7.6 Инструменты Git - Исправление истории

**Исправление истории**

Неоднократно при работе с Git, вам может потребоваться по какой-то причине исправить вашу историю коммитов. Одно из преимуществ Git заключается в том, что он позволяет вам отложить принятие решений на самый последний момент. Область подготовленных изменений позволяет вам решить, какие файлы попадут в коммит непосредственно перед её выполнением; благодаря команде stash вы можете решить, что не хотите продолжать работу над какими-то изменениями; также вы можете изменить уже совершённые коммиты так, чтобы они выглядели совершенно другим образом. В частности, можно изменить порядок коммитов, сообщения или изменённые в коммитах файлы, объединить вместе или разбить на части, полностью удалить коммит — но только до того, как вы поделитесь своими наработками с другими.

В данном разделе вы узнаете, как выполнять эти очень полезные задачи. Таким образом, перед тем, как поделиться вашими наработками с другими, вы сможете привести вашу историю коммитов к нужному виду.

**Изменение последнего коммита**

Изменение вашего последнего коммита, наверное, наиболее частое исправление истории, которое вы будете выполнять. Наиболее часто с вашим последним коммитом вам будет нужно сделать две основные операции: изменить сообщение коммита или изменить только что сделанный снимок, добавив, изменив или удалив файлы.

Если вы хотите изменить только сообщение вашего последнего коммита, это очень просто:

$ git commit --amend

Эта команда откроет в вашем текстовом редакторе сообщение вашего последнего коммита, для того, чтобы вы могли его исправить. Когда вы сохраните его и закроете редактор, будет создан новый коммит, содержащий это сообщение, который теперь и будет вашим последним коммитом.

Если вы создали коммит и затем хотите изменить зафиксированный снимок, добавив или изменив файлы (возможно, вы забыли добавить вновь созданный файл, когда совершали изначальный коммит), то процесс выглядит в основном так же. Вы добавляете в индекс необходимые изменения, редактируя файл и выполняя для него git add или git rm для отслеживаемого файла, а последующая команда git commit --amend берет вашу текущую область подготовленных изменений и делает её снимок для нового коммита.

Вы должны быть осторожными, используя этот приём, так как при этом изменяется SHA-1 коммита. Поэтому как и с операцией rebase — не изменяйте ваш последний коммит, если вы уже отправили её в общий репозиторий.

**Изменение сообщений нескольких коммитов**

Для изменения коммита, расположенного раньше в вашей истории, вам нужно обратиться к более сложным инструментам. В Git отсутствуют инструменты для изменения истории, но вы можете использовать команду rebase, чтобы перебазировать группу коммитов туда же на HEAD, где они были изначально, вместо перемещения их в другое место. С помощью интерактивного режима команды rebase, вы можете останавливаться после каждого нужного вам коммита и изменять сообщения, добавлять файлы или делать что-то другое, что вам нужно. Вы можете запустить rebase в интерактивном режиме, добавив опцию -i к git rebase. Вы должны указать, какие коммиты вы хотите изменить, передав команде коммит, на который нужно выполнить перебазирование.

Например, если вы хотите изменить сообщения последних трёх коммитов, или сообщение какого-то одного коммита этой группы, то передайте как аргумент команде git rebase -i родителя последнего коммита, который вы хотите изменить — HEAD~2^ или HEAD~3. Может быть, проще будет запомнить ~3, так как вы хотите изменить последние три коммита; но не забывайте, что вы, в действительности, указываете четвертый коммит с конца — родителя последнего коммита, который вы хотите изменить:

$ git rebase -i HEAD~3

Напомним, что это команда перебазирования — каждый коммит, входящий в диапазон HEAD~3..HEAD, будет изменён вне зависимости от того, изменили вы сообщение или нет. Не включайте в такой диапазон коммит, который уже был отправлен на центральный сервер: сделав это, вы можете запутать других разработчиков, предоставив вторую версию одних и тех же изменений.

