Introduction

Système de gestion de version

Système logiciel permettant de maintenir et gérer toutes les versions d'un ensemble de fichiers.

Pourquoi un système de gestion de version ?

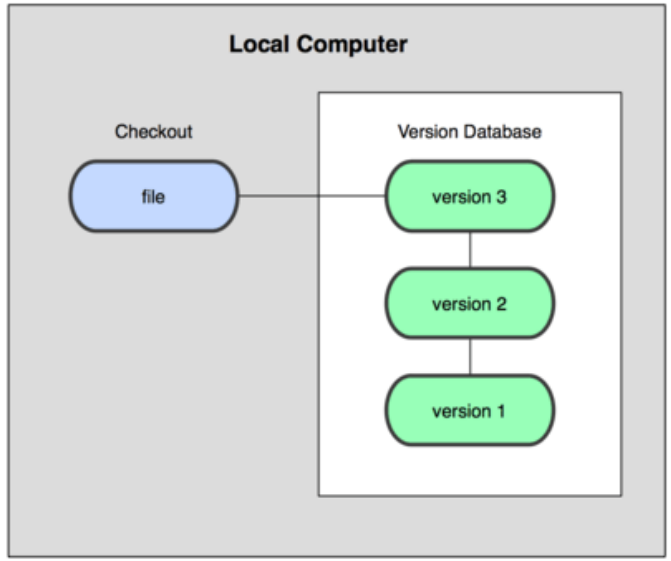
- Revenir aisément à une version précédente.
- Suivre l'évolution du projet au cours du temps.
- Permettre le travail en parallèle sur des parties disjointes du projet et gérer les modifications concurrentes.
- Faciliter la détection et la correction d'erreurs.
- ...

Ouvrage de référence

`http://git-scm.com/book`

Différents types de systèmes de gestion de version

Système de gestion de version local



Avantages

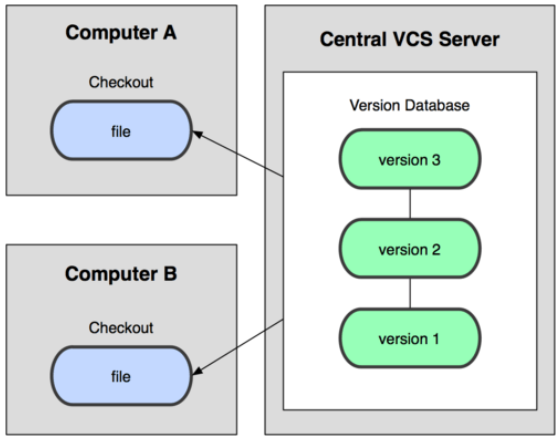
- Gestion et utilisation très simples.

Inconvénients

- Est très sensible aux pannes.
- Ne permet pas la collaboration.

Différents types de systèmes de gestion de version

Système de gestion de version centralisé



Avantages

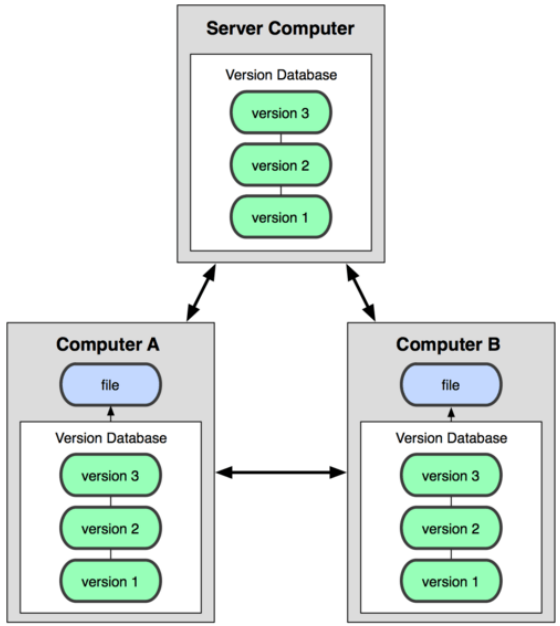
- Structurellement simple.
- Gestion et utilisation simples.

Inconvénients

- Est très sensible aux pannes.
- Inadapté aux très grands projets et/ou avec une forte structure hiérarchique.

Différents types de systèmes de gestion de version

Système de gestion de version distribué



Avantages

- Moins sensible aux pannes.
- Adapté aux très grands projets et/ou avec une forte structure hiérarchique.

Inconvénients

- Gestion et utilisation plus compliquées.
- Peut devenir très complexe structurellement.

Git

Carte d'identité

Système de gestion de version distribué (DVCS).

