…彻底搞懂 Git-Rebase

2018/12月/11 22:07:00 发布在 技术博文9376 次阅读

使用 Git 已经好几年了，却始终只是熟悉一些常用的操作。对于 Git Rebase 却很少用到，直到这一次，不得不用。

一、起因

上线构建的过程中扫了一眼代码变更，突然发现，`commit` 提交竟然多达 `62` 次。我们来看看都提交了什么东西：

43. feat:add chapter_id (detail)
44. feat:add chapter_id (detail)
45. bug fixed (detail)
46. split(token) (detail)
47. delete needless code (detail)
48. update: sc print (detail)
49. update: client api param (detail)
50. update: for test (detail)
51. update: set print (detail)
52. add: print alert (detail)
53. udpate: rm alert (detail)
54. update:delete alert (detail)
55. update:go to report page,use goBack() (detail)
56. update:add alert logs (detail)
57. update:add alert logs (detail)
58. update:add alert logs (detail)
59. update:add alert logs (detail)
60. update:add alert logs (detail)
61. update:remove alert logs (detail)
62. fix: print guide (detail)

这里我们先不说 `git` 提交规范，就单纯这么多次无用的 `commit` 就很让人不舒服。可能很多人觉得无所谓，无非是多了一些提交纪录。

> 然而，并非如此，你可能听过破窗效应，编程也是如此！

二、导致问题

1.不利于代码 `review`

设想一下，你要做 `code review` ，结果一个很小的功能，提交了 `60` 多次，会不会有一些崩溃?

2.会造成分支污染

你的项目充满了无用的 `commit` 纪录，如果有一天线上出现了紧急问题，你需要回滚代码，却发现海量的 `commit` 需要一条条来看。

> 遵循项目规范才能提高团队协作效率，而不是随心所欲。

三、Rebase 场景一：如何合并多次提交纪录?

> 基于上面所说问题，我们不难想到：每一次功能开发， 对多个 commit 进行合并处理。

这时候就需要用到 `git rebase` 了。这个命令没有太难，不常用可能源于不熟悉，所以我们来通过示例学习一下。

1.我们来合并最近的 4 次提交纪录，执行：

```
git rebase -i HEAD~4
```

2.这时候，会自动进入 `vi` 编辑模式:

```
s cacc52da add: qrcode
s f072ef48 update: indexeddb hack
s 4e84901a feat: add indexedDB floder
s 8f33126c feat: add test2.js

# Rebase 5f2452b2..8f33126c onto 5f2452b2 (4 commands)
#
# 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
# d, drop = remove commit
#
# 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.
```

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

有几个命令需要注意一下：

- 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
- d, drop = remove commit

按照如上命令来修改你的提交纪录：

```
s cacc52da add: qrcode
s f072ef48 update: indexeddb hack
s 4e84901a feat: add indexedDB floder
p 8f33126c feat: add test2.js
```

3.如果保存的时候，你碰到了这个错误：

```
error: cannot 'squash' without a previous commit
```

> 注意不要合并先前提交的东西，也就是已经提交远程分支的纪录。

4.如果你异常退出了 `vi` 窗口，不要紧张：

```
git rebase --edit-todo
```

这时候会一直处在这个编辑的模式里，我们可以回去继续编辑，修改完保存一下：

```
git rebase --continue
```

5.查看结果

```
git log
```

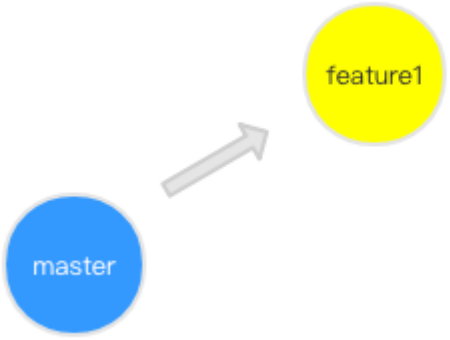
> 三次提交合并成了一次，减少了无用的提交信息。

```
commit 29cb7f28bba9f7a053c99672430f1c1fceb2d0cf (HEAD -> master)
Author: chenfengyanyu <jartto@qq.com>
Date:   Fri Dec 7 18:07:42 2018 +0800

    update: indexeddb hack

    feat: add indexedDB floder

    feat: add test2.js

commit 5f2452b23632323330953a4e2ebc4f87e030c1b9
Author: chenfengyanyu <jartto@qq.com>
Date:   Wed Dec 5 23:22:48 2018 +0800

    add: git-flow123

commit 5265a99b385fde500f4b84b8ecb97be79f6f01c7
Author: chenfengyanyu <jartto@qq.com>
Date:   Wed Dec 5 22:58:41 2018 +0800

    add: gitflow
```

