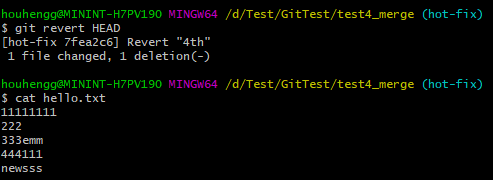
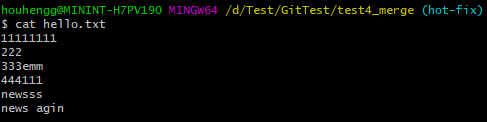
**1.git revert（一个commit < ——> 另一个commit ）**

该命令用于撤销某次操作，此次操作之前和之后的commit和history都会保留，并且把这次撤销作为一次最新的提交。

git revert HEAD 撤销前一次 commit

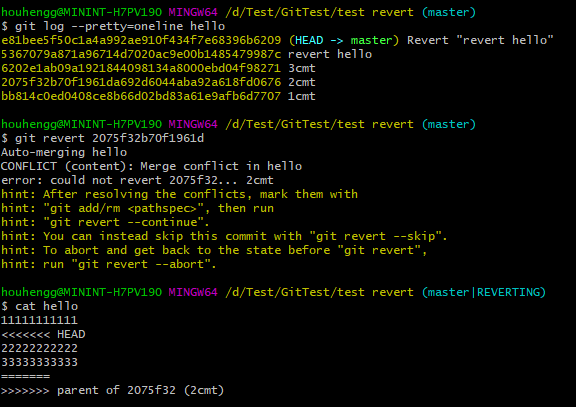
git revert HEAD^ 撤销前前一次 commit



git revert commit （比如：fa042ce57ebbe5bb9c8db709f719cec2c58ee7ff）

该命令用于撤销指定的版本，撤销也会作为一次提交进行保存。

如果出现类似merge冲突的情况，也就是让你做出选择，和merge冲突相似，===上面的是当前的冲突代码，==下面是历史版本的冲突代码，这里就需要手动更改。不需要checkout。



**2.git reset**

git reset 命令语法格式如下：

git reset [--soft | --mixed | --hard] [HEAD]

~~--mixed 为默认，可以不用带该参数，用于重置暂存区的文件与上一次的提交(commit)保持一致，工作区文件内容保持不变。~~

常用的命令：

git reset HEAD^

git reset --soft HEAD

git reset --hard HEAD

git reset --hard HEAD~3 # 回退上上上一个版本

git reset --hard [版本号] # 回退到某个版本回退点之前的所有信息。

~~git reset --hard origin/master # 将本地的状态回退到和远程的一样~~

git reset HEAD [file] 命令用于取消已缓存的内容。

HEAD 说明：

HEAD 表示当前版本

HEAD^ 上一个版本

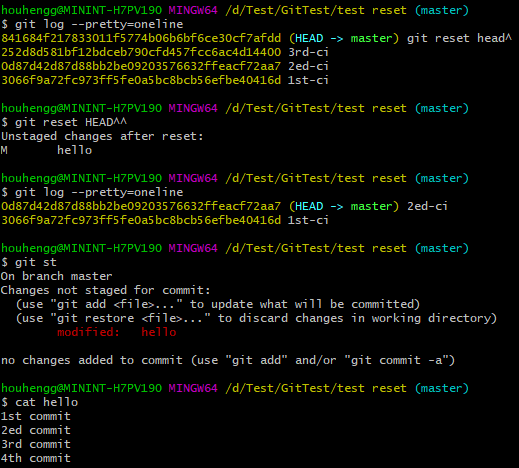
HEAD^^ 上上一个版本。以此类推...

