Eskimon

Zeste de savoir

Refaire l'histoire avec git

12 août 2019

Table des matières

Lors d'un projet informatique, gérer ses sources et leur historique est crucial. C'est pourquoi des logiciels comme **git** ont vu le jour, afin de suivre les versions et modifications.

Dans le meilleur des mondes, chaque commit est bien exécuté et les choses sont faites dans l'ordre. Les fichiers ne sont pas édités sans arrêt, et donc tous les changements sont bien atomiques. Ça, c'est la théorie...

Dans la pratique, il peut arriver que l'on ait besoin de toucher à l'historique git pour plusieurs raisons, comme fusionner des commits entre eux (pour n'en avoir plus qu'un) ou encore fusionner une branche avec une autre, pour suivre les changements de l'une dans l'autre. Eh bien tout cela peut se résumer en une commande : **rebase**.

Ce tutoriel **introductif** va vous proposer de **découvrir** et mettre en pratique ces différents aspects au travers d'un exemple simple et complet. À la fin de ce tutoriel, vous saurez :

— déplacer des commits pour réécrire un historique ;
— fusionner deux commits entre eux ;
— fusionner une branche dans une autre proprement.

i

Pour suivre ce tutoriel, quelques pré-requis sont nécessaires :
— savoir se servir de git et ses opérations de base (commit, add, branch) ;
— comprendre les concepts de branche et d'historique.

!

Dans ce tutoriel nous allons toucher l'historique d'un dépôt git. Bien que cela soit souvent sans risque pour votre dépôt, les opérations que nous allons faire peuvent tout autant très bien casser votre historique de modifications, si vous n'êtes pas assez vigilant.
Faites donc attention lorsque vous les appliquez sur vos projets. Évitez aussi tant que possible de faire ce genre d'opérations une fois vos changements poussés sur le serveur commun, pour ne pas embrouiller le travail de vos collègues.

Contenu masqué n°1

1. Préparation de notre environnement

1.1. La situation

Cette année Bob s'organise, il se fait des fiches de révision pour son examen d'histoire. Du coup, en bon programmeur, il décide de les rédiger en markdown pour pouvoir les enrichir avec des illustrations et autres bricoles.

Il a ainsi rédigé 5 fichiers différents pour 5 périodes différentes :

— la révolution industrielle (la_revolution_industrielle.md) ;
— la Première Guerre mondiale (la_premiere_guerre_mondiale.md) ;
— l'entre-deux-guerres (entre_guerre.md) ;
— la Seconde Guerre mondiale (la_seconde_guerre_mondiale.md) ;
— les Trente Glorieuses (30_glorieuses.md).

Seulement il y a un problème, Bob a été malade pendant l'année. Du coup, ses fichiers ont été écrits dans le désordre : il a rédigé ses cours au fur et à mesure puis, quand il avait du temps, il a fait du rattrapage en prenant les cours d'Alice.

Il se retrouve alors avec l'historique de commits git suivant :

```
218bfcb Conclusion de la première guerre mondiale - rattrapage
b1cc9fd L'entre deux guerres
70c08d1 Début de la seconde guerre mondiale - rattrapage
ce99651 Les 30 glorieuses
61f6523 La seconde guerre mondiale - fin
b52fa42 La première guerre mondiale
8c310df La révolution industrielle
47901da Ajout des titres des parties
5b56868 Creation des fichiers du programme d'histoire
```

C'est cet historique que nous allons essayer de réordonner. On peut voir que les premiers commits correspondent au début du programme scolaire, puis ensuite Bob a été malade. Il a alors fait une pause, puis repris le cours dans l'ordre. Ensuite, il a rattrapé les cours manquants en recopiant les notes d'un autre élève, en partant du plus récent dans le programme au plus vieux.

L'historique montrée est antéchronologique, le commit le plus récent (celui que l'on vient de faire) est en haut de la liste, le plus vieux est en bas.

1.2. Le dépôt exemple

Afin que vous puissiez faire des tests chez vous, je vous mets à disposition un dépôt git possédant cette situation initiale. Vous pouvez le récupérer dans cette archive .

Dans ce dépôt, vous devez trouver deux branches qui ont pour noms Bob et Alice. La branche Bob possède les écrits de Bob, tandis que celle d'Alice possède quelques anecdotes qu'elle a voulu lui donner plus tard. Pour l'instant, concentrez-vous sur la branche de Bob.

```
git checkout Bob
```

Vous pouvez alors vérifier les commits et leur mauvais rangement. Vous devriez obtenir la liste que l'on a vue plus tôt.

```
git log --pretty=oneline --abbrev-commit
```

J'utiliserai la commande ci-dessus via un alias `git logo` (pour "log oneline"). Vous pouvez le rajouter en éditant votre fichier `.gitconfig` ou via la commande `git config --global alias.logo 'log --pretty=oneline --abbrev-commit'`.