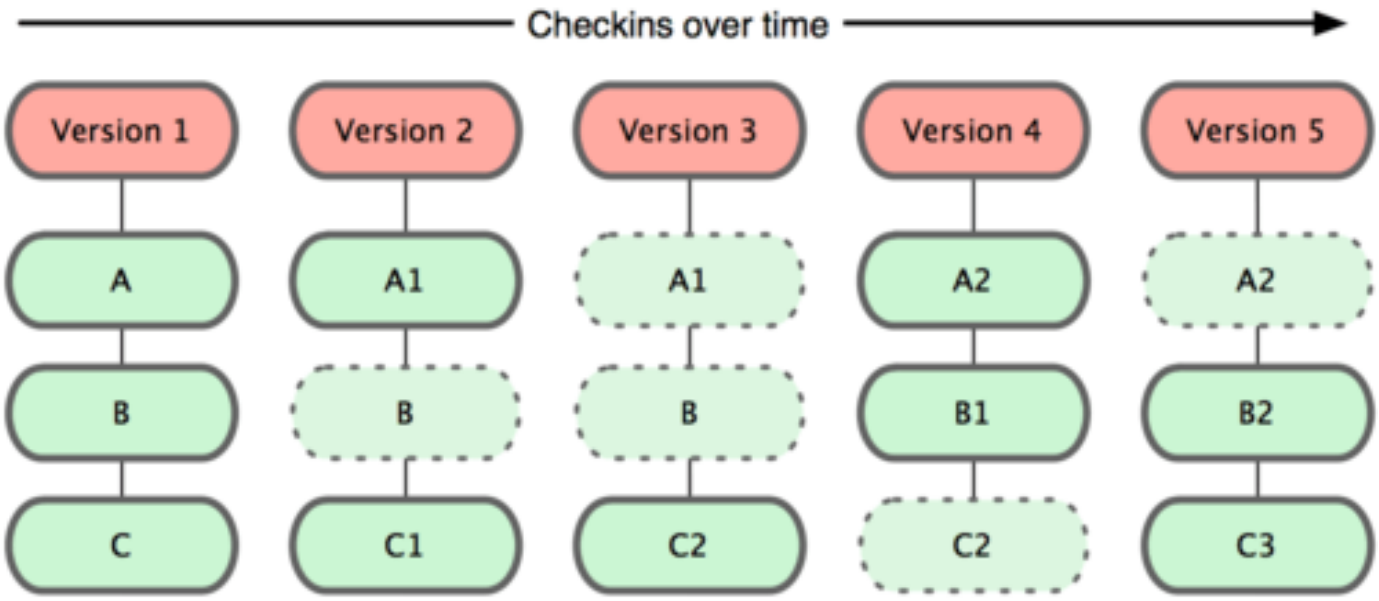
Bref historique (Wikipedia)

- De 1991 à 2002, le noyau Linux était développé sans utiliser de système de gestion de version.
- A partir de 2002, la communauté a commencé à utiliser BitKeeper, un DVCS propriétaire.
- En 2005, suite à un contentieux, BitKeeper retire la possibilité d'utiliser gratuitement son produit. Linus Torvalds lance le développement de Git et après seulement quelques mois de développement, Git héberge le développement du noyau Linux.

Git

Les principes de base

Un dépôt Git est une sorte de système de fichiers (base de données), enregistrant les versions de fichiers d'un projet à des moments précis au cours du temps sous forme d'instantanés.



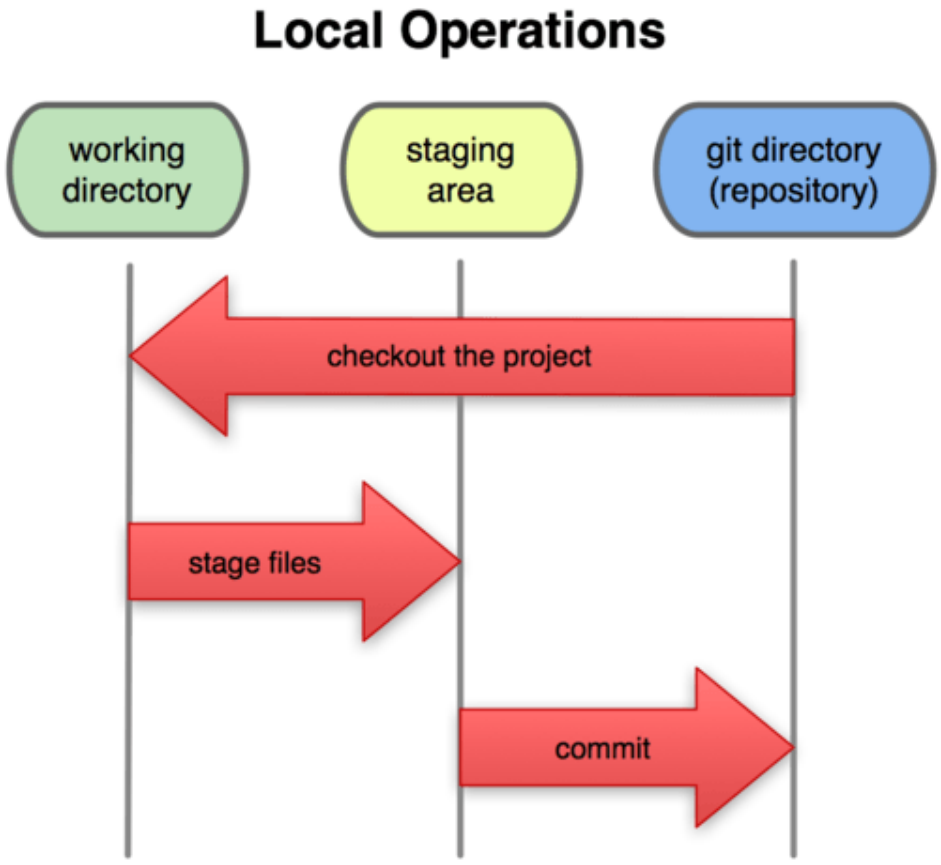
Comment fonctionne le processus de versionnement ?

