

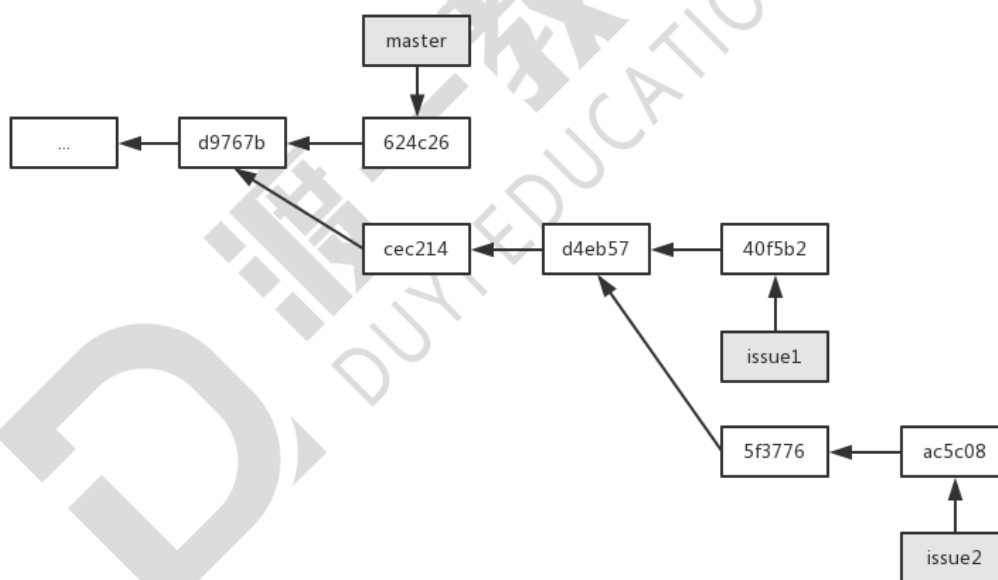
深入理解 git rebase

变基 (rebase)

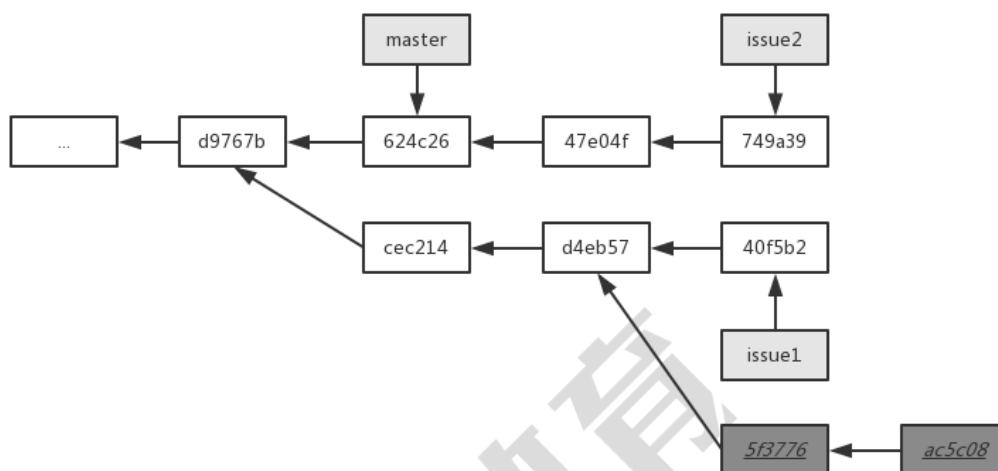
rebase, 有垫底, 基底的意思, Git rebase 我们称它为 Git 变基, 即在当前分支外另一点上重新应用当前分支的提交历史, 变基是 Git 整合变更的一种方式, 另一种方式是什么呢 (当然是合并了)? 变基的基础使用已经在上一篇中提到, 此处就不重复了, 下面会阐述更深入或者说对于初学者更不常用的 Git 变基知识点。

多主题分支变基 (git rebase -onto)

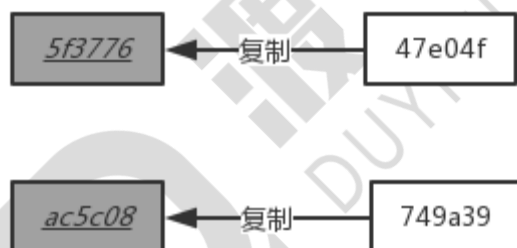
无论曾经, 还是未来, 只要你一直使用 Git, 早晚会遇到这样一种情况: 从主干切出了某一分支 issue1, 进行了一些提交后, 有另一个需求, 我们在 issue1 分支切出新分支 issue2, 进行了提交, 这期间其他成员对 master 主干分支进行了更新, 结构图如下:



现在, issue2 分支开发完, 我们需要将其变更并入主线, 但是 issue1 分支的变更还是继续保持独立, 如果直接进行简单变基, cec214, d4eb57 也将被并入主线, 这不是我们想要的结果, 我们只希望将 issue2 分支上进行的变更和提交并入主线, 即 5f3776, ac5c08, 这时需要使用 git rebase -onto 指令