Contenu masqué n°2

Si git vous ennuie à cause des droits sur les fichiers (compatibilité entre Windows, Linux, etc.) alors entrez la commande suivante, qui vous permettra de désactiver la surveillance du changement des droits : `git config core.filemode false`

1.3. Les objectifs

Nos objectifs seront donc multiples. Afin de conserver un dépôt propre, nous allons effectuer une à une les opérations suivantes :

— mise dans l'ordre chronologique du cours des différents fichiers (car je suis maniaque) ;
— fusion des commits consécutifs traitant du même fichier et de la même partie "logique" ;
— fusion de la branche d'Alice pour "enrichir" notre branche de son contenu.

À la fin, vous serez passés maîtres de la commande `rebase` et de ses différents cas d'utilisations.

2. Déplacer des commits

À l'heure actuelle, on a un historique un peu farfelu. Dans l'idéal, on aimerait que les éléments se suivent chronologiquement, voire que l'on ait uniquement un seul commit par période.

Ainsi, on va essayer d'obtenir le résultat suivant :

```
----------- AVANT -----------
218bfcb Conclusion de la première guerre mondiale - rattrapage
b1cc9fd L'entre deux guerres
70c08d1 Début de la seconde guerre mondiale - rattrapage
ce99651 Les 30 glorieuses
61f6523 La seconde guerre mondiale - fin
b52fa42 La première guerre mondiale
8c310df La révolution industrielle
47901da Ajout des titres des parties
5b56868 Creation des fichiers du programme d'histoire

----------- APRES -----------
ce99651 Les 30 glorieuses
61f6523 La seconde guerre mondiale - fin
70c08d1 Début de la seconde guerre mondiale - rattrapage
b1cc9fd L'entre deux guerres
218bfcb Conclusion de la première guerre mondiale - rattrapage
b52fa42 La première guerre mondiale
8c310df La révolution industrielle
47901da Ajout des titres des parties
5b56868 Creation des fichiers du programme d'histoire
```

Comme vous pouvez le constater, de nombreux commits ont littéralement changé de place ! C'est ça que nous allons faire ici, déplacer des commits !

Et aussi impressionnant que cela puisse paraître, il va suffire d'utiliser une seule commande à bon escient pour le faire : `git rebase`. Mais attention, on ne l'utilise pas n'importe comment.

Pour l'utiliser, on va devoir lui spécifier le commit **le plus ancien** devant rester tel quel. Dans notre cas, nous souhaitons tout remettre en ordre jusqu'à "La premiere guerre mondiale" (b52fa42) qui, lui, ne bouge pas. On va alors lancer le rebase **en mode interactif** jusqu'à ce commit :

```
git rebase -i b52fa42^
```

Attention à ne pas oublier l'option `-i` pour le mode **interactif** ainsi que le `^` après l'identifiant du commit ! Ce dernier sert à indiquer que l'on veut remonter jusqu'à ce commit *inclus*.

Une nouvelle fenêtre s'ouvre alors avec plein de choses passionnantes :

```
pick b52fa42 La première guerre mondiale
pick 61f6523 La seconde guerre mondiale - fin
pick ce99651 Les 30 glorieuses
pick 70c08d1 Début de la seconde guerre mondiale - rattrapage
pick b1cc9fd L'entre deux guerres
pick 218bfcb Conclusion de la première guerre mondiale - rattrapage

# Rebase 8c310df..218bfcb onto 8c310df
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
```

Si vous comprenez l'anglais, vous avez peut-être déjà deviné comment faire. Sinon, voici la procédure.

Dans cette affichage, vous avez la liste des commits jusqu'au dernier que vous souhaitez garder tel quel. L'opération est maintenant simple, il va falloir déplacer les lignes pour les mettre dans l'ordre que vous voulez. L'ordre en question sera celui que l'on a vu juste au-dessus. Laissez les "pick" en début de ligne, ils sont là pour signifier que vous souhaitez utiliser le commit en question.

Vous avez réussi ? Très bien, vous devriez obtenir quelque chose comme ça avant de valider :

```
pick b52fa42 La première guerre mondiale
pick 218bfcb Conclusion de la première guerre mondiale - rattrapage
pick b1cc9fd L'entre deux guerres
pick 70c08d1 Début de la seconde guerre mondiale - rattrapage
pick 61f6523 La seconde guerre mondiale - fin
pick ce99651 Les 30 glorieuses

# Et en dessous le blabla précédent
```

Sauvegardez puis quittez l'éditeur. Le rebase se lance alors automatiquement... et vous crie dessus, c'est un échec !

? Que s'est-il passé ?

Si vous utilisez la commande `git status` vous allez voir qu'il existe un conflit sur le fichier "la_seconde_guerre_mondiale.md". En l'ouvrant, vous verrez des marqueurs `<<<<<<<` et `>>>>>>>` que git a rajoutés pour vous signaler les endroits où il n'arrive pas à faire une chose logique.

C'est donc à vous de jouer en éditant manuellement le fichier, pour qu'il ait l'allure escomptée. En l'occurrence, c'est simplement une ligne blanche qui l'ennuie. Supprimez-là, ainsi que les marqueurs, puis sauvegarder le fichier.

Nous allons maintenant signaler à git que le conflit est résolu en faisant un :

```
git add la_seconde_guerre_mondiale.md
```

Cela nous permet de rajouter le fichier dans l'index, puis on lui demande de continuer le rebase avec :

```
git rebase --continue
```

Git vous demandera alors de confirmer le message de commit (ce que vous ferez), puis continuera son bonhomme de chemin.