Git

Les principes de base

3 sections d'un projet Git :

- Le répertoire Git/dépôt : contient les méta-données et la base de données des objets du projet.
- Le répertoire de travail : extraction unique d'une version du projet depuis la base de données du dépôt.
- La zone de transit/d'index : simple fichier contenant des informations à propos de ce qui sera pris en compte lors de la prochaine soumission.

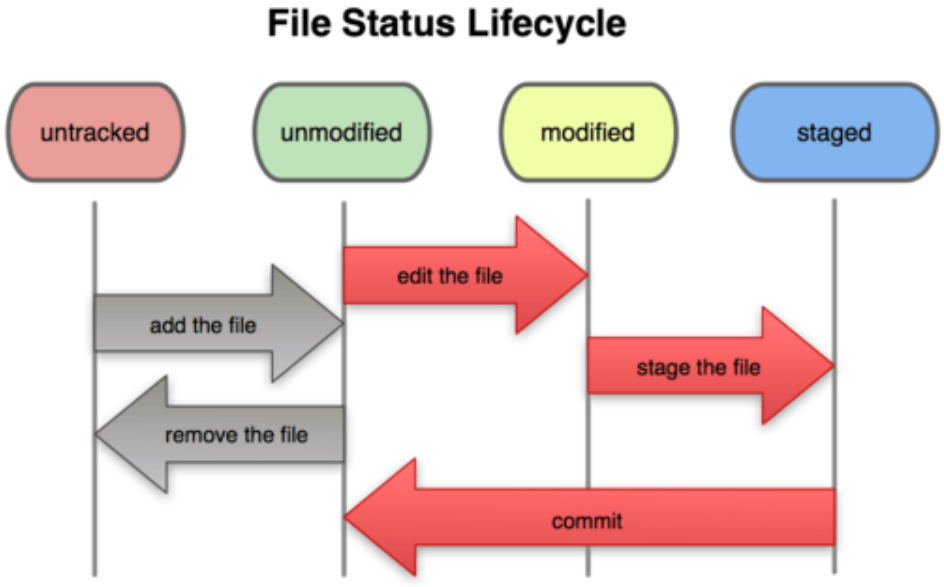


Git

Les principes de base

4 états d'un fichier dans Git :

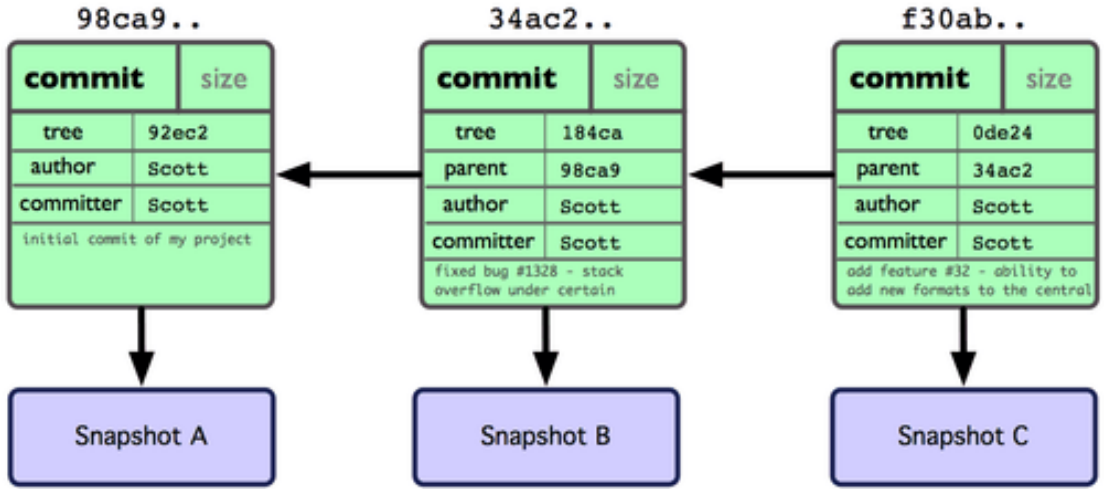
- non versionné : fichier n'étant pas ou plus géré par Git ;
- non modifié : fichier sauvegardé de manière sûre dans sa version courante dans la base de données du dépôt ;
- modifié : fichier ayant subi des modifications depuis la dernière fois qu'il a été soumis ;
- indexé (staged) : idem, sauf qu'il en sera pris un instantané dans sa version courante lors de la prochaine soumission (commit).



