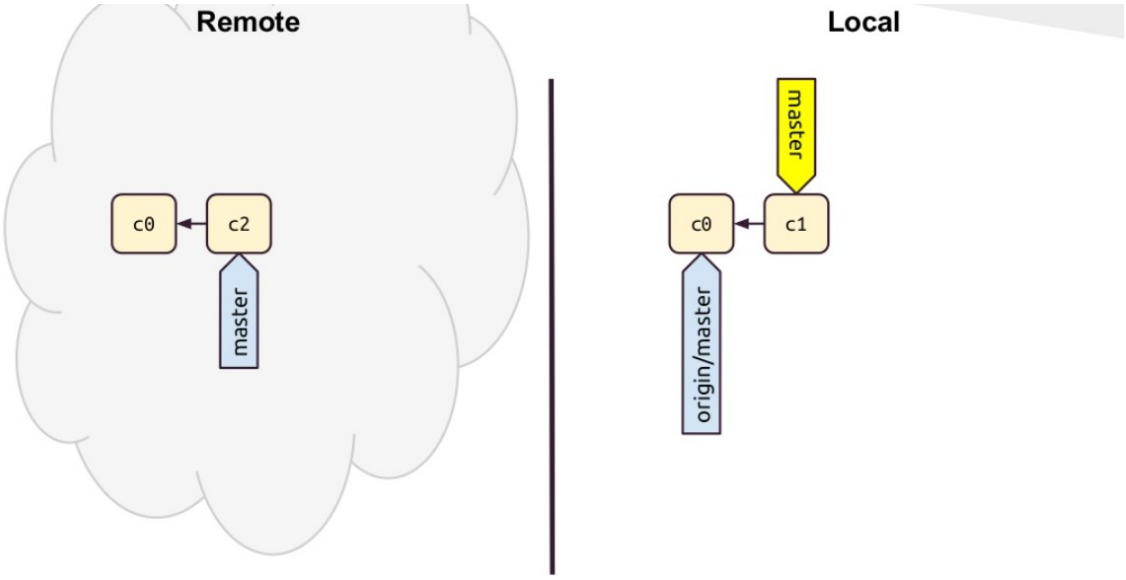
Pour synchroniser ses modifications avec celles partagées sur la branche *master/origin*



Commit local

Commit partagé

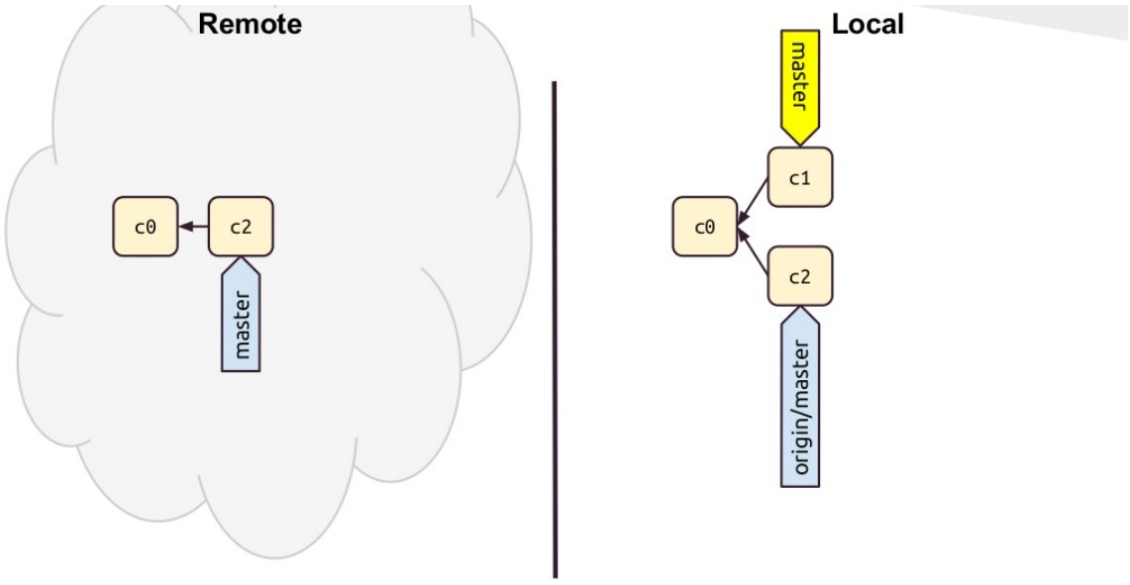
Une modification a été partagé sur la branche *master/orign*



Commit partagé

Recopier en local le commit partagé (Fecth)

La commande *Fetch* me permet de retrouver en local les dernières modifications partagées



Fetch

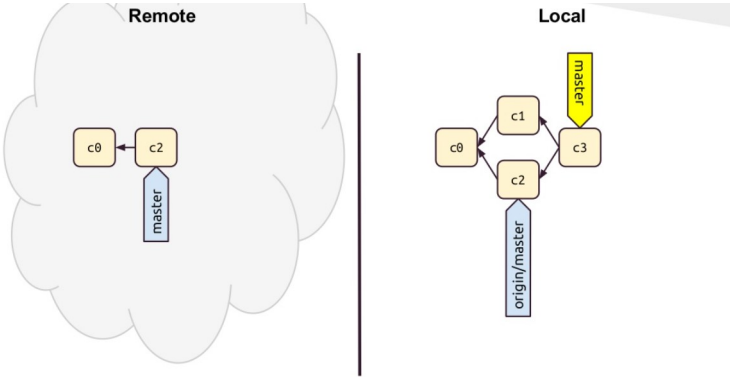
Attention

Il peut exister des conflits entre les révisions partagées de la branche locale *master* et celle du serveur *origin/master*

Il faut lever 1 à 1 ces conflits afin de pouvoir tirer une nouvelle version du projet qui n'engendrera pas de régression.

