

## 26 | Facebook怎样实现代码提交的原子性？

2019-10-23 葛俊

研发效率破局之道

[进入课程 >](#)



讲述：葛俊

时长 24:26 大小 22.39M



你好，我是葛俊。今天，我们继续来聊聊如何通过 Git 提高代码提交的原子性吧。

在上一篇文章中，我给你详细介绍了 Git 助力提高代码提交原子性的五条基础操作，今天我们来再看看 Facebook 的开发人员具体是如何使用这些操作来实现提交的原子性的。

为了帮助你更直观地理解、学习，在这篇文章里，我会与你详细描述工作场景，并列出具体的命令。同时，我还把这些命令的输出也都放到了文章里，供你参考。所以，这篇文章会比较长、比较细。不过不要担心，这些内容都是日常工作中的自然流程，阅读起来也会比较顺畅。

在 Facebook，开发人员最常使用两种 Git 工作流：

使用一个分支，完成所有需求的开发；

使用多个分支，每个分支支持一个需求的开发。

两种工作流都利用 Git 的超强功能来提高代码原子性。这里的“需求”包括功能开发和缺陷修复，用大写字母 A、B、C 等表示；每个需求都可能包含有多个提交，每个提交用需求名 + 序号表示。比如，A 可能包含 A1、A2 两个提交，B 只包含 B1 这一个提交，而 C 包含 C1、C2、C3 三个提交。

需要强调的是，这两种工作流中的一个分支和多个分支，都是在开发者本地机器上的分支，不是远程代码仓中的功能分支。我在前面 [第 7 篇文章](#)中提到过，Facebook 的主代码仓是不使用功能分支的。

另外，这两种 Git 工作流对代码提交原子性的助力作用，跟主代码仓是否使用单分支开发没有关系。也就是说，即使你所在团队的主仓没有使用 Facebook 那样的单分支开发模式，仍然可以使用这两种工作流来提高代码提交的原子性。

接下来，我们就先看看第一种工作流，也就是使用一个分支完成所有需求的开发。

## 工作流一：使用一个分支完成所有需求的开发

这种工作流程的最大特点是，**使用一个分支上的提交链，大量使用 `git rebase -i` 来修改提交链上的提交**。这里的提交链，指的是当前分支上，还没有推送到远端主仓共享分支的所有提交。

首先，我们需要设置一个本地分支来开发需求，通过这个分支和远端主仓的共享分支进行交互。本地分支通常直接使用 master 分支，而远端主仓的共享分支一般是 origin/master，也叫作上游分支（upstream）。

一般来说，在 git clone 的时候，master 是默认已经产生，并且是已经跟踪 origin/master 了的，你不需要做任何设置，可以查看 .git/config 文件做确认：


```
1 > cat .git/config
2 ...
3 [remote "origin"]
4   url = git@github.com:jungejason/git-atomic-demo.git
```

 复制代码

```
5 fetch = +refs/heads/*:refs/remotes/origin/*
6 [branch "master"]
7   remote = origin
8   merge = refs/heads/master
```

可以看到，branch "master"里有一个 remote = origin 选项，表明 master 分支在跟踪 origin 这个上游仓库；另外，config 文件里还有一个 remote "origin"选项，列举了 origin 这个上游仓库的地址。

当然，除了直接查看 config 文件外，Git 还提供了命令行工具。你可以使用 git branch -vv 查看某个分支是否在跟踪某个远程分支，然后再使用 git remote show去查看远程代码仓的细节。

 复制代码

```
1 ## 查看远程分支细节
2 > git branch -vv
3   master          5055c14 [origin/master: behind 1] Add documentation for getRandom
4
5
6 ## 查看分支跟踪的远程代码仓细节
7 > git remote show origin
8 * remote origin
9   Fetch URL: git@github.com:jungejason/git-atomic-demo.git
10  Push URL: git@github.com:jungejason/git-atomic-demo.git
11  HEAD branch: master
12  Remote branch:
13    master tracked
14  Local branches configured for 'git pull':
15    master merges with remote master
16  Local ref configured for 'git push':
17    master pushes to master (fast-forwardable)
18 11:07:36 (master2) jasonge@Juns-MacBook-Pro-2.local:~/jksj-repo/git-atomic-dem
```

因为 config 文件简单直观，所以我常常直接到 config 文件里面查看和修改来完成这些操作。关于远程跟踪上游代码仓分支的更多细节，比如产生新分支、设置上游分支等，你可以参考 [Git: Upstream Tracking Understanding](#) 这篇文章。

设置好分支之后，我们来看看**这个工作流中的具体步骤**。

## 单分支工作流具体步骤

单分支工作流的步骤，大致包括以下 4 步：

1. 一个原子性的功能完成后，使用 [🔗 第 25 篇文章](#)中提到的改变提交顺序的方法，把它放到距离 origin/master 最近的地方。
2. 把这个提交发到代码审查系统 Phabricator 上进行质量检查，包括代码审查和机器检查。在等待质量检查结果的同时，继续其他提交的开发。
3. 如果没有通过质量检查，则需要对提交进行修改，修改之后返回第 2 步。
4. 如果通过质量检查，就把这个提交推送到主代码仓的共享分支上，然后继续其他分支的开发，回到第 1 步。

请注意第二步的目的是，确保入库代码的质量，你可以根据实际情况进行检查。比如，你可以通过提交 PR 触发机器检查的工作流，也可以运行单元测试自行检查。如果没有任何质量检查的话，至少也要进行简单手工验证，让进入到远程代码仓的代码有起码的质量保障。


接下来，我设计了一个案例，尽量模拟我在 Facebook 的真实开发场景，与你讲述这个工作流的操作步骤。大致场景是这样的：我本来在开发需求 A，这时来了更紧急的需求 B。于是，我开始开发 B，把 B 分成两个原子性提交 B1 和 B2，并在 B1 完成之后最先推送到远程代码仓共享分支。

这个案例中，提交的改动很简单，但里面涉及了很多开发技巧，可供你借鉴。

## 阶段 1：开始开发需求 A

某天，我接到开发需求 A 的任务，要求在项目中添加一个 README 文件，对项目进行描述。

我先添加一个简单的 README.md 文件，然后用 `git commit -am 'readme'` 快速生成一个提交 A1，确保代码不会丢失。

 复制代码

```
1  ## 文件内容
2  > cat README.md
3  ## This project is for demoing git
4
5
6  ## 产生提交
```

```

7 > git commit -am 'readme'
8 [master 0825c0b] readme
9 1 file changed, 1 insertion(+)
10 create mode 100644 README.md
11
12
13 ## 查看提交历史
14 > git log --oneline --graph
15 * 0825c0b (HEAD -> master) readme
16 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
17 ...
18
19
20 ## 查看提交细节
21 > git show
22 commit 0825c0b6cd98af11b171b52367209ad6e29e38d1 (HEAD -> master)
23 Author: Jason Ge <gejun_1978@yahoo.com>
24 Date: Tue Oct 15 12:45:08 2019
25
26     readme
27
28 diff --git a/README.md b/README.md
29 new file mode 100644
30 index 0000000..789cfa9
31 --- /dev/null
32 +++ b/README.md
33 @@ -0,0 +1 @@
34 +## This project is for demoing git

```

这时，A1 是 master 上没有推送到 origin/master 的唯一提交，也就是说，是提交链上的唯一提交。

请注意，A1 的 Commit Message 很简单，就是“readme”这 6 个字符。在把 A1 发出去做代码质量检查之前，我需要添加 Commit Message 的细节。

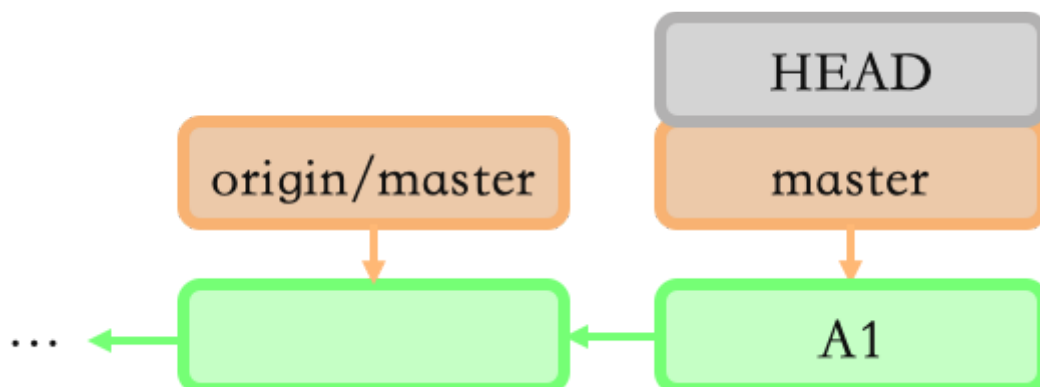



图 1 提交链状态第 1 步

## 阶段 2：开始开发需求 B


这时，来了另外一个紧急需求 B，要求是添加一个 endpoint getRandom。开发时，我不切换分支，直接在 master 上继续开发。

首先，我写一个 getRandom 的实现，并进行简单验证。

 复制代码

```
1  ## 用 VIM 修改
2  > vim index.js
3
4
5  ## 查看工作区中的改动
6  > git diff
7  diff --git a/index.js b/index.js
8  index 986fcd8..06695f6 100644
9  --- a/index.js
10 +++ b/index.js
11 @@ -6,6 +6,10 @@ app.get('/timestamp', function (req, res) {
12     res.send('' + Date.now())
13   })
14
15 +app.get('/getRandom', function (req, res) {
16 +  res.send('' + Math.random())
17 +})
18 +
19 app.get('/', function (req, res) {
20   res.send('hello world')
21 })
22
23
24 ## 用命令行工具 httpie 验证结果
25 > http localhost:3000/getRandom
26 HTTP/1.1 200 OK
27 Connection: keep-alive
28 Content-Length: 19
29 Content-Type: text/html; charset=utf-8
30 Date: Tue, 15 Oct 2019 03:49:15 GMT
31 ETag: W/"13-U1KCE8QRuz+dioGnmVwMkEWypYI"
32 X-Powered-By: Express
33
34 0.25407324324864167
```

