**Git操作 - charry-pick命令详解**

对于多分支的代码库，将代码从一个分支转移到另一个分支是常见需求。

这时分两种情况：

* 一种情况是，你需要另一个分支所有代码的变动，那么就采用合并（git merge）。
* 另一种情况是，你只需要另一个分支的部分代码变动（某一个或几个提交），这时可以采用git cherry-pick。

**1、cherry-pick命令介绍**

git cherry-pick命令可以理解为"挑拣"提交，它会获取某一分支中的一个或者多个提交，并作为一个新的提交（或新的多个提交），引入到你当前的工作分支上。

也就是当我们需要在本地合并其他分支的提交时，如果我们不想对整个分支进行合并，而是只想将某一次提交合并到本地当前分支上，那么就要使用git cherry-pick了。

git cherry-pick命令说明：

|  |
| --- |
| Bash git cherry-pick [<options>] <commit-ish>...  常用options:  --quit 退出当前的chery-pick序列  --continue 继续当前的chery-pick序列  --abort 取消当前的chery-pick序列，恢复当前分支  -n, --no-commit 不自动提交  -e, --edit 编辑提交信息 |

**2、cherry-pick命令基本用法**

git cherry-pick命令的作用，就是将指定的提交（commit）应用于其他分支。

|  |
| --- |
| Bash $ git cherry-pick <commitHash> |

上面命令就会将指定的提交commitHash，应用于当前分支。这会在当前分支产生一个新的提交，当然它们的哈希值会不一样。

举例来说，代码仓库有master和feature两个分支。

|  |
| --- |
| Bash  a - b - c - d Master  \  e - f - g Feature |

现在将提交f应用到master分支。

|  |
| --- |
| Bash # 切换到 master 分支 $ git checkout master  # Cherry pick 操作 $ git cherry-pick f |

上面的操作完成以后，代码库就变成了下面的样子。

|  |
| --- |
| Bash  a - b - c - d - f Master  \  e - f - g Feature |

从上面可以看到，master分支的末尾增加了一个提交f。

git cherry-pick命令的参数，不一定是提交的哈希值，分支名也是可以的，表示转移该分支的最新提交。

|  |
| --- |
| Bash $ git cherry-pick feature |

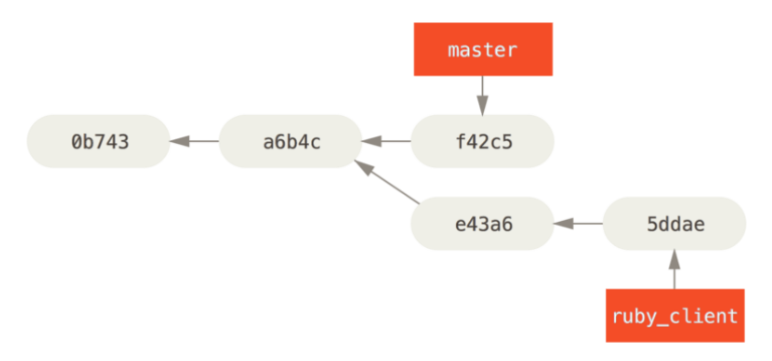
上面代码表示将feature分支的最近一次提交，转移到当前分支上。

如下：

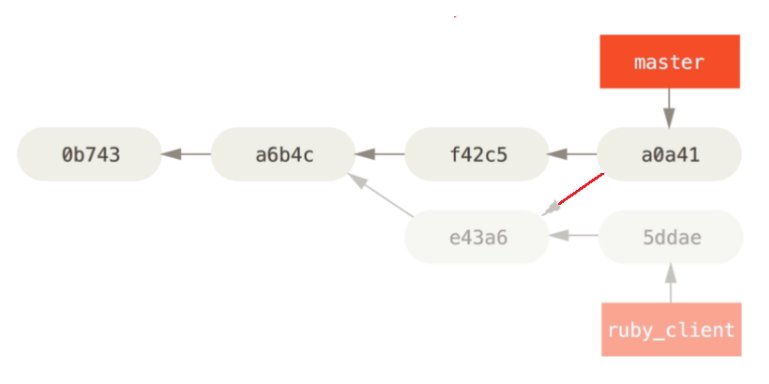
|  |
| --- |
| Bash  a - b - c - d - g Master  \  e - f - g Feature |

我们可以通过插图来观察：

cherry-pick操作前：



cherry-pick操作后：



我们可以看到a0a41是cherry-pick操作新生成的新提交，且形成了一个三方合并，会容易产生冲突。

**3、转移多个提交**

cherry-pick操作支持一次转移多个提交。

|  |
| --- |
| Bash $ git cherry-pick <HashA> <HashB> |

上面的命令将 A 和 B 两个提交应用到当前分支。这会在当前分支生成两个对应的新提交。

如果想要转移一系列的连续提交，可以使用下面的简便语法。

|  |
| --- |
| Bash $ git cherry-pick A..B |

上面的命令可以转移从 A 到 B 的所有提交。

它们必须按照正确的顺序放置：提交 A 必须早于提交 B，否则命令将失败，但不会报错。

注意，使用上面的命令，提交 A 将不会包含在git cherry-pick选择的提交中。如果要包含提交 A，可以使用下面的语法。

|  |
| --- |
| Bash $ git cherry-pick A^..B |

**4、其他用法**

把另一个分支中没有的提交，合并到当前工作分支。

使用命令：git cherry-pick ..< branchname > 或者 git cherry-pick ^HEAD < branchname >

以上两个命令作用相同，表示把一个分支中独有的提交合并到另一个分支上，共有的提交不进行合并。

假设：现在我的仓库中有三个分支，其提交历史如下：

|  |
| --- |
| Bash  C<---D<---E branch2  / master A<---B   \  F<---G<---H branch3  |  HEAD |

