3.6 Les branches avec Git - Rebaser

Rebaser

Dans Git, il y a deux façons d'intégrer les modifications d'une branche dans une autre : en fusionnant (`merge`) et en rebasant (`rebase`). Dans ce chapitre, vous apprendrez la signification de rebaser, comment le faire, pourquoi c'est un outil plutôt ébouriffant et dans quels cas il est déconseillé de l'utiliser.

Les bases

Si vous revenez à un exemple précédent du chapitre sur la fusion (voir la figure 3-27), vous remarquerez que votre travail a divergé et que vous avez ajouté des *commits* sur deux branches différentes.

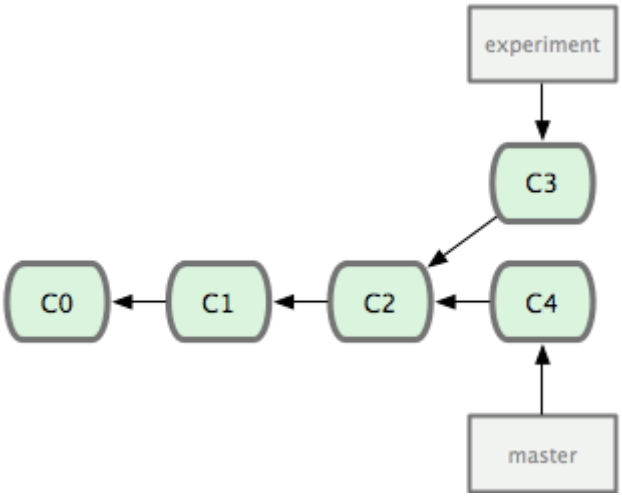

Figure 3-27. Votre historique divergent initial.

Comme nous l'avons déjà expliqué, le moyen le plus simple pour intégrer ensemble ces branches est la fusion via la commande `merge`. Cette commande réalise une fusion à trois branches entre les deux derniers instantanés de chaque branche (C3 et C4) et l'ancêtre commun le plus récent (C2), créant un nouvel instantané (et un *commit*), comme montré par la figure 3-28.

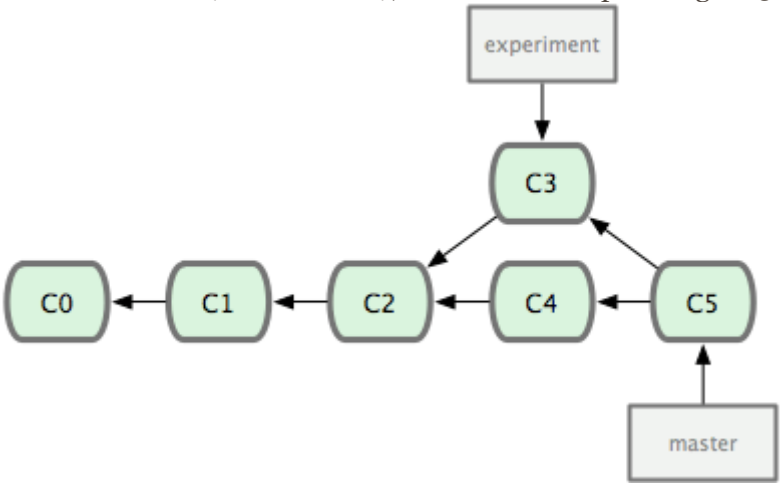

Figure 3-28. Fusion d'une branche pour intégrer les historiques divergents.
Cependant, il existe un autre moyen : vous pouvez prendre le patch de la modification introduite en C3 et le réappliquer sur C4. Dans Git, cette action est appelée *rebaser*. Avec la commande `rebase`, vous prenez toutes les modifications qui ont été validées sur une branche et vous les rejouez 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 entre chaque validation de la branche sur laquelle vous êtes, en les sauvant dans des fichiers temporaires, en basculant sur la branche destination et en réappliquant chaque modification dans le même ordre. La figure 3-29 illustre ce processus.