为确保代码不丢失，我用 `git commit -am 'random'` 命令生成了一个提交 B1：

 复制代码

```
1 ## 产生提交
2 > git commit -am 'random'
3 [master 7752df4] random
4 1 file changed, 4 insertions(+)
5
6
7 ## 查看提交历史
8 > git log --oneline --graph
9 * 7752df4 (HEAD -> master) random
10 * 0825c0b readme
11 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
12 ...
13
14
15 ## 查看提交细节
16 > git show
17 commit f59a4084e3a2c620bdec49960371f8cc93b86825 (HEAD -> master)
18 Author: Jason Ge <gejun_1978@yahoo.com>
19 Date: Tue Oct 15 11:55:06 2019
20
21 random
22
23 diff --git a/index.js b/index.js
24 index 986fcd8..06695f6 100644
25 --- a/index.js
26 +++ b/index.js
27 @@ -6,6 +6,10 @@ app.get('/timestamp', function (req, res) {
28     res.send('' + Date.now())
29 })
30
31 +app.get('/getRandom', function (req, res) {
32 +  res.send('' + Math.random())
33 +})
34 +
35 app.get('/', function (req, res) {
36   res.send('hello world')
37 })
```

B1 的 Commit Message 也很简陋，因为当前的关键任务是先把功能运行起来。

现在，我的提交链上有 A1 和 B1 两个提交了。



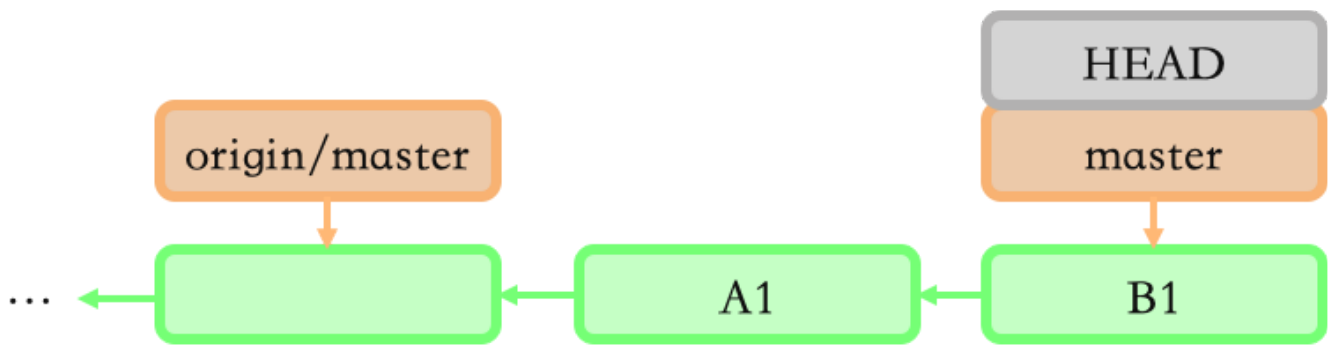


图 2 提交链状态第 2 步

接下来，我需要进行需求 B 的进一步开发：在 README 文件中给这个新的 endpoint 添加说明。

复制代码

```
1 > git diff
2 diff --git a/README.md b/README.md
3 index 789cfa9..7b2b6af 100644
4 --- a/README.md
5 +++ b/README.md
6 @@ -1,3 @@
7  ## This project is for demoing git
8  +
9  +You can visit endpoint getRandom to get a random real number.
```

我认为这个改动是 B1 的一部分，所以我用 git commit --amend 把它添加到 B1 中。

复制代码

```
1 ## 添加改动到 B1
2 > git add README.md
3 > git commit --amend
4 [master 27c4d40] random
5 Date: Tue Oct 15 11:55:06 2019 +0800
6 2 files changed, 6 insertions(+)
7
8
9 ## 查看提交历史
10 > git log --oneline --graph
11 * 27c4d40 (HEAD -> master) random
12 * 0825c0b readme
13 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
```



现在，我的提交链上还是 A1 和 B1' 两个提交。这里的 B1' 是为了区别之前的 B1，B1 仍然存在代码仓中，不过是不再使用了而已。

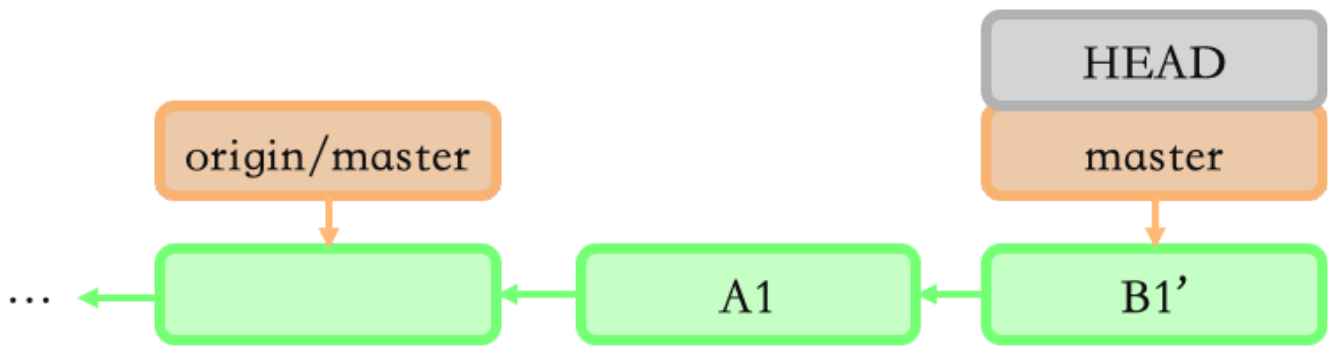


图 3 提交链状态第 3 步

### 阶段 3：拆分需求 B 的代码，把 B1'提交检查系统

这时，我觉得 B1' 的功能实现部分，也就是 index.js 的改动部分，可以推送到 origin/master 了。

不过，文档部分也就是 README.md 文件的改动，还不够好，而且功能实现和文档应该分成两个原子性提交。于是，我将 B1' 拆分为 B1'' 和 B2 两部分。

 复制代码

```
1 ## 将 B1'拆分
2 > git reset HEAD^
3 Unstaged changes after reset:
4 M README.md    ## 这个将是 B2 的内容
5 M index.js     ## 这个将是 B1''的内容
6
7 > git status
8 On branch master
9 Your branch is ahead of 'origin/master' by 1 commit.
10 (use "git push" to publish your local commits)
11 Changes not staged for commit:
12 (use "git add <file>..." to update what will be committed)
13 (use "git checkout -- <file>..." to discard changes in working directory)
14 modified:   README.md
15 modified:   index.js
16 no changes added to commit (use "git add" and/or "git commit -a")
17
18 > git add index.js
19 > git commit    ## 这里我认真填写 B1''的 Commit Message
20
21 > git add README.md
```

```

22 > git commit    ## 这里我认真填写 B2 的 Commit Message
23
24
25 ## 查看提交历史
26 * 68d813f (HEAD -> master) [DO NOT PUSH] Add documentation for getRandom endpoint
27 * 7d43442 Add getRandom endpoint
28 * 0825c0b readme
29 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp

```

现在，提交链上有 A1、B1' '、B2 三个提交。

请注意，在这里我把功能实现和文档分为两个原子性提交，只是为了帮助说明我需要把 B1' 进行原子性拆分而已，在实际工作中，很可能功能实现和文档就应该放在一个提交当中。

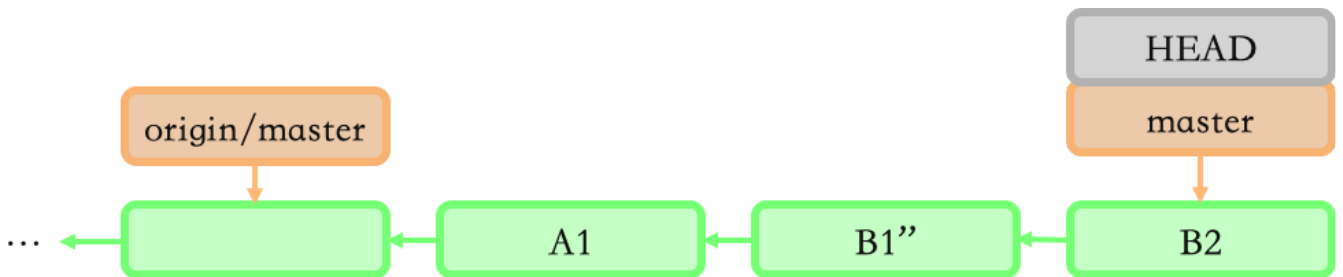


图 4 提交链状态第 4 步

提交 B1' 拆开之后，为了把 B1' ' 推送到 origin/master 上去，我需要要把 B1' ' 挪到 A1 的前面。首先，运行 `git rebase -i origin/master`。


```

1 > git rebase -i origin/master
2
3 ## 下面是弹出的编辑器
4 pick 0825c0b readme                ## 这个是 A1
5 pick 7d43442 Add getRandom endpoint ## 这个是 B1''
6 pick 68d813f [DO NOT PUSH] Add documentation for getRandom endpoint
7
8 # Rebase 7b6ea30..68d813f onto 7b6ea30 (3 commands)
9 ...

