

Formation GIT

Partie 2

Novembre 12, 2019

Objectifs

- Gérer les branches (créer, basculer, modifier, supprimer, fusionner).
- Gérer les dépôts distants avec les branches locales/distantes (push, pull, etc.)
- Gérer les conflits entre les différentes branches locales et distantes.
- Gérer les accès aux dépôts pour différents utilisateurs.

Plan global

Formation GIT

Partie II

1. Les branches avec Git (dépôt local et distant).
2. Atelier GITHUB / BitBucket

1. Les branches avec GIT

Plan

1. Introduction aux branches
2. Branches et fusions : les bases
3. Gestion des branches
4. Travailler avec les branches
5. Branches de suivi à distance
6. Rebaser (Rebasing)

1. Les branches avec git

1.1. Introduction aux branches

Introduction

Un commit ⇒

- + Git stocke un objet commit qui contient un **pointeur vers l'instantané** (snapshot) du contenu indexé.
- + Cet objet contient également les **noms et prénoms de l'auteur**, le **message renseigné** ainsi que des **pointeurs vers le ou les commits qui précèdent directement ce commit** :
- + aucun parent pour le commit initial,
- + un parent pour un commit normal et
- + multiples parents pour un commit qui résulte de la fusion d'une ou plusieurs branches.

Exemple

1. Créer un dépôt projet_01.
2. Créer 3 fichiers dans ce dépôt
3. Initialiser le dépôt avec `$ git init`
4. Ajouter les 3 fichiers (README.md, test.rb, licence) à l'index
5. Faire un commit pour valider l'ajout de ces 3 fichiers

L'indexation des fichiers génère les actions suivantes :

- ⇒ Calculer une empreinte (checksum) pour chacun (via la fonction de hachage SHA-1),
- ⇒ Stocker cette version du fichier dans le dépôt Git (Git les nomme blobs)
- ⇒ Ajouter cette empreinte à la zone d'index (staging area)

```
$ git add README test.rb LICENSE
```

```
$ git commit -m 'initial commit of my project'
```

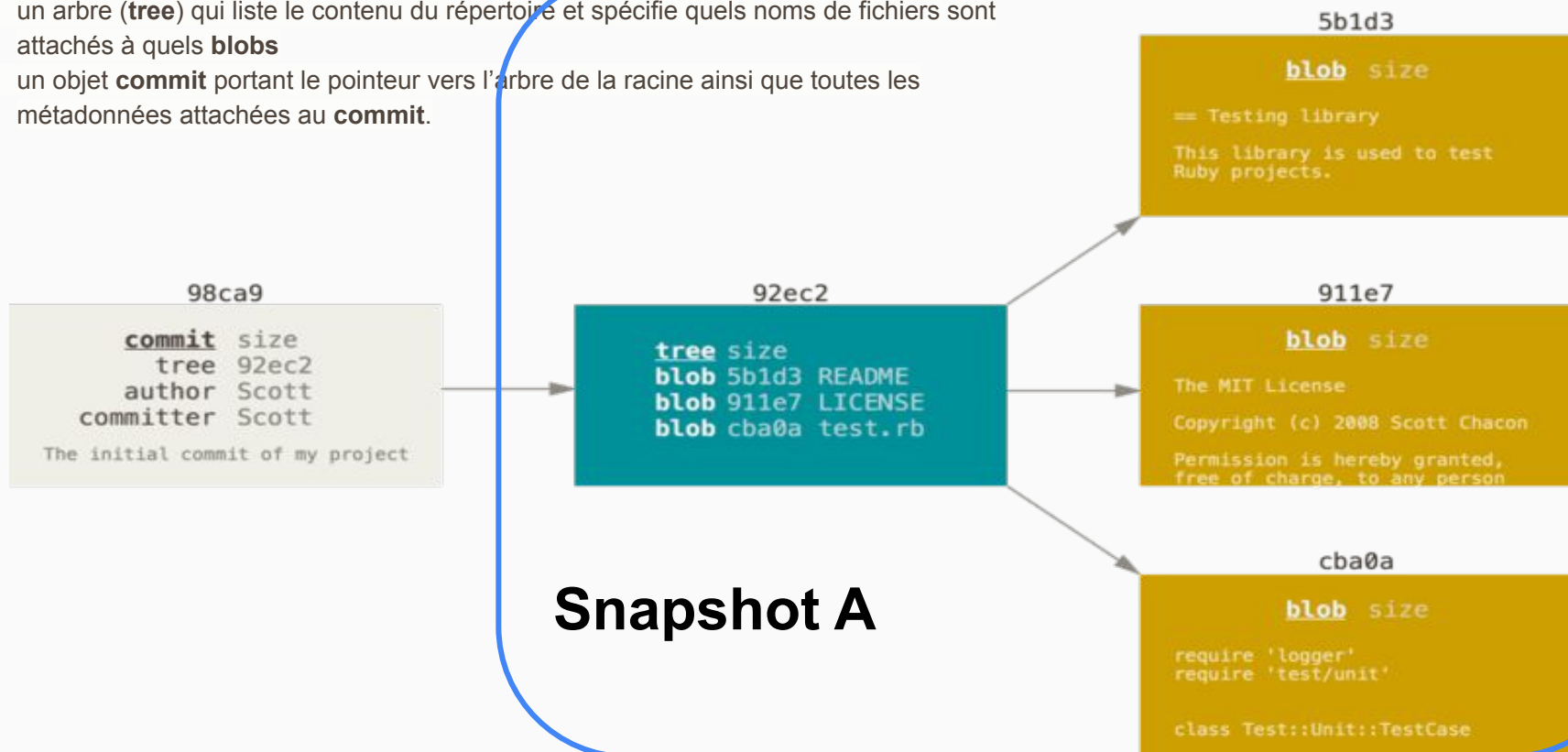
Git commit génère les actions suivantes :

- ⇒ Git calcule l'empreinte de chaque sous-répertoire (ici, seulement pour le répertoire racine) et stocke ces objets de type arbre dans le dépôt Git.
- ⇒ Git crée alors un objet commit qui contient les métadonnées et un pointeur vers l'arbre de la racine du projet de manière à pouvoir recréer l'instantané à tout moment.
- ⇒ Le dépôt Git contient à présent cinq objets :
 - un blob pour le contenu de chacun de vos trois fichiers,
 - un arbre (tree) qui liste le contenu du répertoire et spécifie quels noms de fichiers sont attachés à quels blobs,
 - un objet commit portant le pointeur vers l'arbre de la racine et les métadonnées attachées au commit.

Un commit et son arbre

Le dépôt Git contient à présent cinq objets :

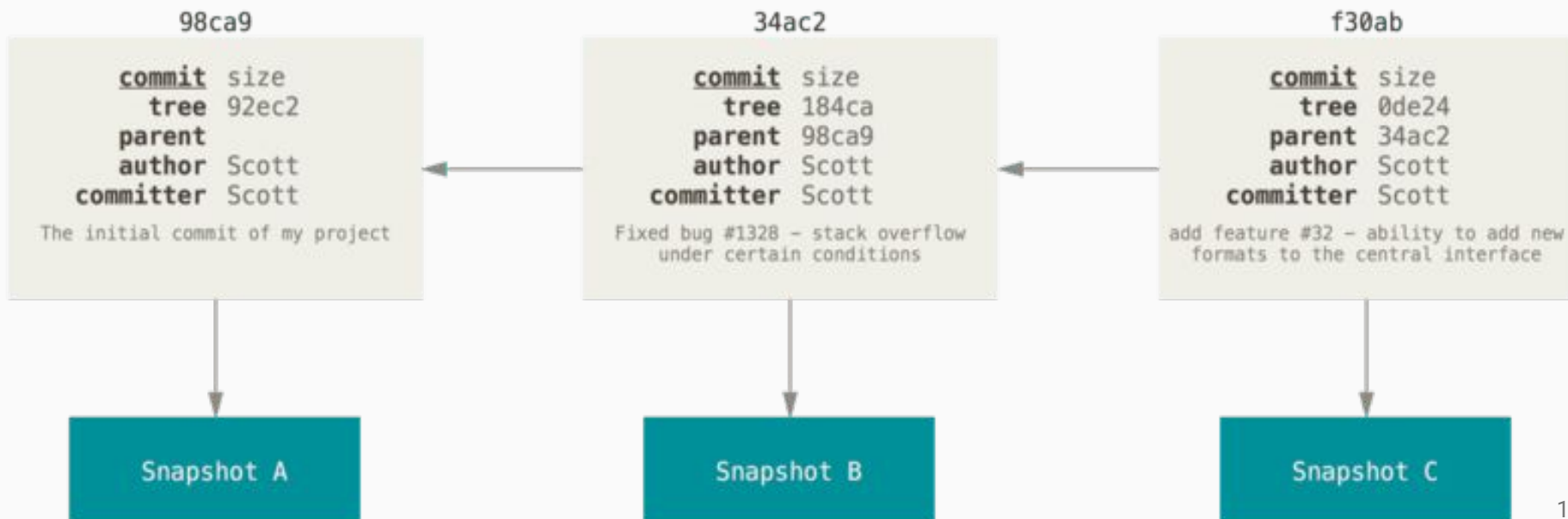
- un **blob** pour le contenu de chacun de vos trois fichiers,
- un arbre (**tree**) qui liste le contenu du répertoire et spécifie quels noms de fichiers sont attachés à quels **blobs**
- un objet **commit** portant le pointeur vers l'arbre de la racine ainsi que toutes les métadonnées attachées au **commit**.



Commit et leurs parents

Si on fait des modifications et validez à nouveau, le prochain commit stocke un pointeur vers le commit précédent.

⇒ **Une branche dans Git est simplement un pointeur léger et déplaçable vers un de ces commits.** La branche par défaut dans Git s'appelle master. Au fur et à mesure des validations, la branche master pointe vers le dernier des commits réalisés. À chaque validation, le pointeur de la branche master avance automatiquement.



Déplacer un commit

```
$ git reset --hard 34ac2
```

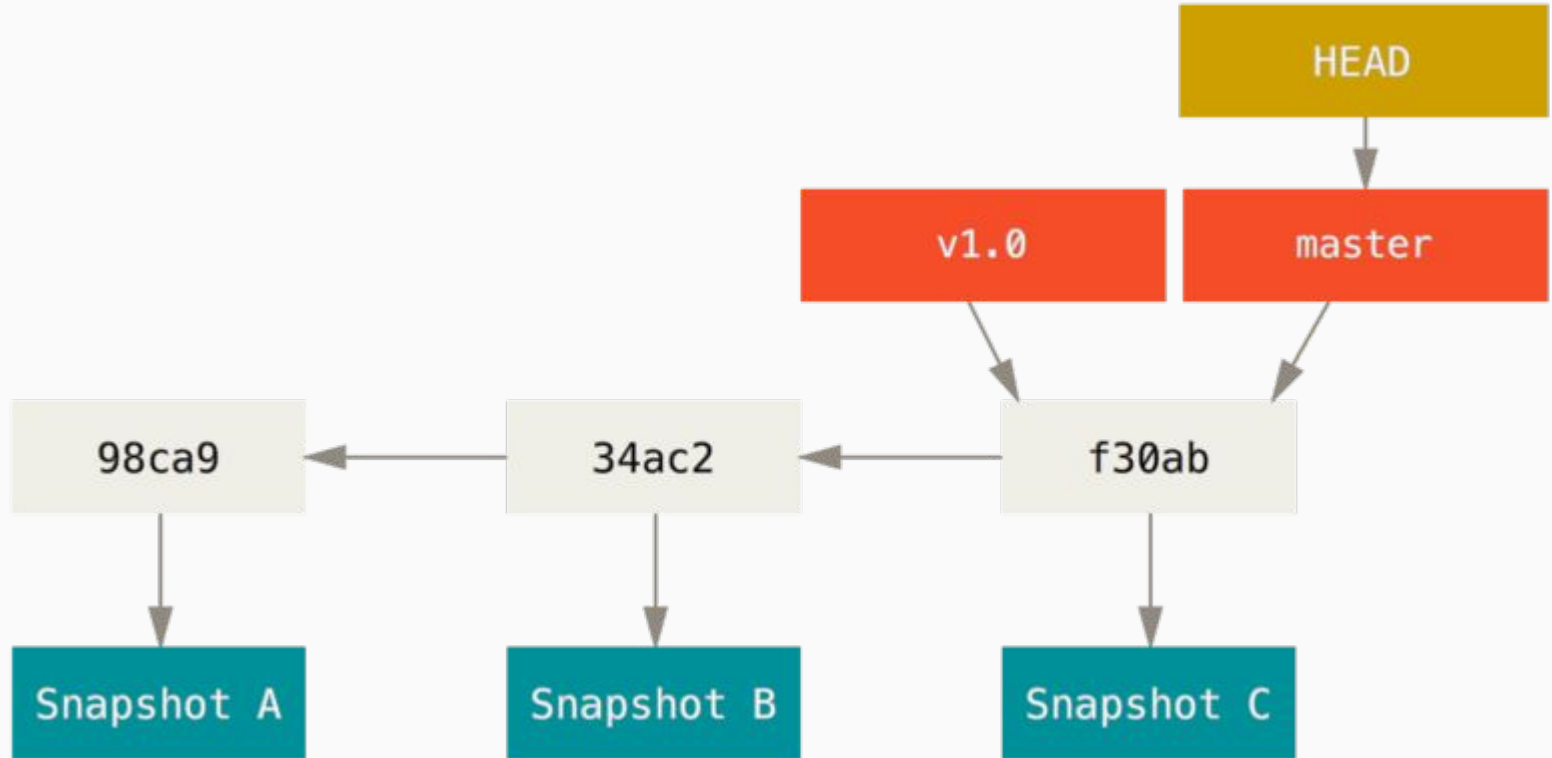
La commande ci dessus déplace la branche en cours vers le commit “34ac2”.

⇒ cela écrasera le commit actuel de votre branche et, par conséquent, tout son historique. On peut perdre du travail en émettant cette commande.

Il est conseillé d'exécuter cette commande sur une nouvelle branche au lieu de celle en cours.

La branche **master** n'est pas une branche spéciale. Elle est identique à toutes les autres branches. La seule raison pour laquelle chaque dépôt en a une est que la commande **git init** la crée par défaut et que la plupart des gens ne s'embêtent pas à la changer.

Une branche et l'historique de ses commits



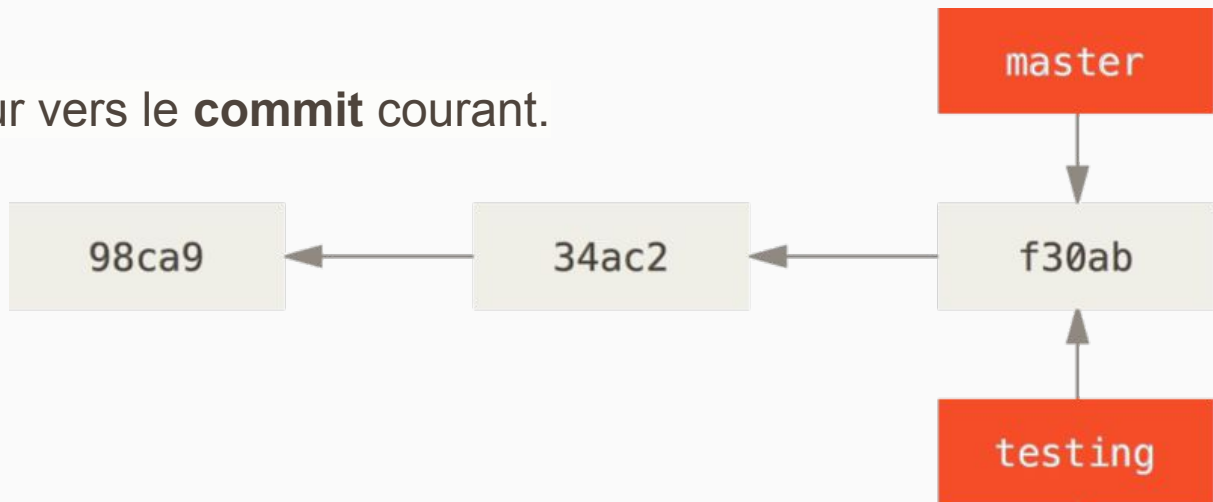
Créer une nouvelle branche

Créer une branche \Rightarrow créer un nouveau pointeur.

Créez une nouvelle branche nommée `testing` :

```
$ git branch testing
```

Cela crée un nouveau pointeur vers le **commit** courant.

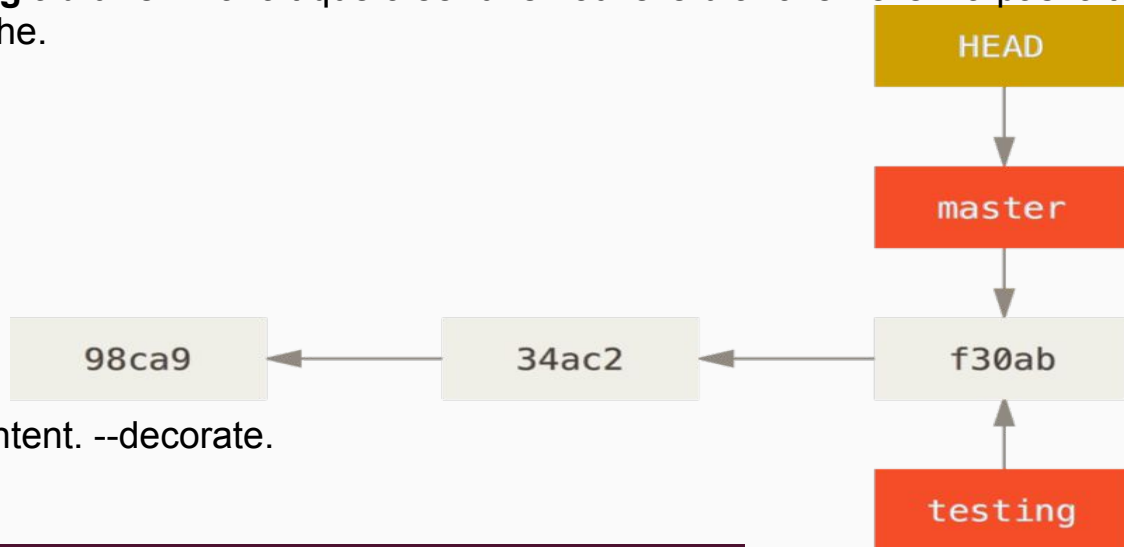


HEAD pointant vers une branche

Comment Git connaît-il alors la branche sur laquelle vous vous trouvez ?