四、Rebase 场景二：分支合并

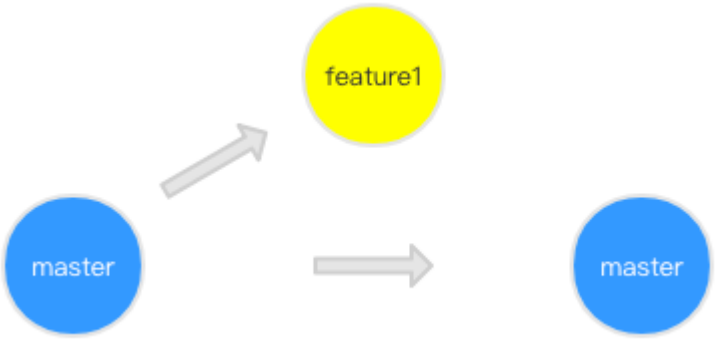
1.我们先从 `master` 分支切出一个 `dev` 分支，进行开发：

```
git:(master) git checkout -b feature1
```



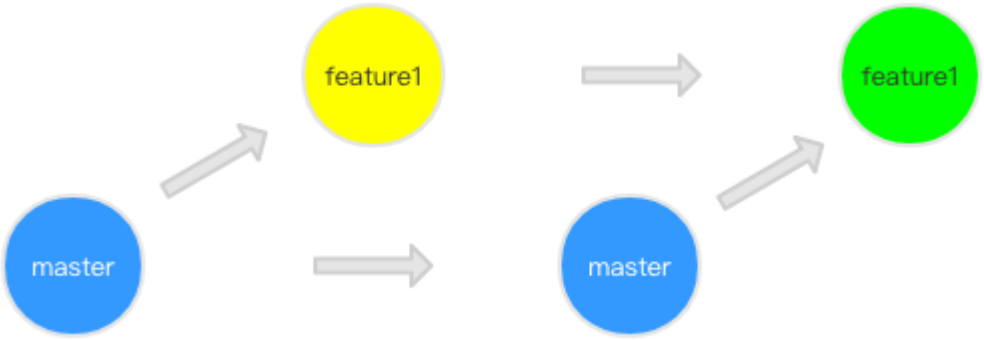
2.这时候，你的同事完成了一次 `hotfix` ，并合并入了 `master` 分支，此时 `master` 已经领

先于你的 feature1 分支了：



3.恰巧，我们想要同步 master 分支的改动，首先想到了 merge ，执行：

```
git:(feature1) git merge master
```

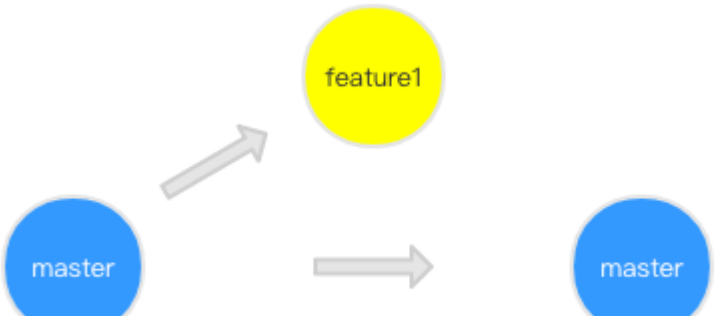


图中绿色的点就是我们合并之后的结果，执行：

```
git:(feature1) git log
```

就会在记录里发现一些 merge 的信息，但是我们觉得这样污染了 commit 记录，想要保持一份干净的 commit ，怎么办呢？这时候， git rebase 就派上用场了。

4.让我们来试试 git rebase ，先回退到同事 hotfix 后合并 master 的步骤：

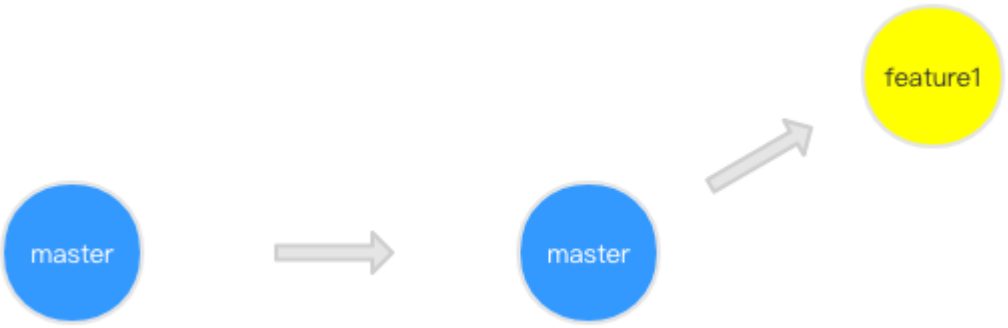


5.使用 rebase 后来看看结果：

```
git:(feature1) git rebase master
```

> 这里补充一点： rebase 做了什么操作呢?