Lever les conflits entre développement local et partagé (pull)

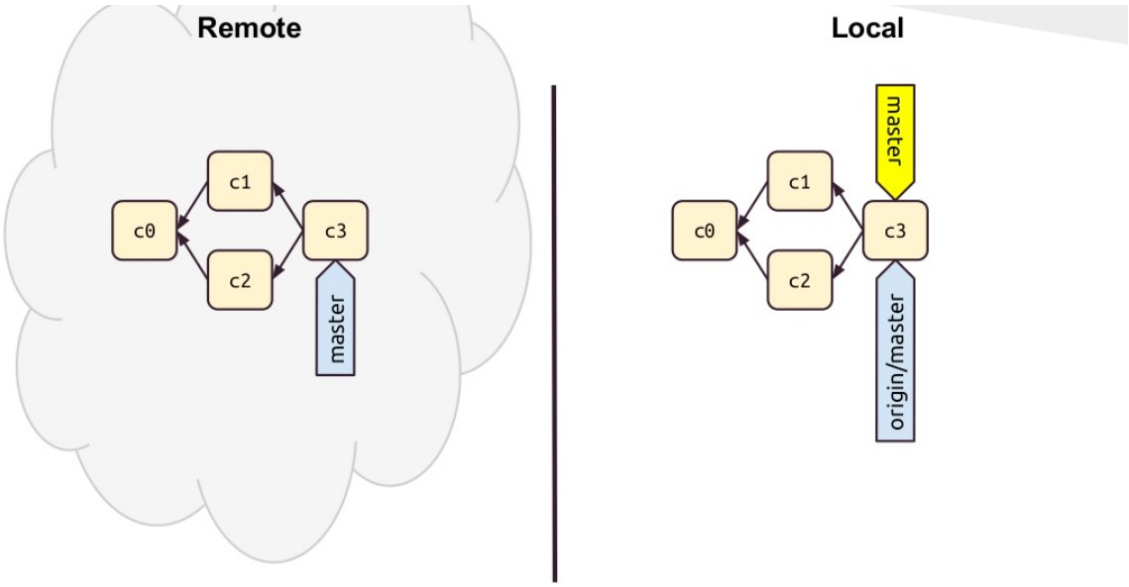
Je tire en local une nouvelle version à partir de ma version locale (*master*) et de celle partagée (*origin/master*)



Pull

Partagé notre modification (push)

Après avoir levé les risques de conflits, je partage ma modification (*push*)



Push

D. Gestion des sources - L'essentiel de Git 2

Fondamental : Git et les branches

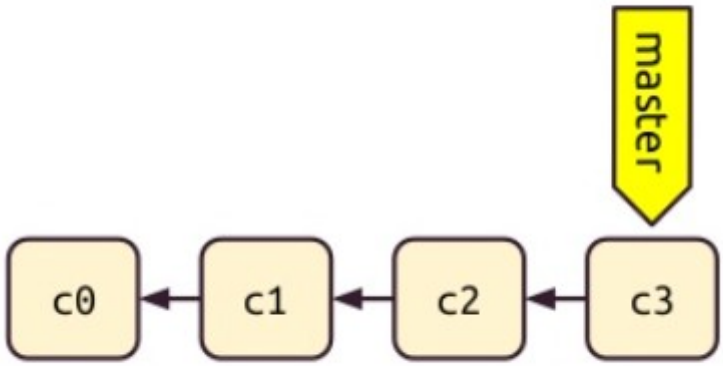
Git est conçu pour faciliter le travail de plusieurs contributeurs sur un même produit.

C'est pour cela que l'utilisation des branches et la fusion de branches (merge) est facilité.

Ci dessous un scénario classique de gestion des révisions avec Git.

Gestion en configuration sans conflit

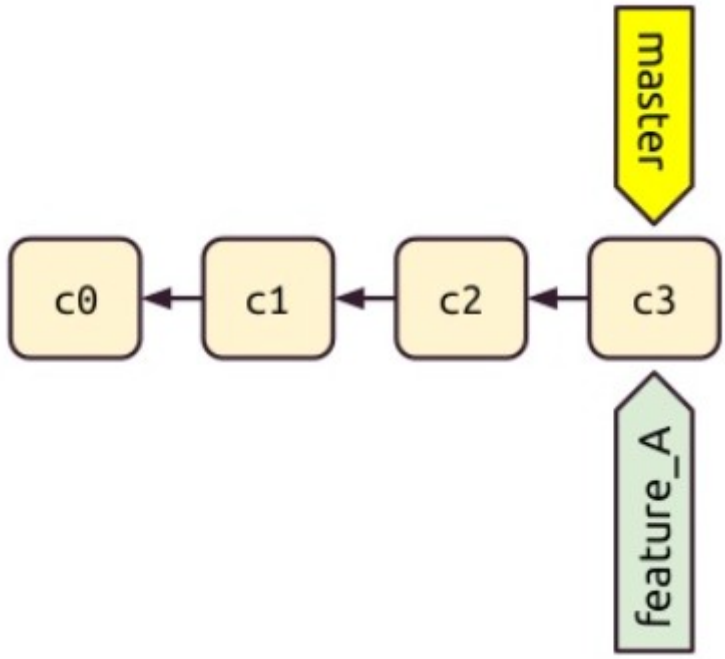
Au début, les révisions se suivent sans entrer en conflit...