Il conserve à cet effet un pointeur spécial appelé HEAD.

⇒ il s'agit simplement d'un pointeur sur la branche locale où vous vous trouvez. Dans ce cas, vous vous trouvez toujours sur master. En effet, la commande **git branch** n'a fait que créer une nouvelle branche - elle n'a pas fait basculer la copie de travail vers cette branche.



git log montre vers quoi les branches pointent. --decorate.

```
$ git log --oneline --decorate
```

```
f30ab (HEAD, master, test) add feature #32 - ability to add new
34ac2 fixed bug #ch1328 - stack overflow under certain conditions
98ca9 initial commit of my project
```


Visualisation des commits dans gitKraken

The screenshot shows the GitKraken application window. The top menu bar includes File, Edit, View, and Help. Below it is a toolbar with icons for New Tab, Release Notes, temp, laravel, New Tab, proj02, formation_docs, and ex1.3files. The main interface is divided into several sections. On the left, there's a sidebar with 'repository' set to 'ex1.3files' and 'branch' set to 'master'. Below this, a list of branches shows 'master' as the current branch. The central area displays a commit history graph with three commits: 'commit 3 : ajout ligne au fichier README', 'commit 2 : ajout ligne au fichier Licence', and 'commit 1 : commit initial - ajout des 3 fichiers'. The right sidebar shows details for 'commit 3', including the commit hash 'b8e1b6', the author 'Lassaad BAATI', and the date '18/11/2019 @ 19:46'.

GitKraken

File Edit View Help

repository ex1.3files branch master

Undo Redo Pull Push Branch Stash Pop Boards

Viewing 2/2 Show All

Filter (Ctrl + Alt + f)

LOCAL 2/2

master

testing

REMOTE 0/0

commit 3 : ajout ligne au fichier README

commit 2 : ajout ligne au fichier Licence

commit 1 : commit initial - ajout des 3 fichiers

commit: b8e1b6

commit 3 : ajout ligne au fichier README

Lassaad BAATI
authored 18/11/2019 @ 19:46

parent: fa6819

This screenshot shows the same GitKraken application but with the 'testing' branch selected. The sidebar on the left now shows 'testing' as the active branch. The commit history graph remains the same, showing the same three commits. The right sidebar still displays details for 'commit 3', but it also includes a section for '1 modified' files, listing 'README.md'. The bottom right corner has tabs for 'Path', 'Tree', and 'View all files'.

GitKraken

File Edit View Help

repository ex1.3files branch testing

Undo Redo Pull Push Branch Stash Pop Boards

Viewing 2/2 Show All

Filter (Ctrl + Alt + f)

LOCAL 2/2

master

testing

REMOTE 0/0

TAGS 0/0

SUBMODULES 0

commit 3 : ajout ligne au fichier README

commit 2 : ajout ligne au fichier Licence

commit 1 : commit initial - ajout des 3 fichiers

commit: b8e1b6

commit 3 : ajout ligne au fichier README

Lassaad BAATI
authored 18/11/2019 @ 19:46

parent: fa6819

1 modified

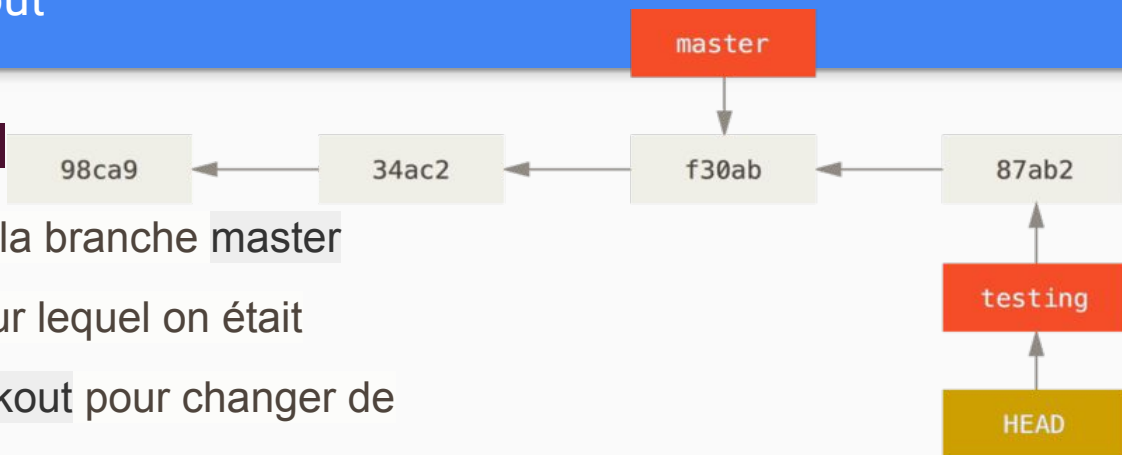
README.md

Path Tree View all files

HEAD est déplacé lors d'un checkout

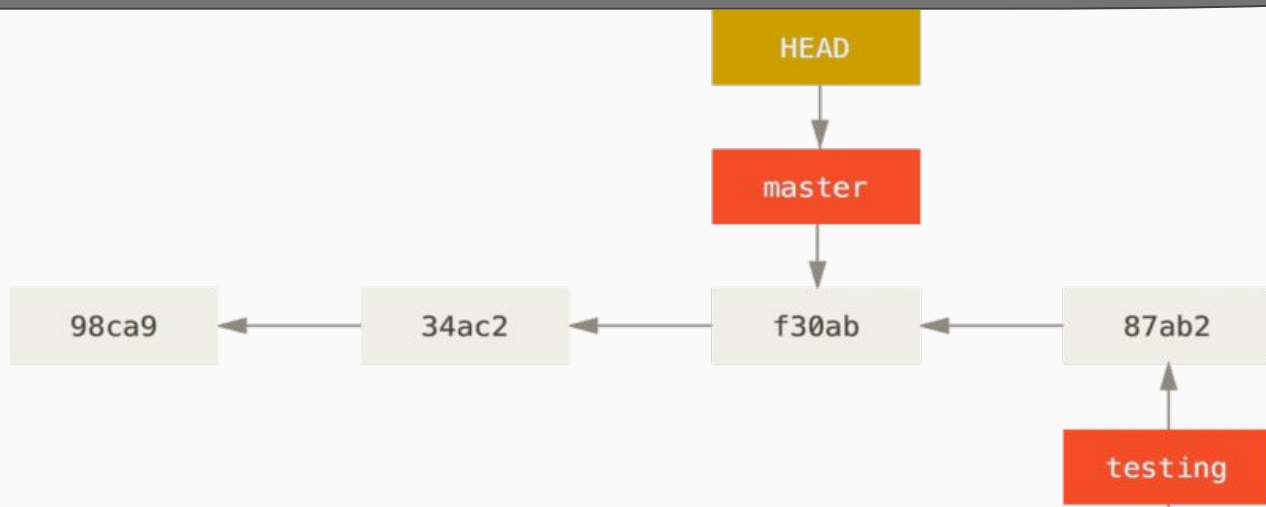
```
$ echo "add line" >> test.rb  
$ git commit -a -m 'made a change'
```

La branche `testing` a avancé tandis que la branche `master` pointe toujours sur le **commit** (`f30ab`) sur lequel on était lorsqu'on a lancé la commande `git checkout` pour changer de branche.



Retour sur la branche `master` :

```
$ git checkout master
```



Changer de branche modifie les fichiers dans votre répertoire de travail

Il est important de noter que lorsque vous changez de branche avec Git, les fichiers de votre répertoire de travail sont modifiés.

Si vous basculez vers une branche plus ancienne, votre répertoire de travail sera remis dans l'état dans lequel il était lors du dernier **commit** sur cette branche.

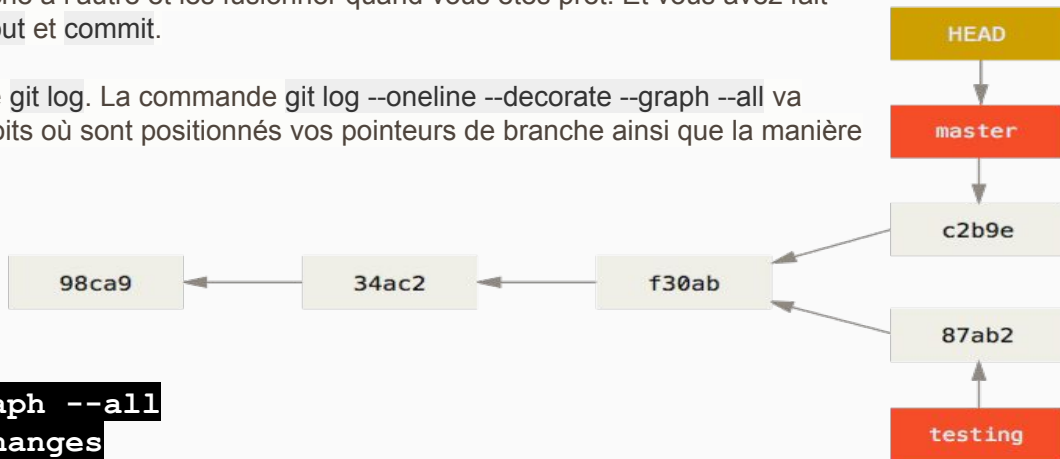
Si git n'est pas en mesure d'effectuer cette action proprement, il ne vous laissera pas changer de branche.

Divergence d'historique

```
$ vim test.rb  
$ git commit -a -m 'made other changes'
```

L'historique du projet a divergé. Vous avez créé une branche et basculé dessus, y avez réalisé des modifications, puis vous avez rebasculé sur la branche principale et réalisé d'autres modifications. Ces deux modifications sont isolées dans des branches séparées : vous pouvez basculer d'une branche à l'autre et les fusionner quand vous êtes prêt. Et vous avez fait tout ceci avec de simples commandes : branch, checkout et commit.

Vous pouvez également voir ceci grâce à la commande `git log`. La commande `git log --oneline --decorate --graph --all` va afficher l'historique de vos **commits**, affichant les endroits où sont positionnés vos pointeurs de branche ainsi que la manière dont votre historique a divergé.



```
$ git log --oneline --decorate --graph --all  
* c2b9e (HEAD, master) made other changes  
| * 87ab2 (test) made a change  
|/  
* f30ab add feature #32 - ability to add new formats to the  
* 34ac2 fixed bug #ch1328 - stack overflow under certain conditions  
* 98ca9 initial commit of my project
```

Créer et basculer vers la branche testing

```
lassaad@my-leeds-ws:~/repos/ex1.3files$ git branch
* master
lassaad@my-leeds-ws:~/repos/ex1.3files$
lassaad@my-leeds-ws:~/repos/ex1.3files$ git branch testing
lassaad@my-leeds-ws:~/repos/ex1.3files$ git status
Sur la branche master
rien à valider, la copie de travail est propre
lassaad@my-leeds-ws:~/repos/ex1.3files$ git log --graph --all --decorate
* commit 6f3177ebaec702b293b49035e45eae863550e615 (HEAD -> master, testing)
  Author: Lassaad BAATI <lassaad.baati@gmail.com>
  Date:   Mon Nov 18 20:41:40 2019 +0100

    commit 3 : ajout ligne au fichier README

* commit d004b82122b3b24374a768a42edfccbd8905d0ba (tag: v1.1)
  Author: Lassaad BAATI <lassaad.baati@gmail.com>
  Date:   Mon Nov 18 20:39:55 2019 +0100

    commit 2 : ajout ligne au fichier Licence

* commit 08ebdf1593a58d71260864948e73f0b1c7d857db (tag: v1.0)
  Author: Lassaad BAATI <lassaad.baati@gmail.com>
  Date:   Mon Nov 18 20:38:55 2019 +0100

    commit 1 : commit initial - ajout des 3 fichiers
lassaad@my-leeds-ws:~/repos/ex1.3files$
lassaad@my-leeds-ws:~/repos/ex1.3files$ git branch
* master
  testing
lassaad@my-leeds-ws:~/repos/ex1.3files$ git checkout testing
Basculement sur la branche 'testing'
lassaad@my-leeds-ws:~/repos/ex1.3files$ git branch
  master
* testing
lassaad@my-leeds-ws:~/repos/ex1.3files$
```



```

lassaad@my-leeds-ws:~/repos/ex1.3files$
lassaad@my-leeds-ws:~/repos/ex1.3files$ git branch
* master
  testing
lassaad@my-leeds-ws:~/repos/ex1.3files$ git checkout testing
Basculement sur la branche 'testing'
lassaad@my-leeds-ws:~/repos/ex1.3files$ git branch
  master
* testing
lassaad@my-leeds-ws:~/repos/ex1.3files$ git status
Sur la branche testing
rien à valider, la copie de travail est propre
lassaad@my-leeds-ws:~/repos/ex1.3files$

```

```

lassaad@my-leeds-ws:~/repos/ex1.3files$ git log --graph --all --decorate
* commit ba666d53c56d1c016014bc654b495a4a95ea69c6 (HEAD -> testing)
  Author: Lassaad BAATI <lassaad.baati@gmail.com>
  Date:   Mon Nov 18 21:25:16 2019 +0100

    commit 4: add line in README.md

* commit 6f3177ebaec702b293b49035e45eae863550e615 (master)
  Author: Lassaad BAATI <lassaad.baati@gmail.com>
  Date:   Mon Nov 18 20:41:40 2019 +0100

    commit 3 : ajout ligne au fichier README

* commit d004b82122b3b24374a768a42edfccbd8905d0ba (tag: v1.1)
  Author: Lassaad BAATI <lassaad.baati@gmail.com>
  Date:   Mon Nov 18 20:39:55 2019 +0100

    commit 2 : ajout ligne au fichier Licence

* commit 08ebdf1593a58d71260864948e73f0b1c7d857db (tag: v1.0)
  Author: Lassaad BAATI <lassaad.baati@gmail.com>
  Date:   Mon Nov 18 20:38:55 2019 +0100

    commit 1 : commit initial - ajout des 3 fichiers
lassaad@my-leeds-ws:~/repos/ex1.3files$ git checkout master
Basculement sur la branche 'master'

```

```

lassaad@my-leeds-ws:~/repos/ex1.3files$ git checkout master
Basculement sur la branche 'master'
lassaad@my-leeds-ws:~/repos/ex1.3files$ git log --graph --all --decorate
* commit ba666d53c56d1c016014bc654b495a4a95ea69c6 (testing)
  Author: Lassaad BAATI <lassaad.baati@gmail.com>
  Date:   Mon Nov 18 21:25:16 2019 +0100

    commit 4: add line in README.md

* commit 6f3177ebaec702b293b49035e45eae863550e615 (HEAD -> master)
  Author: Lassaad BAATI <lassaad.baati@gmail.com>
  Date:   Mon Nov 18 20:41:40 2019 +0100

    commit 3 : ajout ligne au fichier README

* commit d004b82122b3b24374a768a42edfccbd8905d0ba (tag: v1.1)
  Author: Lassaad BAATI <lassaad.baati@gmail.com>
  Date:   Mon Nov 18 20:39:55 2019 +0100

    commit 2 : ajout ligne au fichier Licence

* commit 08ebdf1593a58d71260864948e73f0b1c7d857db (tag: v1.0)
  Author: Lassaad BAATI <lassaad.baati@gmail.com>
  Date:   Mon Nov 18 20:38:55 2019 +0100





    commit 1 : commit initial - ajout des 3 fichiers
lassaad@my-leeds-ws:~/repos/ex1.3files$

```

Introduction aux branches

```
lassaad@my-leeds-ws:~/repos/temp$ git log --graph --decorate --all
* commit 703d744d1299e6ba04996fa6c741fee41500dc86 (HEAD -> master)
| Author: Lassaad BAATI <lassaad.baati@gmail.com>
| Date:   Mon Nov 11 23:17:27 2019 +0100
|
|     commit add line in README add master branch
|
|
| * commit 491356f4a6bb2fd026643a16473eaacf6edcc62c (testing)
| | Author: Lassaad BAATI <lassaad.baati@gmail.com>
| | Date:   Mon Nov 11 23:15:33 2019 +0100
| |
| |     commit add line in LICENCE ad testing branch
| |
| |
| * commit b99eced887a85d5dedbf12d6708b63c4801f2f7e
| / Author: Lassaad BAATI <lassaad.baati@gmail.com>
|   Date:   Mon Nov 11 23:06:15 2019 +0100
|   |
|   |     commit a partir de la branche testing pour modifier README
|   |
|   |
| * commit e5f0da4bdfedae2c9dce38d7a23e4386a2979a46
|   Author: Lassaad BAATI <lassaad.baati@gmail.com>
|   Date:   Mon Nov 11 22:56:18 2019 +0100
|   |
|   |     initial commit of my project
```

Fichier Éditer Vue Aide

 **testing** commit 4: add line in README.md
 **master** commit 3 : ajout ligne au fichier README
 **v1.1** commit 2 : ajout ligne au fichier Licence
 **v1.0** commit 1 : commit initial - ajout des 3 fichiers

Lassaad BAATI <lassaad.baati@gmail.com>
 Lassaad BAATI <lassaad.baati@gmail.com>
 Lassaad BAATI <lassaad.baati@gmail.com>
 Lassaad BAATI <lassaad.baati@gmail.com>

2019-11-18 21:25:16
 2019-11-18 20:41:40
 2019-11-18 20:39:55
 2019-11-18 20:38:55

Id SHA1 : 6f3177ebaec702b293b49035e45eae863550e615

Colonne

2 /

4

Recherche ↓ ↑ commit contient :