```

[复制代码](#)

然后，我把针对 B1' ' 的那一行挪到第一行，保存退出。

 复制代码

```
1 pick 7d43442 Add getRandom endpoint ## 这个是 B1''
2 pick 0825c0b readme ## 这个是 A1
3 pick 68d813f [DO NOT PUSH] Add documentation for getRandom endpoint
4
5 # Rebase 7b6ea30..68d813f onto 7b6ea30 (3 commands)
6 ...
```

git rebase -i 命令会显示运行成功，使用 git log 命令可以看到我成功改变了提交的顺序。

 复制代码

```
1 > git rebase -i origin/master
2 Successfully rebased and updated refs/heads/master.
3
4 > git log --oneline --graph
5 * 86126f7 (HEAD -> master) [DO NOT PUSH] Add documentation for getRandom endpo
6 * 7113c16 readme
7 * 4d37768 Add getRandom endpoint
8 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
```

现在，提交链上有 B1' ' '、A1'、B2' 三个提交了。请注意，B2' 也是一个新的提交。虽然我只是交换了 B1' ' 和 A 的顺序，但 git rebase 的操作是重新应用，产生出了三个新提交。

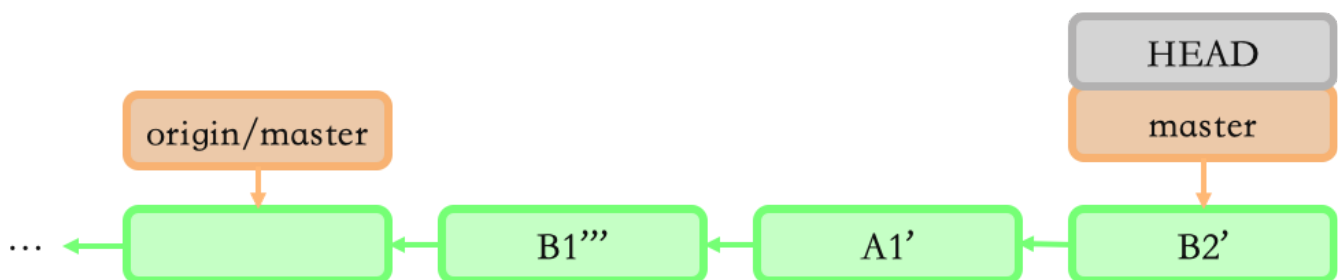


图 5 提交链状态第 5 步

现在，我可以把 B1' ' ' 发送给质量检查系统了。

首先，产生一个临时分支 temp 指向 B2'，确保能回到原来的代码；然后，用 git reset --hard 命令把 master 和 HEAD 指向 B1' ' '。

```

1 > git branch temp
2 > git reset --hard 4d37768
3 HEAD is now at 4d37768 Add getRandom endpoint
4
5 ## 检查提交链
6 > git log --oneline --graph
7 * 4d37768 (HEAD -> master) Add getRandom endpoint
8 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
9 ...

```

这时，提交链中只有 B1' ' '。当然，A1' 和 B2' 仍然存在，只是不在提交链里了而已。

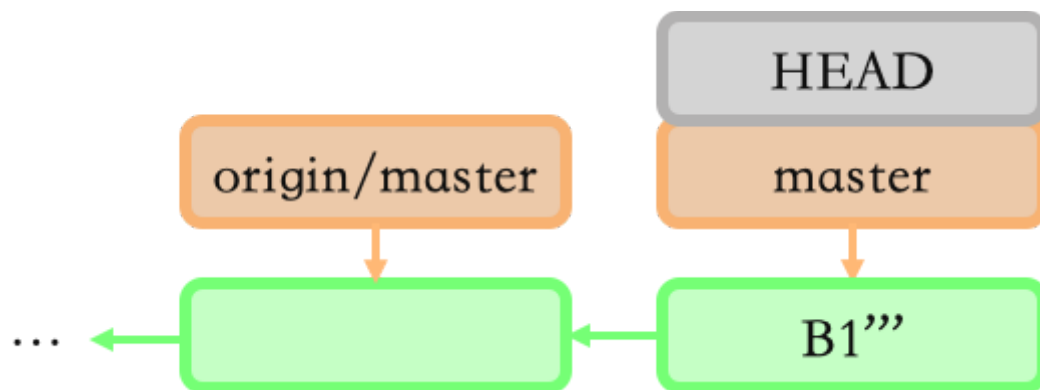


图 6 提交链状态第 6 步

最后，运行命令把 B1' ' ' 提交到 Phabricator 上，结束后使用 `git reset --hard temp` 命令重新把 HEAD 指向 B2' 。

```

1 ## 运行 arc 命令把 B'''提交到 Phabricator 上
2 > arc diff
3
4
5 ## 重新把 HEAD 指向 B2'
6 > git reset --hard temp
7 HEAD is now at 86126f7 [DO NOT PUSH] Add documentation for getRandom endpoint
8
9
10 ## 检查提交链
11 > git log --oneline --graph
12 * 86126f7 (HEAD -> master, temp, single-branch-step-5) [DO NOT PUSH] Add docum
13 * 7113c16 readme
14 * 4d37768 Add getRandom endpoint

```

```
15 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
```

这时，提交链又恢复成为 B1' ' '、A1'、B2' 三个提交了。

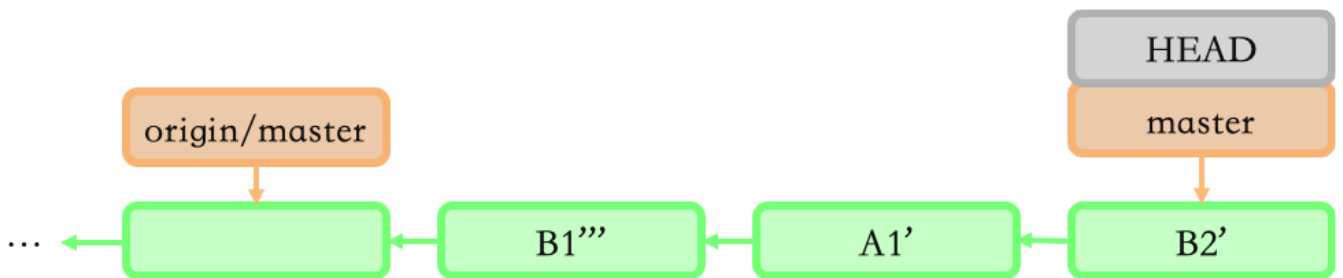


图 7 提交链状态第 7 步

## 阶段 4：继续开发 B2，同时得到 B1 的反馈，修改 B1

把 B1' ' ' 发送到质量检查中心之后，我回到 B2' 继续工作，也就是在 README 文件中继续添加关于 getRandom 的文档。我正在开发的过程中，得到 B1' ' ' 的反馈，要求我对其进行修改。于是，我首先保存当前对 B2' 的修改，用 `git commit --amend` 把它添加到 B2' 中。

 复制代码

```
1 ## 查看工作区中的修改
2 > git diff
3 diff --git a/README.md b/README.md
4 index 8a60943..1f06f52 100644
5 --- a/README.md
6 +++ b/README.md
7 @@ -1,3 +1,4 @@
8  ## This project is for demoing git
9
10 You can visit endpoint getRandom to get a random real number.
11 +The end endpoint is `/getRandom`.
12
13
14 ## 把工作区中的修改添加到 B2' 中
15 > git add README.md
16 > git commit --amend
17 [master 7b4269c] [DO NOT PUSH] Add documentation for getRandom endpoint
18 Date: Tue Oct 15 17:17:18 2019 +0800
19 1 file changed, 3 insertions(+)
20 * 7b4269c (HEAD -> master) [DO NOT PUSH] Add documentation for getRandom endpo
21 * 7113c16 readme
22 * 4d37768 Add getRandom endpoint
```

```
23 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
24 ...
```