可以看出,执行--onto 变基后,在主线上复制了 issue2 分支的所有提交对象,此处所谓复制,是在主线创建新提交对象,而其提交内容及备注复制自 issue2 分支的提交对象,对于如下:



GIT REBASE --ONTO

上例使用了 git rebase --onto 变基目标分支 master 变基过渡分支 issue1 变基当前分支 issue2 指令,此指令解释如下:

- 变基当前分支,即当前执行变基的分支 issue2;
- 变基目标分支,即当前执行变基分支的变基指向分支 master;
- 变基过渡分支,即当前执行变基分支过滤提交对象的目标分支 issue1 (当前分支从过渡分支切出);
- 取出 issue2 分支上进行的所有提交对象,以补丁 (patches) 形式在主线 master 分支上重新应用,创建新提交

对象,然后将分支指针移动到最后一个新创建的提交对象。

需要注意的是此时 master 分支指针并没有更新到新提交对象,我们需要手动进行合并
此时 master 分支和 issue2 分支指针均指向主线最新提交对象 749a39,执行合并指令时输出的第一行信息指明当前主线分支从 624c26 推进到 749a39,第二行 Fast-forward 说明是快速推进合并方式。

变基与冲突

接下来,如果 issue1 分支开发测试完成,可以继续对该分支进行变基,合并:

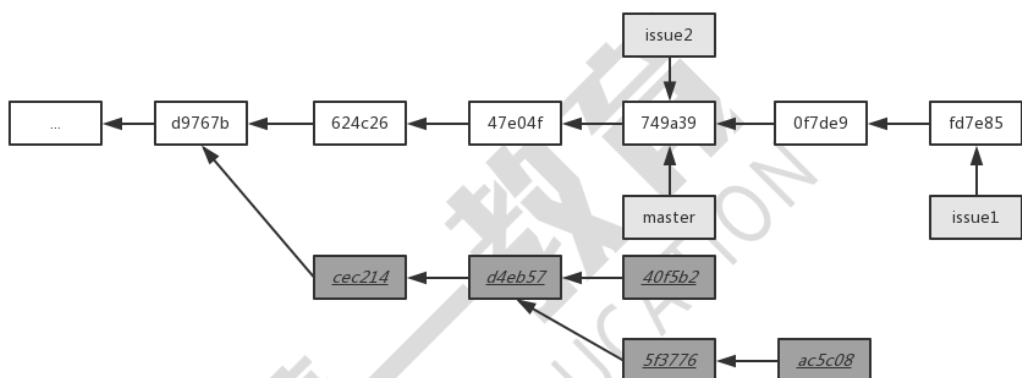
注:注意此处红线标注处 No changes -, 为什么会说没有变更呢?因为在处理变基冲突时,

我把此次提交的变更删除了，该文件没有变化，后文你会发现这样处理的意图。

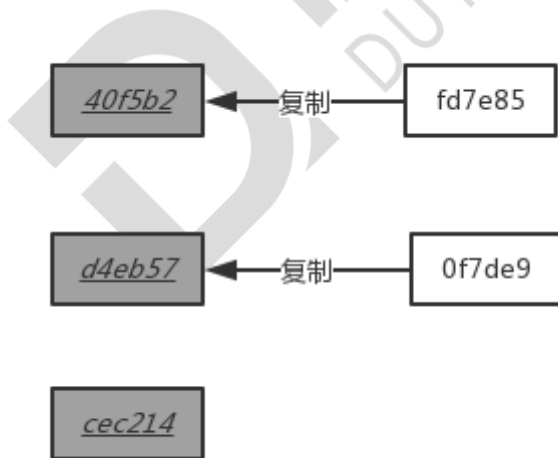
需要注意的有两点：

- 在执行 `git rebase --continue` 指令时，需要使用 `git add` 指令标记冲突已解决（和 SVN 一样有这个过程）；
- 继续变基完成时，我们看到输出了两行 **Applying:** 提交对象备注信息，这里两行是重点。

目前，分支结构如下：



issue1 分支并入主线，其提交历史也全部并入主线，提交对象复制关系如下：



我们发现，在 issue1 分支上有三次提交，然而并入主线只创建（复制）了两个提交对象，这是因为我们在处理 cec214 提交对象的变基冲突时，把此提交对象的变更删除了，即未发现变更，Git 未将此提交对象并入主线。