Git

Les principes de base

Chaque soumission (commit) donne lieu à la création d'un objet "commit" contenant un pointeur vers un instantané du contenu dont les modifications étaient indexées au moment de la soumission (ensemble d'objets permettant de stocker un instantané des fichiers concernés d'une part et de reproduire la structure du répertoire projet et de ses sous-répertoires d'autre part), quelques méta-données (auteur, message) et un pointeur vers l'objet "commit" précédent.



Initialiser un dépôt

```
$ git init
```

Afficher l'état des fichiers du répertoire courant

```
$ git status
```

- *Untracked files* : fichiers non versionnés.
- *Changes to be committed* : modifications (ajout, suppression, changements) chargées en zone de transit (staging area), ou indexées.
- *Changes not staged for commit* : modifications n'ayant pas été chargées en zone de transit (ou indexées).

Git

Les commandes de base

Indexer l'ajout ou les changements d'un fichier

```
$ git add [-p] <fichier>
```

Annuler les modifications indexées d'un fichier

```
$ git reset <fichier>
```

Annuler les modifications non encore indexées d'un fichier

```
$ git checkout [--] <fichier>
```

Indexer la suppression d'un fichier

```
$ git rm <fichier>
```

Déversionner un fichier

```
$ git rm --cached <fichier>
```

Afficher le détail des modifications non indexées

```
$ git diff
```

Afficher le détail des modifications indexées

```
$ git diff --staged
```

Soumettre les modifications indexées en zone de transit

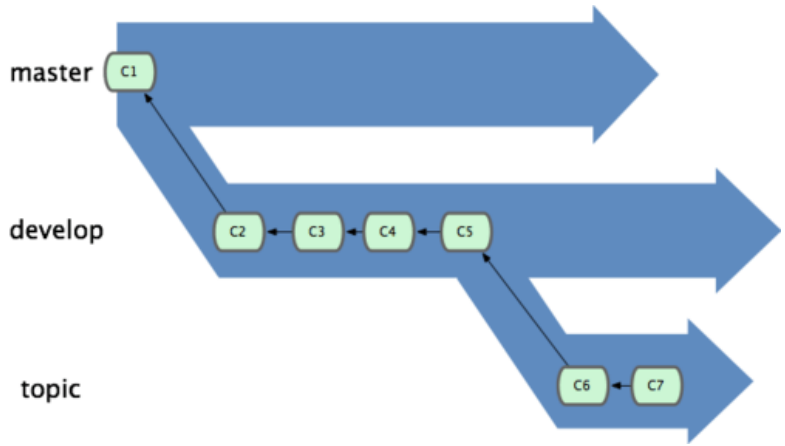
```
$ git commit
```

Voir l'historique des soumissions

```
$ git log
```

Qu'est-ce qu'une branche dans un projet ?

C'est une ligne d'évolution divergent de la ligne d'évolution courante, celles-ci se poursuivant indépendemment l'une de l'autre.



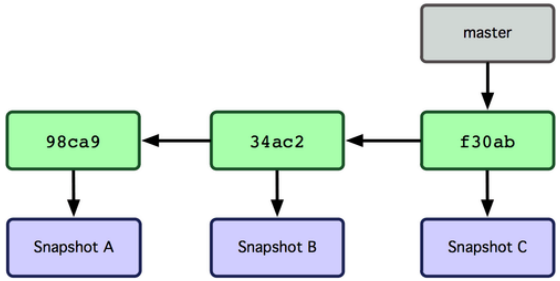
Pourquoi des branches ?

- Pouvoir se lancer dans des évolutions ambitieuses en ayant toujours la capacité de revenir à une version stable que l'on peut continuer à maintenir indépendemment.
- Pouvoir tester différentes implémentations d'une même fonctionnalité de manière indépendante.

Git

Branches

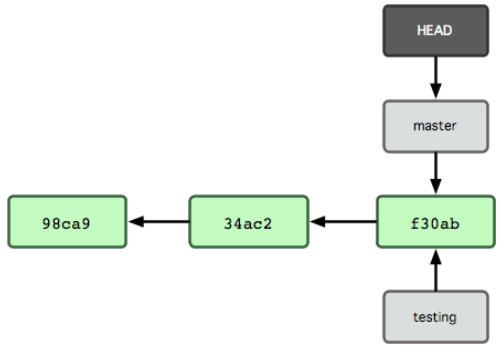
Une branche dans Git est tout simplement un pointeur vers un objet "commit". Par défaut, il en existe une seule, nommée "master".



Créer une nouvelle branche

```
$ git branch <branche>
```

HEAD est un pointeur spécial vers la branche sur laquelle on travaille actuellement (extraite dans le répertoire de travail).