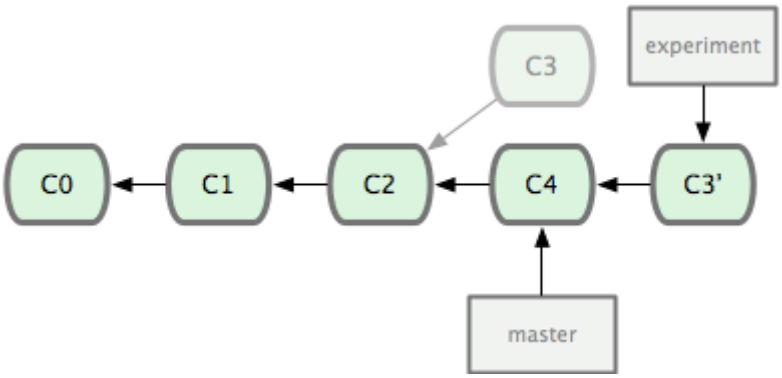

Figure 3-29. Rebaser les modifications introduites par C3 sur C4.
À ce moment, vous pouvez retourner sur la branche `master` et réaliser une fusion en avance rapide (voir figure 3-30).

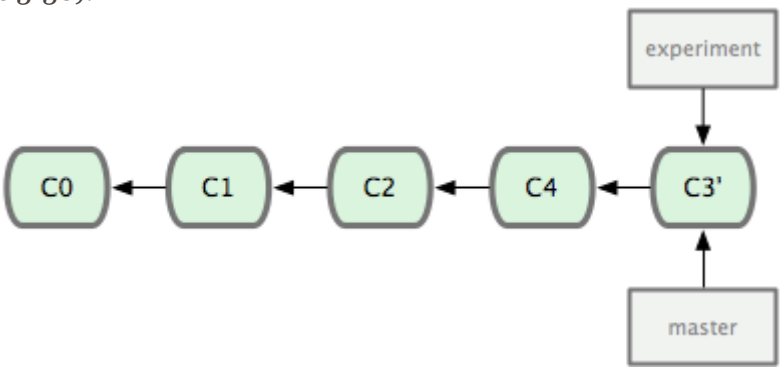

Figure 3-30. Avance rapide sur la branche `master`.

À présent, l'instantané pointé par C3' 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 rebaser pour vous assurer que les patchs que vous envoyez s'appliquent correctement 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 patchs 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 rebase 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.

Rebasages plus intéressants

Vous pouvez aussi faire rejouer votre rebasage sur autre chose qu'une branche. Prenez l'historique de la figure 3-31 par exemple. Vous avez créé une branche pour un sujet spécifique (`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.