Выполнение этой команды отобразит в вашем текстовом редакторе список коммитов, в нашем случае, например, следующее:

pick f7f3f6d changed my name a bit

pick 310154e updated README formatting and added blame

pick a5f4a0d added cat-file

# Rebase 710f0f8..a5f4a0d onto 710f0f8

#

# 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

#

# These lines can be re-ordered; they are executed from top to bottom.

#

# If you remove a line here THAT COMMIT WILL BE LOST.

#

# However, if you remove everything, the rebase will be aborted.

#

# Note that empty commits are commented out

Важно отметить, что коммиты перечислены в порядке, противоположном порядку, который вы обычно видите при использовании команды log. Если вы выполните log, то увидите следующее:

$ git log --pretty=format:"%h %s" HEAD~3..HEAD

a5f4a0d added cat-file

310154e updated README formatting and added blame

f7f3f6d changed my name a bit

Обратите внимание на обратный порядок. Команда rebase в интерактивном режиме предоставит вам скрипт, который она будет выполнять. Она начнет с коммита, который вы указали в командной строке (HEAD~3) и повторит изменения, внесённые каждым из коммитов, сверху вниз. Наверху отображается самый старый коммит, а не самый новый, потому что он будет повторен первым.

Вам необходимо изменить скрипт так, чтобы он остановился на коммите, который вы хотите изменить. Для этого измените слово ‘pick’ на слово ‘edit’ напротив каждого из коммитов, после которых скрипт должен остановиться. Например, для изменения сообщения только третьего коммита, измените файл следующим образом:

edit f7f3f6d changed my name a bit

pick 310154e updated README formatting and added blame

pick a5f4a0d added cat-file

Когда вы сохраните сообщение и выйдете из редактора, Git переместит вас к самому раннему коммиту из списка и вернёт вас в командную строку со следующим сообщением:

$ git rebase -i HEAD~3

Stopped at f7f3f6d... changed my name a bit

You can amend the commit now, with

git commit --amend

Once you’re satisfied with your changes, run

git rebase --continue

Эти инструкции говорят вам в точности то, что нужно сделать. Введите

$ git commit --amend

Измените сообщение коммита и выйдите из редактора. Затем выполните

$ git rebase --continue

Эта команда автоматически применит два оставшиеся коммита и завершится. Если вы измените ‘pick’ на ‘edit’ в других строках, то можете повторить эти шаги для соответствующих коммитов. Каждый раз Git будет останавливаться, позволяя вам исправить коммит, и продолжит, когда вы закончите.

**Переупорядочивание коммитов**

Вы также можете использовать интерактивное перебазирование для переупорядочивания или полного удаления коммитов. Если вы хотите удалить коммит “added cat-file” и изменить порядок, в котором были внесены два оставшихся, то вы можете изменить скрипт перебазирования с такого:

pick f7f3f6d changed my name a bit

pick 310154e updated README formatting and added blame

pick a5f4a0d added cat-file

на такой:

pick 310154e updated README formatting and added blame

pick f7f3f6d changed my name a bit

Когда вы сохраните скрипт и выйдете из редактора, Git переместит вашу ветку на родителя этих коммитов, применит 310154e, затем f7f3f6d и после этого остановится. Вы, фактически, изменили порядок этих коммитов и полностью удалили коммит “added cat-file”.

**Объединение коммитов**

С помощью интерактивного режима команды rebase также можно объединить несколько коммитов в один. Git добавляет полезные инструкции в сообщение скрипта перебазирования:

#

# 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

#

# These lines can be re-ordered; they are executed from top to bottom.

#

# If you remove a line here THAT COMMIT WILL BE LOST.

#

# However, if you remove everything, the rebase will be aborted.

#

# Note that empty commits are commented out

Если вместо “pick” или “edit” вы укажете “squash”, Git применит изменения из текущего и предыдущего коммитов и предложит вам объединить их сообщения. Таким образом, если вы хотите из этих трёх коммитов сделать один, вы должны изменить скрипт следующим образом:

pick f7f3f6d changed my name a bit

squash 310154e updated README formatting and added blame

squash a5f4a0d added cat-file

Когда вы сохраните скрипт и выйдете из редактора, Git применит изменения всех трёх коммитов и затем вернёт вас обратно в редактор, чтобы вы могли объединить сообщения коммитов:

# This is a combination of 3 commits.

# The first commit's message is:

changed my name a bit

# This is the 2nd commit message:

updated README formatting and added blame

# This is the 3rd commit message:

added cat-file

После сохранения сообщения, вы получите один коммит, содержащий изменения всех трёх коммитов, существовавших ранее.