Git

Branches

Voir les branches du dépôt local

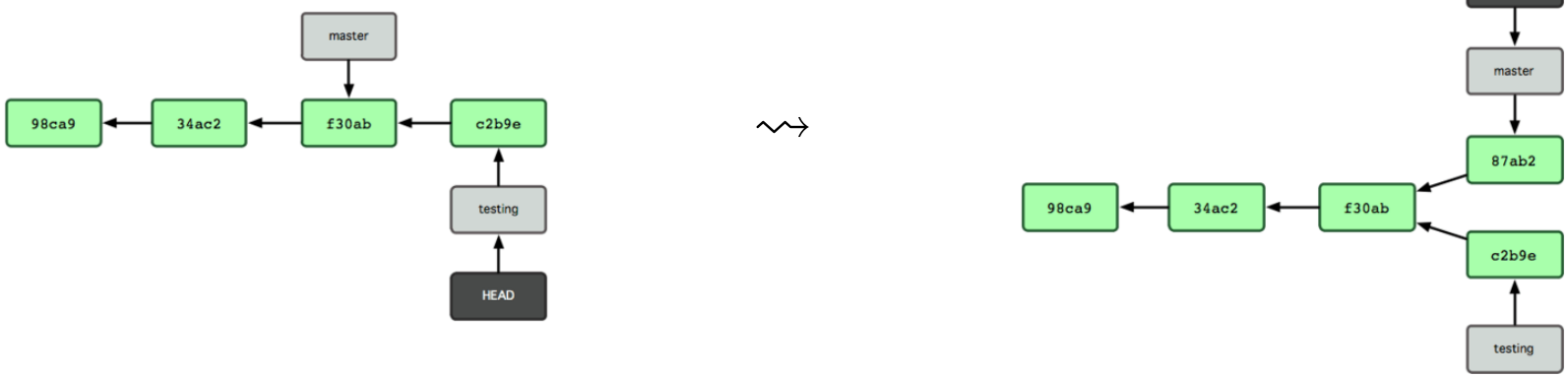
```
$ git branch
```

Supprimer une branche

```
$ git branch -d <branche>
```

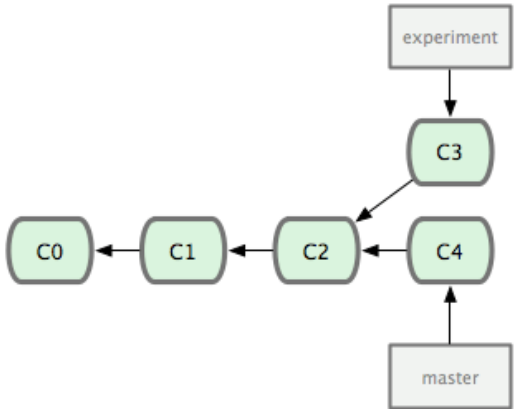
Passer à une branche donnée (mise à jour de l'index et du répertoire de travail, ainsi que du pointeur HEAD)

```
$ git checkout <branche>
```



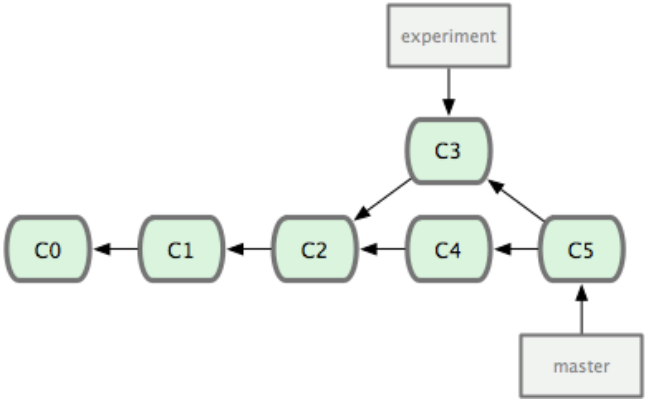
Git

Branches

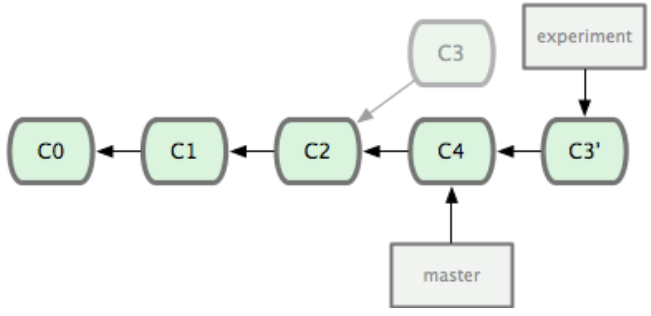


Deux façons pour incorporer les modifications d'une branche dans la branche courante.

Fusionner (merge)



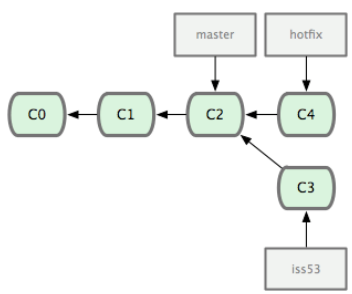
Rebaser (rebase)

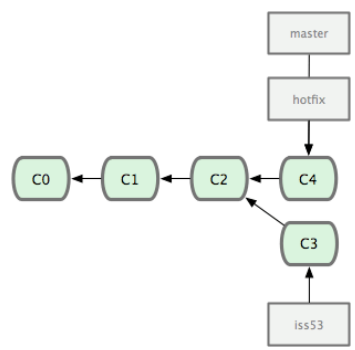


Fusionner les modifications d'une branche donnée dans la branche courante (HEAD)

```
$ git merge <branche>
```

- Si l'objet "commit" pointé par `<branche>` est déjà un ancêtre de l'objet "commit" courant (HEAD), alors rien n'est fait.
- Si l'objet "commit" pointé par `<branche>` est un descendant de l'objet "commit" courant, seul le pointeur de la branche courante est déplacé sur l'objet "commit" concerné par la fusion ("fast-forward").