首先， git 会把 feature1 分支里面的每个 commit 取消掉；
其次，把上面的操作临时保存成 patch 文件，存在 .git/rebase 目录下；
然后，把 feature1 分支更新到最新的 master 分支；
最后，把上面保存的 patch 文件应用到 feature1 分支上；



从 commit 记录我们可以看出来， feature1 分支是基于 hotfix 合并后的 master ，自然而然的成为了最领先的分支，而且没有 merge 的 commit 记录，是不是感觉很舒服了。

6.在 rebase 的过程中，也许会出现冲突 conflict 。在这种情况， git 会停止 rebase 并会让你去解决冲突。在解决完冲突后，用 git add 命令去更新这些内容。

> 注意，你无需执行 git-commit，只要执行 continue

```
git rebase --continue
```

这样 git 会继续应用余下的 patch 补丁文件。

7.在任何时候，我们都可以用 --abort 参数来终止 rebase 的行动，并且分支会回到

`rebase` 开始前的状态。

```
git rebase —abort
```

五、更多 Rebase 的使用场景

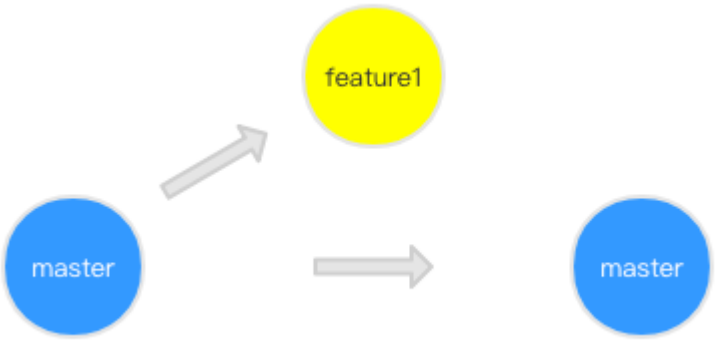
> git-rebase 存在的价值是：对一个分支做「变基」操作。

1.当我们在一个过时的分支上面开发的时候，执行 `rebase` 以此同步 `master` 分支最新变动；
2.假如我们要启动一个放置了很久的并行工作，现在有时间来继续这件事情，很显然这个分支已经落后了。这时候需要在最新的基准上面开始工作，所以 `rebase` 是最合适的选择。

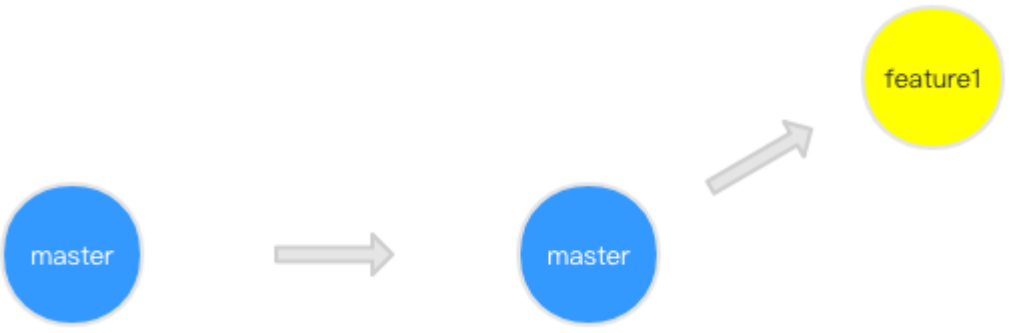
六、为什么会是危险操作?

根据上文来看，`git-rebase` 很完美，解决了我们的两个问题：
1.合并 `commit` 记录，保持分支整洁；
2.相比 `merge` 来说会减少分支合并的记录；

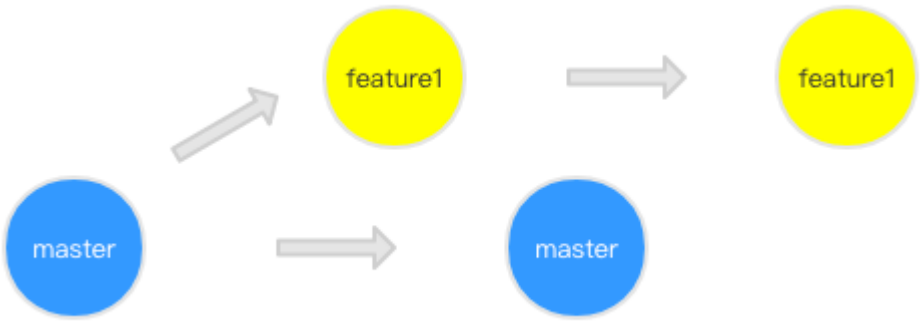
如果你提交了代码到远程，提交前是这样的：



提交后远程分支变成了这样：



而此时你的同事也在 feature1 上开发，他的分支依然还是：



那么当他 pull 远程 master 的时候，就会有丢失提交纪录。这就是为什么我们经常听到有人说 git rebase 是一个危险命令，因为它改变了历史，我们应该谨慎使用。

> 除非你可以肯定该 feature1 分支只有你自己使用，否则请谨慎操作。

结论：只要你的分支上需要 rebase 的所有 commits 历史还没有被 push 过，就可以安全地使用 git-rebase 来操作。

七、参考：

rebase
git-rebase 使用总结
git 中的 rebase操作
git-rebase vs git-merge 详解