可以使用 ～数字表示：HEAD~1 上一个版本

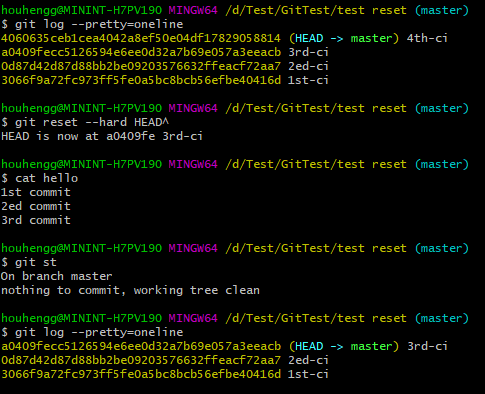
~~--soft 参数用于回退到某个版本;~~

~~--hard 参数撤销工作区中所有未提交的修改内容，将暂存区与工作区都回到上一次版本，并删除之前的所有信息提交：~~

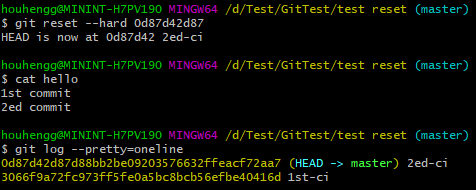
1.运行 git reset HEAD^命令

运行git reset HEAD命令更改了head的指向，指向了历史版本。并且状态由本地库变成了工作区(待add)，文件并没有被改变。类似于git reset –soft命令。

2.运行git reset --hard HEAD^命令

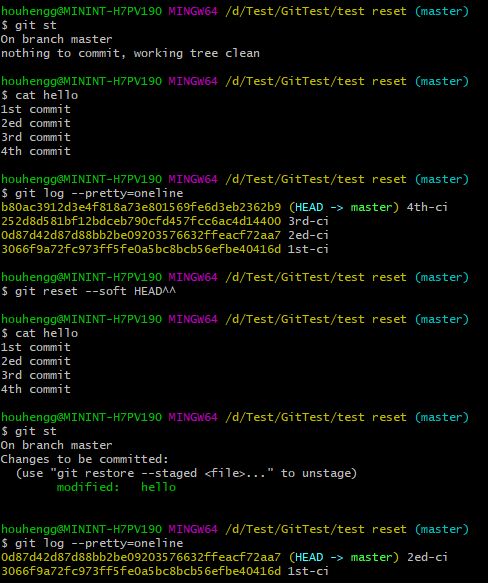


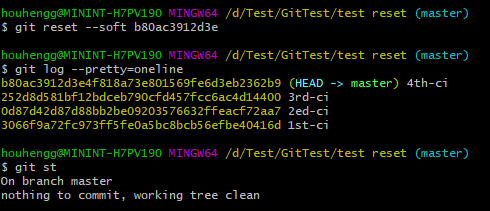
运行git reset --hard [版本号] 命令

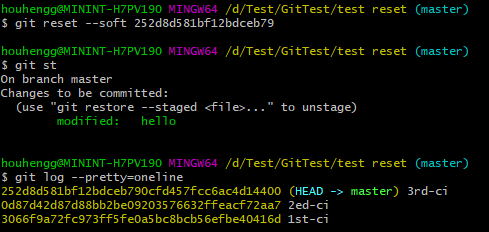


~~git reset --hard origin/master 【暂时无法验证】~~

3.运行git reset --soft HEAD^^ 命令



根据操作记录可以看出用git reset –soft命令后head会指向退回的历史版本，状态由本地库退回到了暂存区，但是暂存区的版本还是操作前的版本即当前版本。可以这么理解，git reset –soft命令只是将提交到本地库的版本退回到历史版本，而暂存区的文件不变，并且状态是暂存区的待提交状态。

这里可以看出soft命令可以回溯到任意版本，在使用的时候，不会删除其他版本。

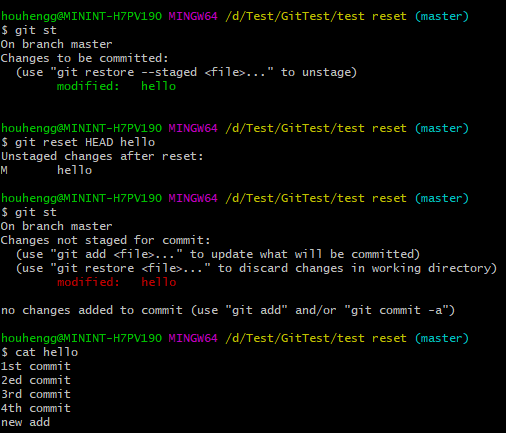
有一点不同的是，如果用soft回溯到最新本版，状态为已提交状态；如果是回溯到其他之前的版本，状态是待提交状态。