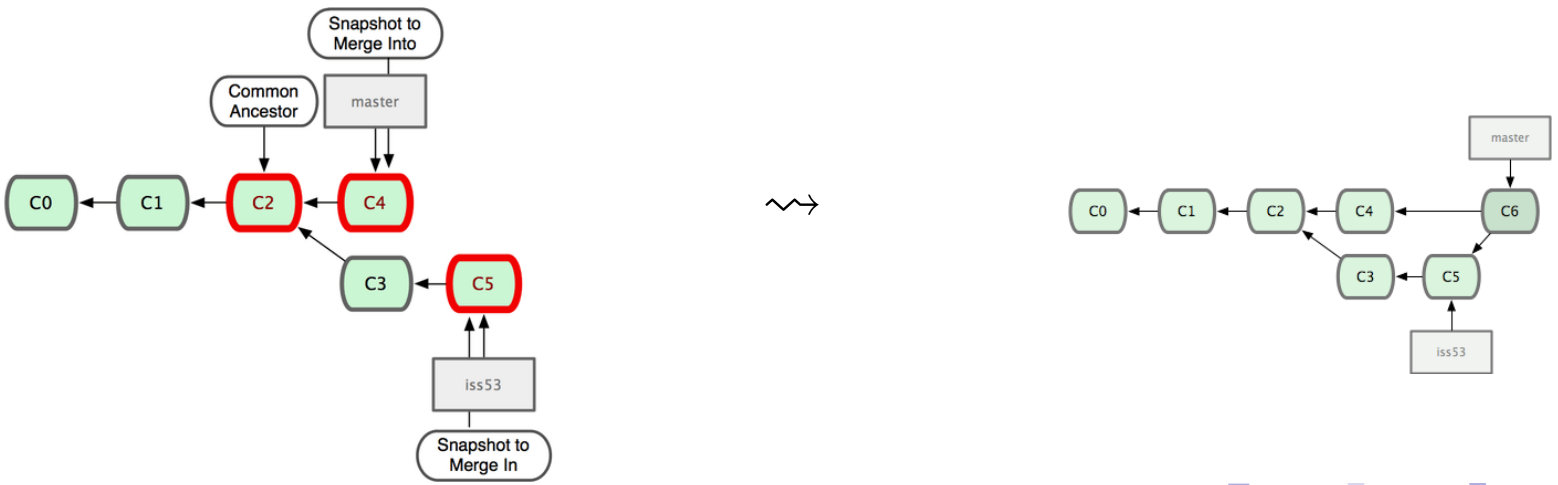




Git

Branches

- Autrement, Git se base sur trois instantanés différents, celui de l'objet “commit” courant, celui de l'objet “commit” pointé par `<branche>` et celui de l'objet “commit” correspondant au plus jeune ancêtre commun des deux premiers objets “commit”, pour créer un nouvel instantané associé à un nouvel objet “commit” par la fusion de deux branches, sur lequel la branche courante pointera à présent.



En cas de conflit empêchant la fusion

- Aucun objet “commit” de fusion n’est créé, mais le processus est mis en pause.
- `git status` donne les fichiers n’ayant pas pu être fusionnés (listés en tant que “unmerged”).
- Git ajoute des marqueurs de résolution de conflits à tout fichier sujet à conflits afin que ceux-ci puissent être résolus à la main.
- Pour marquer les conflits dans un fichier `<fichier>` comme résolus, il faut faire `git add <fichier>`. On peut, après résolution de tous les conflits, soumettre les modifications sous forme d’objet “commit” de fusion avec `git commit` et terminer ainsi le processus de fusion.

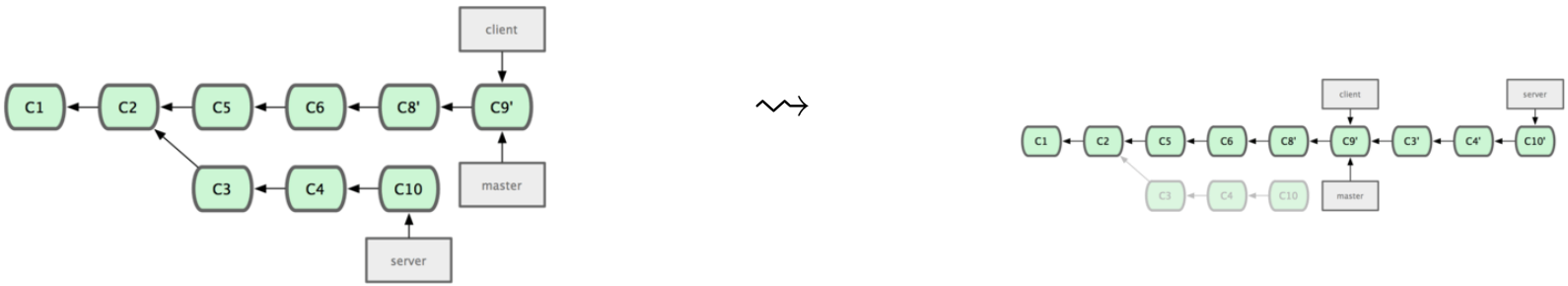
Git

Branches

Rebaser les modifications de la branche courante (HEAD) sur une branche donnée

```
$ git rebase <branche>
```

- Aller à l'objet "commit" correspondant au plus jeune ancêtre commun des deux objets "commit" pointés par la branche courante et `<branche>`.
- Obtenir et sauvegarder les changements introduits depuis ce point par chaque objet "commit" de la branche courante.
- Faire pointer la branche courante sur le même objet "commit" que `<branche>` et réappliquer tous les changements un par un.