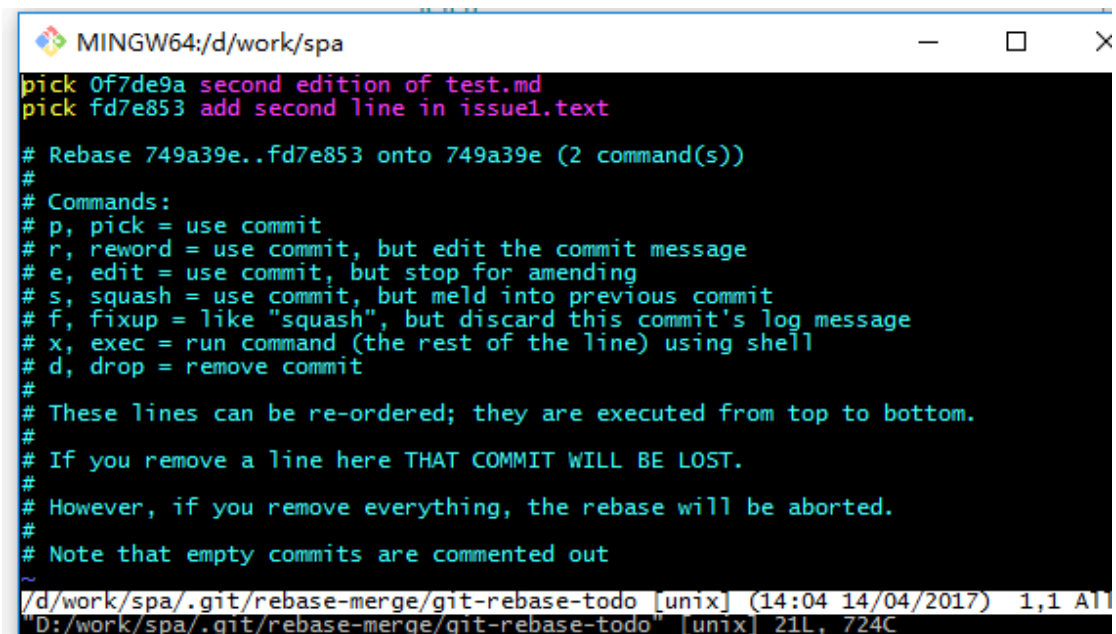
接下来，可以在 master 分支合并 issue1 分支。

交互式变基（`rebase -interactive`）

这一节要介绍的是交互式变基，指令为 `git rebase -i` 或 `git rebase --interactive`，使用该指令可以修改提交历史，其后参数可以是某一特定提交对象 ID 或执行特定提交对象的指针，将输出该提交对象之后的所有提交对象（不包括该提交对象），如 `HEAD~` 表明输出当前分支最新一次提交对象，`HEAD~~` 表明输出当前分支的最新的两次提交对象。

HEAD~~

如下，在 issue1 分支执行指令 `git rebase -i HEAD~~`，会进入编辑模式：



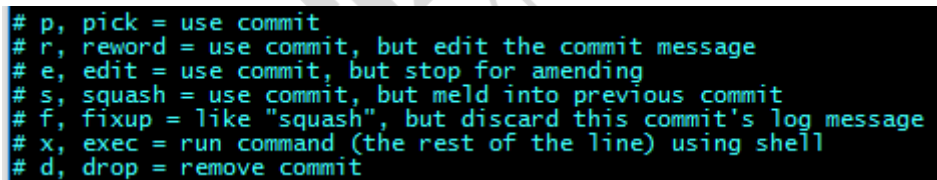
```

MINGW64:/d/work/spa
pick 0f7de9a second edition of test.md
pick fd7e853 add second line in issue1.text

# Rebase 749a39e..fd7e853 onto 749a39e (2 command(s))
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
~
/d/work/spa/.git/rebase-merge/git-rebase-todo [unix] (14:04 14/04/2017) 1,1 All
"D:/work/spa/.git/rebase-merge/git-rebase-todo" [unix] 21L, 724C

```

如图列出了 HEAD~~指向的提交对象之后的所有提交对象，每一行之前默认是 pick 标志，表示该提交对象正在使用，我们可以修改该标记值，随后 Git 根据标记值执行不同操作，标记值与操作对应关系如下：



```

# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit

```

- pick(p)，表明正在使用；
- reword(r)，表明仍然使用该提交对象，但是需修改提交信息；
- edit(e)，使用该提交对象，但是不合并提交对象；
- squash(s)，使用该提交对象，但是将此提交与上一次提交对象合并；
- fixup(f)，同 squash 值，但是丢失此次提交的日志信息；
- exec(x)，后接特定脚本，保存后将执行该脚本；
- drop(d)，移除该提交对象

变基的影响

总结下来，Git 变基的作用也是整合变更，首先在待合并分支执行变基，最后还是归于分支合并，但是在这个过程与直接合并分支还是有差别，正如本文的例子，可以看出变基会保留分支的提交历史，但是是通过将其并入主线保存的，之后关于该分支开发的具体历史及关系，已经被遮盖了，即历史已被修整，而我们通过直接合并分支方式整合变更时，分支的提交记录依然可以以分支的形式独立存在，历史未被修改。

选择变基还是合并

变基会修整历史，然后将分支历史并入主线，可以理解成美化过的历史，而合并则可以不修改历史，让分支历史依然独立存在，可以看作原始的历史。

所以选择变基还是合并，看具体需求，你只是想要一个清晰，明了的历史，并不关系历史的具体来源，你可以首选变基，但是如果你想比较清楚地了解项目不同阶段的原始历史，你可

以选择直接合并。

一个原则

说到最后，还有一个不得不提的原则：

永远不要对已经推到主干分支服务器或者团队其他成员的提交进行变基，我们选择变基还是合并的范围应该在自己当前工作范围内。

