**git rebase 还是 merge的使用场景最通俗的解释**

[](https://www.jianshu.com/u/e2c97b29a17d)

[曹九朵\_](https://www.jianshu.com/u/e2c97b29a17d) 关注

 5.7 2018.06.26 15:19\* 字数 1013 阅读 13957评论 5喜欢 42

**什么是 rebase?**

**git rebase** 你其实可以把它理解成是“重新设置基线”，将你的当前分支重新设置开始点。这个时候才能知道你当前分支于你需要比较的分支之间的差异。  
原理很简单：rebase需要基于一个分支来设置你当前的分支的基线，这基线就是当前分支的开始时间轴向后移动到最新的跟踪分支的最后面，这样你的当前分支就是最新的跟踪分支。这里的操作是基于文件事务处理的，所以你不用怕中间失败会影响文件的一致性。在中间的过程中你可以随时取消rebase 事务。

官方解释: <https://git-scm.com/book/zh/v2/Git->分支-变基

**git rebase 和 git merge 有啥区别？**

**rebase**会把你当前分支的 commit 放到公共分支的最后面,所以叫变基。就好像你从公共分支又重新拉出来这个分支一样。  
举例:如果你从 master 拉了个feature分支出来,然后你提交了几个 commit,这个时候刚好有人把他开发的东西合并到 master 了,这个时候 master 就比你拉分支的时候多了几个 commit,如果这个时候你 rebase develop 的话，就会把你当前的几个 commit，放到那个人 commit 的后面。

rebase

**merge** 会把公共分支和你当前的commit 合并在一起，形成一个新的 commit 提交

merge

**注意:**

* 不要在公共分支使用rebase
* 本地和远端对应同一条分支,优先使用rebase,而不是merge

**抛出问题:**

**为什么不要再公共分支使用rebase?**  
因为往后放的这些 commit 都是新的,这样其他从这个公共分支拉出去的人，都需要再 rebase,相当于你 rebase 东西进来，就都是新的 commit 了

* 1-2-3 是现在的分支状态
* 这个时候从原来的master ,checkout出来一个prod分支
* 然后master提交了4.5，prod提交了6.7
* 这个时候master分支状态就是1-2-3-4-5，prod状态变成1-2-3-6-7
* 如果在prod上用rebase master ,prod分支状态就成了1-2-3-4-5-6-7
* 如果是merge  
  1-2-3-6-7-8  
  ........ |*4-5*|
* 会出来一个8，这个8的提交就是把4-5合进来的提交

**merge和rebase实际上只是用的场景不一样**  
**更通俗的解释一波.**  
比如rebase,你自己开发分支一直在做,然后某一天，你想把主线的修改合到你的分支上,做一次集成,这种情况就用rebase比较好.把你的提交都放在主线修改的头上  
如果用merge，脑袋上顶着一笔merge的8,你如果想回退你分支上的某个提交就很麻烦,还有一个重要的问题,rebase的话,本来我的分支是从3拉出来的,rebase完了之后,就不知道我当时是从哪儿拉出来的我的开发分支  
同样的,如果你在主分支上用rebase, rebase其他分支的修改,是不是要是别人想看主分支上有什么历史,他看到的就不是完整的历史课,这个历史已经被你篡改了

**常用指令**

* git rebase -I dev 可以将dev分支合并到当前分支  
  这里的”-i“是指交互模式。就是说你可以干预rebase这个事务的过程，包括设置commit message，暂停commit等等。
* git rebase –abort 放弃一次合并
* **合并多次commit操作:**  
  1 git rebase -i dev  
  2 修改最后几次commit记录中的pick 为squash  
  3 保存退出,弹出修改文件,修改commit记录再次保存退出(删除多余的change-id 只保留一个)  
  4 git add .  
  5 git rebase --continue