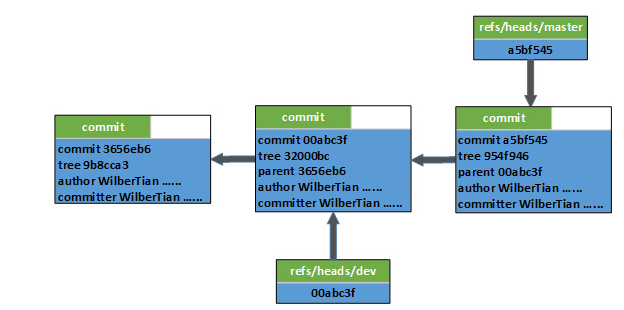
Git的merge和rebase

# merge

下面我们基于本地一个仓库开始介绍，当前仓库的分支情况如下：

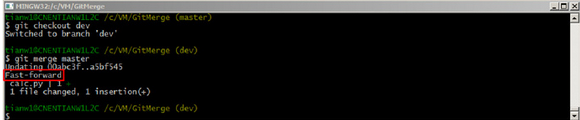


其实，merge命令总结下来会有三种情况发生：

merge命令不生效：当目标分支是当前分支的祖先commit节点，也就是说当前分支已经是最新的了，在这种情况下merge命令没有任何效果。在当前仓库中，当我们把dev分支merge到master的时候，会得到”Already up-to-date.”



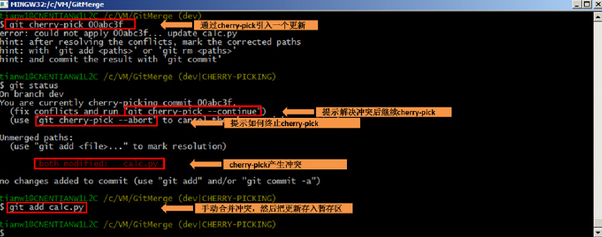
Fast-forward合并模式：当前分支是目标分支的祖先commit节点时，会发生Fast-forward的merge，看下图



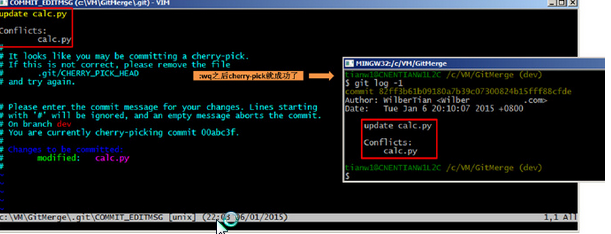
三方合并：Git分支(branch)”中分支合并的内容。如果没有冲突，Git会帮我们完成分支的合并，如果有冲突，就需要我们手动解决冲突了。

# cherry-pick

我们可能会经常碰到这种情况，在分支A上提交了一个更新，但是后来发现我们同样需要在分支B上应用这个更新。那么这时cherry-pick就可以帮助你解决问题。我们要做的就是通过”git reflog”找到A上那个更新的SHA1哈希值，然后切换到B分支上使用”git cherry-pick”：我们要做的就是通过”git reflog”找到A上那个更新的SHA1哈希值，然后切换到B分支上使用”git cherry-pick”。



当我们手动合并过冲突，然后继续执行”cherry-pick”时候，Git会给出友好的交互界面，如果我们不需要更新这个提交的message，我们可以直接”:wq”进行保存退出。到此，这个”cherry-pick”操作就成功了。

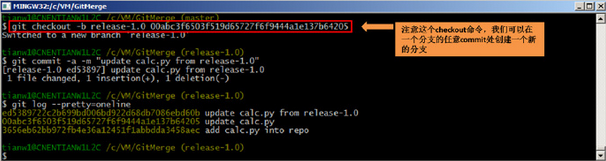


# rebase

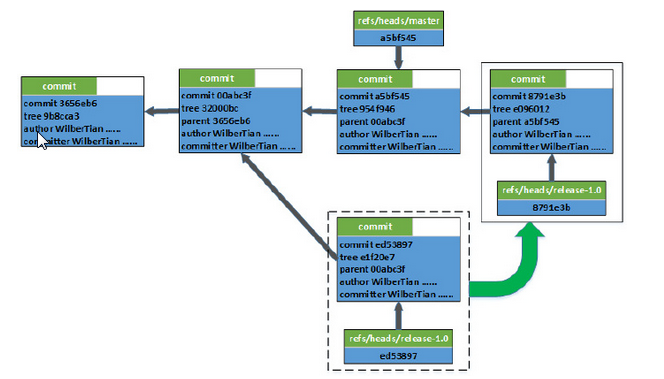
merge的过程中，比较好的就是我们可以看到分支的历史信息，但是，如果分支很多的时候，这个分支历史可能就会变得很复杂了。如果我们使用rebase，那么提交的历史会保持线性。