Un autre conflit similaire apparait alors, résolvez-le de la même manière.

À la fin, git doit afficher le message `Successfully rebased and updated refs/heads/Bob.` pour nous informer que tout va bien.

Si vous réaffichez votre historique, vos commits sont maintenant dans l'ordre ! Félicitations, voilà une bonne chose de faite !

3. Fusionner des commits

Contenu masqué n°3

Cette fois-ci nous allons **fusionner** des commits pour réduire ces derniers, et surtout les rendre cohérents ! (Parce qu'avoir un commit pour une faute d'orthographe, c'est désagréable !)

On va donc chercher à atteindre le schéma suivant :

```
-------- AVANT --------
56701fc Les 30 glorieuses
a63009c Début de la seconde guerre mondiale - rattrapage
752c8cd L'entre deux guerres
328d49e Conclusion de la première guerre mondiale - rattrapage
b52fa42 La première guerre mondiale
```

```
8c310df La révolution industrielle
47901da Ajout des titres des parties
5b56868 Creation des fichiers du programme d'histoire

-------- APRES --------
d55d7d3 Les 30 glorieuses
3107653 La seconde guerre mondiale
ca137f6 L'entre deux guerres
ebfa63b La première guerre mondiale
8c310df La révolution industrielle
47901da Ajout des titres des parties
5b56868 Creation des fichiers du programme d'histoire
```

Oui, nous allons aussi en profiter pour mettre à jour un message de commit !

Là encore, c'est la magie de la commande `rebase` qui va nous être utile. Comme précédemment, on va la lancer sur le dernier commit qui ne bouge pas, donc `b52fa42 La première guerre mondiale`. Ce qui nous donne :

```
git rebase -i b52fa42^
```

La machine se met en route et nous affiche le menu permettant de faire les modifications. Nous allons cette fois-ci lui dire de fusionner le commit 8f3d90c avec son prédécesseur et nous en profiterons pour éditer le message de commit de e4b80f96. On utilisera pour cela le mot-clé "fixup" ou "squash" pour fusionner (le dernier permet de changer le message de commit lors de la fusion), et nous utiliserons "reword" pour éditer juste le message du second commit à modifier.

Voici la séquence :

```

```

Sauvegardez, quittez, puis laissez la magie opérer ! Lors du processus, l'éditeur devrait apparaître pour vous demander le nouveau commit pour l'opération de "reword".

Et voilà, c'est déjà fini pour ce morceau ! Plutôt simple, non ?

4. Merger une autre branche

Contenu masqué n°4

Une dernière fonction bien pratique de l'outil rebase est le *merge* (fusion) entre des branches. Ainsi, si vous travaillez sur une branche pour développer quelque chose, mais que vous voulez

récupérer le contenu d'une autre branche pour mettre à jour la vôtre (vous synchroniser avec *master* par exemple), `rebase` peut vous y aider.

Cette fois-ci, on va se servir de rebase non pas en indiquant un commit mais en indiquant la branche que l'on aimerait récupérer dans notre travail. En l'occurrence, on va chercher à récupérer les modifications d'Alice (branche du même nom) qui a pris notre cours puis y a rajouté quelques anecdotes dans son coin.

Voici la petite commande à lancer :

```
1 git rebase Alice
```

Cette fois-ci, pas besoin du mode "interactif" `-i`.

Évidemment, il peut arriver que des conflits se présentent, sinon ce n'est pas drôle... Et évidemment, j'en ai volontairement créé pour cette opération !

Essayez de les corriger avec l'éditeur, puis il suffit de faire un `git add <lefichier>` suivi d'un `git rebase --continue` pour continuer le rebase !

Vous voilà maintenant synchronisés avec la branche d'Alice.

Une dernière petite information pour terminer. Si jamais un rebase devient trop compliqué à gérer, vous pouvez toujours l'abandonner pour revenir à l'état d'avant le lancement de la commande. Pour cela, tapez simplement `git rebase --abort`.

4.0.1. Sources

— le logo d'illustration de ce tutoriel appartient à git ;
— une lecture intéressante .

Contenu masqué

Contenu masqué n°1

Si vous faites partie des développeurs de Zeste de Savoir, sachez que les notes dans les balises "secret" vous sont destinées, pour faire des analogies avec le projet. Retourner au texte.

Contenu masqué n°2

Ici, la branche Bob peut-être assimilée à votre branche de bugfix ou de développement de *feature*. La branche d'Alice sera ensuite vue comme upstream/dev, mais nous y reviendrons. Retourner au texte.

Contenu masqué

Contenu masqué n°3

Cette opération est souvent pratique quand vous écrivez la doc en plusieurs fois, et que vous voulez réunir tout les commits concernant cette dernière en un seul. Retourner au texte.

Contenu masqué n°4

Ce dernier cas est le cas habituel du merge de upstream/dev dans votre branche de travail, afin que git puisse merger votre PR. Ici, essayez d'imaginer la branche d'Alice en tant que upstream/dev. Retourner au texte.