这时，提交链成为 B1' ' '、A1'、B2' ' 三个提交了。

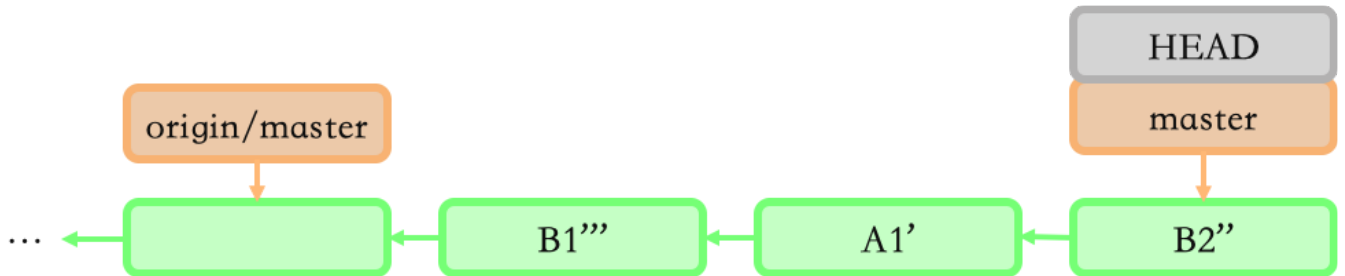


图 8 提交链状态第 8 步

接下来，我使用 [第 25 篇文章](#) 中介绍的基础操作对 B1' ' ' 进行修改。

首先，在 `git rebase -i origin/master` 的文本输入框中，将 `pick B1' ' '` 那一行修改为 `edit B1' ' '`，然后保存退出，`git rebase` 暂停在 B1' ' ' 处：

```
1 > git rebase -i origin/master
2
3
4 ## 以下是弹出编辑器中的文本内容
5 edit 4d37768 Add getRandom endpoint    ## <-- 这一行开头原本是 pick
6 pick 7113c16 readme
7 pick 7b4269c [DO NOT PUSH] Add documentation for getRandom endpoint
8
9
10 ## 以下是保存退出后 git rebase -i origin/master 的输出
11 Stopped at 4d37768... Add getRandom endpoint
12 You can amend the commit now, with
13     git commit --amend
14 Once you are satisfied with your changes, run
15     git rebase --continue
16
17
18 ## 查看提交历史
19 > git log --oneline --graph
20 * 4d37768 (HEAD) Add getRandom endpoint
21 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
```

复制代码

这时，提交链上只有 B1' ' ' 一个提交。

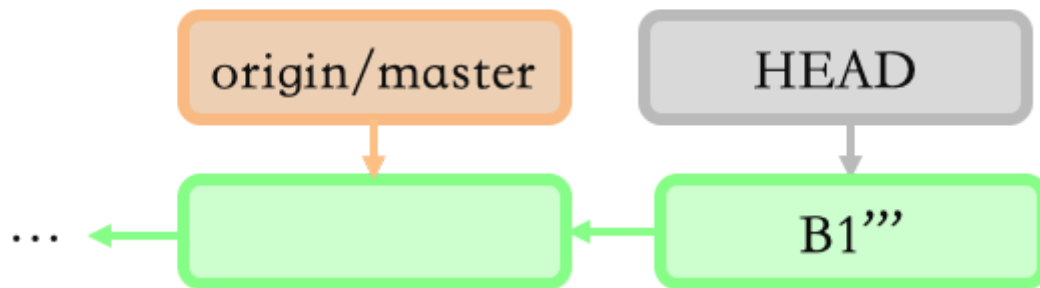


图 9 提交链状态第 9 步

然后，我对 index.js 进行修改，并添加到 B1' ' ' 中，成为 B1' ' ' '。完成之后，再次把 B1' ' ' ' 发送到代码质量检查系统。

复制代码

```
1 ## 根据同事反馈，修改 index.js
2 > vim index.js
3 > git add index.js
4
5
6 ## 查看修改
7 > git diff --cached
8 diff --git a/index.js b/index.js
9 index 06695f6..cc92a42 100644
10 --- a/index.js
11 +++ b/index.js
12 @@ -7,7 +7,7 @@ app.get('/timestamp', function (req, res) {
13   })
14
15   app.get('/getRandom', function (req, res) {
16 -   res.send('' + Math.random())
17 +   res.send('The random number is:' + Math.random())
18   })
19
20   app.get('/', function (req, res) {
21
22
23 ## 把改动添加到 B1'''中。
24 > git commit --amend
25 [detached HEAD 29c8249] Add getRandom endpoint
26 Date: Tue Oct 15 17:16:12 2019 +0800
27 1 file changed, 4 insertions(+)
28 19:17:28 (master|REBASE-i) jasonge@Juns-MacBook-Pro-2.local:~/jksj-repo/git-at
29 > git show
30 commit 29c82490256459539c4a1f79f04823044f382d2b (HEAD)
31 Author: Jason Ge <gejun_1978@yahoo.com>
```



```

32 Date:    Tue Oct 15 17:16:12 2019
33     Add getRandom endpoint
34
35     Summary:
36     As title.
37
38     Test:
39     Verified it on localhost:3000/getRandom
40
41 diff --git a/index.js b/index.js
42 index 986fcd8..cc92a42 100644
43 --- a/index.js
44 +++ b/index.js
45 @@ -6,6 +6,10 @@ app.get('/timestamp', function (req, res) {
46     res.send('' + Date.now())
47 })
48
49 +app.get('/getRandom', function (req, res) {
50 + res.send('The random number is:' + Math.random())
51 +})
52 +
53 app.get('/', function (req, res) {
54     res.send('hello world')
55 })
56
57
58 ## 查看提交链
59 > git log --oneline --graph
60 * 29c8249 (HEAD) Add getRandom endpoint
61 * 7b6ea30 (origin/master, git-add-p) Add a new endpoint to return timestamp
62
63
64 ## 将 B1''''发送到代码审查系统
65 > arc diff

```

这时，提交链只有 B1'''' 一个提交。

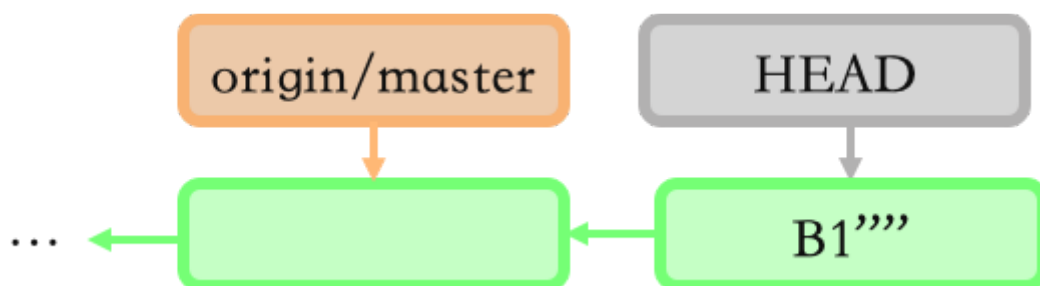



图 10 提交链状态第 10 步

最后，运行 `git rebase --continue` 完成整个 `git rebase -i` 操作。

 复制代码

```
1 > git rebase --continue
2 Successfully rebased and updated refs/heads/master.
3
4
5 ## 查看提交历史
6 > git log --oneline --graph
7 * bc0900d (HEAD -> master) [DO NOT PUSH] Add documentation for getRandom endpo
8 * 1562cc7 readme
9 * 29c8249 Add getRandom endpoint
10 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
11 ...
```