Branche 'Master'

Gestion en configuration sans conflit : préparation de la fonction A

Puis viens le moment de marquer une modification qui correspond à la réalisation d'une fonction A

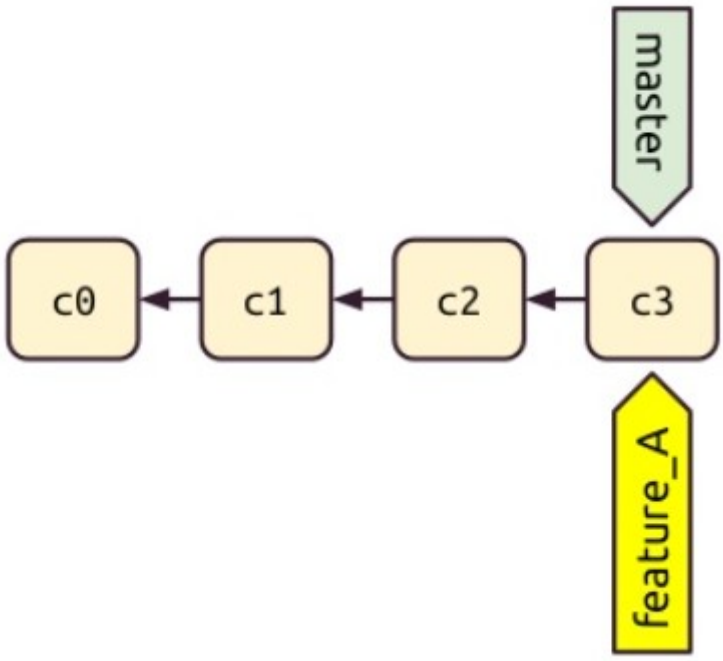


Branche Fonctionnalité A

```
1 git branch feature_A
```

Gestion en configuration sans conflit : passer sur la branche de dev de la fonction A

Maintenant que la branche de développement de la fonction A est créée il faut que je me positionne dessus



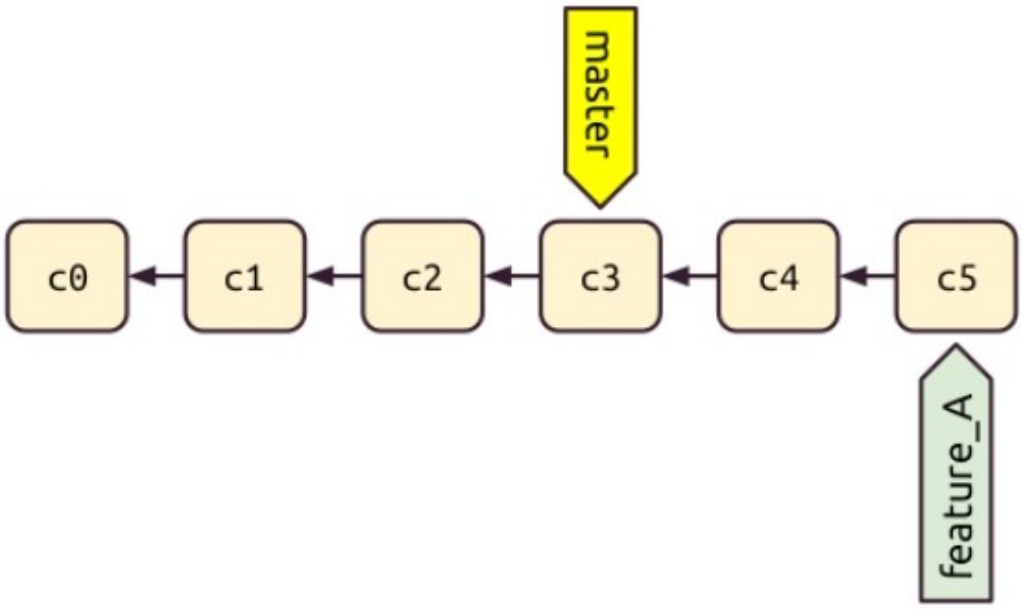
Checkout Fonctionnalité A

```
1 git checkout feature_A
```

Gestion en configuration sans conflit : développement d'une branche

Les modifications intermédiaires se suivent sur la branche de dev de la fonction A sans modifier la branche principale.

