重写历史

很多时候，在 Git 上工作的时候，你也许会由于某种原因想要修订你的提交历史。Git 的一个卓越之处就是它允许你在最后可能的时刻再作决定。你可以在你即将提交暂存区时决定什么文件归入哪一次提交，你可以使用 stash 命令来决定你暂时搁置的工作，你可以重写已经发生的提交以使它们看起来是另外一种样子。这个包括改变提交的次序、改变说明或者修改提交中包含的文件，将提交归并、拆分或者完全删除——这一切在你尚未开始将你的工作和别人共享前都是可以的。

在这一节中，你会学到如何完成这些很有用的任务以使你的提交历史在你将其共享给别人之前变成你想要的样子。

改变最近一次提交

改变最近一次提交也许是最常见的重写历史的行为。对于你的最近一次提交，你经常想做两件基本事情：改变提交说明，或者改变你刚刚通过增加，改变，删除而记录的快照。

如果你只想修改最近一次提交说明，这非常简单：

$ git commit --amend

这会把你带入文本编辑器，里面包含了你最近一次提交说明，供你修改。当你保存并退出编辑器，这个编辑器会写入一个新的提交，里面包含了那个说明，并且让它成为你的新的最近一次提交。

如果你完成提交后又想修改被提交的快照，增加或者修改其中的文件，可能因为你最初提交时，忘了添加一个新建的文件，这个过程基本上一样。你通过修改文件然后对其运行git add或对一个已被记录的文件运行git rm，随后的git commit --amend会获取你当前的暂存区并将它作为新提交对应的快照。

使用这项技术的时候你必须小心，因为修正会改变提交的SHA-1值。这个很像是一次非常小的rebase——不要在你最近一次提交被推送后还去修正它。

修改多个提交说明

要修改历史中更早的提交，你必须采用更复杂的工具。Git没有一个修改历史的工具，但是你可以使用rebase工具来衍合一系列的提交到它们原来所在的HEAD上而不是移到新的上。依靠这个交互式的rebase工具，你就可以停留在每一次提交后，如果你想修改或改变说明、增加文件或任何其他事情。你可以通过给git rebase增加-i选项来以交互方式地运行rebase。你必须通过告诉命令衍合到哪次提交，来指明你需要重写的提交的回溯深度。

例如，你想修改最近三次的提交说明，或者其中任意一次，你必须给git rebase -i提供一个参数，指明你想要修改的提交的父提交，例如HEAD~2或者HEAD~3。可能记住~3更加容易，因为你想修改最近三次提交；但是请记住你事实上所指的是四次提交之前，即你想修改的提交的父提交。

$ git rebase -i HEAD~3

再次提醒这是一个衍合命令——HEAD~3..HEAD范围内的每一次提交都会被重写，无论你是否修改说明。不要涵盖你已经推送到中心服务器的提交——这么做会使其他开发者产生混乱，因为你提供了同样变更的不同版本。

运行这个命令会为你的文本编辑器提供一个提交列表，看起来像下面这样

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

# e, edit = use commit, but stop for amending

# s, squash = use commit, but meld into previous commit

#

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

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

#

很重要的一点是你得注意这些提交的顺序与你通常通过log命令看到的是相反的。如果你运行log，你会看到下面这样的结果：

$ 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

请注意这里的倒序。交互式的rebase给了你一个即将运行的脚本。它会从你在命令行上指明的提交开始(HEAD~3)然后自上至下重播每次提交里引入的变更。它将最早的列在顶上而不是最近的，因为这是第一个需要重播的。

你需要修改这个脚本来让它停留在你想修改的变更上。要做到这一点，你只要将你想修改的每一次提交前面的pick改为edit。例如，只想修改第三次提交说明的话，你就像下面这样修改文件：

edit f7f3f6d changed my name a bit

pick 310154e updated README formatting and added blame

pick a5f4a0d added cat-file

当你保存并退出编辑器，Git会倒回至列表中的最后一次提交，然后把你送到命令行中，同时显示以下信息：

$ git rebase -i HEAD~3

Stopped at 7482e0d... updated the gemspec to hopefully work better

You can amend the commit now, with

git commit --amend

Once you’re satisfied with your changes, run

git rebase --continue

这些指示很明确地告诉了你该干什么。输入

$ git commit --amend

修改提交说明，退出编辑器。然后，运行

$ git rebase --continue

这个命令会自动应用其他两次提交，你就完成任务了。如果你将更多行的 pick 改为 edit ，你就能对你想修改的提交重复这些步骤。Git每次都会停下，让你修正提交，完成后继续运行。

重排提交

你也可以使用交互式的衍合来彻底重排或删除提交。如果你想删除"added cat-file"这个提交并且修改其他两次提交引入的顺序，你将rebase脚本从这个

pick f7f3f6d changed my name a bit

pick 310154e updated README formatting and added blame

pick a5f4a0d added cat-file

改为这个：

pick 310154e updated README formatting and added blame

pick f7f3f6d changed my name a bit

当你保存并退出编辑器，Git 将分支倒回至这些提交的父提交，应用310154e，然后f7f3f6d，接着停止。你有效地修改了这些提交的顺序并且彻底删除了"added cat-file"这次提交。