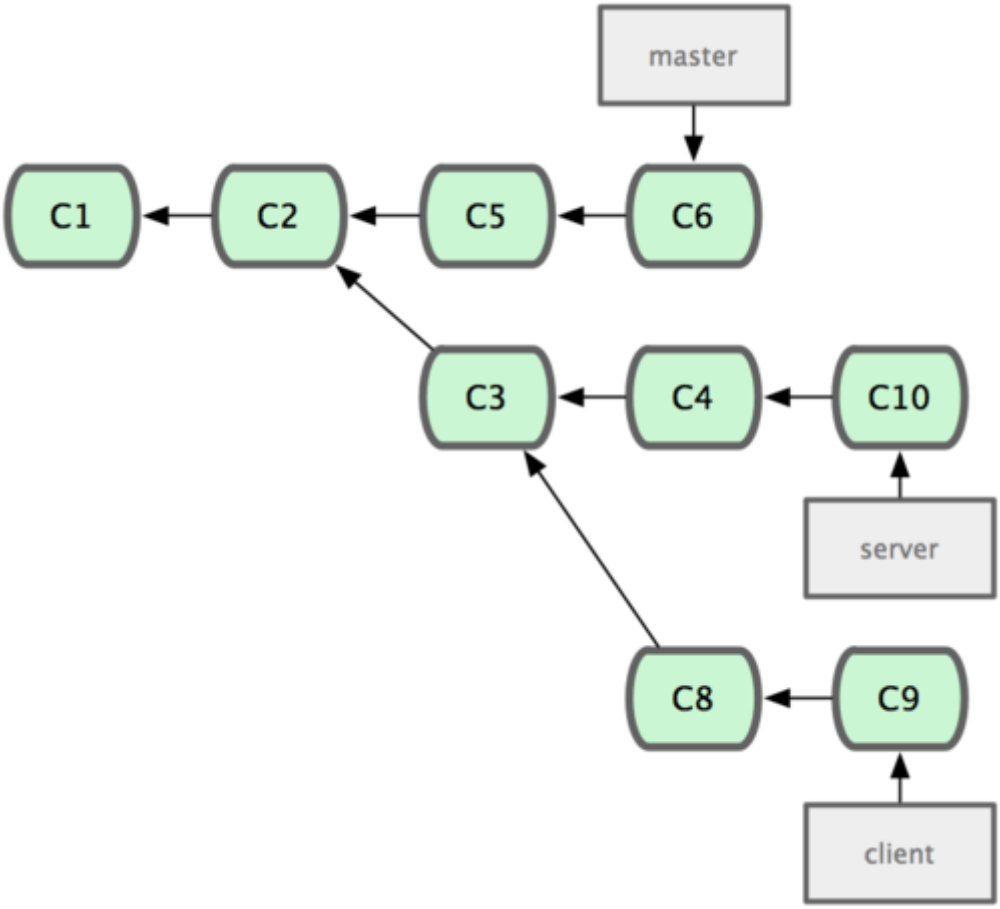

Figure 3-31. Un historique avec une branche qui sort d'une autre branche thématique.

Supposons que vous décidez que vous souhaitez fusionner vos modifications pour le côté client dans votre ligne principale pour une publication mais vous souhaitez retenir les modifications pour la partie serveur jusqu'à ce qu'elles soient un peu plus testées. Vous pouvez récupérer les modifications pour le 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 essence « Extraire la branche client, déterminer les patchs depuis l'ancêtre commun des branches `client` et `serveur` puis les rejouer sur `master` ». C'est assez complexe, mais le résultat visible sur la figure 3-32 est assez impressionnant.

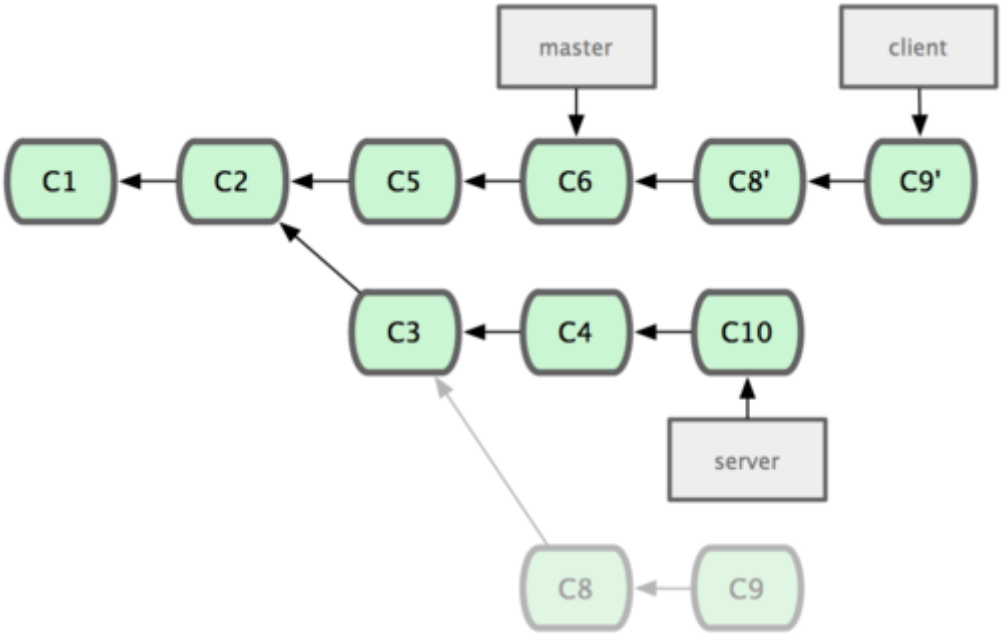

Figure 3-32. Rebaser une branche thématique sur une autre branche.