Git

Travail avec des dépôts distants

Pour collaborer, il est nécessaire de communiquer et d'échanger avec un ou plusieurs dépôts distants hébergeant le même projet (typiquement des dépôts publics associés à une personne, une équipe ou tout le projet).

Les données des dépôts distants (objets "commit" et instantanés) sont entièrement copiées dans le dépôt local, et pour chaque branche `<branche>` d'un dépôt distant `<dépôt>` est maintenue une branche locale `<dépôt>/<branche>` non modifiable, permettant de suivre la position de `<branche>` sur `<dépôt>` localement.

Afficher la liste de tous les dépôts distants du projet

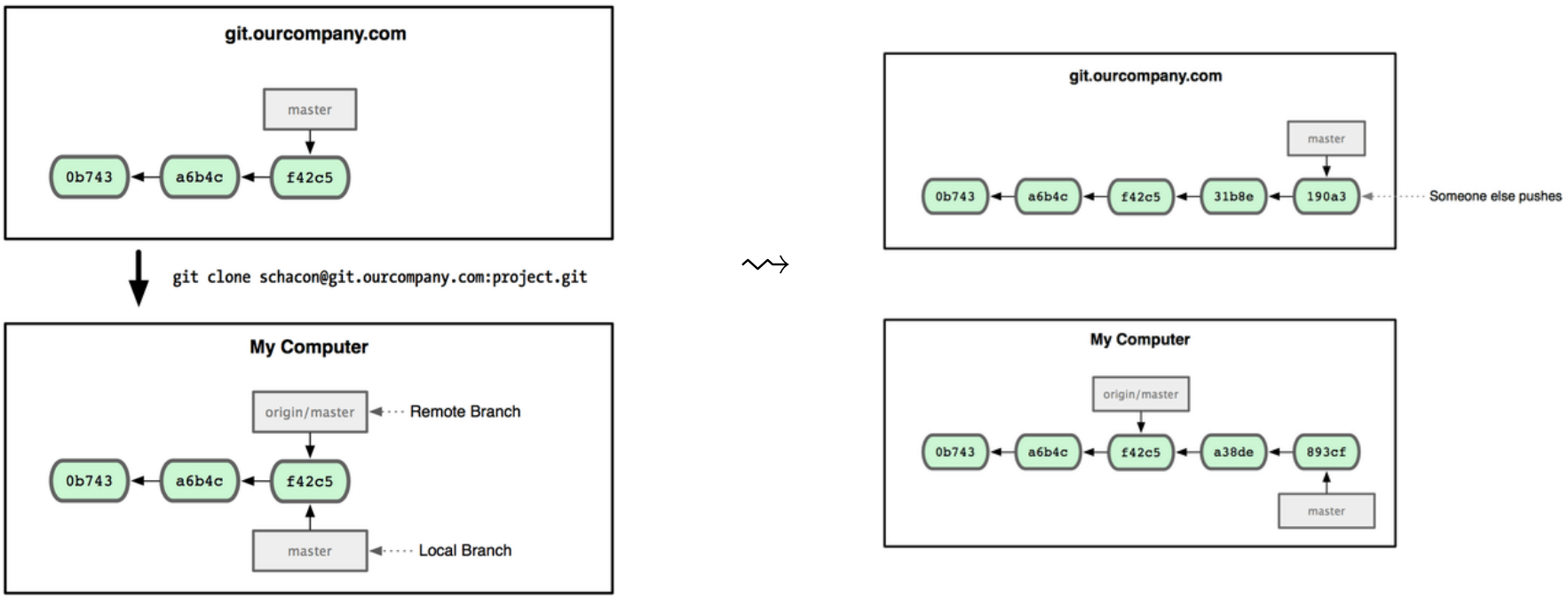
```
$ git remote
```

Git

Travail avec des dépôts distants

Cloner un dépôt distant (automatiquement nommé origin)

```
$ git clone <URL> [<répertoire>]
```