Je me replace sur la branche principale pour récupérer/faire des évolutions

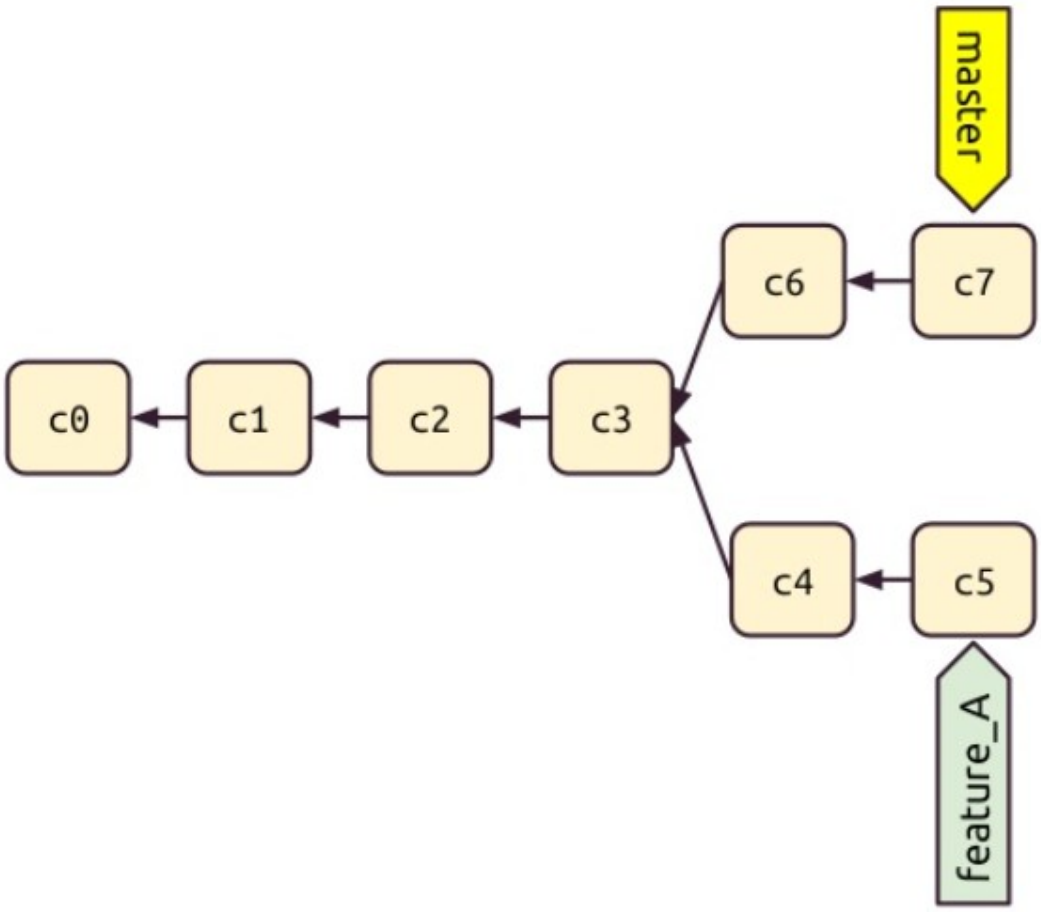


Checout branche Master

```
1 git checkout master
```

Gestion en configuration sans conflit : la branche principale est modifiée

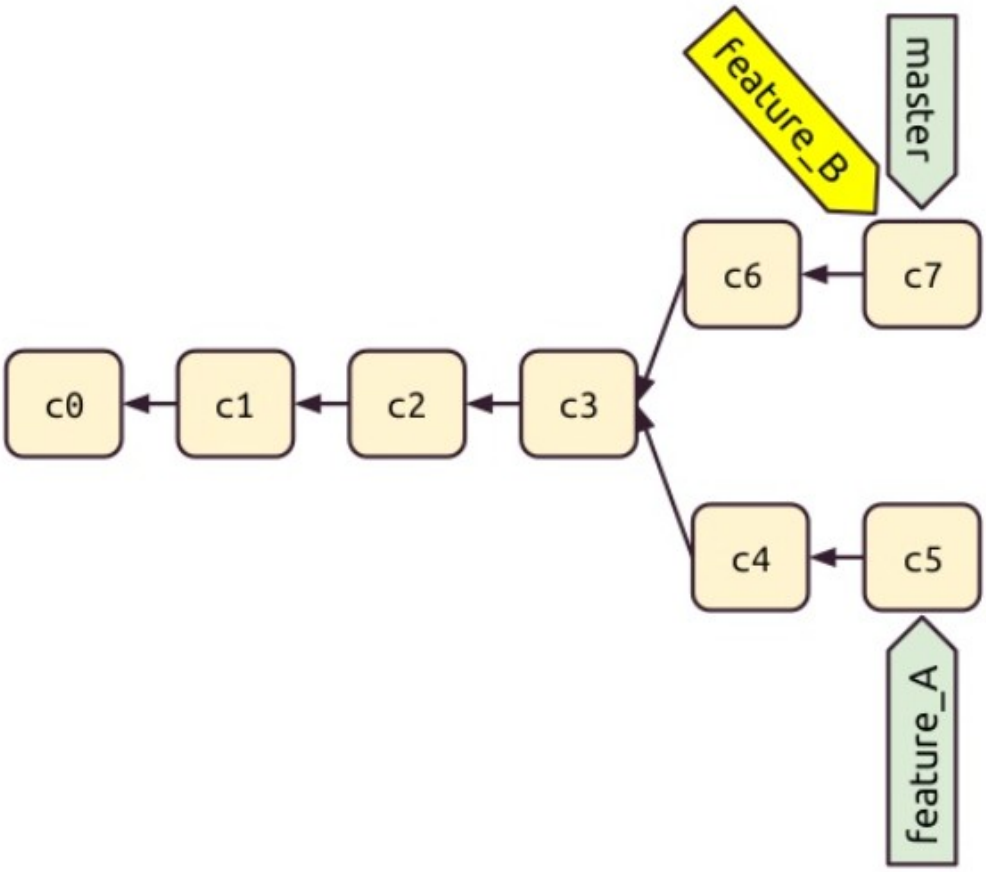
La branche master est modifiée (suite à des *commits* en local ou sur le serveur...)