Exact

Tous les champs

Rechercher

◆ Patch ◆ Arbre

◆ Diff ◆ Ancienne version ◆ Nouvelle version Lignes de contexte

Auteur: Lassaad BAATI <lassaad.baati@gmail.com> 2019-11-18 21:25:16
 Valideur: Lassaad BAATI <lassaad.baati@gmail.com> 2019-11-18 20:41:40
 Parent: [d004b82122b3b24374a768a42edfccbd8905d0ba](#) (commit 2 : ajout ligne au fichier Licence)
 Enfant: [ba666d53c56d1c016014bc654b495a4a95ea69c6](#) (commit 4 : ajout ligne au fichier README)
 Branche: [master](#), [testing](#)
 Suit: [v1.1](#)
 Précède:

commit 3 : ajout ligne au fichier README

----- README.md -----
 index e69de29..a7816e2 100644

@@ -0,0 +1 @@

+line 1 : premiere ligne README

 Commentaires
 README.md

Git ⇒ fusion avec conflits

Quand le processus ci-dessus ne se déroule pas aussi bien. Si on a modifié différemment la même partie du même fichier dans les deux branches qu'on souhaite fusionner, Git ne sera pas capable de réaliser proprement la fusion. Si la résolution du problème #53 a modifié la même section de fichier que le correctif, on obtient un conflit qui ressemblera à ceci :

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Git n'a pas automatiquement créé le commit de fusion. Il a arrêté le processus le temps que vous résolviez le conflit. Si vous voulez vérifier, à tout moment après l'apparition du conflit, quels fichiers n'ont pas été fusionnés, vous pouvez lancer la commande `git status` :

```
$ git status
On branch master
You have unmerged paths.
(fix conflicts and run "git commit")
Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:
      index.html
no changes added to commit (use "git add" and/or "git commit -a")
```

Git : conflit de merge

```
lassaad@my-leeds-ws:~/repos/formation_docs$ git branch
master
* test
testing
lassaad@my-leeds-ws:~/repos/formation_docs$ git merge master
CONFLICT (modification/suppression) : file1 supprimé dans HEAD et modifié dans master. Version master de file1 laissée dans l'arbre.
Fusion automatique de README.md
La fusion automatique a échoué ; réglez les conflits et validez le résultat.
lassaad@my-leeds-ws:~/repos/formation_docs$ git checkout master
file1: needs merge
error: vous devez d'abord résoudre votre index courant
lassaad@my-leeds-ws:~/repos/formation_docs$ git status
Sur la branche test
Vous avez des chemins non fusionnés.
(réglez les conflits puis lancez "git commit")
(utilisez "git merge --abort" pour annuler la fusion)

Modifications qui seront validées :

    modifié :      README.md
    nouveau fichier : "dir1/file\303\251"
    nouveau fichier : dir1/test
    nouveau fichier : dir2/file1

Chemins non fusionnés :
(utilisez "git add/rm <fichier>..." si nécessaire pour marquer comme résolu)

    supprimé par nous :      file1

lassaad@my-leeds-ws:~/repos/formation_docs$ git add *
```

Exercice

1. Créer un dossier temp
2. Ajouter 3 fichiers README LICENCE et test.php
3. Faire un git init
4. Modifier un fichier
5. Ajouter ce fichier à l'index
6. Faire un commit de la dernière modification sur la branche master (Pointé par HEAD)
7. Créer une nouvelle branche testing
8. Basculer sur la nouvelle branche testing
9. Ajouter une ligne au fichier LICENCE
10. Ajouter le fichier modifié à l'index (staged)
11. Faire un commit sur la branche testing (pointé par HEAD).
12. Vérifier le log des commit
13. Basculer sur la branche master
14. Ajouter une ligne au fichier test.rb
15. Visualiser l'état des commit par branche avec la commande suivante

```
$ git log --graph --decorate --all
```

1. Les branches avec git

1.2. Branche et fusion : les bases

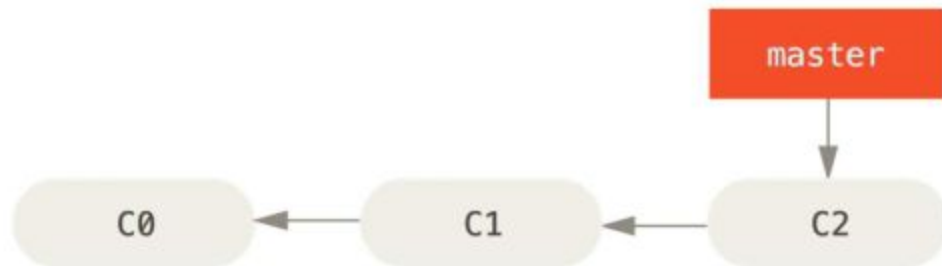
Plan

- Scénario
- Mise en place
- Développement du correctif
- Téléphone
- Modification du site internet
- Retour sur le correctif
- Une fusion un peu spéciale

scénario

- 1) vous travaillez sur un site web ;
 - a) vous travaillez sur un site web ;
 - b) vous créez une branche pour un nouvel article en cours ;
 - c) vous commencez à travailler sur cette branche.
- 2) À cette étape, vous recevez un appel pour vous dire qu'un problème critique a été découvert et qu'il faut le régler au plus tôt. Vous faites donc ce qui suit :
 - a) vous basculez sur la branche de production ;
 - b) vous créez une branche pour y ajouter le correctif ;
 - c) après l'avoir testé, vous fusionnez la branche du correctif et poussez le résultat en production ;
- 3) Basculez sur la branche initiale et continuez votre travail.

- Nous avons notre projet qui ressemble à ça :



- On décide de travailler sur le problème #53, on crée une branche et on se place dessus.

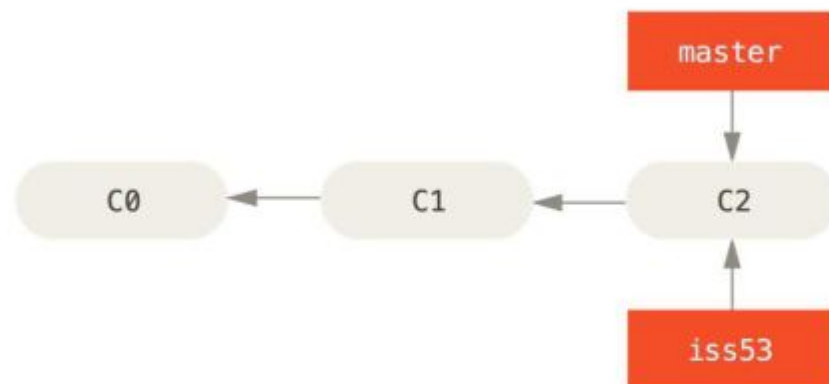
\$> git checkout -b iss53

- Équivalent à :

\$> git branch iss53

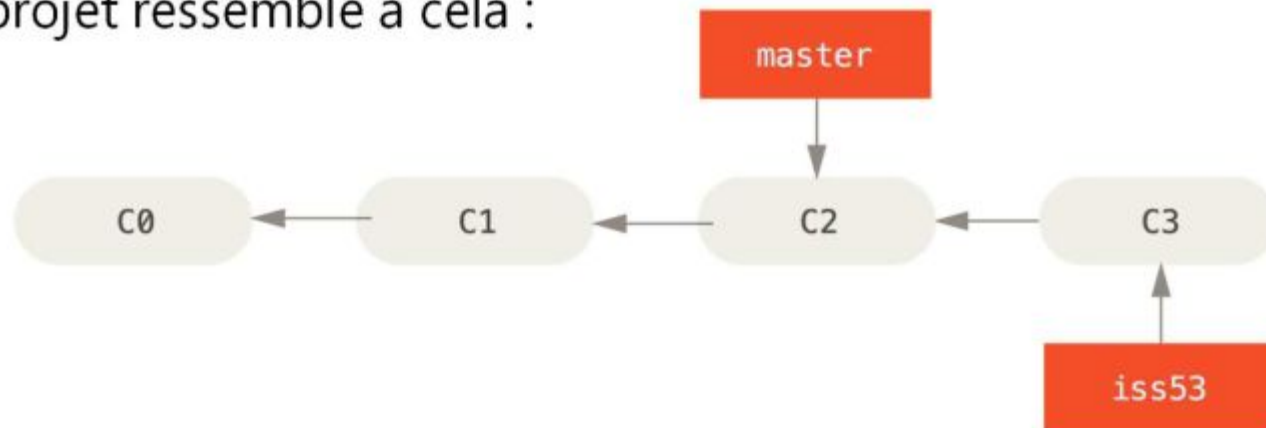
\$> git checkout iss53

- Notre projet ressemble donc à cela et *HEAD* est sur iss53.



- Nous faisons une modification et nous créons un nouveau commit :
`$> git commit -a -m 'Résolution d\'un premier bug'`

- Notre projet ressemble à cela :



- Le téléphone sonne, il faut qu'on fasse immédiatement une modification sur le site internet.

\$> git checkout master

Modification du site internet

- Nous allons développer notre modification urgente en créant une nouvelle branche:

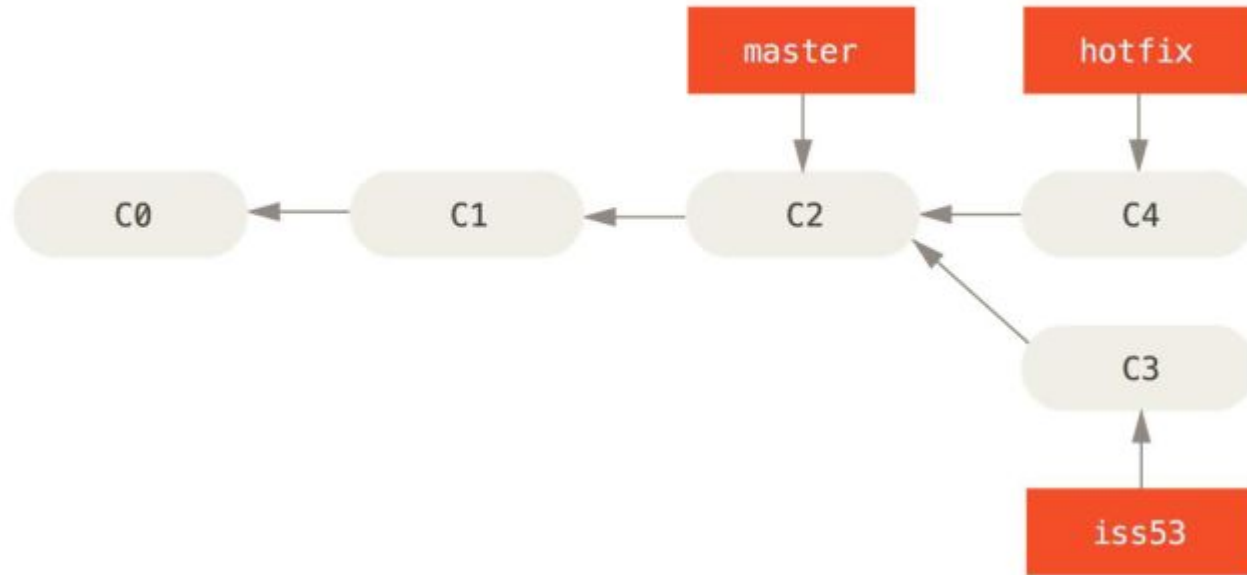
```
$> git checkout -b hotfix
```

- On modifie notre code.
- On crée un commit :

```
$> git commit -a -m 'correction de l\'adresse email'
```

- À quoi ressemble notre arbre ?

- Notre arbre ressemble à :



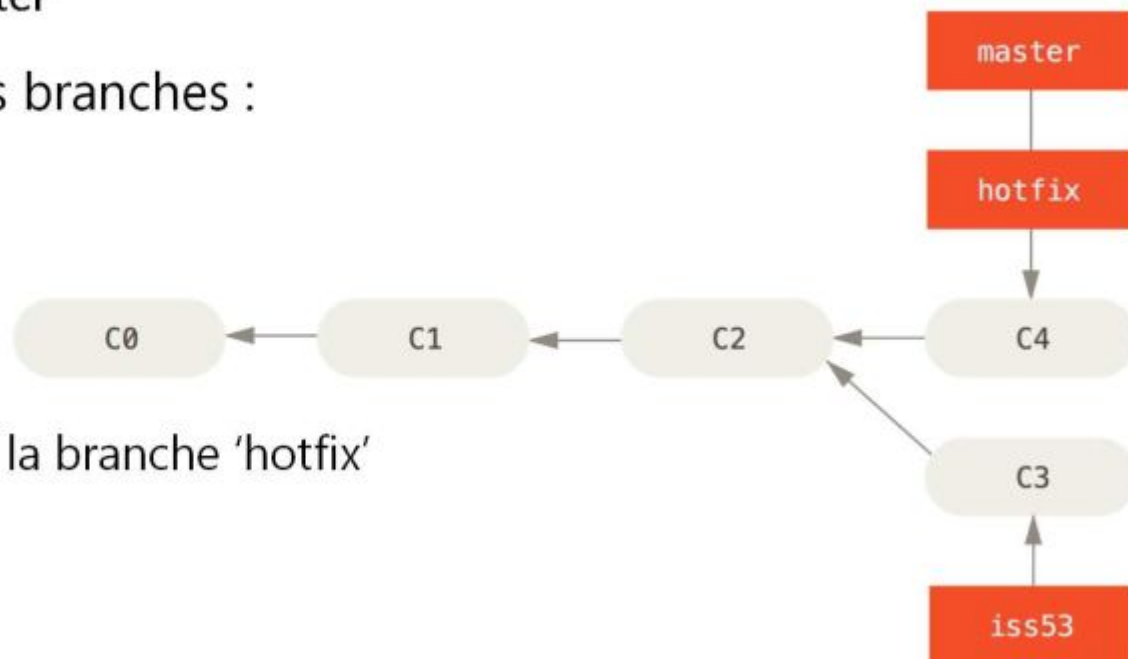
- Maintenant, nous revenons sur master :

\$> git checkout master

- Nous fusionnons les branches :

\$> git merge hotfix

- On peut supprimer la branche 'hotfix'



Retour sur le correctif

- On retourne sur le correctif

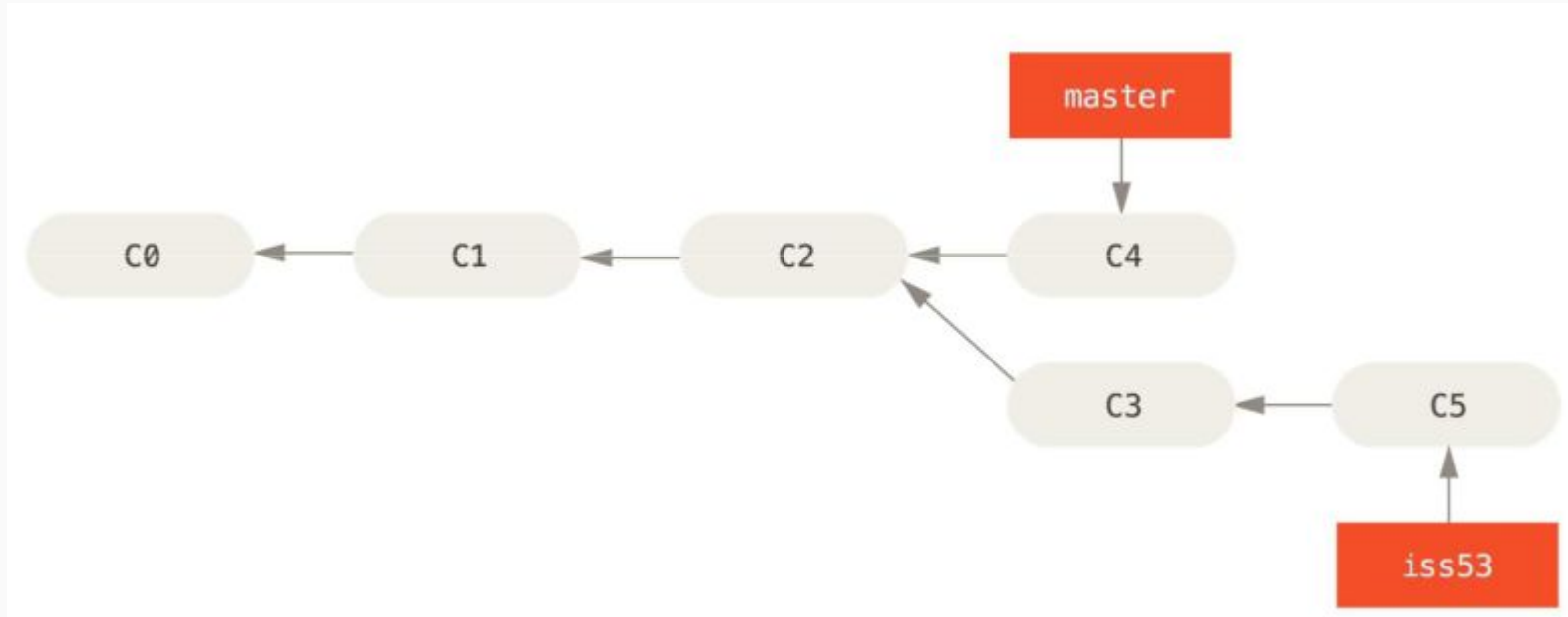
```
$> git checkout iss53
```

- On continue nos développements
- On commit les nouvelles modifications

```
$> git commit -a -m 'Résolution deuxième bug, donc problème #53 corrigé'
```

- À quoi ressemble notre arbre ?