这时，提交链包含 `B1''''`、`A1''`、`B2''''` 三个提交。

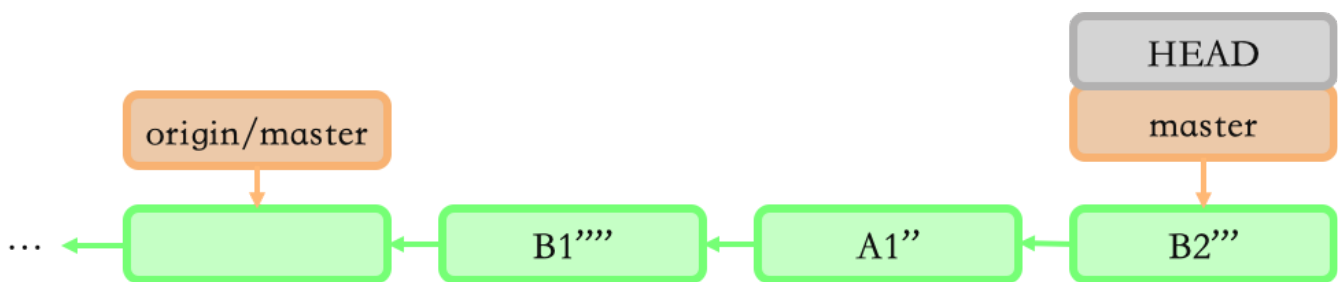



图 11 提交链状态第 11 步

## 阶段 5：继续开发 `A1`，并发出代码审查

这时，我认为 `A1''` 比 `B2''''` 更为紧急重要，于是决定先完成 `A1''` 的工作并发送审查，同样也是使用 `git rebase -i`。


 复制代码

```
1 > git rebase -i HEAD^^ ## 两个 ^^ 表示从当前 HEAD 前面两个提交的地方 rebase
2
3
4 ## git rebase 弹出编辑窗口
5 edit 1562cc7 readme <-- 这一行开头原来是 pick。这个是 A1''
6 pick bc0900d [DO NOT PUSH] Add documentation for getRandom endpoint
7
8
9 ## 保存退出后 git rebase -i HEAD^^ 的结果
10 Stopped at 1562cc7... readme
11 You can amend the commit now, with
```

```
12  git commit --amend
13  Once you are satisfied with your changes, run
14  git rebase --continue
15
16
17  ## 对 A1''修改
18  > vim README.md
19  > git diff
20  diff --git a/README.md b/README.md
21  index 789cfa9..09bcc7d 100644
22  --- a/README.md
23  +++ b/README.md
24  @@ -1,1 @@
25  -## This project is for demoing git
26  +## This project is for demoing atomic commit in git
27
28  > git add README.md
29  > git commit --amend
30
31
32  ## 下面是 git commit 弹出编辑器, 在里面完善 A1''的 Commit Message
33  Add README.md file
34
35  Summary: we need a README file for the project.
36
37  Test: none.
38
39  # Please enter the Commit Message for your changes. Lines starting
40  # with '#' will be ignored, and an empty message aborts the commit.
41  #
42  # Date:      Tue Oct 15 12:45:08 2019 +0800
43  #
44  # interactive rebase in progress; onto 29c8249
45  # Last command done (1 command done):
46  #   edit 1562cc7 readme
47  # Next command to do (1 remaining command):
48  #   pick bc0900d [DO NOT PUSH] Add documentation for getRandom endpoint
49  # You are currently splitting a commit while rebasing branch 'master' on '29c8:
50  #
51  # Changes to be committed:
52  #       new file:   README.md
53  #
54
55
56  ## 保存退出后 git commit 输出结果
57  [detached HEAD 2c66fe9] Add README.md file
58  Date: Tue Oct 15 12:45:08 2019 +0800
59  1 file changed, 1 insertion(+)
60  create mode 100644 README.md
61
62
63  ## 继续执行 git rebase -i
```

```
64 > git rebase --continue
65 Auto-merging README.md
66 CONFLICT (content): Merge conflict in README.md
67 error: could not apply bc0900d... [DO NOT PUSH] Add documentation for getRandom
68 Resolve all conflicts manually, mark them as resolved with
69 "git add/rm <conflicted_files>", then run "git rebase --continue".
70 You can instead skip this commit: run "git rebase --skip".
71 To abort and get back to the state before "git rebase", run "git rebase --abort".
72 Could not apply bc0900d... [DO NOT PUSH] Add documentation for getRandom endpoint
```

这个过程可能会出现冲突，比如在 A1' ' ' 之上应用 B2' ' ' 时可能会出现冲突。冲突出现时，你可以使用 `git log` 和 `git status` 命令查看细节。

 复制代码

```
1 ## 查看当前提交链
2 > git log --oneline --graph
3 * 2c66fe9 (HEAD) Add README.md file
4 * 29c8249 Add getRandom endpoint
5 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
6 ...
7
8
9 ## 查看冲突细节
10 > git status
11 interactive rebase in progress; onto 29c8249
12 Last commands done (2 commands done):
13   edit 1562cc7 readme
14   pick bc0900d [DO NOT PUSH] Add documentation for getRandom endpoint
15 No commands remaining.
16 You are currently rebasing branch 'master' on '29c8249'.
17   (fix conflicts and then run "git rebase --continue")
18   (use "git rebase --skip" to skip this patch)
19   (use "git rebase --abort" to check out the original branch)
20
21 Unmerged paths:
22   (use "git reset HEAD <file>..." to unstage)
23   (use "git add <file>..." to mark resolution)
24
25   both modified:   README.md
26
27 no changes added to commit (use "git add" and/or "git commit -a")
28
29
30 ## 用 git diff 和 git diff --cached 查看更多细节
31 > git diff
32 diff --cc README.md
33 index 09bcc7d,1f06f52..0000000
34 --- a/README.md
```

```
35 +++ b/README.md
36 @@@ -1,1 -1,4 +1,8 @@@
37 ++<<<<<< HEAD
38  + # This project is for demoing atomic commit in git
39  +=====
40  + ## This project is for demoing git
41  +
42  + You can visit endpoint getRandom to get a random real number.
43  + The end endpoint is `/getRandom`.
44  ++>>>>>> bc0900d... [DO NOT PUSH] Add documentation for getRandom endpoint
45
46 > git diff --cached
47 * Unmerged path README.md
```

解决冲突的具体步骤是：

1. 手动修改冲突文件；
2. 使用 git add 或者 git rm 把修改添加到暂存区；
3. 运行 git rebase --continue。于是，git rebase 会把暂存区的内容生成提交，并继续 git-rebase 后续步骤。

 复制代码

```
1 > vim README.md
2
3 ## 这个是初始内容
4 <<<<<< HEAD
5 # This project is for demoing atomic commit in git
6 =====
7 ## This project is for demoing git
8
9 You can visit endpoint getRandom to get a random real number.
10 The end endpoint is `/getRandom`.
11 >>>>>> bc0900d... [DO NOT PUSH] Add documentation for getRandom endpoint
12
13
14 ## 这个是修改后内容，并保存退出
15 # This project is for demoing atomic commit in git
16
17 You can visit endpoint getRandom to get a random real number.
18 The end endpoint is `/getRandom`.
19
20
21 ## 添加 README.md 到暂存区，并使用 git status 查看状态
22 > git add README.md
23 19:51:16 (master|REBASE-i) jasonge@Juns-MacBook-Pro-2.local:~/jksj-repo/git-at
```

```

24
25
26 ## 使用 git status 查看状态
27 > git status
28 interactive rebase in progress; onto 29c8249
29 Last commands done (2 commands done):
30     edit 1562cc7 readme
31     pick bc0900d [DO NOT PUSH] Add documentation for getRandom endpoint
32 No commands remaining.
33 You are currently rebasing branch 'master' on '29c8249'.
34     (all conflicts fixed: run "git rebase --continue")
35
36 Changes to be committed:
37     (use "git reset HEAD <file>..." to unstage)
38
39     modified:   README.md
40
41 ## 冲突成功解决, 继续 git rebase -i 后续步骤
42 > git rebase --continue
43
44
45 ## git rebase 提示编辑 B2''''的 Commit Message
46 [DO NOT PUSH] Add documentation for getRandom endpoint
47
48 Summary:
49 AT.
50
51 Test:
52 None.
53
54
55 ## 保存退出之后 git rebase --continue 的输出
56 [detached HEAD ae38d9e] [DO NOT PUSH] Add documentation for getRandom endpoint
57 1 file changed, 3 insertions(+)
58 Successfully rebased and updated refs/heads/master.
59
60
61 ## 检查提交链
62 * ae38d9e (HEAD -> master) [DO NOT PUSH] Add documentation for getRandom endpo
63 * 2c66fe9 Add README.md file
64 * 29c8249 Add getRandom endpoint
65 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
66

```

这时, 提交链上有 B1' ' ' '、A1' ' ' '、B2' ' ' ' 三个提交。

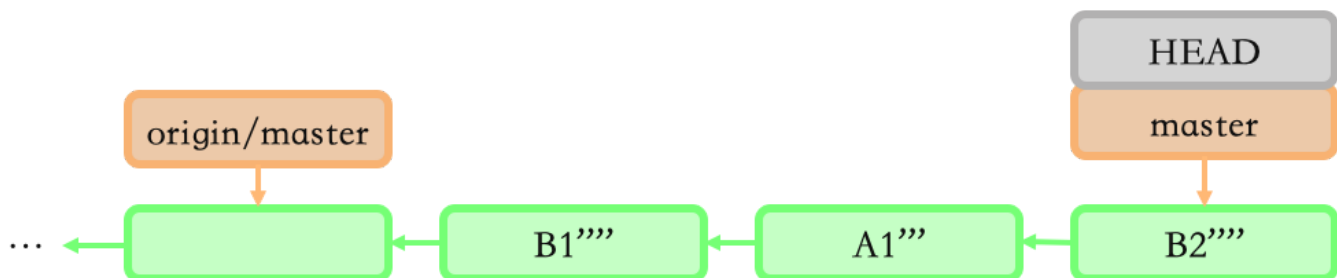


图 12 提交链状态第 12 步

## 阶段 6: B1 检查通过, 推送到远程代码仓共享分支

这时, 我从 Phabricator 得到通知, B1' ' ' ' 检查通过了, 可以将其推送到 origin/master 去了!

首先, 使用 `git fetch` 和 `git rebase origin/master` 命令, 确保本地有远端主代码仓的最新代码。

[复制代码](#)

```
1 > git fetch
2 > git rebase origin/master
3 Current branch master is up to date.
```

然后, 使用 `git rebase -i`, 在 B1' ' ' ' 处暂停:

[复制代码](#)

```
1 > git rebase -i origin/master
2
3 ## 修改第一行开头: pick -> edit
4 edit 29c8249 Add getRandom endpoint
5 pick 2c66fe9 Add README.md file
6 pick ae38d9e [DO NOT PUSH] Add documentation for getRandom endpoint
7
8
9 ## 保存退出结果
10 Stopped at 29c8249... Add getRandom endpoint
11 You can amend the commit now, with
12   git commit --amend
13 Once you are satisfied with your changes, run
14   git rebase --continue
15
16
17 ## 查看提交链
```



```
18 * 29c8249 (HEAD) Add getRandom endpoint
19 * 7b6ea30 (origin/master) Add a new endpoint to return timestamp
20 ...
```

这时，origin/master 和 HEAD 之间只有 B1' ' ' ' 一个提交。

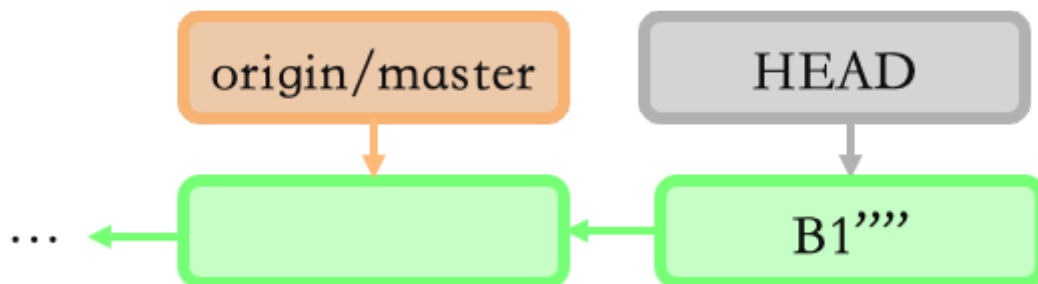


图 13 提交链状态第 13 步

我终于可以运行 `git push origin HEAD:master`，去推送 B1' ' ' ' 了。

注意，当前 HEAD 不在任何分支上，master 分支仍然指向 B2' ' ' '，所以 push 命令需要明确指向远端代码仓 origin 和远端分支 maser，以及本地要推送的分支 HEAD。推送完成之后，再运行 `git rebase --continue` 完成 rebase 操作，把 master 分支重新指向 B2' ' ' '。