Dev branche 'Master'

Gestion en configuration sans conflit : la branche principale est modifiée pour la fonction B

Par précaution je crée une branche pour marque le début de mes modifications sur la branche master : *feature_B*

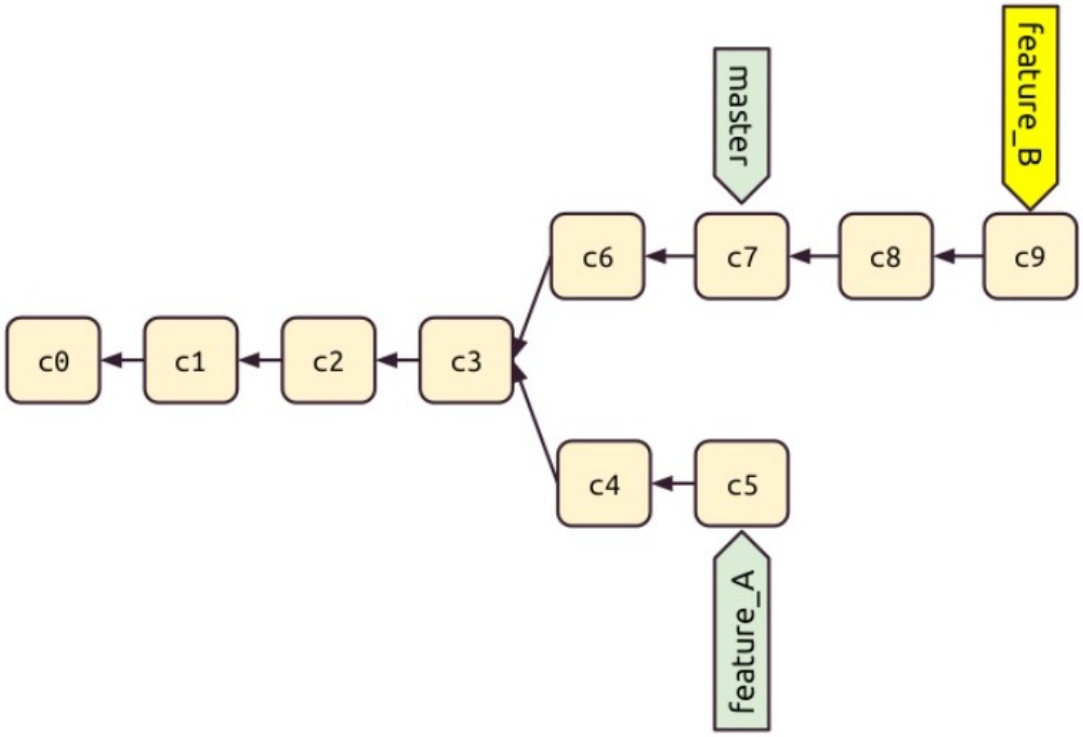


Créer la branche fonctionnalité B

```
1 git checkout feature_B
```

Gestion en configuration sans conflit : développement de la fonction B

Les modifications sont enregistrée sur la branche de la fonction B : *feature_B*



Dev fonctionnalité B

Fondamental : Comment réorganiser les versions du projet

3 branches :

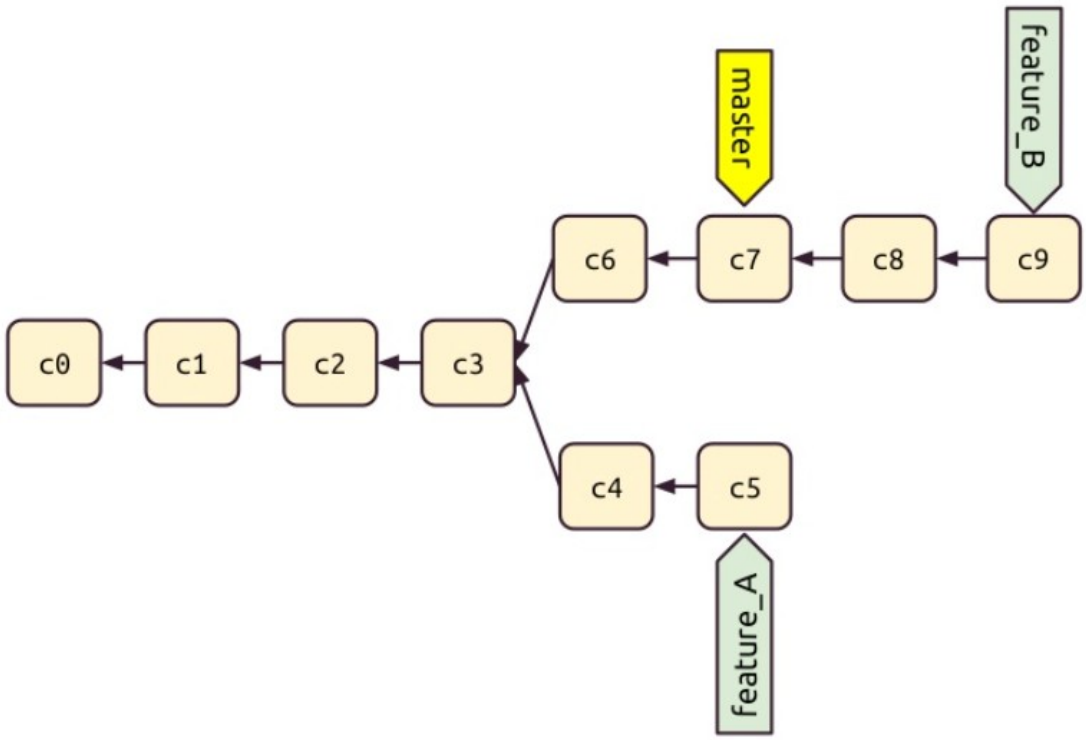
- **master**
- **feature_A**
- **feature_B**

Il est temps de livrer !