Retour sur le correctif



Une fusion un peu spéciale

- On se place sur 'master' :

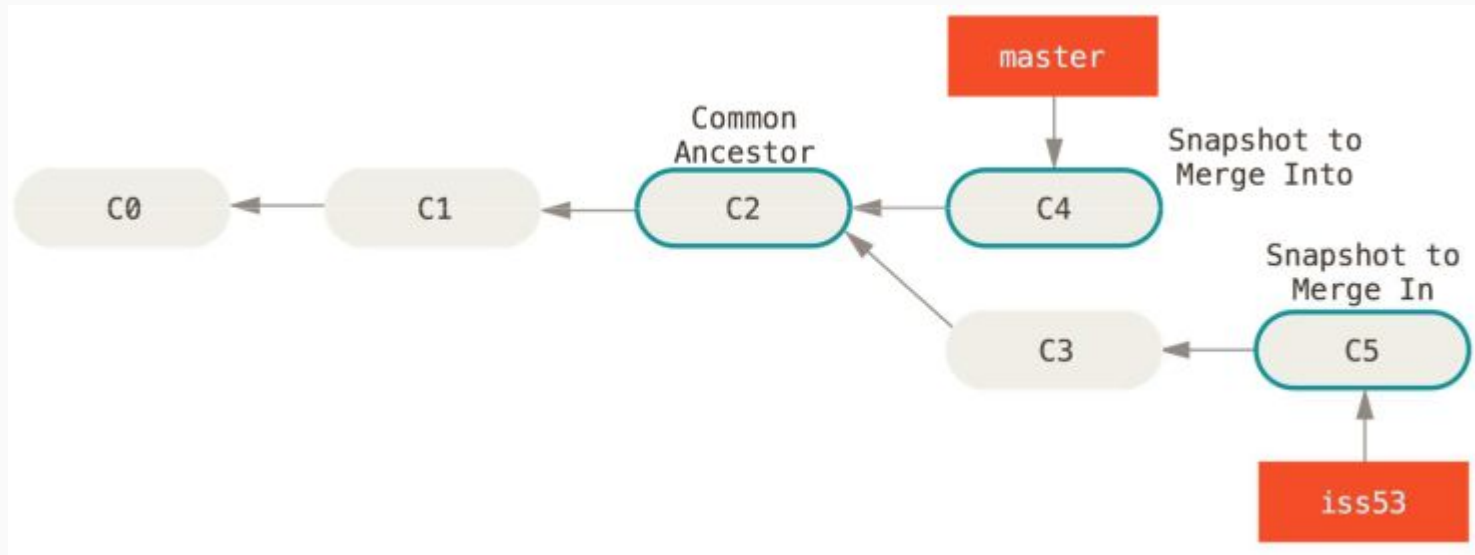
```
$> git checkout master
```

- On fusionne les deux branches

```
$> git merge iss53
```

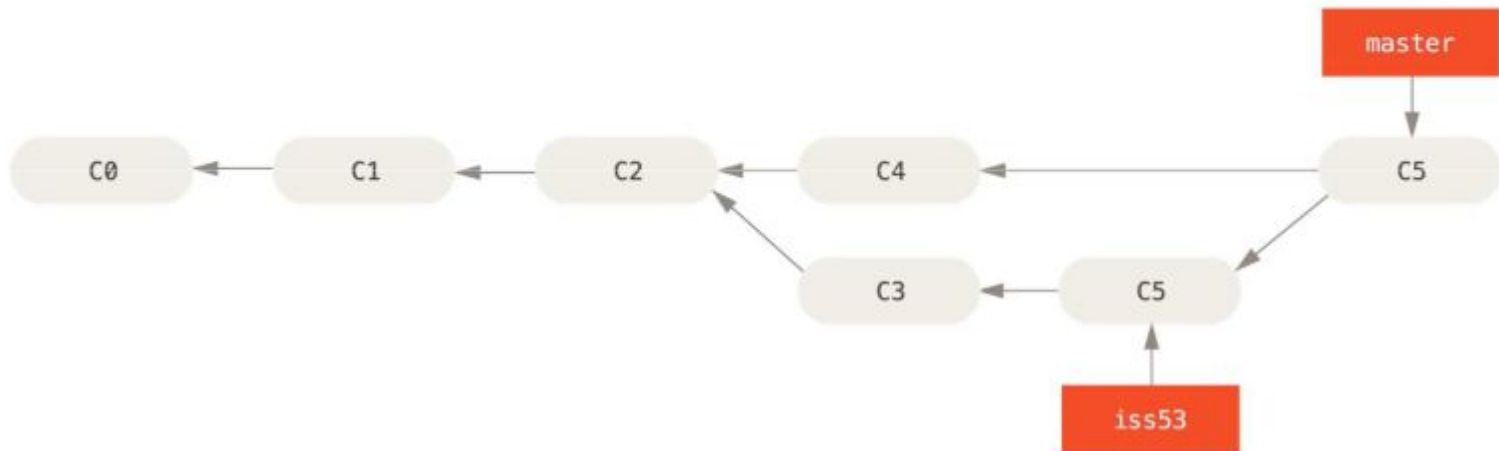
- Qu'est ce qu'il se passe dans ce cas ?

Une fusion un peu spéciale



Une fusion un peu spéciale

- Git crée un nouveau commit qui contient le fusion de C4 et C5.



Un conflit

- Si nous modifions le même fichier dans deux branches qui seront fusionnées, il est possible que Git n'arrive plus à faire une fusion.
- La procédure est la suivante :

```
$> git checkout master
```

```
$> git merge future conflit
```

- Un conflit apparaît, on rentre dans le(s) fichier(s) incriminé(s) et corrige le conflit.

```
$> git add <les fichiers>
```

```
$> git commit -m 'message'
```

1. Les branches avec git

1.3. Gestion des branches

Plan

- Lister les branches
- Les branches fusionnées et non fusionnées
- Supprimer une branche

Lister les branches

- Maintenant que vous avez créé, fusionné et supprimé des branches, regardons de plus près les outils.
- 'git branch', sans argument, permet de lister les branches en local.

```
$ git branch
```

- Pour avoir un peu plus d'informations

```
$ git branch -v
```

Les branches fusionnées et non fusionnées

- Les options `--merged` et `--no-merge` sont des options très utiles.
- Pour avoir la liste des branches déjà fusionnées:

```
$ git branch --merged
```

- Pour avoir la liste des branches pas encore fusionnées :

```
$ git branch --no-merged
```

Supprimer une branche

- Pour supprimer une branche :

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

```
$ git branch -d test
```

```
error: The branch 'test' is not fully merged.
```

```
If you are sure you want to delete it, run 'git branch -D test'.
```

Si vous souhaitez réellement supprimer cette branche et perdre ainsi le travail réalisé, vous pouvez tout de même forcer la suppression avec l'option -D, comme l'indique le message.

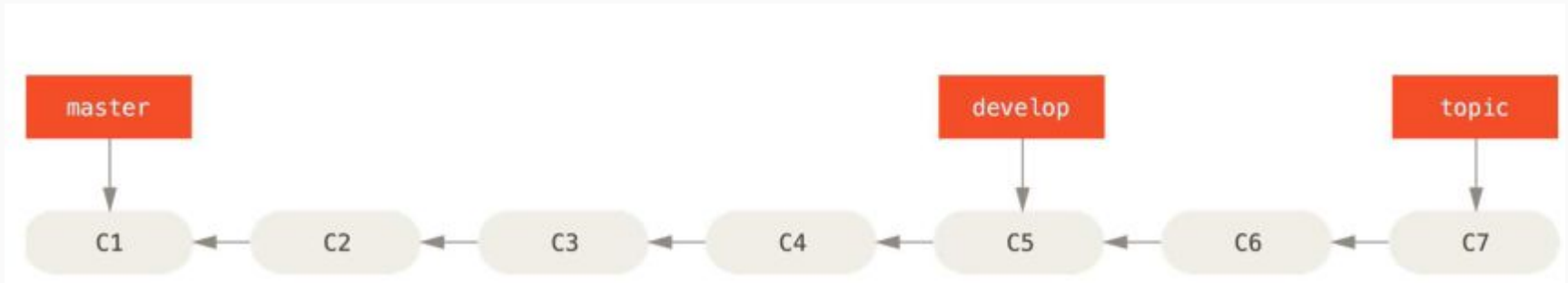
1. Les branches avec git

1.4. Travailler avec les branches

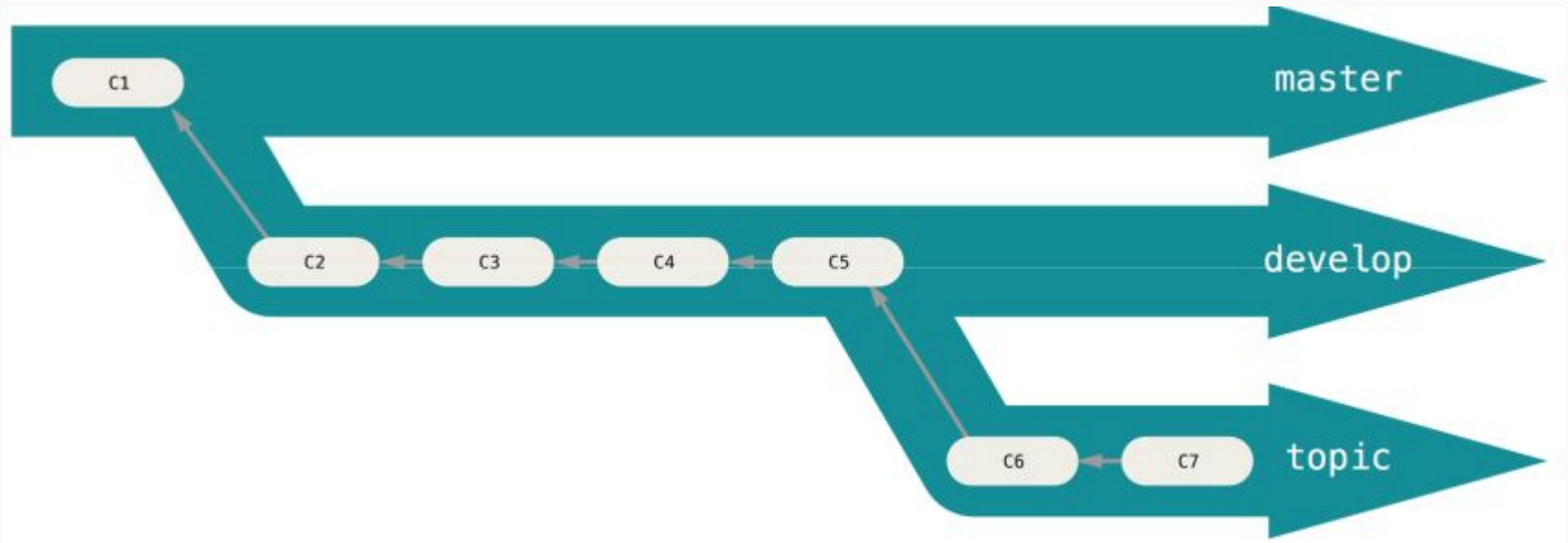
Plan

- Branche au long cours
- Branche thématique

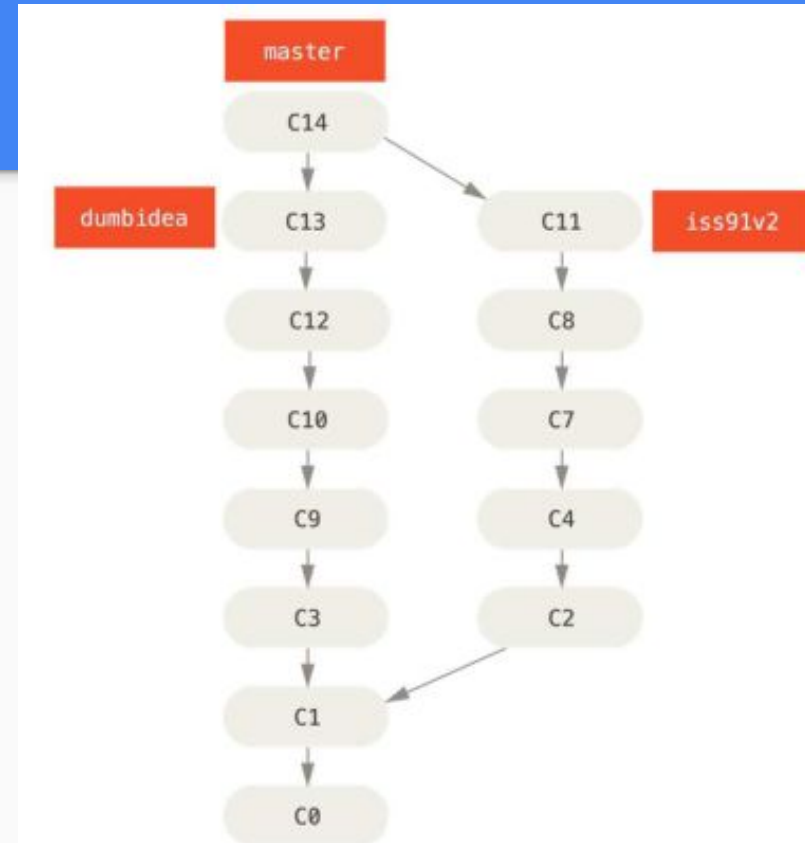
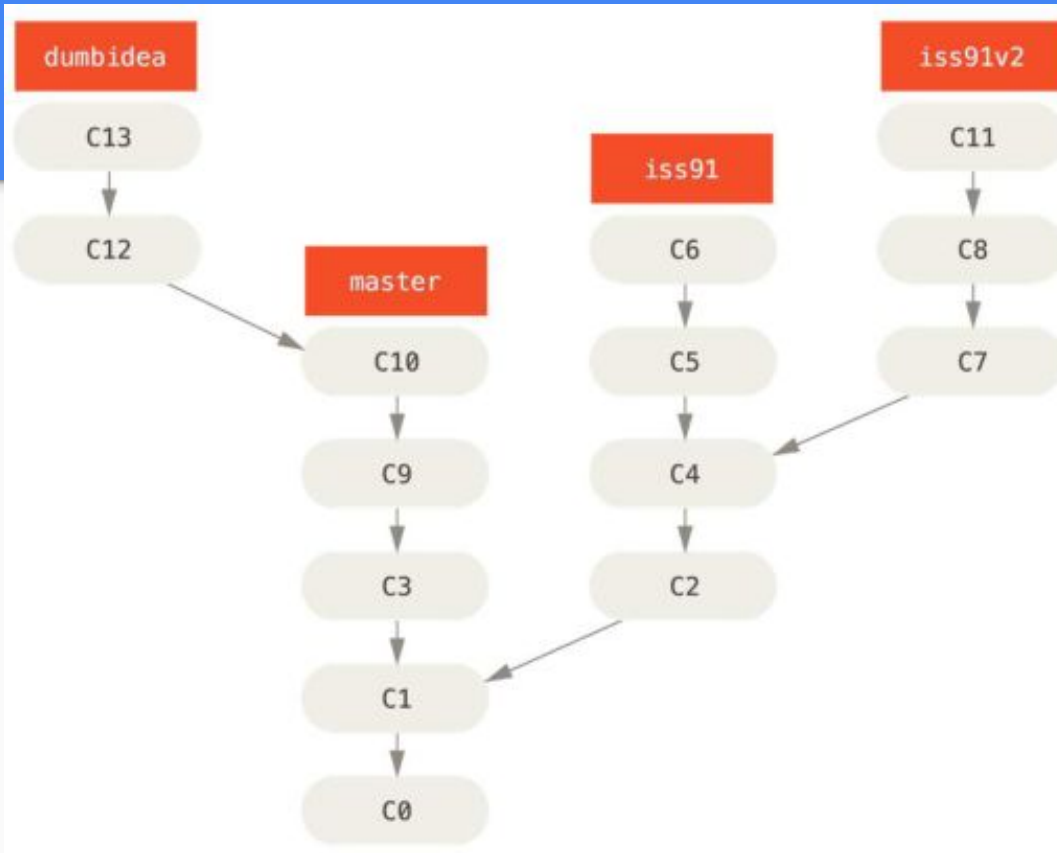
Branche au long cours



Branche au long cours



Branche thématique



Note

origin n'est pas spécial. De la même manière que le nom de branche master n'a aucun sens particulier pour Git, le nom origin n'est pas spécial. Tandis que master est le nom attribué par défaut à votre branche initiale lorsque vous lancez la commande `git init` et c'est la seule raison pour laquelle ce nom est utilisé aussi largement, origin est le nom utilisé par défaut pour un dépôt distant lorsque vous lancez `git clone`.

Si vous lancez à la place `git clone -o booyah`, votre branche de suivi à distance par défaut s'appellera `booyah/master`.

1. Les branches avec git

1.5. Introduction aux branches distantes

Plan

- Branches distantes
- Pousser les branches

Branches distantes

- Les références distantes sont des références (pointeurs) vers les éléments de votre dépôt distant tels que les branches, les tags, etc...

- Pour obtenir la liste complète de ces références :

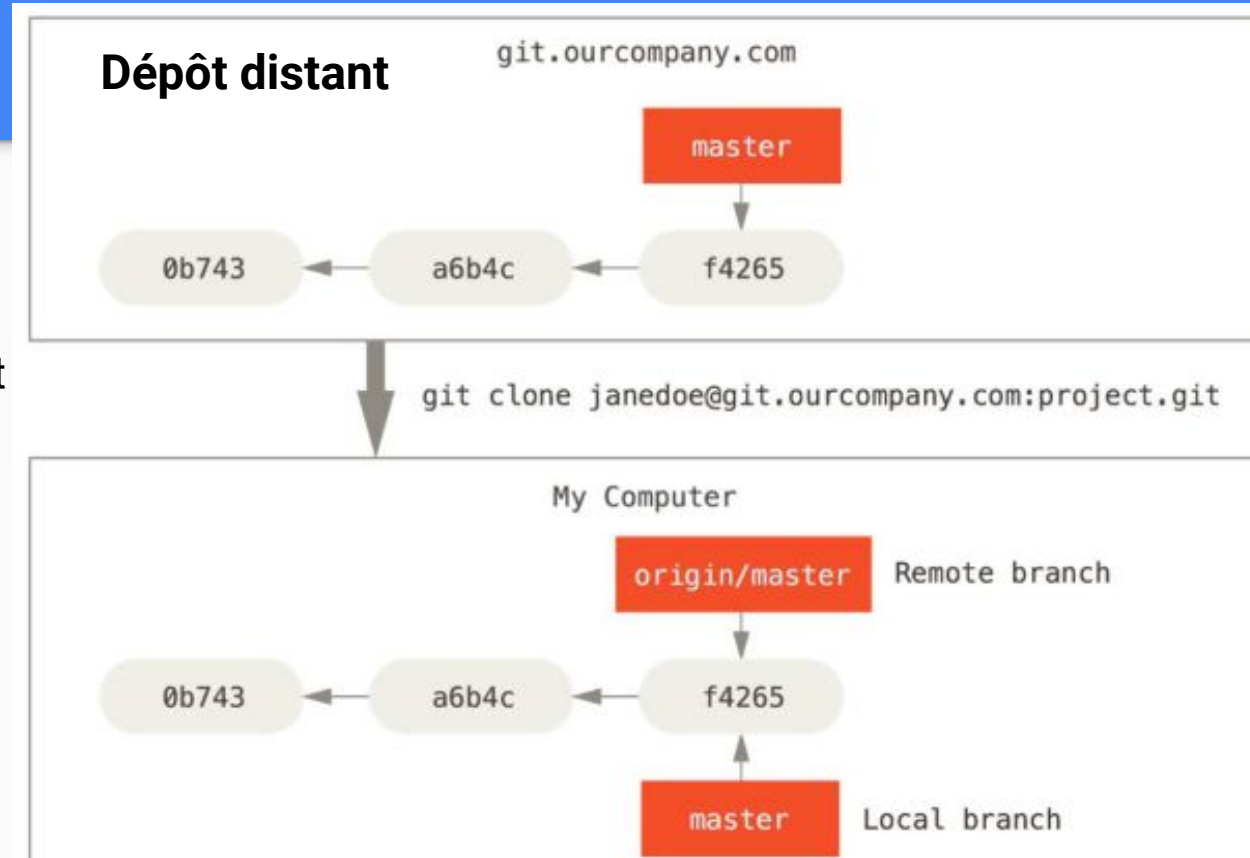
```
$ git ls-remote <remote>
```

- Les branches distantes sont des références (des pointeurs) vers l'état des branches sur votre dépôt distant.