 复制代码

```
1 ## 直接推送。因为当前 HEAD 不在任何分支上，推送失败。
2 > git push
3 fatal: You are not currently on a branch.
4 To push the history leading to the current (detached HEAD)
5 state now, use
6     git push origin HEAD:<name-of-remote-branch>
7
8
9 ## 再次推送，指定远端代码仓 origin 和远端分支 maser，以及本地要推送的分支 HEAD。推送成功
10 > git push origin HEAD:master
11 Enumerating objects: 5, done.
12 Counting objects: 100% (5/5), done.
13 Delta compression using up to 8 threads
14 Compressing objects: 100% (3/3), done.
15 Writing objects: 100% (3/3), 392 bytes | 392.00 KiB/s, done.
16 Total 3 (delta 2), reused 0 (delta 0)
17 remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
18 To github.com:jungejason/git-atomic-demo.git
19     7b6ea30..29c8249 HEAD -> master
```

```

20 > git rebase --continue
21 Successfully rebased and updated refs/heads/master.
22
23 ## 查看提交链
24 > git log --oneline --graph
25 * ae38d9e (HEAD -> master) [DO NOT PUSH] Add documentation for getRandom endpo
26 * 2c66fe9 Add README.md file
27 * 29c8249 (origin/master) Add getRandom endpoint
28 * 7b6ea30 Add a new endpoint to return timestamp
29

```

这时，origin/master 已经指向了 B1' ' ' '，提交链现在只剩下了 A1' ' ' 和 B2' ' ' '。

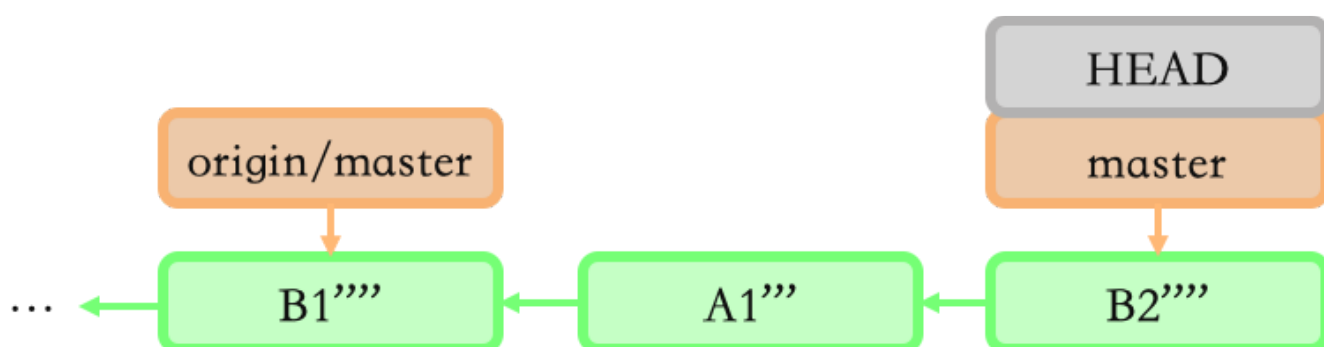


图 14 提交链状态第 14 步

至此，我们完成了在一个分支上同时开发两个需求 A 和 B、把提交拆分为原子性提交，并尽早把完成的提交推送到远端代码仓共享分支的全过程！

这个过程看起来比较复杂，但实际上就是根据上面列举的“单分支工作流”的 4 个步骤执行而已。

接下来，我们再看看使用多个分支，每个分支支持一个需求的开发方式。

## 用本地多分支实现多个需求的提交的原子性

在这种开发工作流下，每个需求都拥有独享的分支。同样的，跟单分支实现提交原子性的方式一样，这些分支都是本地分支，并不是主代码仓上的功能分支。

需要注意的是，在下面的分析中，我只描述每个分支上只有一个提交的简单形式，而至于每个分支上使用多个提交的形式，操作流程与单分支提交链中的描述一样，就不再重复表述

了。

## 多分支工作流具体步骤

分支工作流的具体步骤，大致包括以下 4 步：

1. 切换到某一个分支对某需求进行开发，产生提交。
2. 提交完成后，将其发送到代码审查系统 Phabricator 上进行质量检查。在等待质量检查结果的同时，切换到其他分支，继续其他需求的开发。
3. 如果第 2 步的提交没有通过质量检查，则切换回这个提交所在分支，对提交进行修改，修改之后返回第 2 步。
4. 如果第 2 步的提交通过了质量检查，则切换回这个提交所在分支，把这个提交推送到远端代码仓中，然后回到第 1 步进行其他需求的开发。


接下来，我们看一个开发两个需求 C 和 D 的场景吧。

在这个场景中，我首先开发需求 C，并把它的提交 C1 发送到质量检查中心；然后开始开发需求 D，等到 C1 通过质量检查之后，我立即将其推送到远程共享代码仓中去。

### 阶段 1：开发需求 C

需求 C 是一个简单的重构，把 index.js 中所有的 var 都改成 const。

首先，使用 `git checkout -b feature-c origin/master` 产生本地分支 feature-c，并跟踪 origin/master。

 复制代码

```
1 > git checkout -b feature-c origin/master
2 Branch 'feature-c' set up to track remote branch 'master' from 'origin'.
3 Switched to a new branch 'feature-c'
```

然后，进行 C 的开发，产生提交 C1，并把提交发送到 Phabricator 进行检查。

 复制代码

```
1 ## 修改代码，产生提交
```

```

2 > vim index.js
3 > git diff
4 diff --git a/index.js b/index.js
5 index cc92a42..e5908f0 100644
6 --- a/index.js
7 +++ b/index.js
8 @@ -1,6 +1,6 @@
9 -var port = 3000
10 -var express = require('express')
11 -var app = express()
12 +const port = 3000
13 +const express = require('express')
14 +const app = express()
15
16 app.get('/timestamp', function (req, res) {
17     res.send('' + Date.now())
18 20:54:10 (feature-c) jasonge@Juns-MacBook-Pro-2.local:~/jksj-repo/git-atomic-d
19
20 > git add .
21 20:54:16 (feature-c) jasonge@Juns-MacBook-Pro-2.local:~/jksj-repo/git-atomic-d
22
23 > git commit
24
25
26 ## 填写详细 Commit Message
27 Refactor to use const instead of var
28
29 Summary: const provides more info about a variable. Use it when possible.
30
31 Test: ran `node index.js` and verifeid it by visiting localhost:3000.
32 Endpoints still work.
33
34
35 ## 以下是 Commit Message 保存后退出, git commit 的输出结果
36 [feature-c 2122faa] Refactor to use const instead of var
37 1 file changed, 3 insertions(+), 3 deletions(-)
38
39
40 ## 使用 Phabricator 的客户端, arc, 把当前提交发送给 Phabricator 进行检查
41 > arc diff
42
43
44 ## 查看提交链
45 * 2122faa (HEAD -> feature-c, multi-branch-step-1) Refactor to use const inste
46 * 5055c14 (origin/master) Add documentation for getRandom endpoint
47 ...

```

这时, origin/master 之上只有 feature-c 一个分支, 上面有 C1 一个提交。

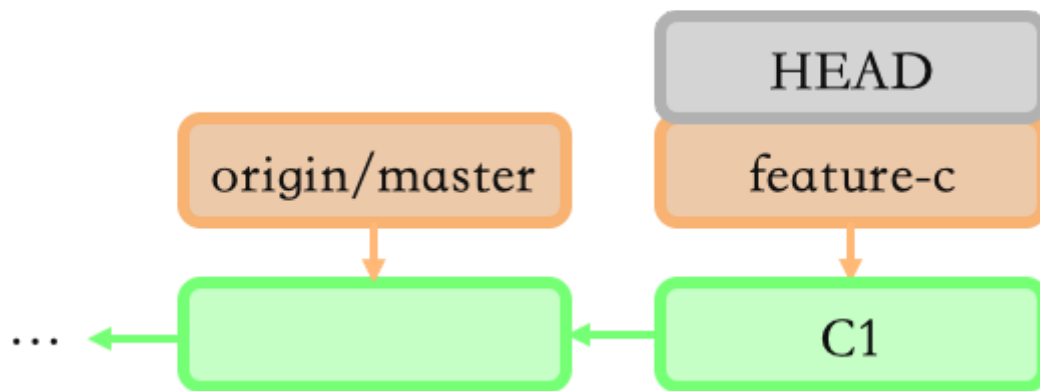


图 15 多分支提交状态第 1 步

## 阶段 2：开发需求 D

C1 发出去进行质量检查后，我开始开发需求 D。需求 D 是在 README.md 中，添加所有 endpoint 的文档。

首先，也是使用 `git checkout -b feature-d origin/master` 产生一个分支 `feature-d` 并跟踪 `origin/master`。