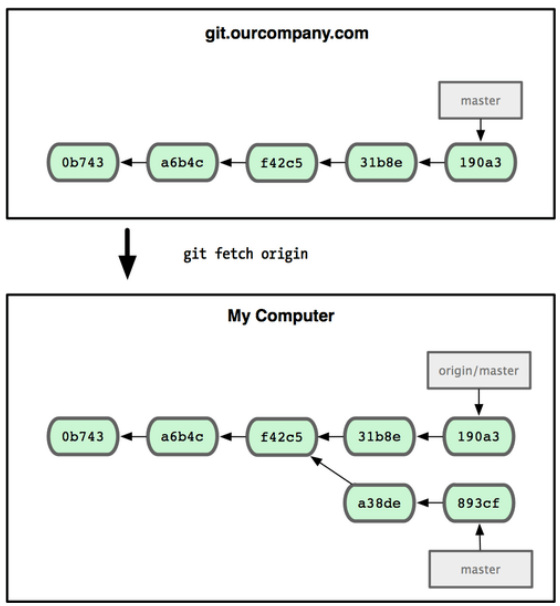


Git

Travail avec des dépôts distants

Récupérer les modifications d'un dépôt distant

```
$ git fetch <dépôt>
```

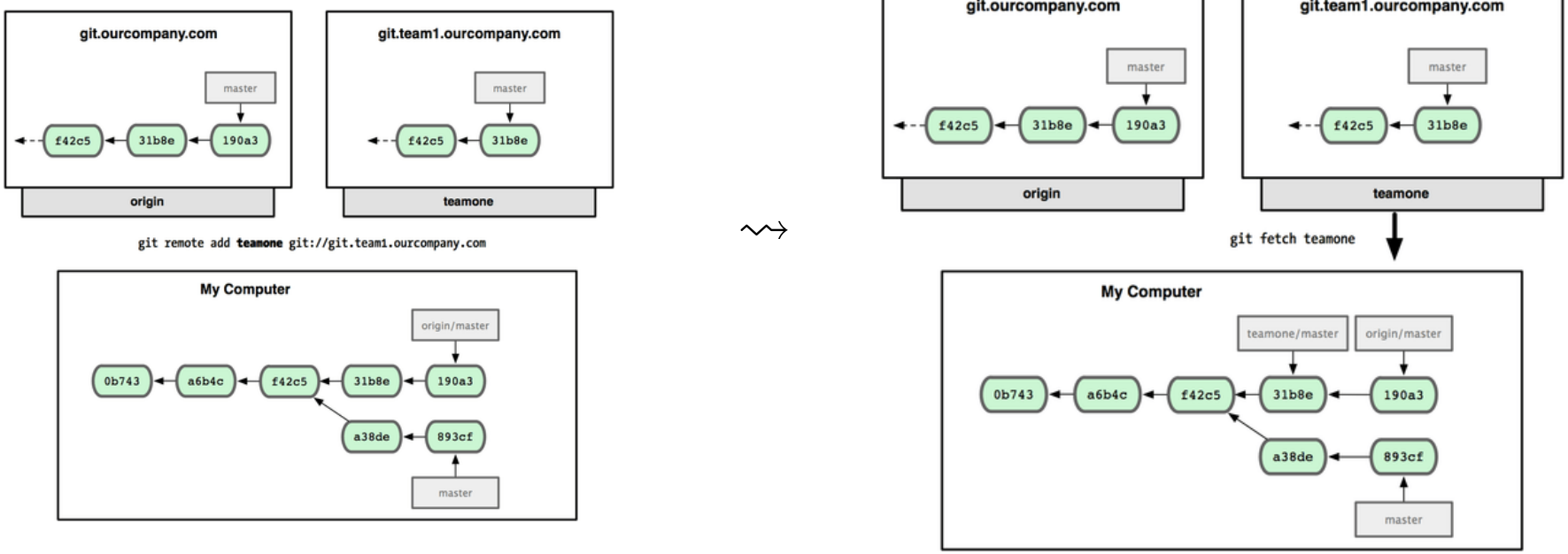


Git

Travail avec des dépôts distants

Ajouter un dépôt distant

```
$ git remote add <nom> <URL>
```



Mettre à jour un dépôt distant donné pour une certaine branche locale

```
$ git push <dépôt> <branche>
```

Attention : ceci fonctionne uniquement si le dernier objet “commit” de la branche concernée du dépôt distant a été récupéré et intégré dans la branche en question du dépôt local, en d'autres termes si la mise à jour de la branche du dépôt distant peut être faite par “fast-forward”.

Combiner `git fetch` et `git merge`

```
git pull <dépôt> <branche>
```

Combiner `git fetch` et `git rebase`

```
git pull --rebase <dépôt> <branche>
```

Conseils

- User et abuser des branches.
- Eviter de systématiquement faire un `git pull`, mais réfléchir à ce qu'il est le plus pertinent de faire (fusionner ou rebaser).

Bonnes pratiques

- Ne jamais rebaser une branche déjà présente sur un dépôt public.
- Respecter les conventions de formattage du message de soumission : titre de 50 caractères au plus, suivi d'une ligne vide puis d'une description détaillée avec des lignes de 72 caractères au plus, le tout au présent de l'impératif (`http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html`).

GitHub

- Service web d'hébergement et de gestion de projets de développement logiciel utilisant le système de gestion de version Git.
- En plus de ce système, on trouvera tout un éco-système incluant notamment un Wiki et un système de tickets (avec MarkDown et référencement des tickets).
- `https://help.github.com/`

Fin

Des questions ?