- Elles sont sous la forme <distant>/<branche>

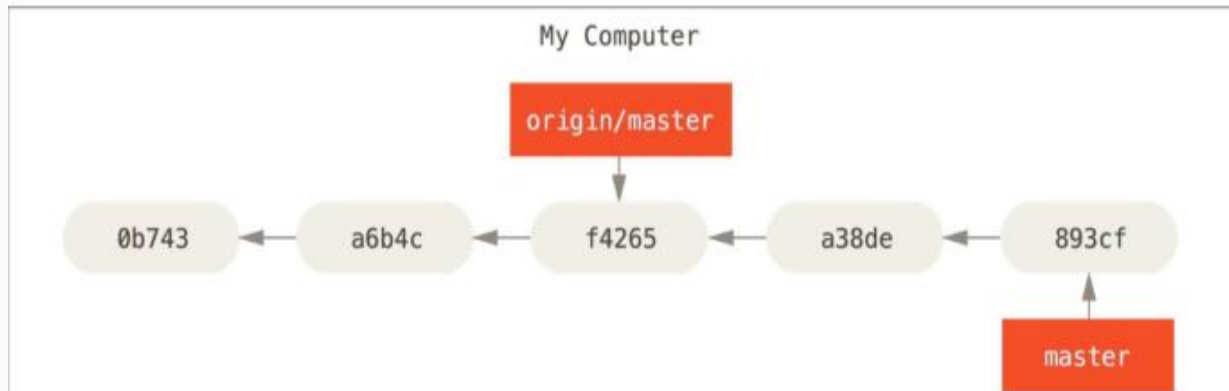
Dépôts distant et local après un clone

Si vous travaillez sur votre branche locale master et que dans le même temps, quelqu'un publie sur git.notresociete.com et met à jour cette même branche master, alors vos deux historiques divergent. Tant que vous restez sans contact avec votre serveur distant, votre pointeur vers origin/master n'avance pas.



Les travaux locaux et distants peuvent diverger

Lancez la commande `git fetch origin` pour synchroniser les travaux. Cette commande recherche le serveur hébergeant origin (`git.notresociete.com`), y récupère toutes les nouvelles données et met à jour votre base de donnée locale en déplaçant votre pointeur `origin/master` vers une nouvelle position, plus à jour.



```

lassaad@my-leeds-ws:~/repos/formation_docs$ git fetch origin
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Dépaquetage des objets: 100% (3/3), fait.
Depuis https://github.com/lassaadbaati/formation_docs
 6ec1f59..d875e88 master    -> origin/master
lassaad@my-leeds-ws:~/repos/formation_docs$ cat README.md
# formation_docs. add string from master branch
Dépot pour la formation git
Bonjour - dernière ligne
Dernière ligne dans la branche test
lassaad@my-leeds-ws:~/repos/formation_docs$ git status
Sur la branche master
Votre branche est en retard sur 'origin/master' de 1 commit, et peut être mise à jour en avance rapide.
(utilisez "git pull" pour mettre à jour votre branche locale)

rien à valider, la copie de travail est propre
lassaad@my-leeds-ws:~/repos/formation_docs$ git pull
Mise à jour 6ec1f59..d875e88
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
lassaad@my-leeds-ws:~/repos/formation_docs$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.

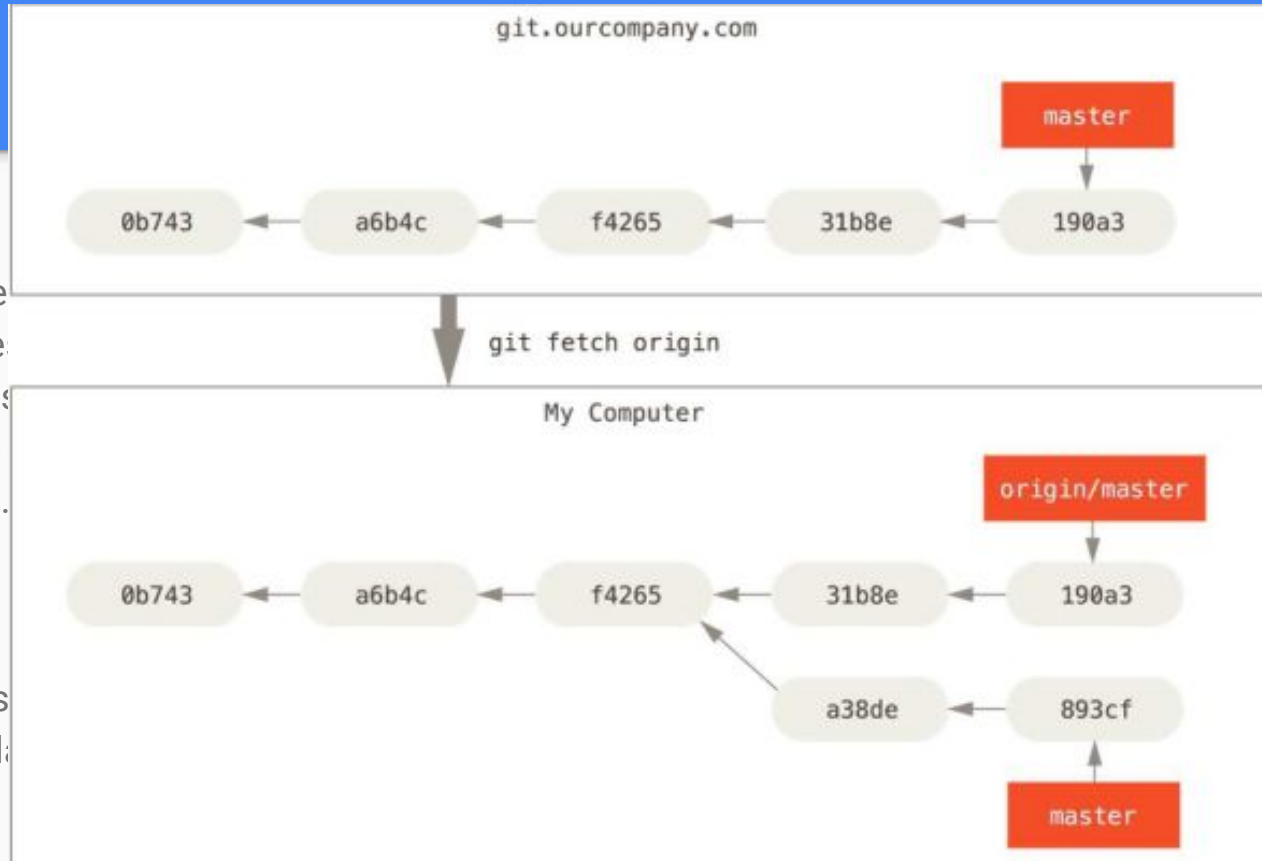
rien à valider, la copie de travail est propre
lassaad@my-leeds-ws:~/repos/formation_docs$ █

```

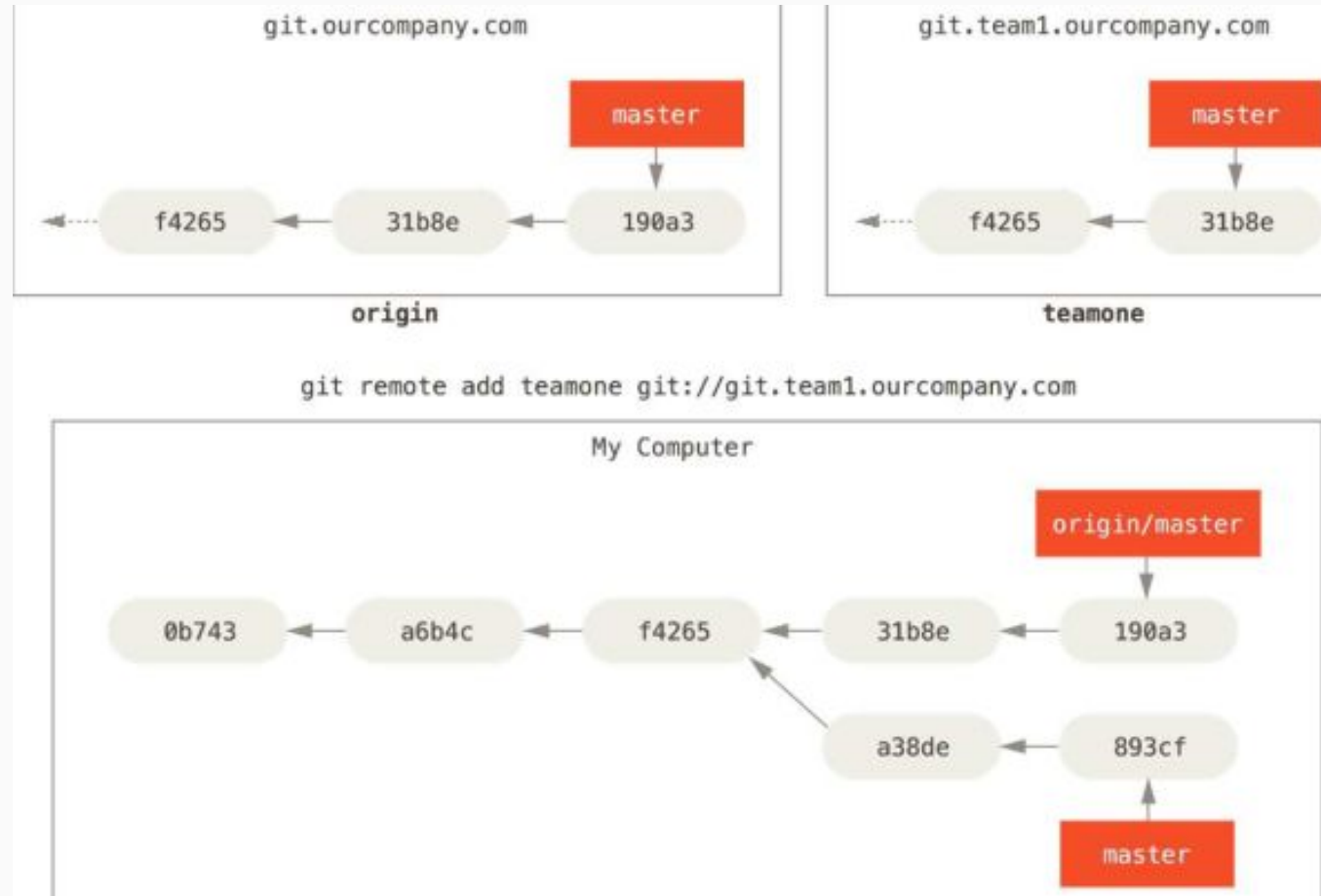
git fetch met à jour vos branches de suivi à distance

Pour démontrer l'usage de multiples serveurs distants et le fonctionnement des branches de suivi à distance pour ces projets distants, supposons que vous avez un autre serveur Git interne qui n'est que par une équipe de développeurs. Ce serveur se trouve sur `git.entreprise1.notresociete.com`.

Vous pouvez l'ajouter aux références distantes de votre projet en lançant la commande **`git remote add`**

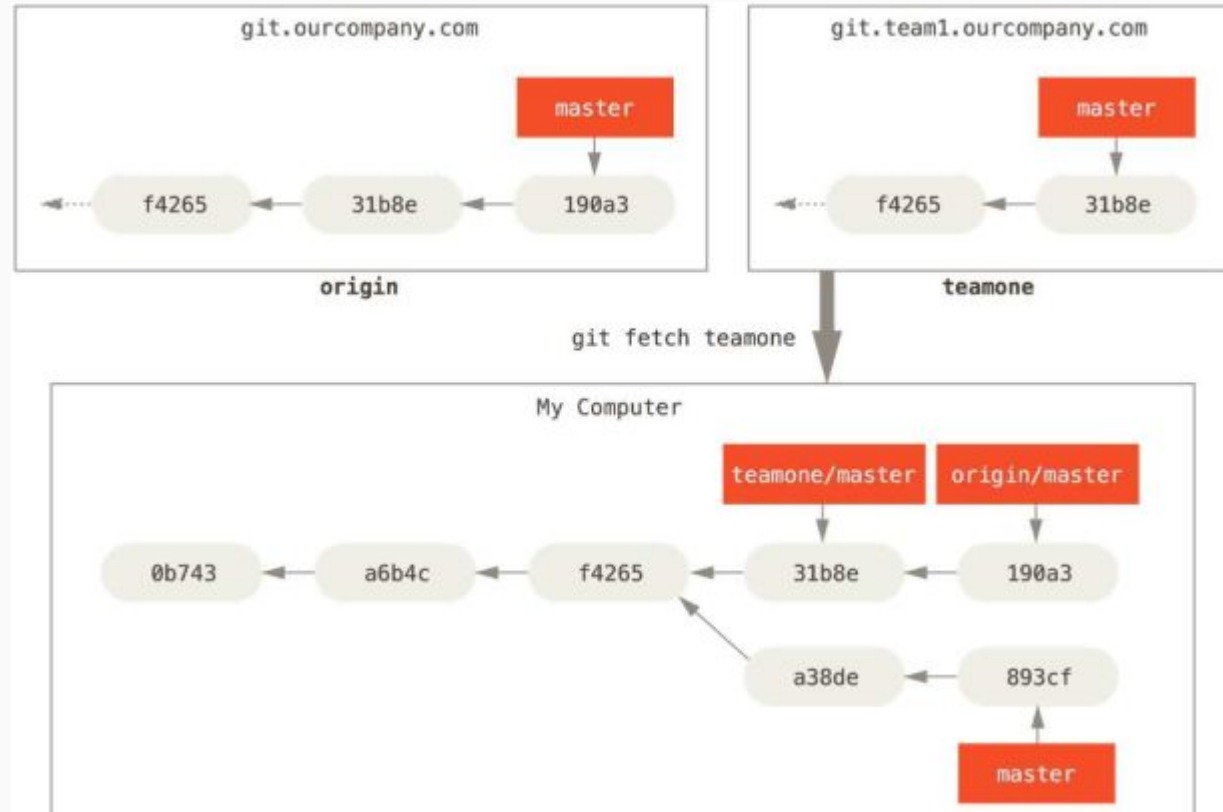


Branche de suivi à distance equipeun/master



Ajout d'un nouveau serveur en tant que référence distante

vous pouvez lancer **git fetch equipeun** pour récupérer l'ensemble des informations du serveur distant equipeun que vous ne possédez pas. Comme ce serveur contient déjà un sous-ensemble des données du serveur origin, Git ne récupère aucune donnée mais initialise une branche de suivi à distance appelée equipeun/master qui pointe sur le même commit que celui vers lequel pointe la branche master de equipeun.



Si vous possédez une branche nommée `correctionserveur` sur laquelle vous souhaitez travailler avec d'autres, vous pouvez la pousser de la même manière que vous avez poussé votre première branche. Lancez `git push (serveur distant) (branche)` :

```
$ git push origin correctionserveur
Counting objects: 24, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (15/15), done.
Writing objects: 100% (24/24), 1.91 KiB | 0 bytes/s, done.
Total 24 (delta 2), reused 0 (delta 0)
To https://github.com/schacon/simplegit
* [new branch]
correctionserveur -> correctionserveur
```

Pousser les branches

- Vos branches locales ne sont pas automatiquement synchronisées sur les serveurs distants.
- Vous devez pousser explicitement les branches que vous souhaitez partager.
- Pour pousser une branche :

```
$ git push <distant> <branche>
```

- La prochaine fois qu'un de vos collègues récupérera les données depuis le serveur, il récupérera la branche distante créée.