Maintenant, vous pouvez faire une avance rapide sur votre branche `master` (voir figure 3-33) :

```
$ git checkout master

$ git merge client
```

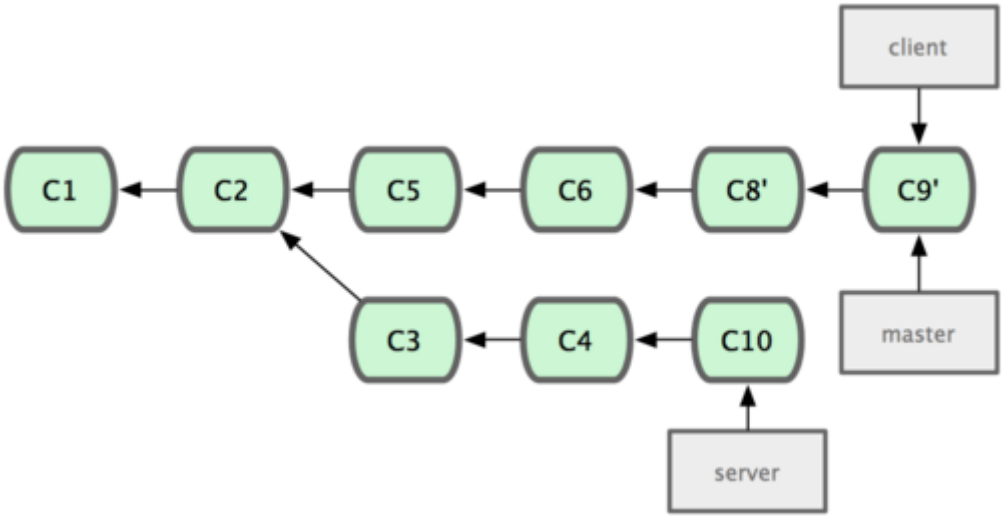

Figure 3-33. Avance rapide sur votre branche `master` pour inclure les modifications de la branche client.

Supposons que vous décidiez de tirer votre branche `serveur` aussi. Vous pouvez rebaser la branche `serveur` sur la branche `master` sans avoir à l'extraire avant en utilisant `git rebase [branchedebase] [branchedesujet]` — qui extrait la branche thématique (dans notre cas, `serveur`) pour vous et la rejoue sur la branche de base (`master`) :

```
$ git rebase master serveur
```

Cette commande rejoue les modifications de `serveur` sur le sommet de la branche `master`, comme indiqué dans la figure 3-34.

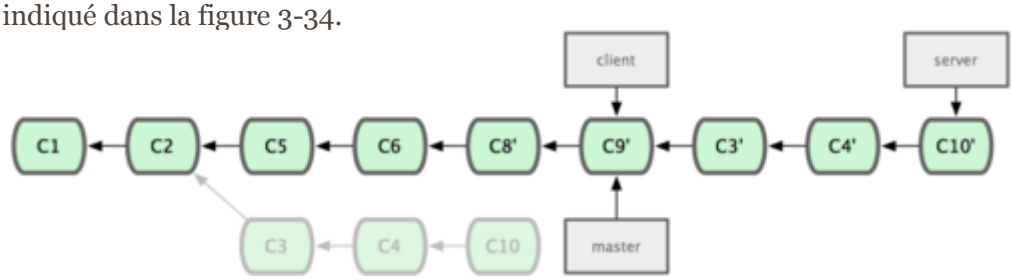

Figure 3-34. Rebaser la branche serveur sur le sommet de la branche `master`.
Ensuite, vous pouvez faire une avance rapide sur la branche de base (`master`) :

```
$ git checkout master

$ git merge serveur
```

Vous pouvez effacer les branches `client` et `serveur` une fois que tout le travail est intégré et que vous n'en avez plus besoin, éliminant tout l'historique de ce processus, comme visible sur la figure 3-35 :

```
$ git branch -d client

$ git branch -d serveur
```

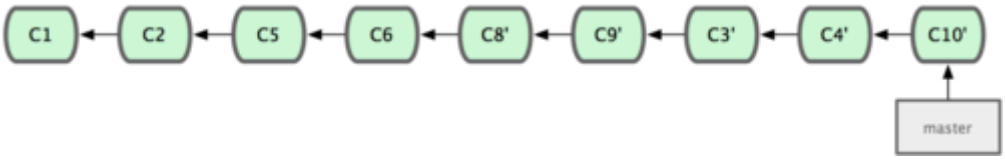

Figure 3-35. L'historique final des *commits*.

Les dangers de rebaser

Ah... mais les joies de rebaser ne viennent pas sans leurs contreparties, qui peuvent être résumées en une ligne :

Ne rebasez jamais des *commits* qui ont déjà été poussés sur un dépôt public.