小结：Soft与hard的区别

根据对git reset hrad 的演示图片可以看出：hard操作将会删除退回之前的所有信息，也就是退回到的历史版本之后的版本的所有信息。操作之后的状态是本地库状态。

而git reset soft命令并不会删除其他版本，也就是运行这个命令后还可以回到以前的任何版本。

4.运行git reset HEAD [file] 命令

该命令将文件由暂存区退回到工作区，但不改变文件内容。

（如果在本地库执行此命令，将无效。）

**3.git revert 和 git reset的区别**

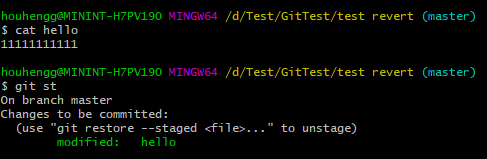
1. git revert是用一次新的commit来回滚之前的commit，git reset是直接删除指定的commit。

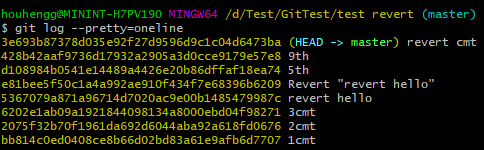
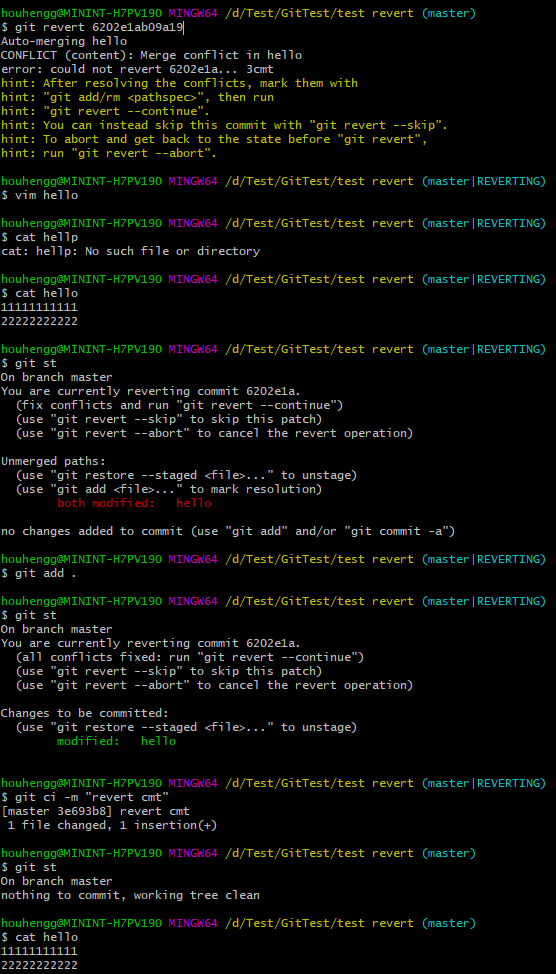
2. 在回滚这一操作上看，效果差不多。但是在日后继续merge以前的老版本时有区别。因为git revert是用一次逆向的commit“中和”之前的提交，

因此日后合并老的branch时，导致这部分改变不会再次出现，但是git reset是之间把某些commit在某个branch上删除，因而和老的branch再次merge时，这些被回滚的commit应该还会被引入。

3. git reset 是把HEAD向后移动了一下，而git revert是HEAD继续前进，只是新的commit的内容和要revert的内容正好相反，能够抵消要被revert的内容。