Pousser les branches

- Il est important de noter que lorsque vous récupérez une nouvelle branche depuis un serveur distant, vous ne créez pas automatiquement une copie locale éditale.
- Mais seulement un pointeur sur la branche distante qui n'est pas directement modifiable.
- Pour fusionner ce travail dans votre branche de travail :

```
$ git merge origin/branchejohnpatrick
```

- Si vous ne souhaitez pas fusionner la branche distante avec votre travail, vous pouvez créer une nouvelle branche et copier le travail dedans :

```
$ git checkout -b branchejohnpatrick origin/branchejohnpatrick
```

```
lassaad@my-leeds-ws:~/repos/formation_docs$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Dépaquetage des objets: 100% (3/3), fait.
Depuis https://github.com/lassaadbaati/formation_docs
d875e88..ed070a5  master    -> origin/master
lassaad@my-leeds-ws:~/repos/formation_docs$ git status
Sur la branche master
Votre branche est en retard sur 'origin/master' de 1 commit, et peut être mise à jour en avance rapide.
(utilisez "git pull" pour mettre à jour votre branche locale)

rien à valider, la copie de travail est propre
lassaad@my-leeds-ws:~/repos/formation_docs$ cat README.md
# formation_docs. add string from master branch
Dépot pour la formation git
Bonjour - derniere ligne
Derniere ligne dans la branche test
ajouter une ligne depuis github
lassaad@my-leeds-ws:~/repos/formation_docs$ git merge origin/master
Mise à jour d875e88..ed070a5
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
lassaad@my-leeds-ws:~/repos/formation_docs$ cat README.md
# formation_docs. add string from master branch
Dépot pour la formation git
Bonjour - derniere ligne
Derniere ligne dans la branche test
ajouter une ligne depuis github
Une nouvelle ligne depuis github
lassaad@my-leeds-ws:~/repos/formation_docs$
```

Créer une branche locale se référent à une branche distante

```
$ git checkout -b correctionserveur origin/correctionserveur
```

```
Branch correctionserveur set up to track remote branch correctionserveur from origin.
```

```
Switched to a new branch 'correctionserveur'
```

Cette commande permet de créer une branche qui se base sur le dépôt distant origin et la branche correctionserveur

```
lassaad@my-leeds-ws:~/repos/formation_docs$ git log --graph --all --oneline
* 1b25b73 (master) commit 9 : add string in line 1 file README from branch master
| * b519d95 (testing) commit 8 : add line to README.md from branch testing
| * 9cd9989 commit 7 : add line to file1 from branch testing
|/
* 3551ef0 commit 6 : Add dir1 et dir2
* de14ced commit 5 : update file A from master
| * c98ca3d (HEAD -> test) commit 4 : Update README.md dans la branche test
| * 654e278 commit 1 : branche test
|/
* 6e30dcf Commit 2: Add files : file1 dir1/file1 AND Update README.md
* 3387779 (origin/master, origin/HEAD) commit 1 : Ajout du fichier Formation Git - Partie 1.pdf
* f5f9a12 Initial commit
```

```
lassaad@my-leeds-ws:~/repos/formation_docs$ git checkout master
Basculement sur la branche 'master'
Votre branche est en avance sur 'origin/master' de 4 commits.
(utilisez "git push" pour publier vos commits locaux)
lassaad@my-leeds-ws:~/repos/formation_docs$ git branch
* master
  test
  testing
lassaad@my-leeds-ws:~/repos/formation_docs$ git status
Sur la branche master
Votre branche est en avance sur 'origin/master' de 4 commits.
(utilisez "git push" pour publier vos commits locaux)

rien à valider, la copie de travail est propre
lassaad@my-leeds-ws:~/repos/formation_docs$
lassaad@my-leeds-ws:~/repos/formation_docs$
lassaad@my-leeds-ws:~/repos/formation_docs$
lassaad@my-leeds-ws:~/repos/formation_docs$ git merge test
Mise à jour 1b25b73..6ec1f59
Fast-forward
 README.md | 1 +
 dir1/file1 | 1 -
 2 files changed, 1 insertion(+), 1 deletion(-)
 delete mode 100644 dir1/file1
```



```

lassaad@my-leeds-ws:~/repos/formation_docs$ git merge test
Mise à jour 1b25b73..6ec1f59
Fast-forward
 README.md | 1 +
 dir1/file1 | 1 -
 2 files changed, 1 insertion(+), 1 deletion(-)
 delete mode 100644 dir1/file1
lassaad@my-leeds-ws:~/repos/formation_docs$ git log --graph --all --oneline
*   6ec1f59 (HEAD -> master, test) commit 10 : commit master branch resoudre conflit
| \
| * 1b25b73 commit 9 : add string in line 1 file README from branch master
| * c98ca3d commit 4 : Update README.md dans la branche test
| * 654e278 commit 1 : branche test
| | * b519d95 (testing) commit 8 : add line to README.md from branch testing
| | * 9cd9989 commit 7 : add line to file1 from branch testing
| | /
| * 3551ef0 commit 6 : Add dir1 et dir2
| * de14ced commit 5 : update file A from master
| /
* 6e30dcf Commit 2: Add files : file1 dir1/file1 AND Update README.md
* 3387779 (origin/master, origin/HEAD) commit 1 : Ajout du fichier Formation Git - Partie 1.pdf
* f5f9a12 Initial commit
lassaad@my-leeds-ws:~/repos/formation_docs$

```

1. Les branches avec git

1.6. Interagir avec les branches distantes

Plan

- Suivre les branches
- Trier une branch (Pulling)
- Suppression de branches distantes

L'extraction d'une branche locale à partir d'une branche distante crée automatiquement ce qu'on appelle une "**branche de suivi**" (**tracking branch**) et la branche qu'elle suit est appelée "**branche amont**" (**upstream branch**).

Les branches de suivi sont des branches locales qui sont en relation directe avec une branche distante.

Si vous vous trouvez sur une branche de suivi et que vous tapez **git push**, Git sélectionne automatiquement le serveur vers lequel pousser vos modifications.

De même, un **git pull** sur une de ces branches récupère toutes les références distantes et fusionne automatiquement la branche distante correspondante dans la branche actuelle.

Suivre les branches

- L'extraction d'une branche locale à partir d'une branche distante crée automatiquement ce qu'on appelle une « branche de suivi ».
- Si nous rentrons la commande 'git push', Git sélectionne automatiquement le serveur vers lequel pousser vos modifications.
- Pour connaître la branche suivie :

```
$ git branch -vv
```

- Pour changer la branche suivie :

```
$ git branch -u origin/nouvellebranchesuivie
```

```
$ git branch --set-upstream-to origin/nouvellebranchesuivie
```

Rattacher une branche locale à une autre dans un dépôt distant

Pour créer une branche locale avec un nom différent de celui de la branche distante, vous pouvez simplement utiliser la première version avec un nom différent de branche locale :

```
$ git checkout -b cs origin/correctionserveur  
Branch cs set up to track remote branch correctionserveur from origin.  
Switched to a new branch 'cs'
```

À présent, votre branche Locale cs Poussera vers Et tirera automatiquement Depuis origin/correctionserveur.

Si vous avez déjà une branche locale et que vous voulez l'associer à une branche distante que vous venez de récupérer ou que vous voulez changer la branche distante que vous suivez, vous pouvez ajouter l'option -u ou --set-upstream-to à la commande git branch à tout moment.

```
$ git branch -u origin/correctionserveur  
Branch correctionserveur set up to track remote branch correctionserveur from origin.
```

Suivre les branches

```
$ git branch -vv  
iss53 7e424c3 [origin/iss53: ahead 2] forgot the brackets  
master 1ae2a45 [origin/master] deploying index fix  
* correctionserveur f8674d9 [equipe1/correction-serveur-ok: ahead 3, behind 1] this should do it  
test 5ea463a trying something new
```

Vous pouvez constater ici que votre branche `iss53` suit `origin/iss53` et est "devant de deux", ce qui signifie qu'il existe deux commits locaux qui n'ont pas été poussés au serveur. On peut aussi voir que la branche `master` suit `origin/master` et est à jour. On peut voir ensuite que notre branche `correctionserveur` suit la branche `correction-serveur-ok` sur notre serveur `equipe1` et est "devant de trois" et "derrière de un", ce qui signifie qu'il existe un commit qui n'a pas été encore intégré localement et trois commits locaux qui n'ont pas été poussés. Finalement, on peut voir que notre branche `test` ne suit aucune branche distante.

Il est important de noter que ces nombres se basent uniquement sur l'état de votre branche distante la dernière fois qu'elle a été synchronisée depuis le serveur. Cette commande n'effectue aucune recherche sur les serveurs et ne travaille que sur les données locales qui ont été mises en cache depuis ces serveurs. Si vous voulez mettre complètement à jour ces nombres, vous devez préalablement synchroniser (fetch) toutes vos branches distantes depuis les serveurs. Vous pouvez le faire de cette façon : `$ git fetch --all; git branch -vv`.

```
lassaad@my-leeds-ws:~/repos/formation_docs$ git remote -v
origin https://github.com/lassaadbaati/formation_docs.git (fetch)
origin https://github.com/lassaadbaati/formation_docs.git (push)
lassaad@my-leeds-ws:~/repos/formation_docs$ git ls-remote origin
338777936a927312975dec2a29e287002a01b851 HEAD
338777936a927312975dec2a29e287002a01b851 refs/heads/master
lassaad@my-leeds-ws:~/repos/formation_docs$ git branch -vv
* master 1b25b73 [origin/master: en avance de 4] commit 9 : add string in line 1 file README from branch master
test c98ca3d commit 4 : Update README.md dans la branche test
testing b519d95 commit 8 : add line to README.md from branch testing
lassaad@my-leeds-ws:~/repos/formation_docs$
```

```
doud@ubuntu:~/work/project-alphorm$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.
```

```
rien à valider, la copie de travail est propre
doud@ubuntu:~/work/project-alphorm$ git branch
iss57
```

```
* master
```

```
doud@ubuntu:~/work/project-alphorm$ git branch -vv
iss57 2a5fece [origin/iss57] app.js: extended = true
* master 2a5fece [origin/master] app.js: extended = true
```

```
doud@ubuntu:~/work/project-alphorm$ git pull
Depuis https://github.com/didouard/project-alphorm
2a5fece..c081ccf master -> origin/master
```

```
Mise à jour 2a5fece..c081ccf
Fast-forward
```

```
app.js | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
doud@ubuntu:~/work/project-alphorm$ gitk --all
```

Tirer une branche (pulling)

Bien que la commande **git fetch** récupère l'ensemble des changements présents sur serveur et qui n'ont pas déjà été rapatriés localement, elle ne modifie en rien le répertoire de travail. Cette commande récupère simplement les données et laisse les fusionner par vous-même.

Cependant, il existe une commande appelée **git pull** qui consiste essentiellement en un **git fetch** immédiatement suivi par un **git merge** dans la plupart des cas. Si vous disposez d'une branche de suivi configurée, soit par une configuration explicite soit en ayant laissé les commandes **clone** ou **checkout** les créer pour vous, **git pull** va examiner quel serveur et quelle branche votre branche courante suit actuellement, synchroniser depuis ce serveur et ensuite essayer de fusionner cette branche distante avec la vôtre.

Il est généralement préférable de simplement utiliser les commandes **fetch** et **merge** explicitement plutôt que de laisser faire la magie de **git pull** qui peut s'avérer source de confusion.

Tirer une branche (pulling)

- La commande '`git fetch`' récupère l'ensemble des changements présents sur le serveur, mais elle ne modifie en rien votre répertoire de travail.
- Il faut faire un '`git merge`' pour fusionner les nouveaux changements.
- La commande '`git pull`' va faire un '`git fetch`' et '`git merge`' d'un coup.

```
$ git pull
```


Suppression de branches distantes

Supposons que vous en avez terminé avec une branche distante – disons que vous et vos collaborateurs avez terminé une fonctionnalité et l’avez fusionnée dans la branche master du serveur distant (ou la branche correspondant à votre code stable). On peut effacer une branche distante en ajoutant `--delete` à `git push`. Si vous souhaitez effacer votre branche `correctionserveur` du serveur :

```
$ git push origin --delete correctionserveur
```

```
To https://github.com/schacon/simplegit
```

```
- [deleted] correctionserveur
```

⇒ Cela ne fait que supprimer le pointeur sur le serveur.

⇒ Le serveur Git garde généralement les données pour un temps jusqu’à ce qu’un processus de nettoyage (garbage collection) passe. De cette manière, si une suppression accidentelle a eu lieu, les données sont souvent très facilement récupérables.

```

lassaad@my-leeds-ws:~/repos/formation_docs$ git log --graph --all --oneline
*   6ec1f59 (HEAD -> master, test) commit 10 : commit master branch resoudre conflit
| \
| * 1b25b73 commit 9 : add string in line 1 file README from branch master
| * | c98ca3d commit 4 : Update README.md dans la branche test
| * | 654e278 commit 1 : branche test
| | * b519d95 (testing) commit 8 : add line to README.md from branch testing
| | * 9cd9989 commit 7 : add line to file1 from branch testing
| | /
| * 3551ef0 commit 6 : Add dir1 et dir2
| * de14ced commit 5 : update file A from master
| /
* 6e30dcf Commit 2: Add files : file1 dir1/file1 AND Update README.md
* 3387779 (origin/master, origin/HEAD) commit 1 : Ajout du fichier Formation Git - Partie 1.pdf
* f5f9a12 Initial commit
lassaad@my-leeds-ws:~/repos/formation_docs$ git fetch
lassaad@my-leeds-ws:~/repos/formation_docs$ git remote -v
origin  https://github.com/lassaadbaati/formation_docs.git (fetch)
origin  https://github.com/lassaadbaati/formation_docs.git (push)
lassaad@my-leeds-ws:~/repos/formation_docs$ git status
Sur la branche master
Votre branche est en avance sur 'origin/master' de 7 commits.
(utilisez "git push" pour publier vos commits locaux)

rien à valider, la copie de travail est propre
lassaad@my-leeds-ws:~/repos/formation_docs$ █

```

Git push Pour mettre à jour le dépôt distant

```
lassaad@my-leeds-ws:~/repos/formation_docs$ git remote -v
origin  https://github.com/lassaadbaati/formation_docs.git (fetch)
origin  https://github.com/lassaadbaati/formation_docs.git (push)
lassaad@my-leeds-ws:~/repos/formation_docs$ git status
Sur la branche master
Votre branche est en avance sur 'origin/master' de 7 commits.
(utilisez "git push" pour publier vos commits locaux)

rien à valider, la copie de travail est propre
lassaad@my-leeds-ws:~/repos/formation_docs$ git push
Username for 'https://github.com': lassaad.baati@gmail.com
Password for 'https://lassaad.baati@gmail.com@github.com':
Décompte des objets: 24, fait.
Delta compression using up to 8 threads.
Compression des objets: 100% (20/20), fait.
Écriture des objets: 100% (24/24), 2.24 KiB | 459.00 KiB/s, fait.
Total 24 (delta 8), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8), completed with 1 local object.
To https://github.com/lassaadbaati/formation_docs.git
   3387779..6ec1f59  master -> master
lassaad@my-leeds-ws:~/repos/formation_docs$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.

rien à valider, la copie de travail est propre
lassaad@my-leeds-ws:~/repos/formation_docs$
```

Git push pour une branche non suivie

```
lassaad@my-leeds-ws:~/repos/formation_docs$ git branch -vv
* master    ed070a5 [origin/master] Update README.md
  test      6ec1f59 commit 10 : commit master branch resoudre conflit
  testing   b519d95 commit 8 : add line to README.md from branch testing
lassaad@my-leeds-ws:~/repos/formation_docs$ git checkout test
Basculement sur la branche 'test'
lassaad@my-leeds-ws:~/repos/formation_docs$
lassaad@my-leeds-ws:~/repos/formation_docs$
lassaad@my-leeds-ws:~/repos/formation_docs$
lassaad@my-leeds-ws:~/repos/formation_docs$
lassaad@my-leeds-ws:~/repos/formation_docs$
lassaad@my-leeds-ws:~/repos/formation_docs$ git push
fatal: La branche courante test n'a pas de branche amont.
Pour pousser la branche courante et définir la distante comme amont, utilisez

    git push --set-upstream origin test

lassaad@my-leeds-ws:~/repos/formation_docs$
```


Push distant

```
lassaad@my-leeds-ws:~/repos/formation_docs$ git branch
master
* test
testing
lassaad@my-leeds-ws:~/repos/formation_docs$ git branch -u origin/testing
error: la branche amont demandée 'origin/testing' n'existe pas
astuce:
astuce: Si vous comptez baser votre travail sur une branche
astuce: amont qui existe déjà sur le serveur distant, vous pouvez
astuce: lancer "git fetch" pour la récupérer.
astuce:
astuce: Si vous comptez pousser une nouvelle branche locale qui suivra
astuce: sa jumelle distante, vous pouvez utiliser "git push -u"
astuce: pour paramétrer le suivi distant en même temps que vous poussez.
lassaad@my-leeds-ws:~/repos/formation_docs$ git push -u origin test:testing
Username for 'https://github.com': lassaad.baati@gmail.com
Password for 'https://lassaad.baati@gmail.com@github.com':
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'testing' on GitHub by visiting:
remote:      https://github.com/lassaadbaati/formation_docs/pull/new/testing
remote:
To https://github.com/lassaadbaati/formation_docs.git
 * [new branch]      test -> testing
La branche 'test' est paramétrée pour suivre la branche distante 'testing' depuis 'origin'.
lassaad@my-leeds-ws:~/repos/formation_docs$
```

1. Les branches avec git

1.7. Rebaser (Rebasing)

Plan

- Introduction
- Les bases
- Rebases plus intéressant
- Les dangers du rebasage
- Rebaser quand vous rebasez
- Rebaser ou fusionner ?