[复制代码](#)

```
1 > git checkout -b feature-d origin/master
2 Branch 'feature-d' set up to track remote branch 'master' from 'origin'.
3 Switched to a new branch 'feature-d'
4 Your branch is up to date with 'origin/master'.
```

然后，开始开发 D，产生提交 D1，并把提交发送到 Phabricator 进行检查。

[复制代码](#)

```
1 ## 进行修改
2 > vim README.md
3
4
5 ## 添加，产生修改，过程中有输入 Commit Message
6 > git add README.md
7 > git commit
8
9
10 ## 查看修改
11 > git show
12 commit 97047a33071420dce3b95b89f6d516e5c5b59ec9 (HEAD -> feature-d, multi-bran
13 Author: Jason Ge <gejun_1978@yahoo.com>
```

```
14 Date:    Tue Oct 15 21:12:54 2019
15
16     Add spec for all endpoints
17
18     Summary: We are missing the spec for the endpoints. Adding them.
19
20     Test: none
21
22 diff --git a/README.md b/README.md
23 index 983cb1e..cbefdc3 100644
24 --- a/README.md
25 +++ b/README.md
26 @@ -1,4 +1,8 @@
27  # This project is for demoing atomic commit in git
28
29  -You can visit endpoint getRandom to get a random real number.
30  -The end endpoint is `/getRandom`.
31  +## endpoints
32  +
33  +* /getRandom: get a random real number.
34  +* /timestamp: get the current timestamp.
35  +* /: get a "hello world" message.
36
37
38  ## 将提交发送到 Phabricator 进行审查
39  > arc diff
40
41
42  ## 查看提交历史
43  > git log --oneline --graph feature-c feature-d
44  * 97047a3 (HEAD -> feature-d Add spec for all endpoints
45  | * 2122faa (feature-c) Refactor to use const instead of var
46  |/
47  * 5055c14 (origin/master) Add documentation for getRandom endpoint
```

这时，origin/master 之上有 feature-c 和 feature-d 两个分支，分别有 C1 和 D1 两个提交。

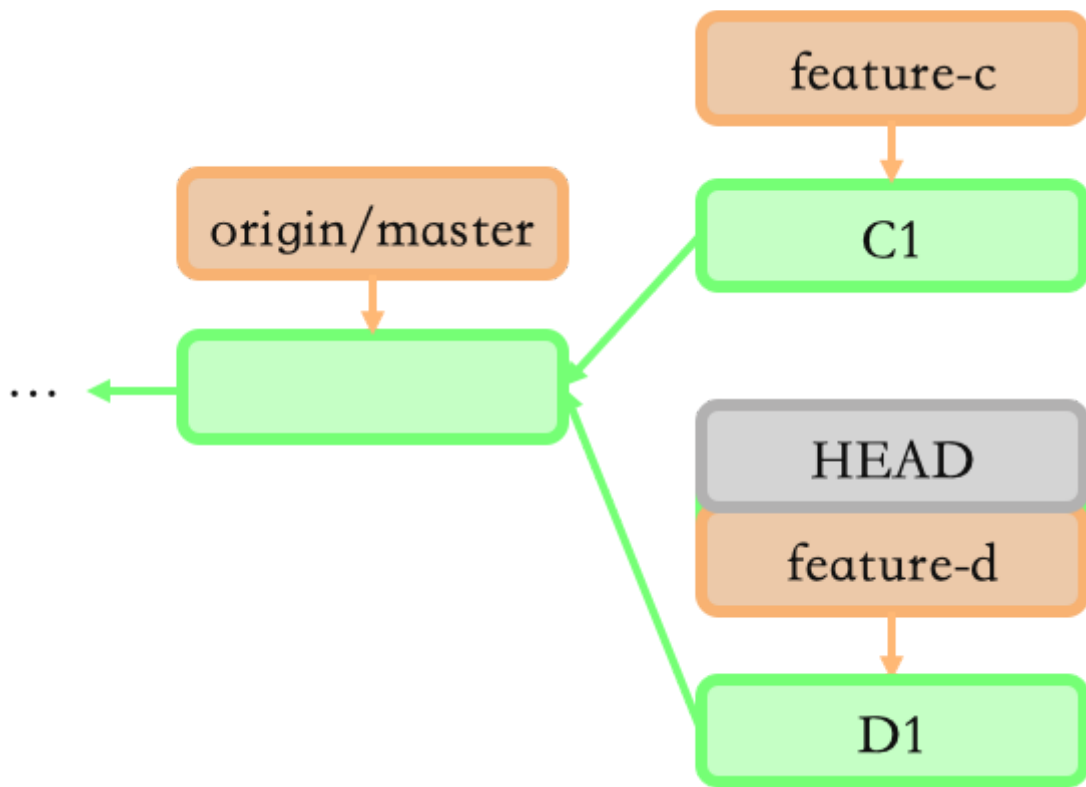


图 16 多分支提交状态第 2 步

### 阶段 3：推送提交 C1 到远端代码仓共享分支

这时，我收到 Phabricator 的通知，C1 通过了检查，可以推送了！首先，我使用 git checkout 把分支切换回分支 feature-c:

[复制代码](#)

```
1 > git checkout feature-c
2 Switched to branch 'feature-c'
3 Your branch is ahead of 'origin/master' by 1 commit.
4   (use "git push" to publish your local commits)
```


然后，运行 git fetch; git rebase origin/master，确保我的分支上有最新的远程共享分支代码：

[复制代码](#)

```
1 > git fetch
2 > git rebase origin/master
3 Current branch feature-c is up to date.
```



接下来, 运行 git push 推送 C1:

 复制代码

```
1 > git push
2 Enumerating objects: 5, done.
3 Counting objects: 100% (5/5), done.
4 Delta compression using up to 8 threads
5 Compressing objects: 100% (3/3), done.
6 Writing objects: 100% (3/3), 460 bytes | 460.00 KiB/s, done.
7 Total 3 (delta 2), reused 0 (delta 0)
8 remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
9 To github.com:jungejason/git-atomic-demo.git
10    5055c14..2122faa  feature-c -> master
11
12
13 ## 查看提交状态
14 * 97047a3 (feature-d) Add spec for all endpoints
15 | * 2122faa (HEAD -> feature-c, origin/master, multi-branch-step-1) Refactor t
16 |/
17 * 5055c14 Add documentation for getRandom endpoint
18 ...
```

这时, origin/master 指向 C1。分支 feature-d 从 origin/master 的父提交上分叉, 上面只有 D1 一个提交。

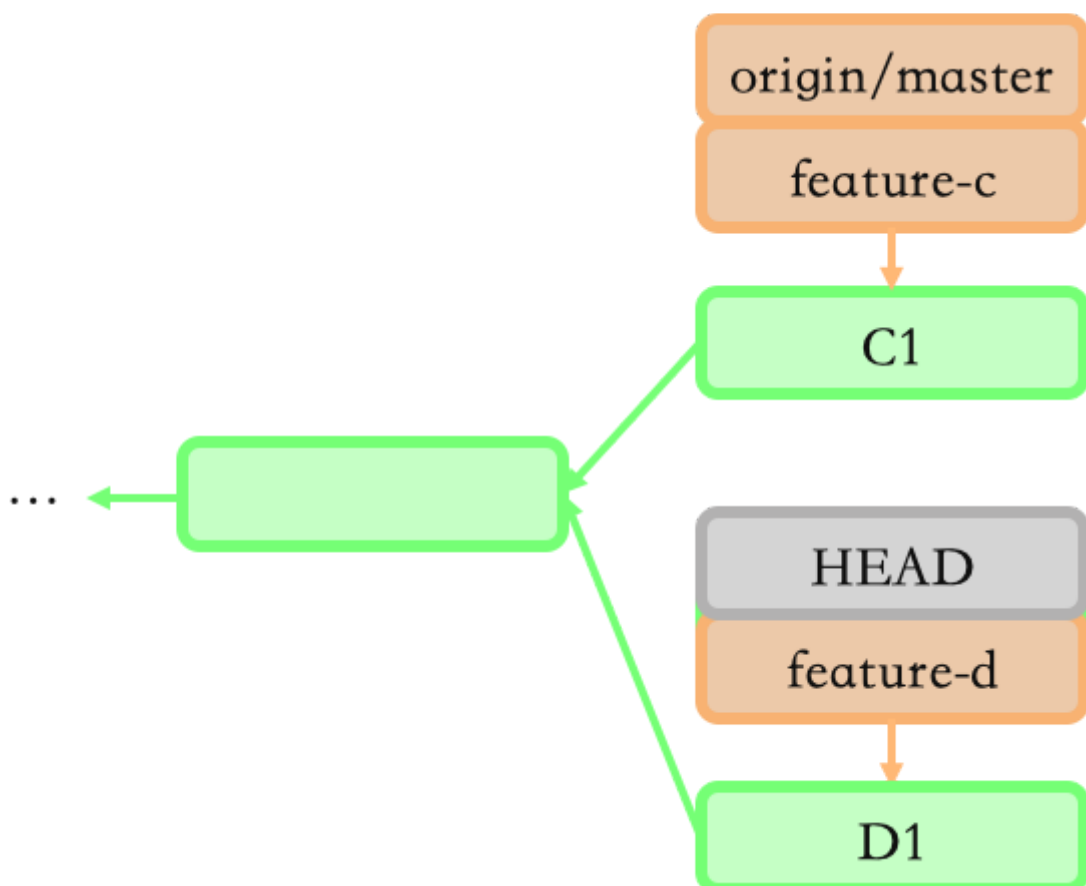


图 17 多分支提交状态第 3 步

## 阶段 4：继续开发 D1

完成 C1 的推送后，我继续开发 D1。首先，用 `git checkout` 命令切换回分支 `feature-d`；然后，运行 `git fetch` 和 `git rebase`，确保当前代码 D1 是包含了远程代码仓最新的代码，以减少将来合并代码产生冲突的可能性。