… reverser *feature_A* dans la branche principale, puis *feature_B*

Gestion en configuration sans conflit : reverser les modifications dans la branche principale 1/4

Se placer sur la branche cible

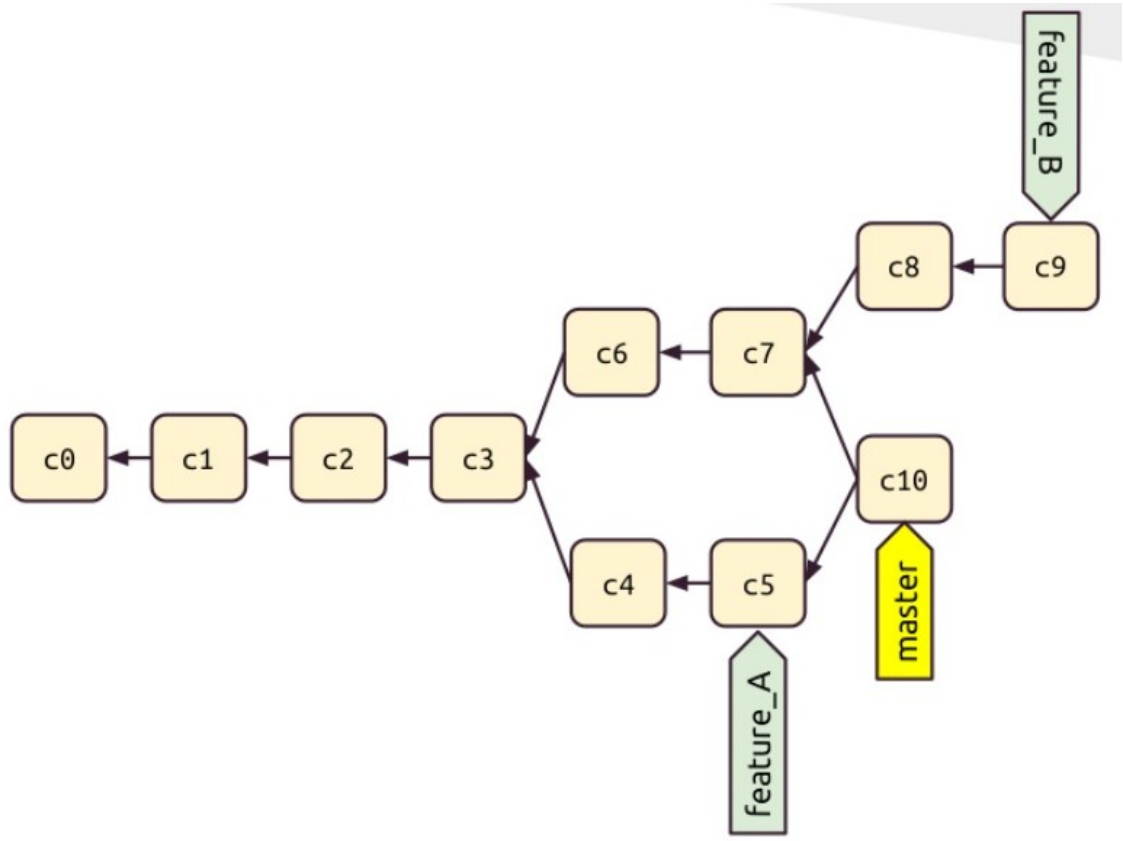


Checkout branche 'Master'

```
1 git chekcout master
```

Gestion en configuration sans conflit : reverser les modifications dans la branche principale 2/4

Fusionner la branche *feature_A* avec la branche *master*



Dev branche 'Master'