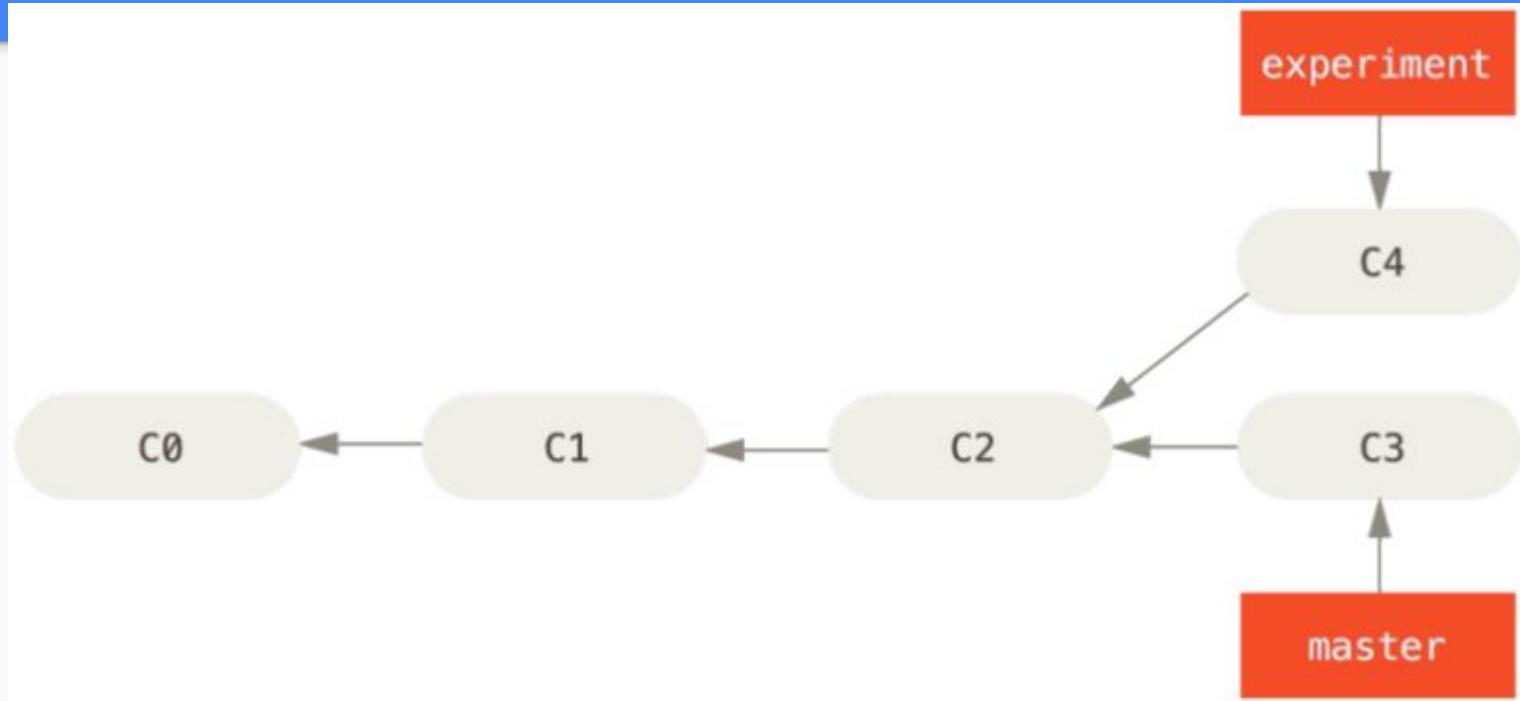
Introduction

- Il y a deux façons d'intégrer des modifications dans une branche :

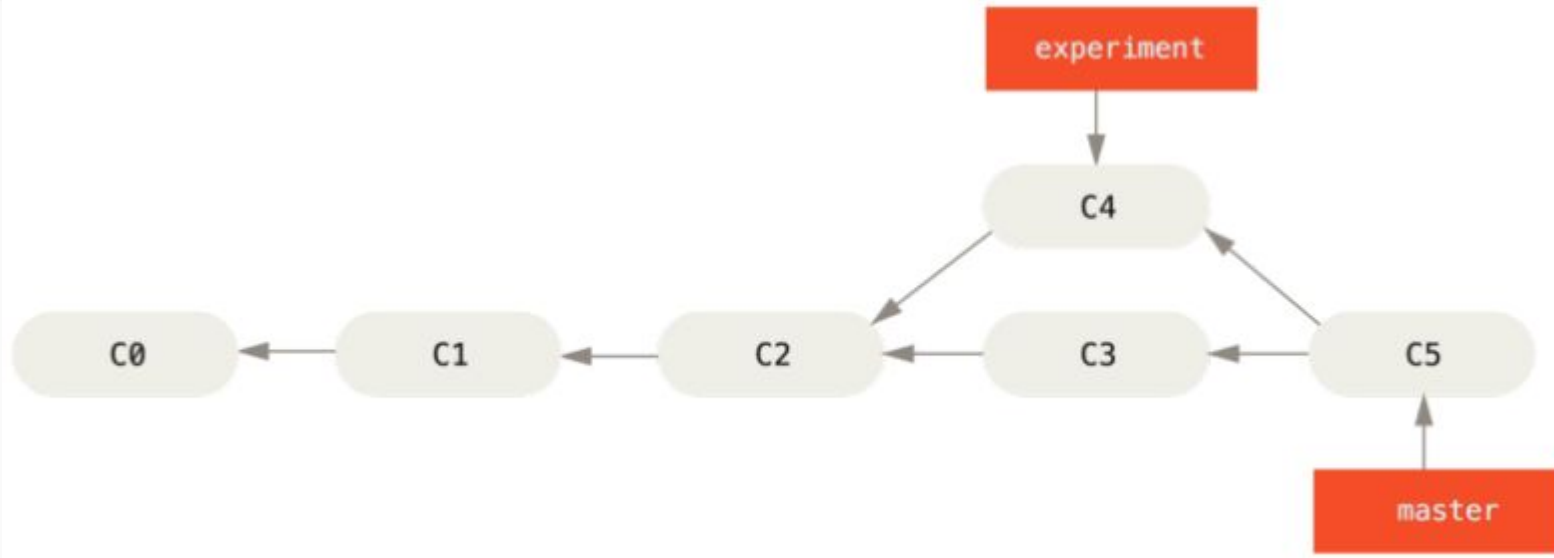
En fusionnant (merge)

En rebasant (rebase)

Historique divergeant simple



Fusion pour intégrer des travaux aux historiques divergeants



Cependant, il existe un autre moyen : vous pouvez prendre **le patch de la modification introduite en C4 et le réappliquer sur C3**. Dans Git, cette action est appelée "**rebaser**" (**rebasing**). Avec la commande rebase, vous pouvez prendre toutes les modifications qui ont été validées sur une branche et les rejouer sur une autre.

Dans cet exemple, vous lanceriez les commandes suivantes :

```
$ git checkout experience
```

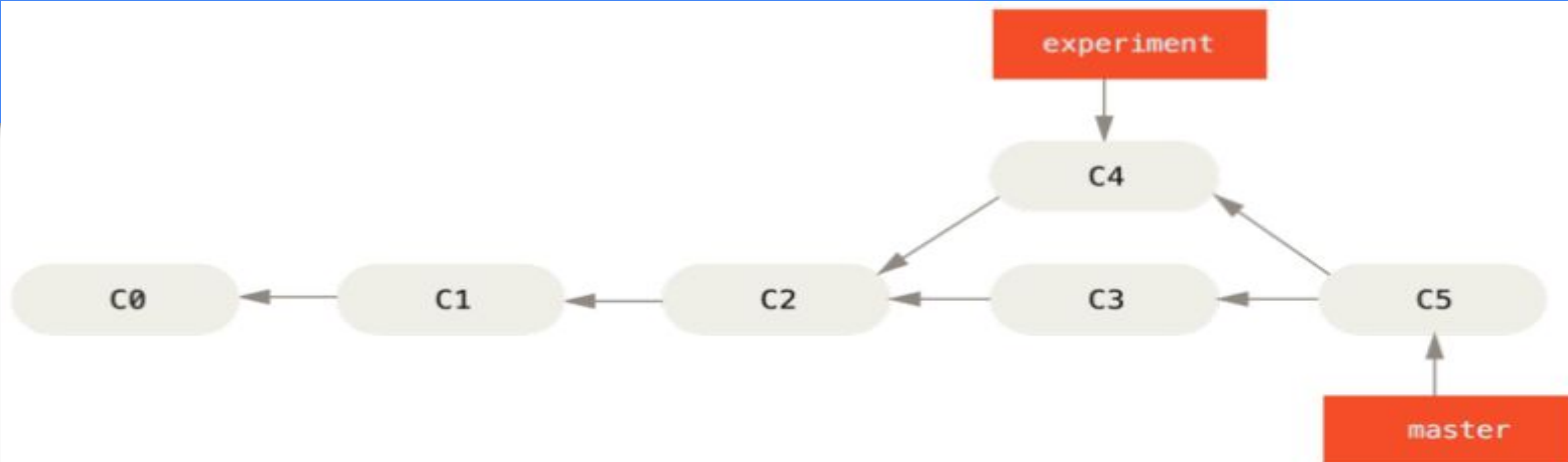
```
$ git rebase master
```

```
First, rewinding head to replay your work on top of it...
```

```
Applying: added staged command
```

Cela fonctionne en cherchant l'ancêtre commun le plus récent des deux branches (celle sur laquelle vous vous trouvez et celle sur laquelle vous rebasez), en récupérant toutes les différences introduites par chaque commit de la branche courante, en les sauvant dans des fichiers temporaires, en réinitialisant la branche courante sur le même commit que la branche de destination et en appliquant finalement chaque modification dans le même ordre.

Rebasage des
modifications
introduites par
C4 sur C3

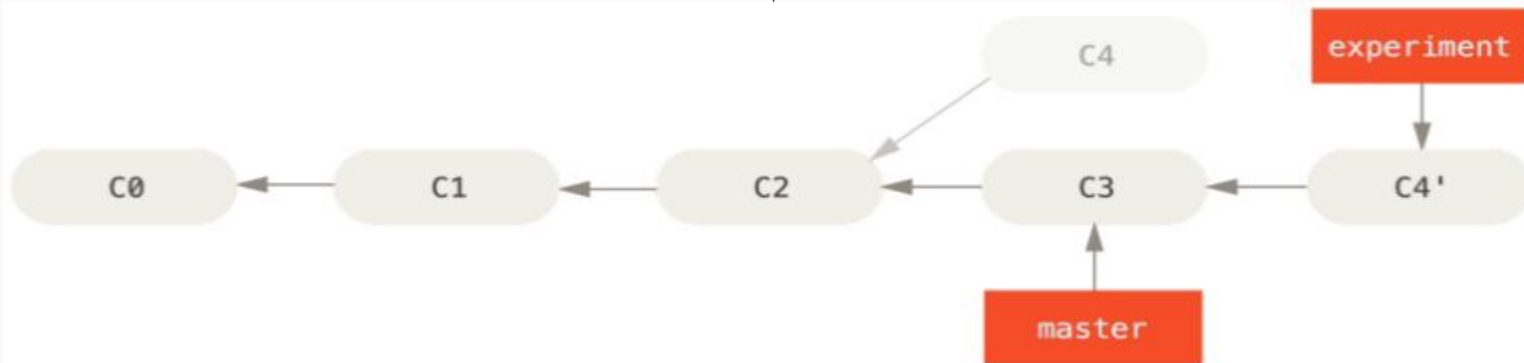


\$ git checkout experience

\$ git rebase master

First, rewinding head to replay your work on top of it...

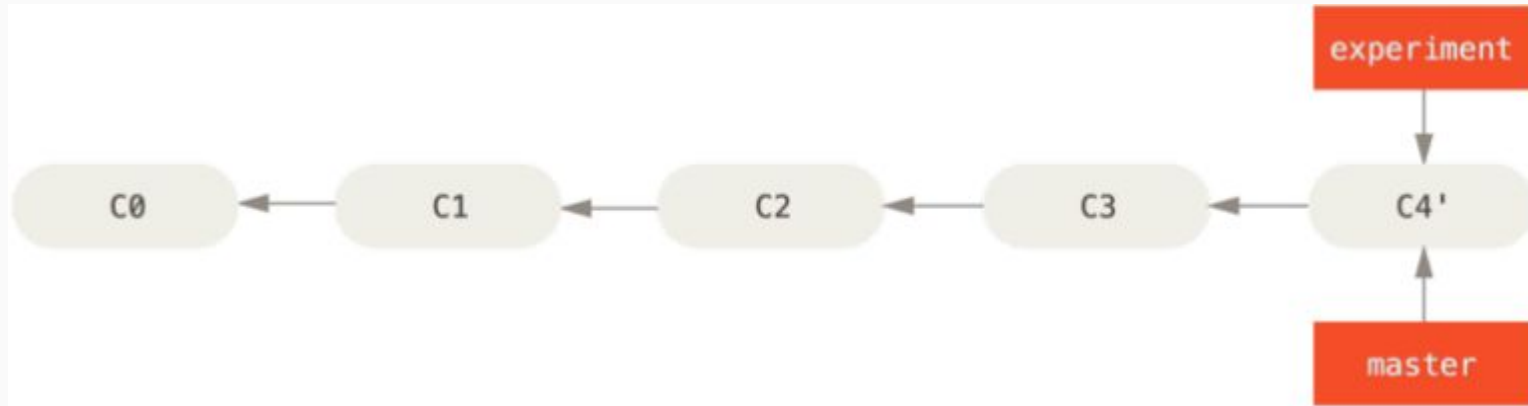
Applying: added staged command



Avance rapide de la branche master

À ce moment, vous pouvez retourner sur la branche master et réaliser une fusion en avance rapide (fast-forward merge).

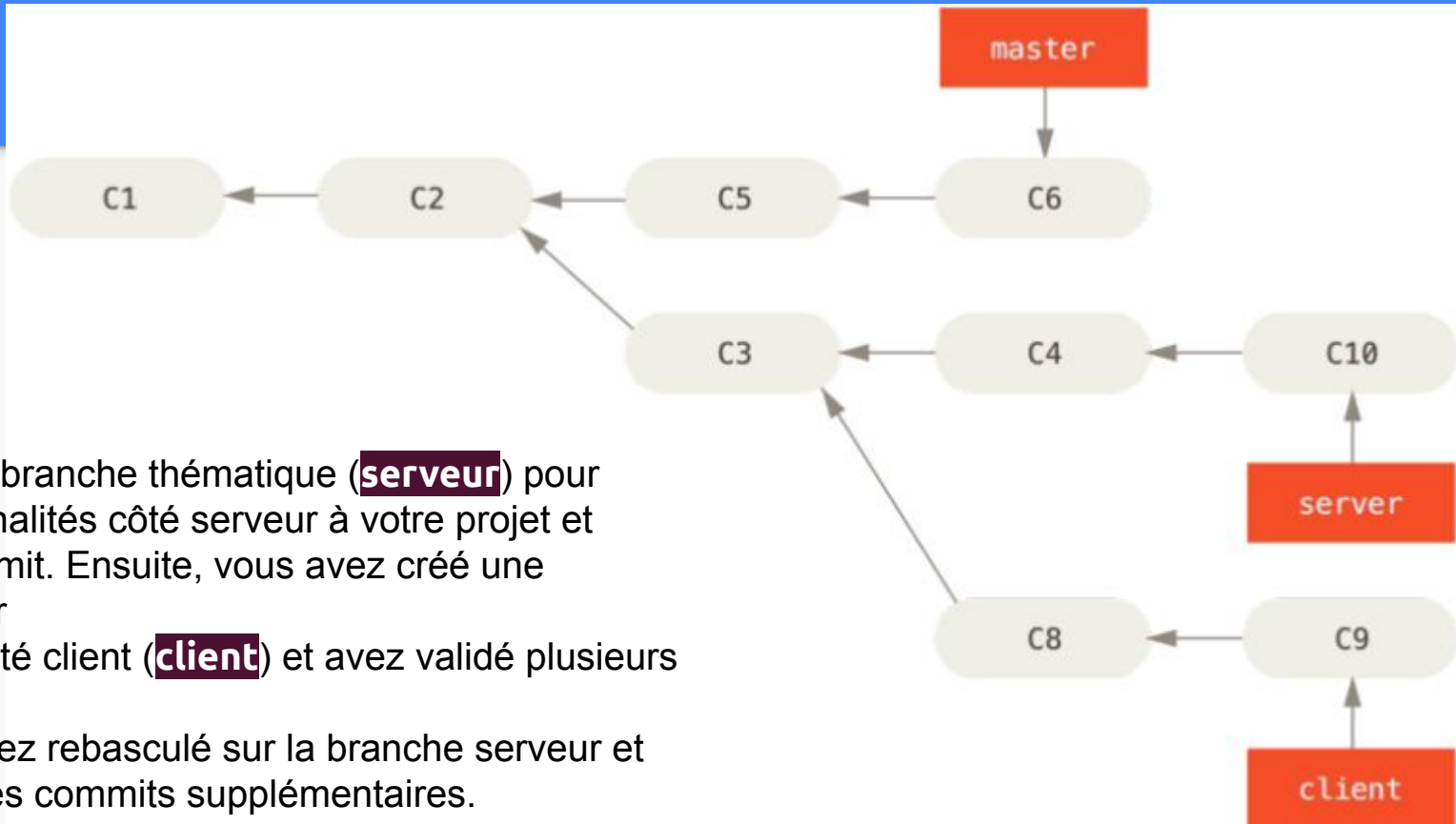
À présent, l'instantané pointé par C4' est exactement le même que celui pointé par C5 dans l'exemple de fusion. Il n'y a pas de différence entre les résultats des deux types d'intégration, mais rebaser rend l'historique plus clair. Si vous examinez le journal de la branche rebasée, elle est devenue linéaire : toutes les modifications apparaissent en série même si elles ont eu lieu en parallèle.



Vous aurez souvent à faire cela pour vous assurer que vos commits s'appliquent proprement sur une branche distante — par exemple, sur un projet où vous souhaitez contribuer mais que vous ne maintenez pas. **Dans ce cas, vous réaliseriez votre travail dans une branche puis vous rebaseriez votre travail sur origin/master quand vous êtes prêt à soumettre vos patches au projet principal.** De cette manière, le mainteneur n'a pas à réaliser de travail d'intégration — juste une avance rapide ou simplement **une application propre.**

Il faut noter que l'instantané pointé par le commit final, qu'il soit le dernier des commits d'une opération de rebasage ou le commit final issu d'une fusion, sont en fait le même instantané — c'est juste que l'historique est différent. Rebaser rejoue les modifications d'une ligne de commits sur une autre dans l'ordre d'apparition, alors que la fusion joint et fusionne les deux têtes.

Un historique avec deux branches thématiques qui sortent l'une de l'autre



Vous avez créé une branche thématique (**serveur**) pour ajouter des fonctionnalités côté serveur à votre projet et avez réalisé un commit. Ensuite, vous avez créé une branche pour ajouter des modifications côté client (**client**) et avez validé plusieurs fois.

Finalement, vous avez rebasculé sur la branche serveur et avez réalisé quelques commits supplémentaires.