Si vous suivez ce conseil, tout ira bien. Sinon, de nombreuses personnes vont vous haïr et vous serez méprisé par vos amis et votre famille.

Quand vous rebasez des données, vous abandonnez les *commits* existants et vous en créez de nouveaux qui sont similaires mais différents. Si vous poussez des *commits* quelque part, que d'autres les tirent et se basent dessus pour travailler, et qu'après coup, vous réécrivez ces *commits* à l'aide de `git rebase` et les poussez à nouveau, vos collaborateurs devront re-fusionner leur travail et les choses peuvent rapidement devenir très désordonnées quand vous essaierez de tirer leur travail dans votre dépôt.

Examinons un exemple expliquant comment rebaser un travail déjà publié sur un dépôt public peut générer des gros problèmes. Supposons que vous clonez un dépôt depuis un serveur central et réalisez quelques travaux dessus. Votre historique de *commits* ressemble à la figure 3-36.

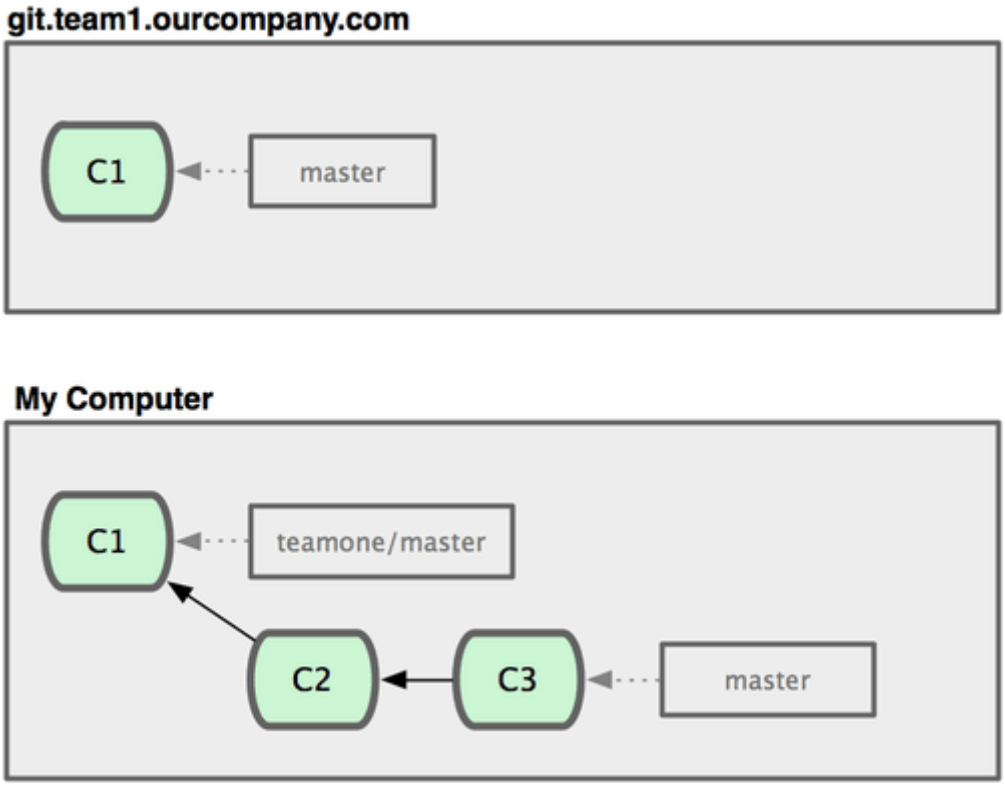

Figure 3-36. Cloner un dépôt et baser du travail dessus.
À présent, une autre personne travaille et inclut une fusion, puis elle pousse ce travail sur le serveur central. Vous le récupérez et vous fusionnez la nouvelle branche distante dans votre copie, ce qui donne l'historique de la figure 3-37.