**Разбиение коммита**

Разбиение коммита отменяет его и позволяет затем по частям индексировать и фиксировать изменения, создавая таким образом столько коммитов, сколько вам нужно. Например, предположим, что вы хотите разбить средний коммит на три. Вместо одного коммита “updated README formatting and added blame” вы хотите получить два разных: первый — “updated README formatting”, и второй — “added blame”. Вы можете добиться этого, изменив в скрипте rebase -i инструкцию для разбиваемого коммита на “edit”:

pick f7f3f6d changed my name a bit

edit 310154e updated README formatting and added blame

pick a5f4a0d added cat-file

Затем, когда скрипт вернёт вас в командную строку, вам нужно будет отменить индексацию изменений этого коммита, и создать несколько коммитов на основе этих изменений. Когда вы сохраните скрипт и выйдете из редактора, Git переместится на родителя первого коммита в вашем списке, применит первый коммит (f7f3f6d), применит второй (310154e), и вернёт вас в консоль. Здесь вы можете отменить коммит с помощью команды git reset HEAD^, которая, фактически, отменит этот коммит и удалит из индекса изменённые файлы. Теперь вы можете добавлять в индекс и фиксировать файлы, пока не создадите требуемые коммиты, а после этого выполнить команду git rebase --continue:

$ git reset HEAD^

$ git add README

$ git commit -m 'updated README formatting'

$ git add lib/simplegit.rb

$ git commit -m 'added blame'

$ git rebase --continue

Git применит последний коммит (a5f4a0d) из скрипта, и ваша история примет следующий вид:

$ git log -4 --pretty=format:"%h %s"

1c002dd added cat-file

9b29157 added blame

35cfb2b updated README formatting

f3cc40e changed my name a bit

И снова, при этом изменились SHA-1 хеши всех коммитов в вашем списке, поэтому убедитесь, что ни один коммит из этого списка ранее не был отправлен в общий репозиторий.

**Удаление коммита**

Если вы хотите избавиться от какого-либо коммита, то удалить его можно во время интерактивного перебазирования rebase -i. Напишите слово “drop” перед коммитом, который хотите удалить, или просто удалите его из списка:

pick 461cb2a This commit is OK

drop 5aecc10 This commit is broken

Из-за того, как Git создаёт объекты коммитов, удаление или изменение коммита влечёт за собой перезапись всех последующих коммитов. Чем дальше вы вернётесь в историю ваших коммитов, тем больше коммитов потребуется переделать. Это может вызвать множество конфликтов слияния, особенно если у вас много последующих коммитов, которые зависят от удалённого.

Если во время подобного перебазирования вы поняли, что это была не очень хорошая идея, то всегда можно остановиться. Просто выполните команду git rebase --abort и ваш репозиторий вернётся в то состояние, в котором он был до начала перебазирования.

Если вы завершили перебазирование, а затем решили, что полученный результат это не то, что вам нужно — воспользуйтесь командой git reflog, чтобы восстановить предыдущую версию вашей ветки. Дополнительную информацию по команде reflog можно найти в разделе [Восстановление данных](https://git-scm.com/book/ru/v2/ch00/r_data_recovery).

|  |  |
| --- | --- |
| **Note** | Drew DeVault создал практическое руководство с упражнениями по использованию git rebase. Найти его можно здесь: <https://git-rebase.io/> |

**Продвинутый инструмент: filter-branch**

Существует ещё один способ изменения истории, который вы можете использовать при необходимости изменить большое количество коммитов каким-то программируемым способом — например, изменить глобально ваш адрес электронной почты или удалить файл из всех коммитов. Для этого существует команда filter-branch, и она может изменять большие периоды вашей истории, поэтому вы, возможно, не должны её использовать кроме тех случаев, когда ваш проект ещё не стал публичным и другие люди ещё не имеют наработок, основанных на коммитах, которые вы собираетесь изменить. Однако, эта команда может быть очень полезной. Далее вы ознакомитесь с несколькими обычными вариантами использованиями этой команды, таким образом, вы сможете получить представление о том, на что она способна.