**Revert的操作：**





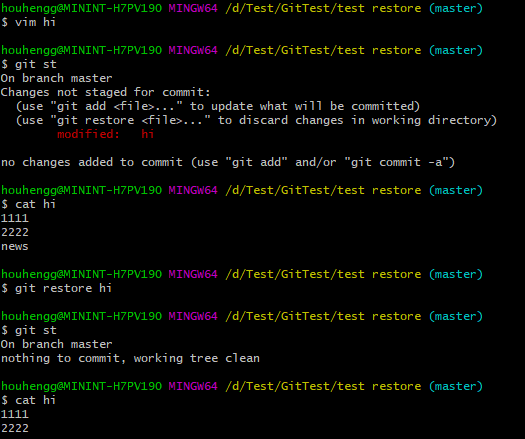
**Git restore**

此命令用于把文件从缓存区撤销，回到未被追踪的状态。

有两个常用的用法：git restore <file>和git restore --staged <file>

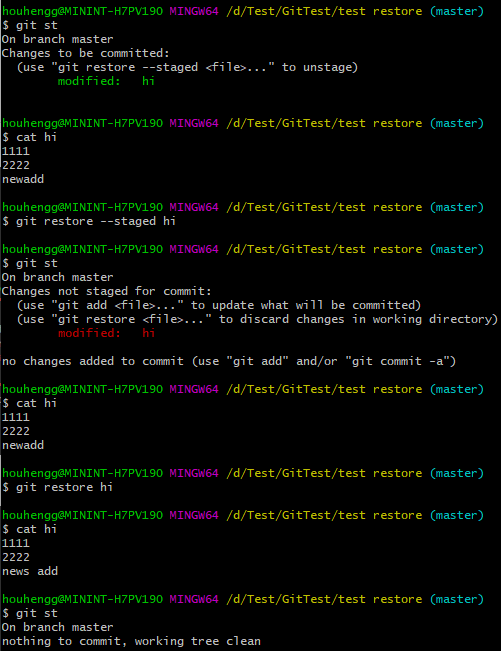
1.git restore <file>

在工作区运行此命令：文件会撤回到之前本地库的状态，并发生修改。【待测试】



在暂存区运行此命令：无效。此时应该修改命令为：git restore –satged <file>

运行命令后，文件退回到工作区状态，根据提示再运行git restore <file>回到之前的本地库，并被修改。



在本地库运行此命令：无任何效果

小结：git restore <file> 将工作区已修改的文件撤回到之前本地库的状态。

Git restore –staged 将已经添加到缓存区的文件撤回到工作区状态，文件不会发生修改。

**~~撤销提交commit：（暂存区<——>本地库）~~**

~~想要撤消刚才的提交操作，可以使用 --amend 选项重新提交：$ git commit --amend 【这里不理解，还需要研究】~~

~~$ git commit --amend命令的描述：~~

~~合并缓存区的修改和最近的一次commit, 然后用生成的新的commit替换掉老的。~~

~~如果缓存区没有内容, 那么利用amend可以修改上一次commit的描述。~~

~~不要对一个公共的commit使用amend：~~

~~amend后生成的commit是一个全新的commit, 之前的老的commit会从项目历史中被删除。~~

~~如果你amend了一个被其他开发者使用的commit, 会严重影响其他开发者。~~

**~~git restore --stage <file>和git restore <file>两个命令区分：~~**

~~1、文件在暂存区且未作修改的情况~~

~~使用git restore --staged <file> 把文件从暂存区移动到工作区，即文件不被追踪；~~

~~2、文件在暂存区且已经修改的情况~~

~~使用git restore --staged <file> 把文件从暂存区移动到工作区，且不会撤销修改的内容；~~

~~使用git restore <file> 文件仍在暂存区且会撤销文件修改的内容；~~

~~3、文件在本地库已经修改的情况~~

~~使用git add <file> 把文件重新放到暂存区，且保留文件的修改；~~

~~使用git restore <file> 文件仍在本地代码库且会撤销文件的修改；~~

~~小结：~~

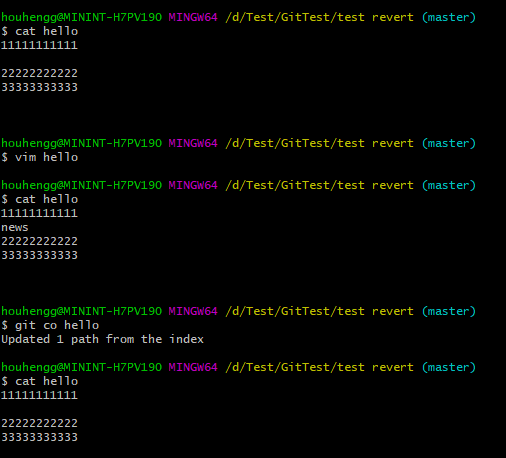
~~对于git restore <file>命令，会撤销文件的修改，使文件恢复到暂存区或本地代码库（取决于文件在修改前的状态）；~~