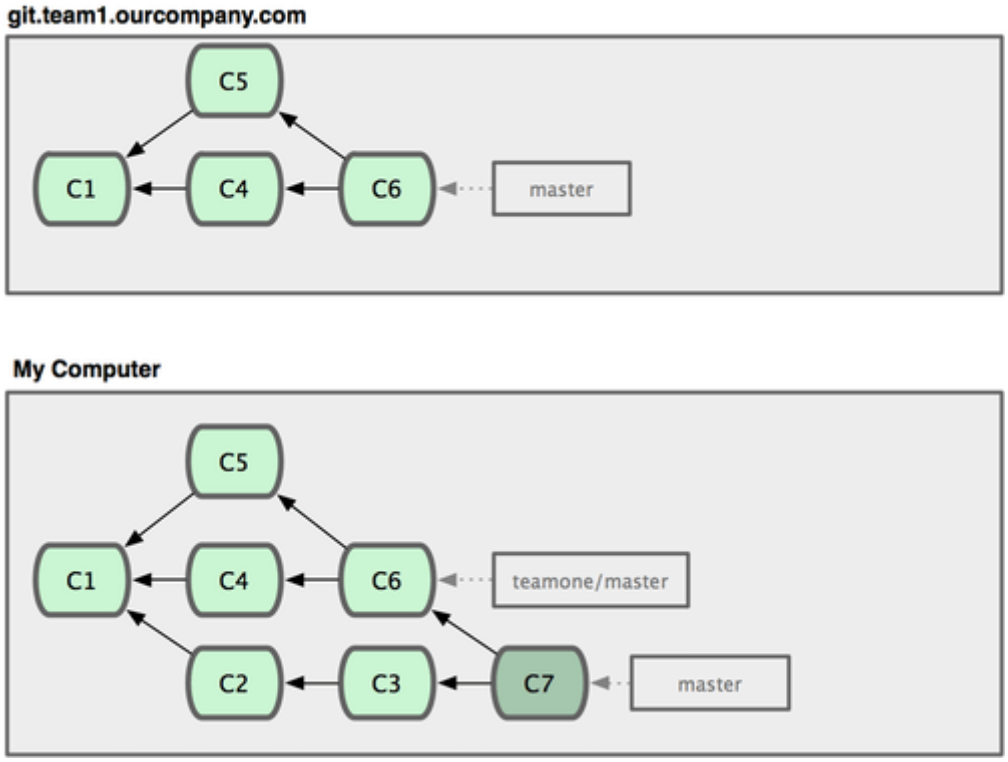

Figure 3-37. Récupération de *commits* et fusion dans votre copie.

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*.

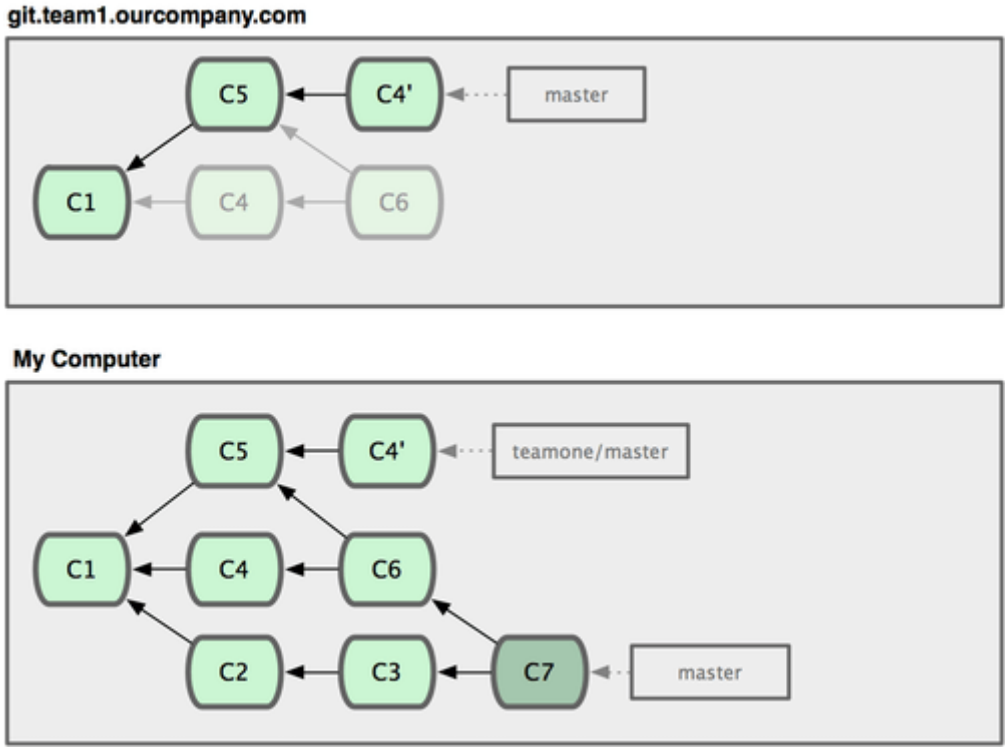

Figure 3-38. Quelqu'un pousse des *commits* rebasés, en abandonnant les *commits* sur lesquels vous avez fondé votre travail.