 复制代码

```
1 > git checkout feature-d
2 Switched to branch 'feature-d'
3 Your branch and 'origin/master' have diverged,
4 and have 1 and 1 different commits each, respectively.
5   (use "git pull" to merge the remote branch into yours)
6 21:38:22 (feature-d) jasonge@Juns-MacBook-Pro-2.local:~/jksj-repo/git-atomic-d
7
8 > git fetch
9 > git rebase origin/master
10 First, rewinding head to replay your work on top of it...
11 Applying: Add spec for all endpoints
12
13
14 ## 查看提交状态
15 > git log --oneline --graph feature-c feature-d
16 * a8f92f5 (HEAD -> feature-d) Add spec for all endpoints
17 * 2122faa (origin/master,) Refactor to use const instead of var
18 ...
```

这时，当前分支为 `feature-d`，上面有唯一一个提交 D1'，而且 D1' 已经变基到了 `origin/master` 上。

需要注意的是，因为使用的是 `git rebase`，没有使用 `git merge` 产生合并提交，所以提交历史是线性的。我在 [第 7 篇文章](#) 中提到过，线性的提交历史对 Facebook 的 CI 自动化意义重大。

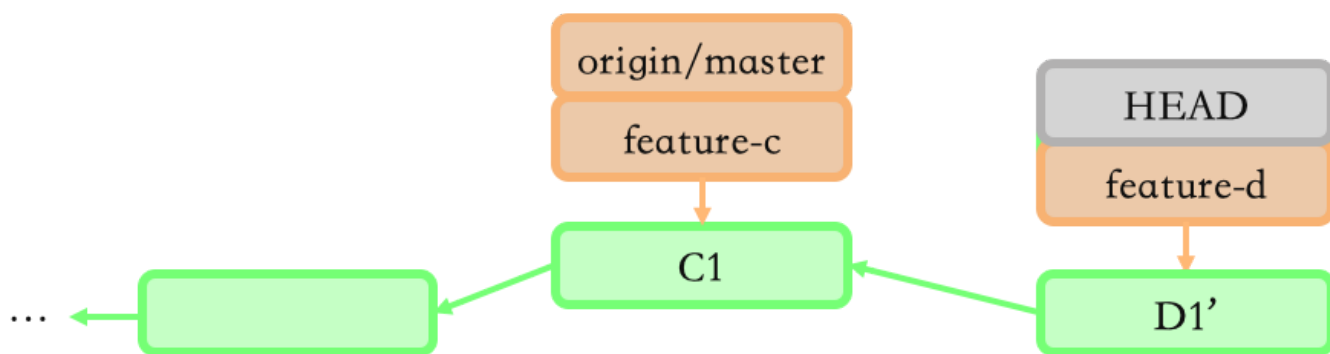


图 18 多分支提交状态第 4 步

至此，我们完成了在两个分支上同时开发 C 和 D 两个需求，并尽早把完成了的提交推送到远端代码仓中的全过程。

虽然在这个例子中，我简化了这两个需求开发的情况，每个需求只有一个提交并且一次就通过了质量检查，但结合在一个分支上完成所有开发需求的流程，相信你也可以推导出每个需求有多个提交，以及质量检查没有通过时的处理方法了。如果这中间还有什么问题的话，那就直接留言给我吧。

接下来，我与你对比下这两种工作流。

## 两种工作流的对比

如果我们要对比这两工作流的话，那就是各有利弊。

单分支开发方式的好处是，不需要切换分支，可以顺手解决一些缺陷修复，但缺点是 rebase 操作多，产生冲突的可能性大。

而多分支方式的好处是，一个分支只对应一个需求，相对比较简单、清晰，rebase 操作较少，产生冲突的可能性小，但缺点是不如单分支开发方式灵活。

无论是采用哪一种工作流，都有几个需要注意的地方：

不要同时开发太多的需求，否则分支管理花销太大；

有了可以推送的提交就尽快推送到远端代码仓，从而减少在本地的管理成本，以及推送时产生冲突的可能性；

经常使用 `git fetch` 和 `git rebase`，确保自己的代码在本地持续与远程共享分支的代码在做集成，降低推送时冲突的可能性

最后，我想说的是，如果你对 Git 不是特别熟悉，我推荐你先尝试第二种工作流。这种情况 `rebase` 操作较少，相对容易上手一些。

## 小结

今天，我与你详细讲述了，在 Facebook 开发人员借助 Git 的强大功能，实现代码提交的原子性的两种工作流。

第一种工作流是，在一个单独的分支上进行多个需求的开发。总结来讲，具体的工作方法是：把每一个需求的提交都拆分为比较小的原子提交，并使用 `git rebase -i` 的功能，把可以进行质量检查的提交，放到提交链的最底部，也就是最接近 `origin/master` 的地方，然后发送到代码检查系统进行检查，之后继续在提交链的其他提交处工作。如果提交没有通过检查，就对它进行修改再提交检查；如果检查通过，就马上把它推送到远端代码仓的共享分支去。在等待代码检查时，继续在提交链的其他提交处工作。

第二种工作流是，使用多个分支来开发多个需求，每个分支对应一个需求。与单分支开发流程类似，我们尽快把当前可以进行代码检查的提交，放到离 `origin/master` 最近的地方；然后在代码审查时，继续开发其他提交。与单分支开发流程不同的是，切换工作任务时，需要切换分支。

这两种工作流，无论哪一种都能大大促进代码提交的原子性，从而同时提高个人及团队的研发效能。

我把今天的案例放到了 GitHub 上的 [🔗 git-atomic-demo](#) 代码仓里，并标注出了各个提交状态产生的分支。比如，`single-branch-step-14` 就是单分支流程中的第 14 个状态，`multi-branch-step-4` 就是多分支流程中的第 4 个状态。

## 思考题

1. 在对提交链的非当前提交，比如 `HEAD^`，进行修改时，除了使用 `git rebase -i`，在它暂停的时候进行修改，你还其他办法吗？你觉得这种方法的利弊是什么？
2. Git 和文字编排系统 LaTeX 有一个有趣的共同点，你知道是什么吗？

感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎你把这篇文章分享给更多的朋友一起阅读。我们下期再见！



# 研发效率破局之道

Facebook 研发效率工作法

葛俊

前 Facebook 内部工具团队 Tech Lead



新版升级：点击「👤 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 25 | 玩转Git：五种提高代码提交原子性的基本操作

下一篇 27 | 命令行：不只是酷，更重要的是能提高个人效能

## 精选留言 (3)

写留言



Weining Cao

2019-10-23

从来不用rebase, 一直用merge, 看来要好好学习下rebase了

作者回复: <http://onlywei.github.io/explain-git-with-d3/#rebase>

交互式的界面学rebase。推荐看看：)





我来也

2019-10-23

学习了，这两种方式我都会，也经常用。  
我比较喜欢线性的提交历史，不喜欢merge。

但有个比较纠结的是。  
比如采用第二种方式：...

展开 ∨

作者回复: 1-----

> 但有个比较纠结的是...

我一般是使用为rebase或者cherry-pick，把这些分支上的都放到合并到一个分支上，不使用merge --no-ff命令。

至于提交的粒度，就要考虑原子性了。尽量是单个提交只完成单个需求。当然，有些时候如果bug fix很小，也可以选择把多个bug fix合并成一个提交。

2-----

> 1. 先备份当前分支，比如git branch temp

> 2. git reset --hard HEAD^

> 3. 修改并git commit --amend --no-edit

> 4. git cherry-pick 后面的commit 😊

> 或者git checkout temp ; git rebase xxx 刚才修改后的commit（不知道不可行）

这两种办法都可以，都很好！

还有第三个办法，我一般是把提交交换顺序，把要修改的提交放到HEAD处修改。

另外，我还第一次见到--no-edit。我以前都是用 commit --amend -a -C HEAD来完成类似功能：)



Johnson

2019-10-23

第一种太烧脑了，对开发人员的git技能要求太高了，大部分的开发人员都不能正确驾驭，感觉工作中更多的是以第二种为主，然后在某个branch内部结合第一种的情况比较容易驾驭。很头疼的问题是很多老同事没有深入学习git的主动性和激情，稍微复杂点儿的操作就让别人帮忙操作。

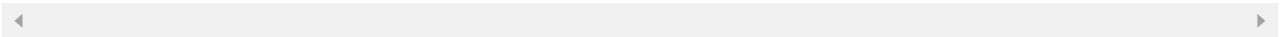
展开 ∨

作者回复: > 第二种为主, 然后在某个branch内部结合第一种的情况比较容易驾驭

的确是这样。第1种操作的确是要求比较高。而且如果有比较多的提交的话, rebase的overhead会比较大。所以单独的分支上不推荐, 产生太多的提交。

> 很头疼的问题是很多老同事没有深入学习git的主动性和激情

如果他们能够意识到git的强大能够帮助自己提高研发效率的话, 可能就会更愿意投入一些。能不能给他们搞点集体培训什么的? 对团队效能提升帮助应该不错



1