如果我使用 git cherry-pick ..branch2 命令或者 git cherry-pick ^HEAD branch2 命令，那么会将属于branch2分支的祖先，但不属于branch3分支的祖先的所有提交，都引入到当前分支branch3上，并生成新的提交。

执行命令后的提交历史如下：

|  |
| --- |
| Bash  C<---D<---E branch2  / master A<---B   \  F<---G<---H<---C<---D<---E branch3  |  HEAD |

**5、配置选项说明**

git cherry-pick命令的常用配置项如下。

* **（1）-e，--edit**：打开外部编辑器，编辑提交信息。
* **（2）-n，--no-commit**：只更新工作区和暂存区，不产生新的提交。
* **（3）-x**：在提交信息的末尾追加一行(cherry picked from commit ...)，方便以后查到这个提交是如何产生的。
* **（4）-s，--signoff**：在提交信息的末尾追加一行操作者的签名，表示是谁进行了这个操作。
* **（5）-m parent-number，--mainline parent-number**：如果原始提交是一个合并节点，来自于两个分支的合并，那么git cherry-pick命令默认将执行失败，因为它不知道应该采用哪个分支的代码变动。 而-m配置项告诉 Git，应该采用哪个分支的变动。它的参数parent-number是一个从1开始的整数，代表原始提交的父分支编号。 示例：$ git cherry-pick -m 1 <commitHash> 上面命令表示，git cherry-pick命令采用提交commitHash来自编号1的父分支的变动。 一般来说，1号父分支是接受变动的分支（the branch being merged into），2号父分支是作为变动来源的分支（the branch being merged from）。

**6、代码冲突**

如果操作过程中发生代码冲突，git cherry-pick命令会停下来，让用户决定如何继续操作。

**（1）--continue**

用户解决代码冲突后，第一步将修改的文件重新加入暂存区（git add .），第二步使用下面的命令，让git cherry-pick命令过程继续执行。

|  |
| --- |
| Bash $ git cherry-pick --continue |

**（2）--abort**

发生代码冲突后，放弃合并，回到操作前的样子。

**（3）--quit**

发生代码冲突后，退出git cherry-pick命令，但是不回到操作前的样子。

**7、转移到另一个代码库**

git cherry-pick命令也支持转移另一个代码库的提交，方法是先将该库加为远程仓库。

|  |
| --- |
| Bash # 添加了一个远程仓库`target`。 $ git remote add target git://gitUrl |

然后，将远程代码抓取到本地。

|  |
| --- |
| Bash # 将远程代码仓库抓取到本地。 $ git fetch target |

接着，检查一下要从远程仓库转移的提交，获取它的哈希值。

|  |
| --- |
| Bash $ git log target/master |

target/master是本地的远程跟踪分支（以后会说明）。

最后，使用git cherry-pick命令转移提交到当前工作分支。

|  |
| --- |
| Bash $ git cherry-pick <commitHash> |

**8、常见问题**

**（1）问题1**

The previous cherry-pick is now empty, possibly due to conflict resolution.：先前的cherry-pick操作为空，可能是由于解决冲突所致。

原因：在执行git cherry-pick命令时出现冲突，解决冲突后，本地分支中内容和cherry-pick操作之前相比没有改变。因此在以后的操作中，继续git cherry-pick命令或执行其他命令时，由于此时还处于上次cherry-pick状态，都会提示该信息，表示可能是由于解决冲突造成上一次cherry-pick操作的内容是空的。

解决方案：

1. 执行git cherry-pick --abort命令，取消上次cherry-pick操作。
2. 执行git commit --allow-empty命令，表示允许空提交。

**（2）问题2**

fatal: You are in the middle of a cherry-pick – cannot amend.：致命：您处于cherry-pick操作中，无法修改。

原因：在cherry-pick操作时出现冲突，没有解决冲突就执行git commit --amend命令，从而会提示该信息。

解决方案：

1. 首先在执行git commit --amend命令之前，先解决冲突。
2. 完成这次cherry-pick操作： $ git add . $ git cherry-pick --continue

|  |
| --- |
| 参考：   * [https://www.ruanyifeng.com/blog/2020/04/git-cherry-pick.html](https://links.jianshu.com/go?to=https%3A%2F%2Fwww.ruanyifeng.com%2Fblog%2F2020%2F04%2Fgit-cherry-pick.html) * [https://blog.csdn.net/FightFightFight/article/details/81039050](https://links.jianshu.com/go?to=https%3A%2F%2Fblog.csdn.net%2FFightFightFight%2Farticle%2Fdetails%2F81039050) |

作者：繁华似锦Fighting 链接：https://www.jianshu.com/p/231b80486641 来源：简书 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。