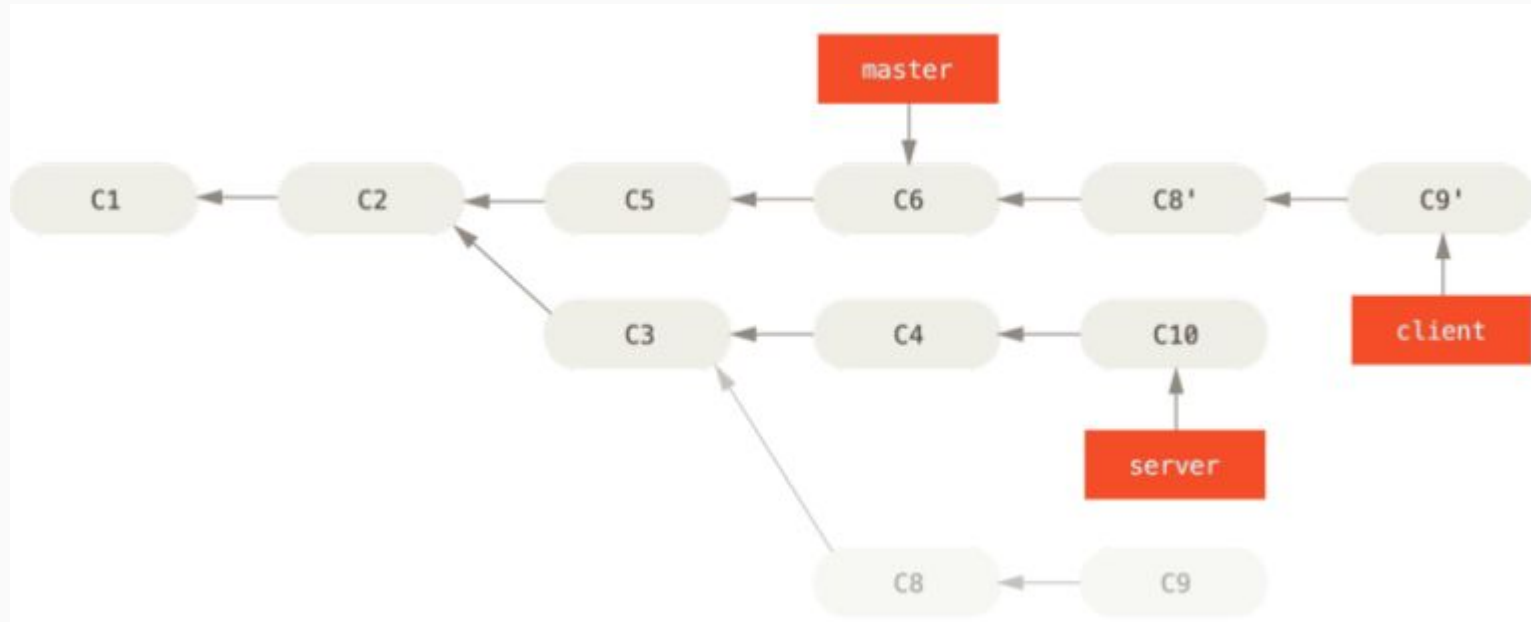
Rebaser deux branches thématiques l'une sur l'autre

Supposons que vous décidez que vous souhaitez fusionner vos modifications du côté **client** dans votre ligne principale pour une publication (release) mais vous souhaitez retenir les modifications de la partie serveur jusqu'à ce qu'elles soient un peu mieux testées. Vous pouvez récupérer les modifications du côté client qui ne sont pas sur le serveur (C8 et C9) et les rejouer sur la branche master en utilisant l'option `--onto` de `git rebase` :

```
$ git rebase --onto master serveur client
```

Cela signifie en substance **"Extraire la branche client, déterminer les patches depuis l'ancêtre commun des branches client et serveur puis les rejouer sur master "**. C'est assez complexe, mais le résultat est assez impressionnant.

Rebaser deux branches thématiques l'une sur l'autre



Maintenant, vous pouvez faire une avance rapide sur votre branche master (cf. Avance rapide sur votre branche master pour inclure les modifications de la branche client):

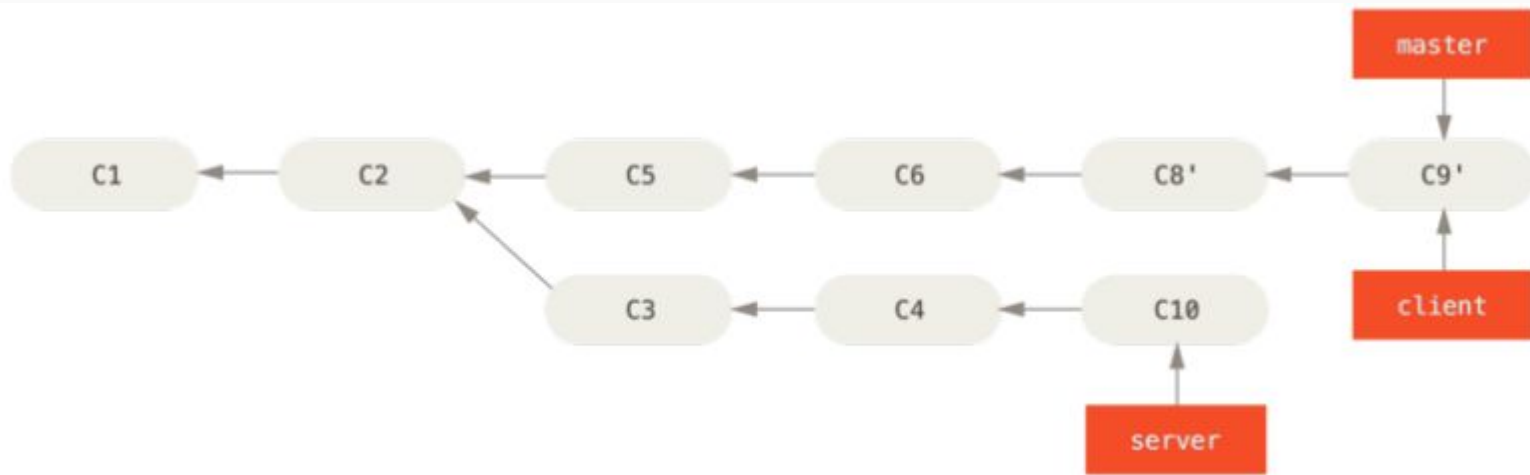
```
$ git checkout master  
$ git merge client
```

Avance rapide sur votre branche master pour inclure les modifications de la branche client

Maintenant, vous pouvez faire une avance rapide sur votre branche master (cf. Avance rapide sur votre branche master pour inclure les modifications de la branche client):

```
$ git checkout master
```

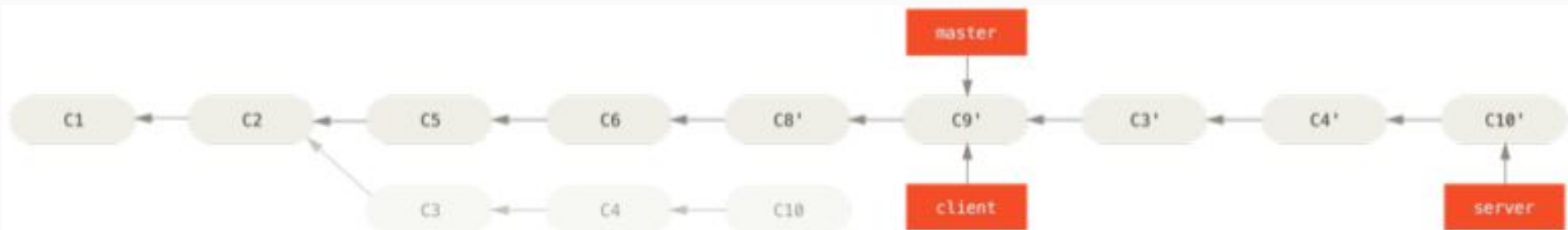
```
$ git merge client
```



Rebasage de la branche serveur sur le sommet de la branche master

```
$ git rebase master serveur
```

Cette commande rejoue les modifications de serveur sur le sommet de la branche master,



Vous pouvez ensuite faire une avance rapide sur la branche de base (master) :

```
$ git checkout master
```

```
$ git merge serveur
```

Vous pouvez effacer les branches client et serveur une fois que tout le travail est intégré

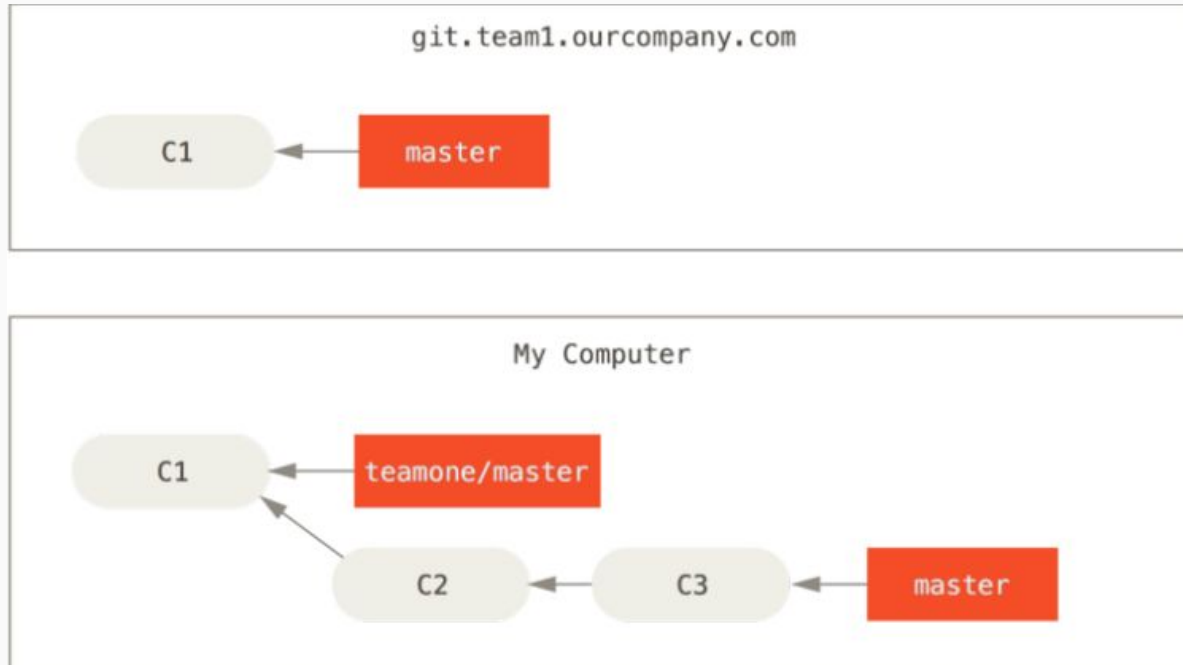
```
$ git branch -d client
```

```
$ git branch -d serveur
```



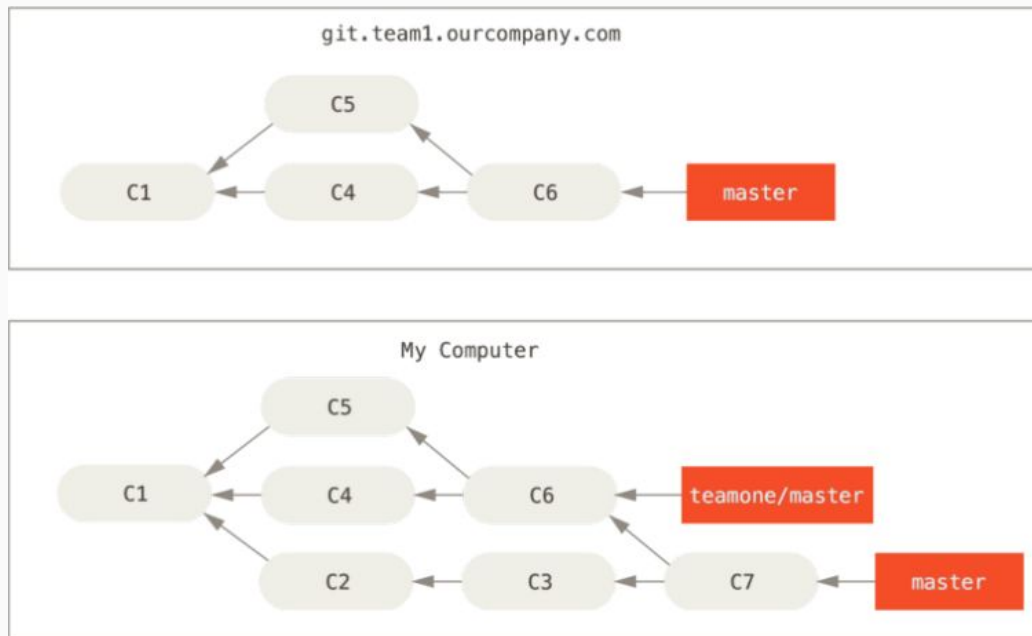
Dangers du rebasage

Cloner un dépôt et baser du travail dessus

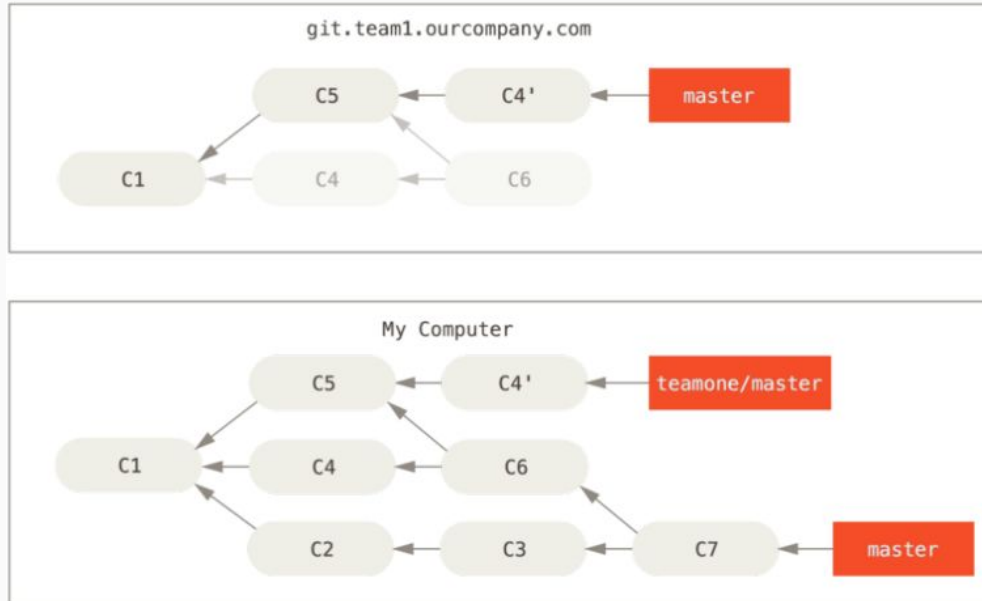


Récupération de commits et fusion dans votre copie

À présent, une autre personne travaille et inclut une fusion, puis elle pousse ce travail sur le serveur central. Vous le récupérez et vous fusionnez la nouvelle branche distante dans votre copie, ce qui donne l'historique suivant :

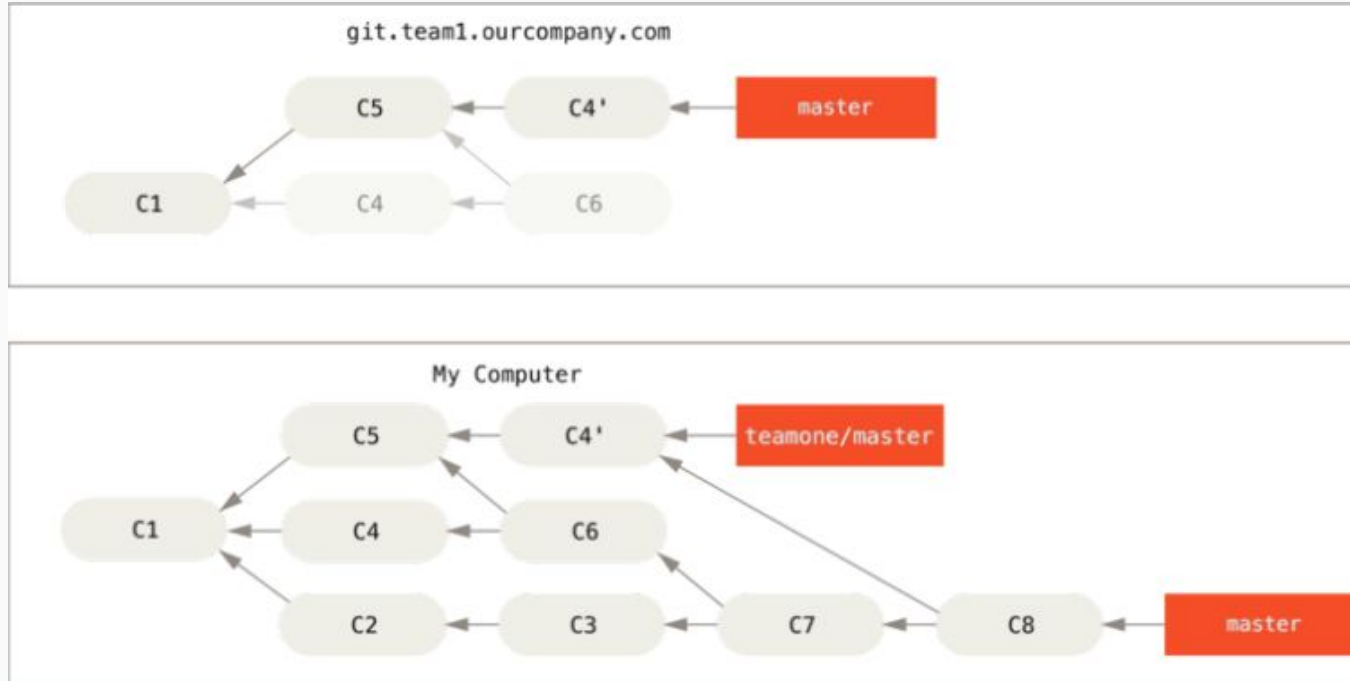


Ensuite, la personne qui a poussé le travail que vous venez de fusionner décide de faire marche arrière et de rebaser son travail. Elle lance un **git push --force** pour forcer l'écrasement de l'historique sur le serveur. Vous récupérez alors les données du serveur, qui vous amènent les nouveaux commits.



Vous fusionnez le même travail une nouvelle fois dans un nouveau commit de fusion

Vous êtes désormais tous les deux dans le pétrin. Si vous faites un git pull, vous allez créer un commit de fusion incluant les deux historiques et votre dépôt ressemblera à ça :



Rebaser ou fusionner

- C'est très important de conserver votre historique intact.
- Une fusion (merge) va créer un commit
- Un rebase va réagencer les commits et suivre l'historique

Github & BitBucket

- Créons-nous un compte sur : <https://github.com>
- N'oubliez pas de vérifier votre compte grâce à l'email que vous avez reçu.

Accès par SSH

- Si vous voulez joindre les serveurs de github par SSH, il faut ajouter vos clefs SSH.
- Setting -> SSH keys
- Et vous ajoutez votre clef publique.

Votre adresse électronique

- Github utilise les adresses électroniques pour faire correspondre les commits Git aux utilisateurs.
- Pour ajouter les adresses électroniques :

Setting -> Emails

Authentification en 2 facteurs

- L'authentification à deux facteurs est un mécanisme d'authentification qui est devenu très populaire récemment pour réduire les risques de corruption de votre compte si votre mot de passe est dérobé.

Setting -> Security

- Soit par une application
- Soit par un SMS