~~对于git restore --staged <file>命令，把文件从暂存区撤回到工作区，保留文件最后一次修改的内容；~~

撤销修改：

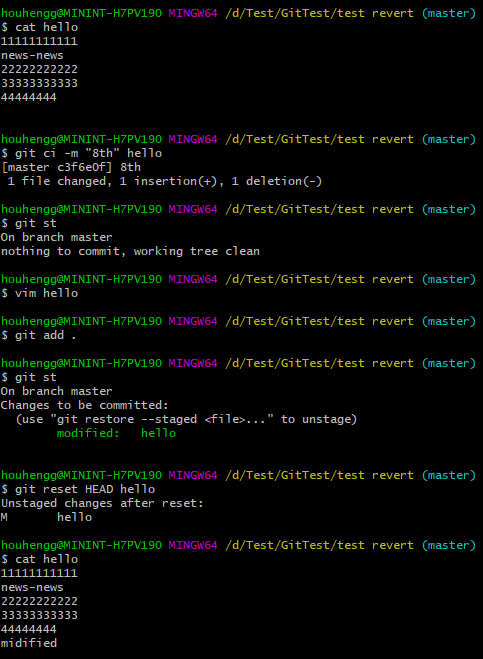
1）在没有git add之前

$ git checkout -- file 撤销指定文件的修改

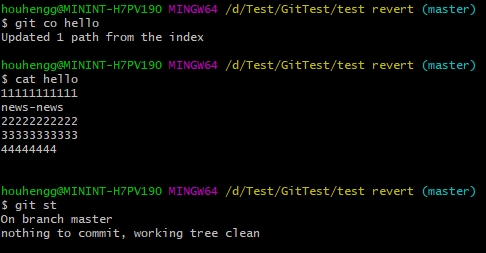


2） git add了，但没有git commit

$ git reset HEAD file 把add到暂存区的文件回退到工作区（**上面已经测试过**）



$ git checkout file 更新



3） git add了，也git commit了

这种情况只能回溯历史版本了，下面介绍版本回溯

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

回溯：

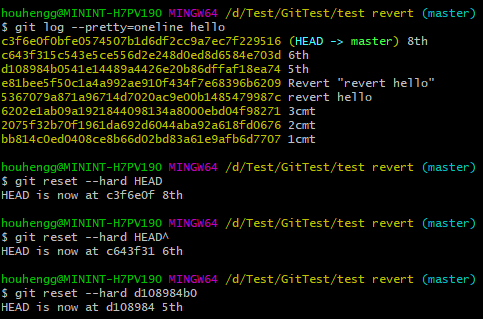
git log 查看提交日志，commit后面对应的是版本ID

git reflog 最前面黄色的7个符号就是版本ID的前七位

git reset --hard ID或ID的前七位 回溯到具体某个版本

git reset --hard HEAD^ 回溯上个版本

git reset --hard HEAD^^ 回溯上上个版本



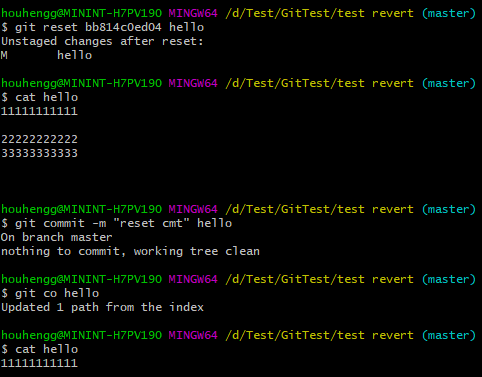
回溯到某一文件的历史版本：必须按照操作全部执行，如果在reset后查看file，此时还没有回到历史版本，checkout后才回溯到历史版本。