压制(Squashing)提交

交互式的衍合工具还可以将一系列提交压制为单一提交。脚本在 rebase 的信息里放了一些有用的指示：

#

# Commands:

# p, pick = use commit

# e, edit = use commit, but stop for amending

# s, squash = use commit, but meld into previous commit

#

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

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

#

如果不用"pick"或者"edit"，而是指定"squash"，Git 会同时应用那个变更和它之前的变更并将提交说明归并。因此，如果你想将这三个提交合并为单一提交，你可以将脚本修改成这样：

pick f7f3f6d changed my name a bit

squash 310154e updated README formatting and added blame

squash a5f4a0d added cat-file

当你保存并退出编辑器，Git 会应用全部三次变更然后将你送回编辑器来归并三次提交说明。

# 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

当你保存之后，你就拥有了一个包含前三次提交的全部变更的单一提交。

拆分提交

拆分提交就是撤销一次提交，然后多次部分地暂存或提交直到结束。例如，假设你想将三次提交中的中间一次拆分。将"updated README formatting and added blame"拆分成两次提交：第一次为"updated README formatting"，第二次为"added blame"。你可以在rebase -i脚本中修改你想拆分的提交前的指令为"edit"：

pick f7f3f6d changed my name a bit

edit 310154e updated README formatting and added blame

pick a5f4a0d added cat-file

然后，这个脚本就将你带入命令行，你重置那次提交，提取被重置的变更，从中创建多次提交。当你保存并退出编辑器，Git 倒回到列表中第一次提交的父提交，应用第一次提交（f7f3f6d），应用第二次提交（310154e），然后将你带到控制台。那里你可以用git reset HEAD^对那次提交进行一次混合的重置，这将撤销那次提交并且将修改的文件撤回。此时你可以暂存并提交文件，直到你拥有多次提交，结束后，运行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在脚本中应用了最后一次提交（a5f4a0d），你的历史看起来就像这样了：

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

1c002dd added cat-file

9b29157 added blame

35cfb2b updated README formatting

f3cc40e changed my name a bit

再次提醒，这会修改你列表中的提交的 SHA 值，所以请确保这个列表里不包含你已经推送到共享仓库的提交。

核弹级选项: filter-branch

如果你想用脚本的方式修改大量的提交，还有一个重写历史的选项可以用——例如，全局性地修改电子邮件地址或者将一个文件从所有提交中删除。这个命令是filter-branch，这个会大面积地修改你的历史，所以你很有可能不该去用它，除非你的项目尚未公开，没有其他人在你准备修改的提交的基础上工作。尽管如此，这个可以非常有用。你会学习一些常见用法，借此对它的能力有所认识。

从所有提交中删除一个文件

这个经常发生。有些人不经思考使用git add .，意外地提交了一个巨大的二进制文件，你想将它从所有地方删除。也许你不小心提交了一个包含密码的文件，而你想让你的项目开源。filter-branch大概会是你用来清理整个历史的工具。要从整个历史中删除一个名叫password.txt的文件，你可以在filter-branch上使用--tree-filter选项：

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

Rewrite 6b9b3cf04e7c5686a9cb838c3f36a8cb6a0fc2bd (21/21)

Ref 'refs/heads/master' was rewritten

--tree-filter选项会在每次检出项目时先执行指定的命令然后重新提交结果。在这个例子中，你会在所有快照中删除一个名叫 password.txt 的文件，无论它是否存在。如果你想删除所有不小心提交上去的编辑器备份文件，你可以运行类似git filter-branch --tree-filter "find \* -type f -name '\*~' -delete" HEAD的命令。

你可以观察到 Git 重写目录树并且提交，然后将分支指针移到末尾。一个比较好的办法是在一个测试分支上做这些然后在你确定产物真的是你所要的之后，再 hard-reset 你的主分支。要在你所有的分支上运行filter-branch的话，你可以传递一个--all给命令。

将一个子目录设置为新的根目录

假设你完成了从另外一个代码控制系统的导入工作，得到了一些没有意义的子目录（trunk, tags等等）。如果你想让trunk子目录成为每一次提交的新的项目根目录，filter-branch也可以帮你做到：

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

Rewrite 856f0bf61e41a27326cdae8f09fe708d679f596f (12/12)

Ref 'refs/heads/master' was rewritten

现在你的项目根目录就是trunk子目录了。Git 会自动地删除不对这个子目录产生影响的提交。

全局性地更换电子邮件地址

另一个常见的案例是你在开始时忘了运行git config来设置你的姓名和电子邮件地址，也许你想开源一个项目，把你所有的工作电子邮件地址修改为个人地址。无论哪种情况你都可以用filter-branch来更换多次提交里的电子邮件地址。你必须小心一些，只改变属于你的电子邮件地址，所以你使用--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

这个会遍历并重写所有提交使之拥有你的新地址。因为提交里包含了它们的父提交的SHA-1值，这个命令会修改你的历史中的所有提交，而不仅仅是包含了匹配的电子邮件地址的那些。