rebase的原理：先将当前分支的HEAD引用指向目标分支和当前分支的共同祖先commit节点，然后将当前分支上的commit一个个apply到目标分支上，apply完以后再将HEAD引用指向当前分支。下面我们看个实例。

下面就开始rebase的介绍，我们会基于master新建一个release-1.0的分支，并在该分支上提交一个更新。



这时，我们在release-1.0分支上执行”git rebase master”，就会得到下面的对象关系图。

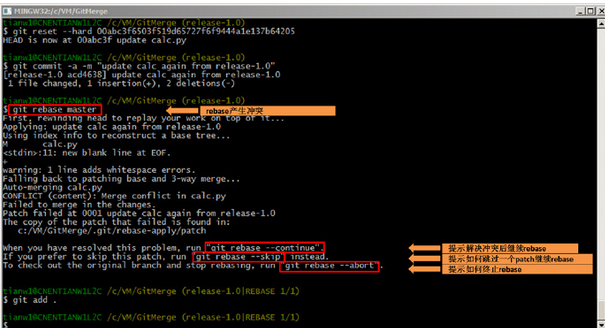


根据rebase的工作原理进行分析：

1. 把当前分支的HEAD引用指向”00abc3f”
2. 然后将当前分支上的commit一个个apply到目标分支，这里就是把”ed53897″更新apply到master上；注意，如果没有冲突，这里将直接生成一个新的更新
3. 最后更新当前分支的HEAD引用执行最新的提交”8791e3b”

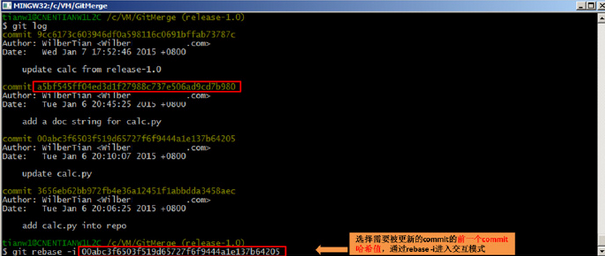
这个就是rebase操作，可以看到通过rebase操作的话，我们的commit历史会一直保持线性。在这种情况下，当我们切换到master分支，然后进行”git merge release-1.0″分支合并时，master将会直接是Fast-forward合并模式，commit历史也会是线性的。

当然rebase操作也会产生冲突，当一个冲突发生的时候，我们可以skip过当前的patch（一定要当心，不要随便使用，以免丢失更新）；也可以手动的解决冲突，然后继续rebase操作

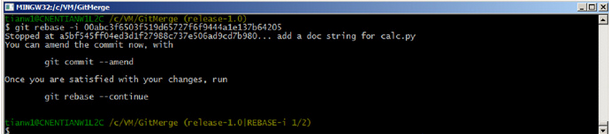


# rebase交互模式

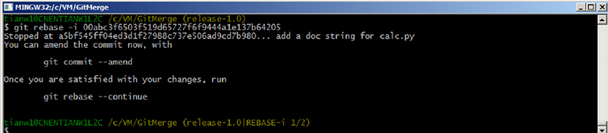
其实，rebase还有别的很强大功能，比如rebase交互模式，通过交互模式我们可以改变commit的信息，删除commit，合并commit以及更改commit的顺序等等。假如我们现在想要更新一个commit的信息，我们就可以使用rebase交互模式，找到commit的哈希值，然后进入交互模式。



根据rebase的操作提示，我们选择edit操作，然后保存退出。



这时，Git将会提示我们，是进行更改，还是可以继续操作。这里我们通过”git commit –amend”进入编辑模式。



在编辑模式中对commit进行更新，然后保存退出，继续rebase操作