$ git log file 查看指定文件的提交日志

$ git reset 版本ID file 把文件回溯到指定版本，放在暂存区

$ git commit -m “提交说明“ file 把暂存区指定文件写入本地库

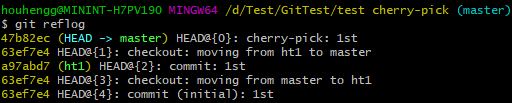
$ git checkout file 更新

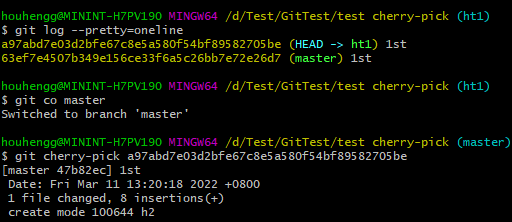


4. git cherry-pick

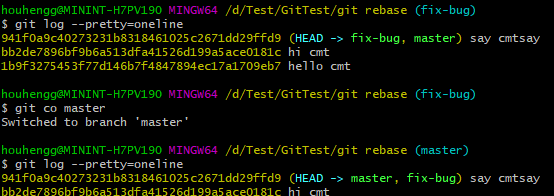
该命令可以选择其他分支中的一个或几个commit(s)来进行操作（操作对象是其他分支的commit）。

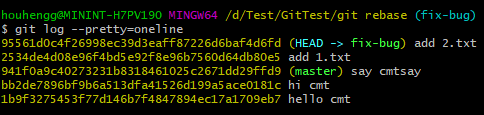
例如，假设我们有个稳定版本的分支，叫v2.0，另外还有个开发版本的分支v3.0，我们不能直接把两个分支合并，这样会导致稳定版本混乱，但是又想增加一个v3.0中的功能到v2.0中，这里就可以使用cherry-pick了。

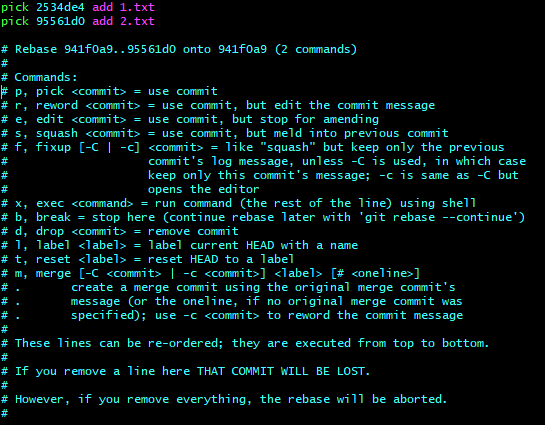
常用命令：git cherry-pick <commit id>

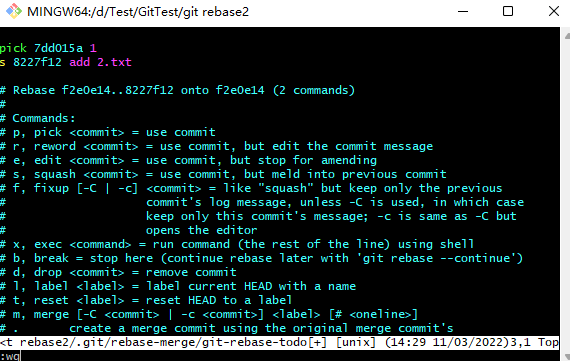


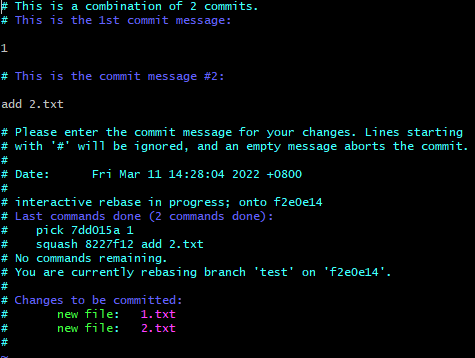
5. git rebase

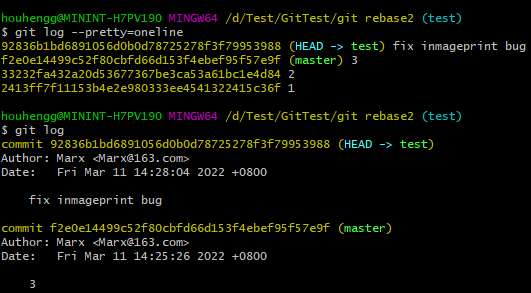












rebase需要基于一个分支来设置你当前的分支的基线，这基线就是当前分支的开始时间轴向后移动到最新的跟踪分支的最后面，这样你的当前分支就是最新的跟踪分支。这里的操作是基于文件事务处理的，所以你不用怕中间失败会影响文件的一致性。在中间的过程中你可以随时取消rebase 事务。