À ce moment, vous devez fusionner son travail une nouvelle fois, même si vous l'avez déjà fait. Rebaser change les empreintes SHA-1 de ces *commits*, ce qui les rend nouveaux aux yeux de Git, alors qu'en fait, vous avez déjà le travail de C4 dans votre historique (voir figure 3-39).

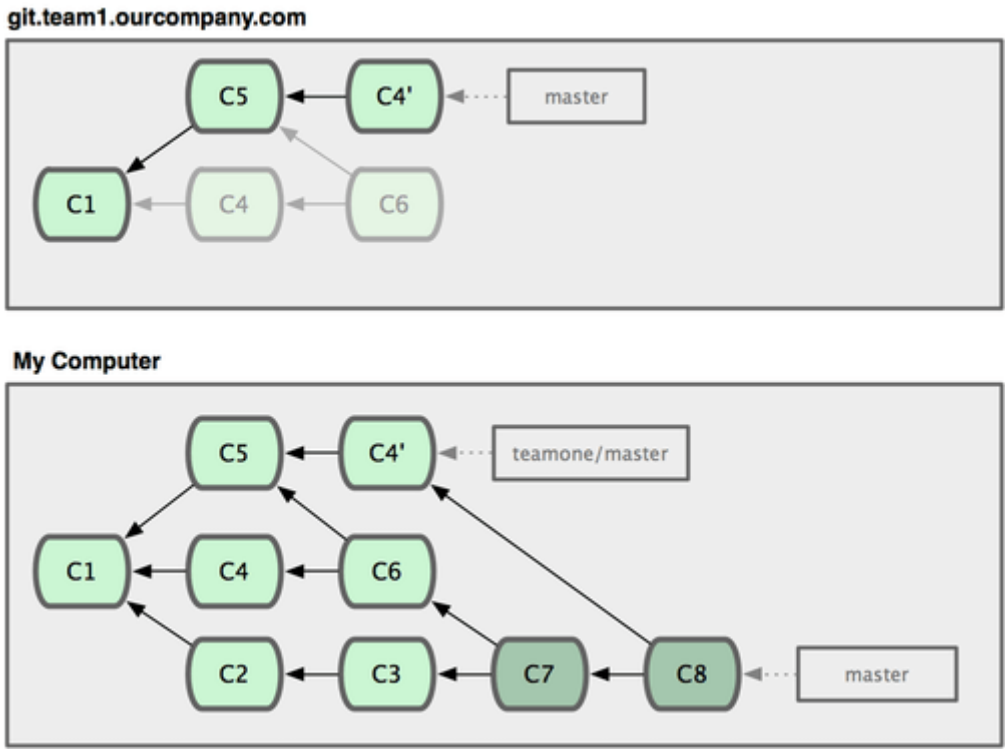

Figure 3-39. Vous fusionnez le même travail une nouvelle fois dans un nouveau *commit* de fusion.

Vous devez fusionner ce travail pour pouvoir continuer à suivre ce développeur dans le futur. Après fusion, votre historique contient à la fois les *commits* C4 et C4', qui ont des empreintes SHA-1 différentes mais introduisent les même modifications et ont les mêmes messages de validation. Si vous lancez `git log` lorsque votre historique ressemble à ceci, vous verrez deux *commits* qui ont la même date d'auteur et les mêmes messages, ce qui est déroutant. De plus, si vous poussez cet historique sur le serveur, vous réintroduirez tous ces *commits* rebasés sur le serveur central, ce qui va encore plus dérouter les autres développeurs.

Si vous considérez le fait de rebaser comme un moyen de nettoyer et réarranger des *commits* avant de les pousser et si vous vous en tenez à ne rebaser que des *commits* qui n'ont jamais été publiés, tout ira bien. Si vous tentez de rebaser des *commits* déjà publiés sur lesquels les gens ont déjà basé leur travail, vous allez au devant de gros problèmes énervants.