**Удаление файла из каждого коммита**

Такое случается довольно часто. Кто-нибудь случайно зафиксировал огромный бинарный файл, неосмотрительно выполнив git add ., и вы хотите отовсюду его удалить. Возможно, вы случайно зафиксировали файл, содержащий пароль, а теперь хотите сделать ваш проект общедоступным. В общем, утилиту filter-branch вы, вероятно, захотите использовать, чтобы привести к нужному виду всю вашу историю. Для удаления файла passwords.txt из всей вашей истории вы можете использовать опцию --tree-filter команды filter-branch:

$ git filter-branch --tree-filter 'rm -f passwords.txt' HEAD

Rewrite 6b9b3cf04e7c5686a9cb838c3f36a8cb6a0fc2bd (21/21)

Ref 'refs/heads/master' was rewritten

Опция --tree-filter выполняет указанную команду после переключения на каждый коммит и затем повторно фиксирует результаты. В данном примере, вы удаляете файл passwords.txt из каждого снимка вне зависимости от того, существует он или нет. Если вы хотите удалить все случайно зафиксированные резервные копии файлов, созданные текстовым редактором, то вы можете выполнить нечто подобное git filter-branch --tree-filter 'rm -f \*~' HEAD.

Вы можете посмотреть, как Git изменит деревья и коммиты, а затем уже переместить указатель ветки. Как правило, хорошим подходом будет выполнение всех этих действий в тестовой ветке и, после проверки полученных результатов, установка на неё указателя основной ветки. Для выполнения filter-branch на всех ваших ветках, вы можете передать команде опцию --all.

**Установка поддиректории корневой директорией проекта**

Предположим, вы выполнили импорт из другой системы управления исходным кодом и получили в результате поддиректории, которые не имеют никакого смысла (trunk, tags и так далее). Если вы хотите сделать поддиректорию trunk корневой директорией для каждого коммита, команда filter-branch может помочь вам в этом:

$ git filter-branch --subdirectory-filter trunk HEAD

Rewrite 856f0bf61e41a27326cdae8f09fe708d679f596f (12/12)

Ref 'refs/heads/master' was rewritten

Теперь вашей новой корневой директорией проекта будет являться поддиректория trunk. Git также автоматически удалит коммиты, которые не затрагивали эту поддиректорию.

**Глобальное изменение адреса электронной почты**

Ещё один типичный случай возникает, когда вы забыли выполнить git config для настройки своего имени и адреса электронной почты перед началом работы, или, возможно, хотите открыть исходные коды вашего рабочего проекта и изменить везде адрес вашей рабочей электронной почты на персональный. В любом случае вы можете изменить адрес электронный почты сразу в нескольких коммитах с помощью команды filter-branch. Вы должны быть осторожны, чтобы изменить только свои адреса электронной почты, для этого используйте опцию --commit-filter:

$ git filter-branch --commit-filter '

if [ "$GIT\_AUTHOR\_EMAIL" = "schacon@localhost" ];

then

GIT\_AUTHOR\_NAME="Scott Chacon";

GIT\_AUTHOR\_EMAIL="schacon@example.com";

git commit-tree "$@";

else

git commit-tree "$@";

fi' HEAD

Эта команда пройдёт по всем коммитам и установит в них ваш новый адрес. Так как коммиты содержат значения SHA-1 хешей их родителей, эта команда изменяет SHA-1 хеш каждого коммита в вашей истории, а не только тех, которые соответствовали адресам электронной почты.