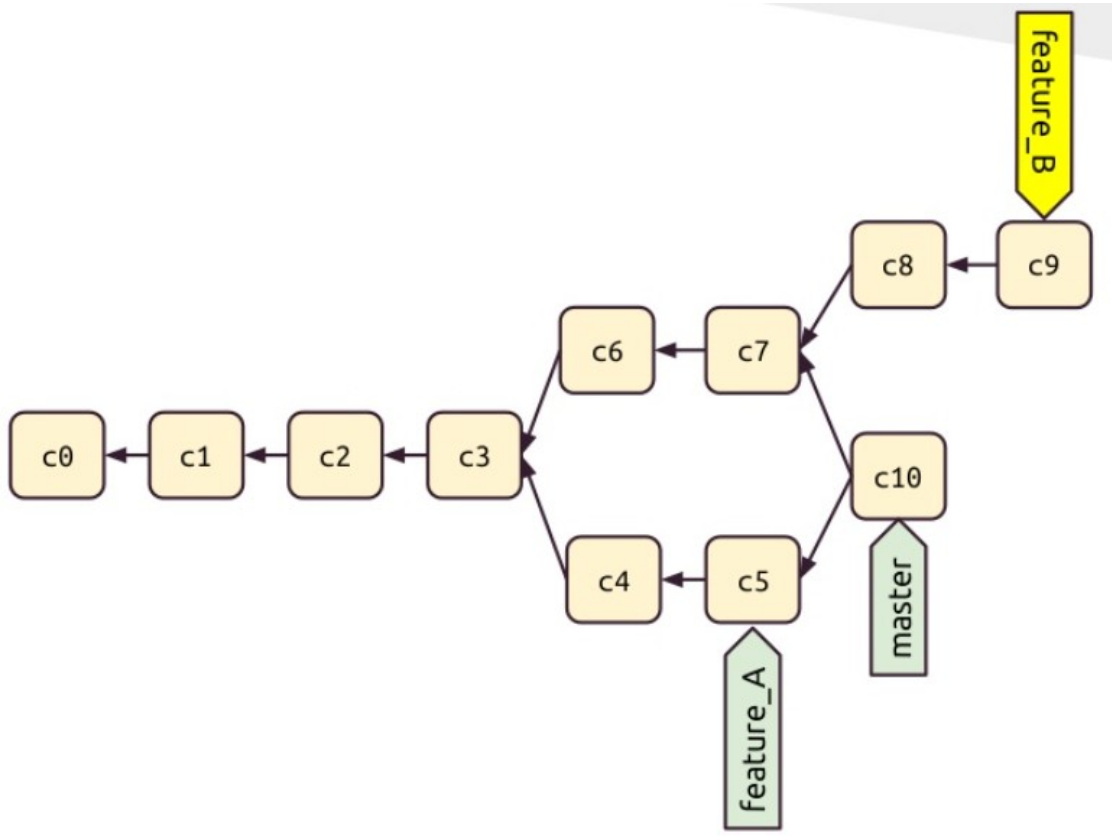
```
1 git merge feature_A
```

Gestion en configuration sans conflit : reverser les modifications dans la branche principale 3/4

J'aimerais conserver dans ma branche principale les révisions effectuées pour la branche de la fonction B : *feature_B*

Il me faut donc fusionner la branche feature_B avec la branche master mais en conservant les révisions : rebase...

Je me positionne sur la branche B

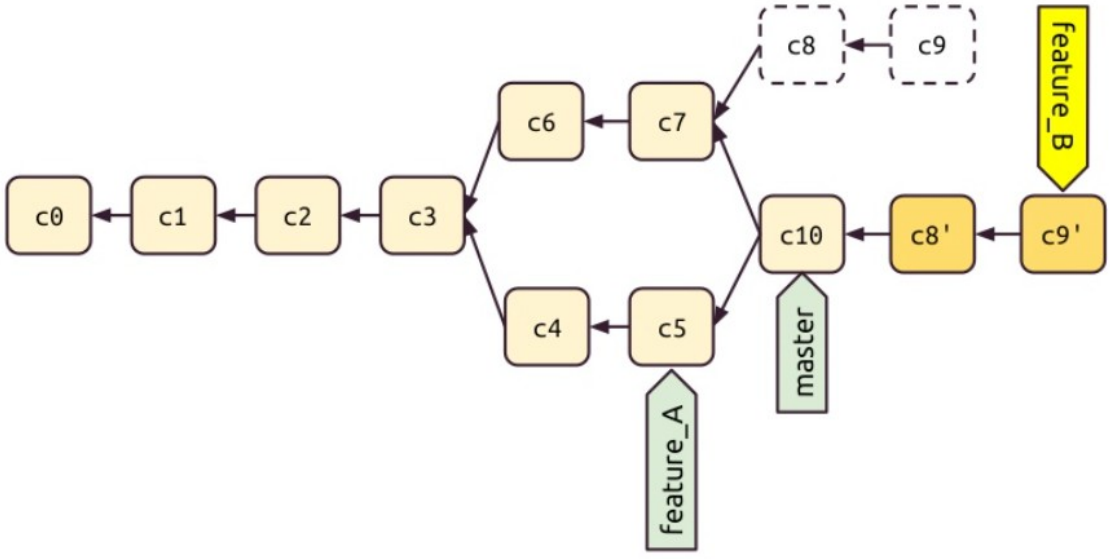


Checkout branche fonctionnalité B

```
1 git checkout feature_B
```

Gestion en configuration sans conflit : reverser les modifications dans la branche principale 4/4

Mes révisions de la branche feature_B vont être ajoutées à la branche master...



Rebase de la branche fonctionnalité B sur la branche 'Master'

```
1 git rebase master
```

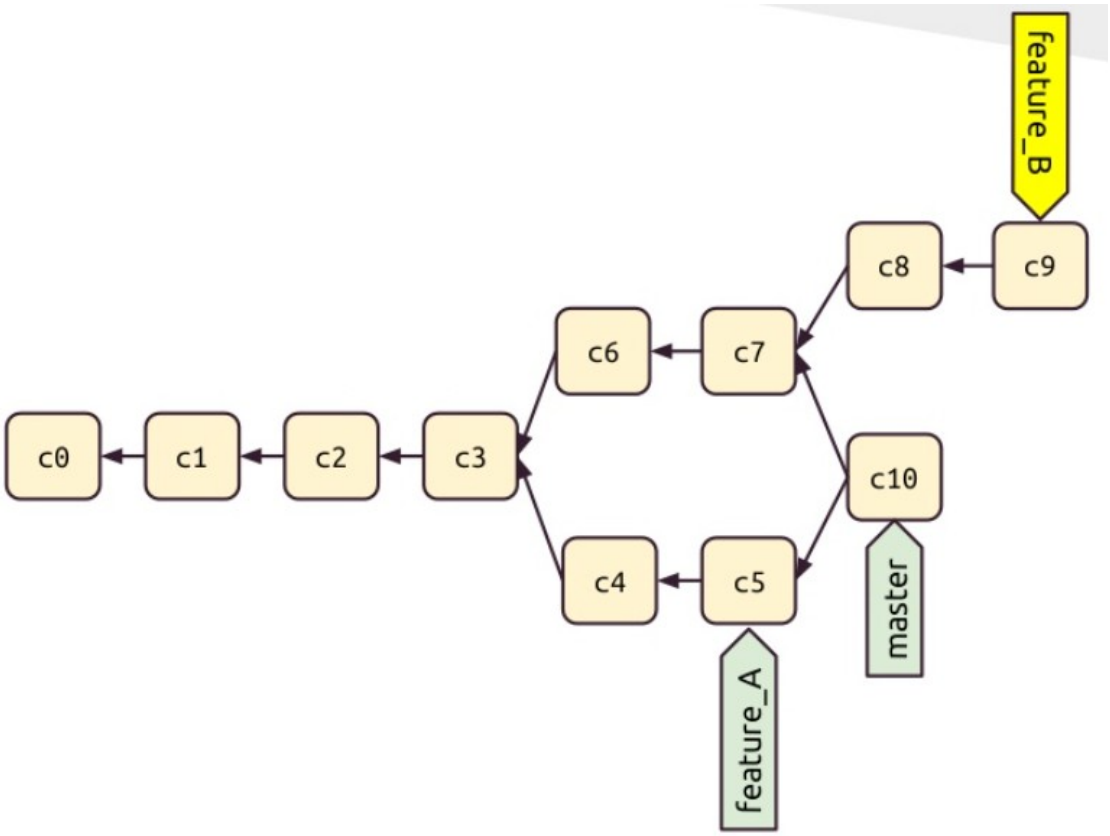
Fondamental : Git : entre rebase et merge

Git permet de faire entre les branches :

merge : les 2 branches fusionnent, une révision spécifique au merge est créée

rebase : la 2ème branche est fusionnée avec la 1er en collant les révisions à la suite (pas de nouvelle révision)

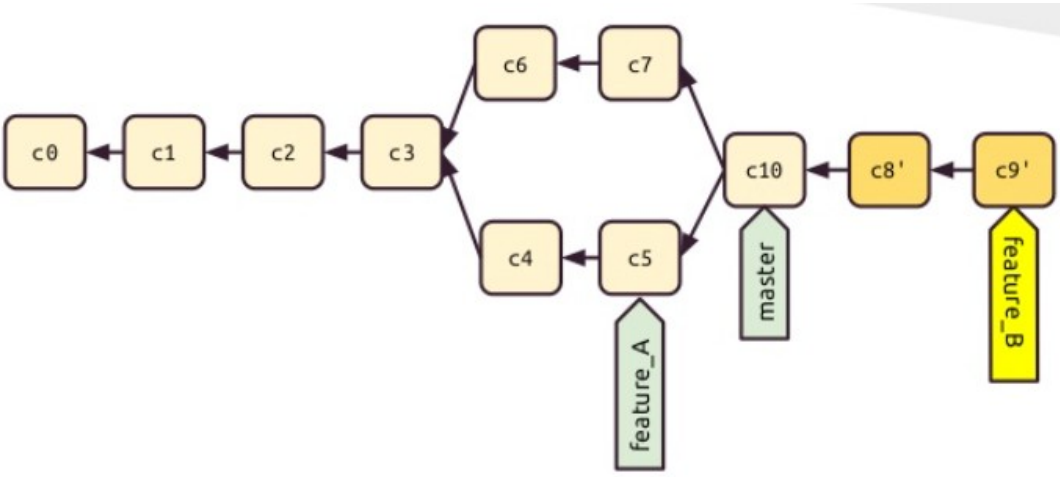
Soit, à partir des révisions du projet ci-dessous



Checkout branche fonctionnalité B

Rebase d'une branche

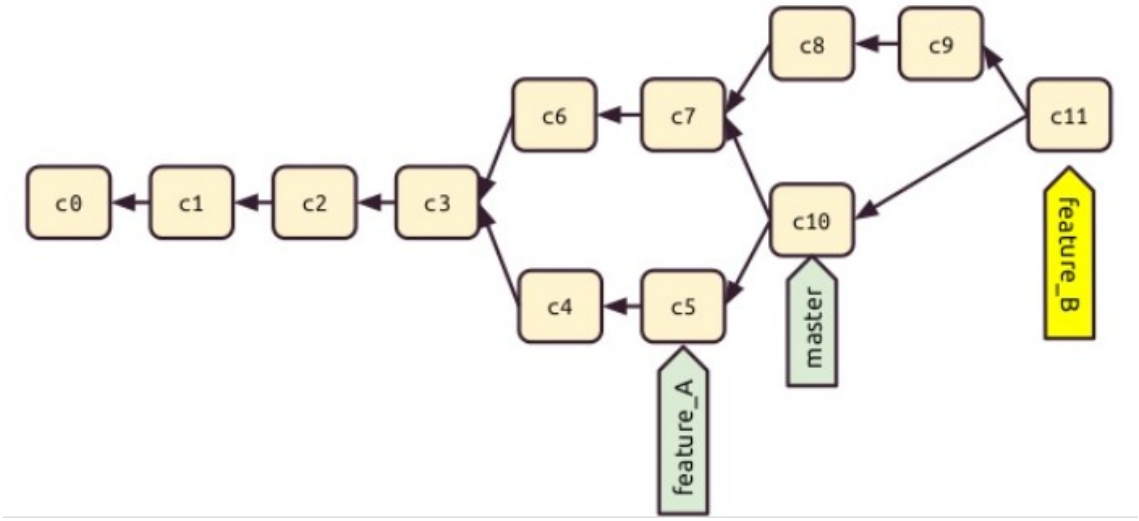
On conserve les mêmes révisions mais sur la branche de fusion



Rebase

Merge des 2 branches

On crée une nouvelle révision résultat de la fusion des 2 branches



Merge

Fondamental : Rebase vs Merge

Le mécanisme de rebase rend plus lisible et maintenable l'arbre des versions du projet

Complément

Pour s'entraîner avec git :

- http://pcottle.github.io/learnGitBranching/
- *https://try.github.io/levels/1/challenges/1*[1]

E. Tutoriels Git

Tutoriels Git

07-EX10_TutorielsGit : Les meilleurs tutoriels Git

1 - https://try.github.io/levels/1/challenges/12