
This book is available in English.

Full translation available in

български език,
Español,
Français,
日本語,
한국어,
Nederlands,
Русский,
Українська
简体中文,

Partial translations available in

Čeština,
Indonesian,
Polski,
Српски,
Tagalog,
繁體中文,

Translations started for

Беларуская,
Deutsch,
فارسی,
Ελληνικά,
Italiano,
Bahasa Melayu,
Polski,
Português (Brasil),
Türkçe,
Ўзбекча.

The source of this book is hosted on GitHub.
Patches, suggestions and comments are welcome.

Chapters ▾

1st Edition

.2 Git 分支 - 分支的新建与合并

分支的新建与合并

现在让我们来看一个简单的分支与合并的例子，实际工作中大体也会用到这样的工作流程：

1. 开发某个网站。
2. 为实现某个新的需求，创建一个分支。
3. 在这个分支上开展工作。

假设此时，你突然接到一个电话说有个很严重的问题需要紧急修补，那么可以按照下面的方式处理：

1. 返回到原先已经发布到生产服务器上的分支。

2. 为这次紧急修补建立一个新分支，并在其中修复问题。
3. 通过测试后，回到生产服务器所在的分支，将修补分支合并进来，然后再推送到生产服务器上。
4. 切换到之前实现新需求的分支，继续工作。

分支的新建与切换

首先，我们假设你正在项目中愉快地工作，并且已经提交了几次更新（见图 3-10）。

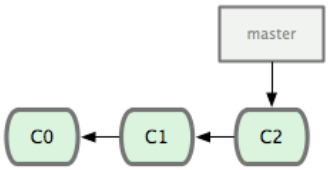

图 3-10. 一个简短的提交历史

现在，你决定要修补问题追踪系统上的 #53 问题。顺带说明下，Git 并不同任何特定的问题追踪系统打交道。这里为了说明要解决的问题，才把新建的分支取名为 iss53。要新建并切换到该分支，运行 `git checkout` 并加上 `-b` 参数：

```
$ git checkout -b iss53
Switched to a new branch 'iss53'
```

这相当于执行下面这两条命令：

```
$ git branch iss53
$ git checkout iss53
```

图 3-11 示意该命令的执行结果。

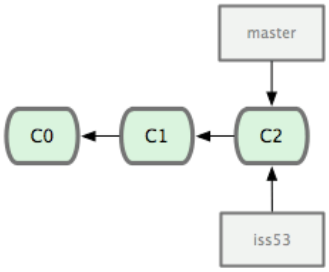

图 3-11. 创建了一个新分支的指针

接着你开始尝试修复问题，在提交了若干次更新后，`iss53` 分支的指针也会随着向前推进，因为它就是当前分支（换句话说，当前的 `HEAD` 指针正指向 `iss53`，见图 3-12）：

```
$ vim index.html
$ git commit -a -m 'added a new footer [issue 53]'
```

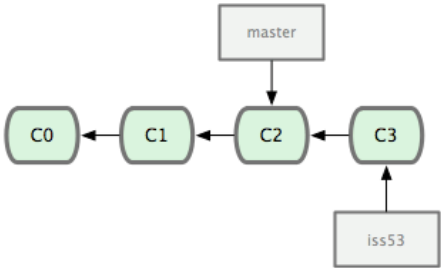

图 3-12. iss53 分支随工作进展向前推进

现在你就接到了那个网站问题的紧急电话，需要马上修补。有了 Git，我们就不需要同时发布这个补丁和 `iss53` 里作出的修改，也不需要

在创建和发布该补丁到服务器之前花费大力气来复原这些修改。唯一需要的仅仅是切换回 `master` 分支。

不过在此之前，留心你的暂存区或者工作目录里，那些还没有提交的修改，它会和你即将检出的分支产生冲突从而阻止 Git 为你切换分支。切换分支的时候最好保持一个清洁的工作区域。稍后会介绍几个绕过这种问题的办法（分别叫做 stashing 和 commit amending）。目前已经提交了所有的修改，所以接下来可以正常转换到 `master` 分支：

```
$ git checkout master
Switched to branch 'master'
```

此时工作目录中的内容和你在解决问题 #53 之前一模一样，你可以集中精力进行紧急修补。这一点值得牢记：Git 会把工作目录的内容恢复为检出某分支时它所指向的那个提交对象的快照。它会自动添加、删除和修改文件以确保目录的内容和你当时提交时完全一样。

接下来，你得进行紧急修补。我们创建一个紧急修补分支 `hotfix` 来开展工作，直到搞定（见图 3-13）：

```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
$ vim index.html
$ git commit -a -m 'fixed the broken email address'
[hotfix 3a0874c] fixed the broken email address
 1 files changed, 1 deletion(-)
```

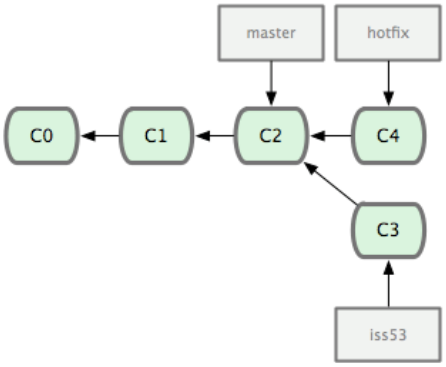

图 3-13. hotfix 分支是从 master 分支所在点分化出来的

有必要作些测试，确保修补是成功的，然后回到 `master` 分支并把它合并进来，然后发布到生产服务器。用 `git merge` 命令来进行合并：

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 README | 1 -
 1 file changed, 1 deletion(-)
```

请注意，合并时出现了“Fast forward”的提示。由于当前 `master` 分支所在的提交对象是要并入的 `hotfix` 分支的直接上游，Git 只需把 `master` 分支指针直接右移。换句话说，如果顺着一个分支走下去可以到达另一个分支的话，那么 Git 在合并两者时，只会简单地把指针右移，因为这种单线的历史分支不存在任何需要解决的分歧，所以这种合并过程可以称为快进（Fast forward）。

现在最新的修改已经在当前 `master` 分支所指向的提交对象中了，可以部署到生产服务器上去了（见图 3-14）。

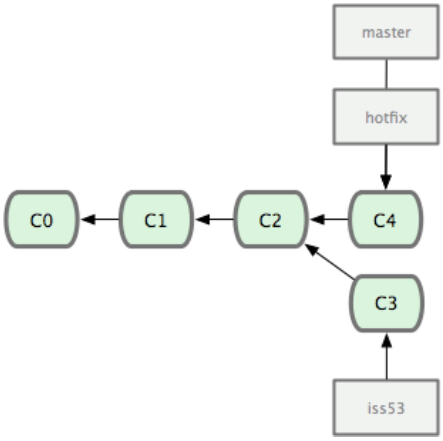

图 3-14. 合并之后，master 分支和 hotfix 分支指向同一位置。

在那个超级重要的修补发布以后，你想要回到被打扰之前的工作。由于当前 `hotfix` 分支和 `master` 都指向相同的提交对象，所以 `hotfix` 已经完成了历史使命，可以删掉了。使用 `git branch` 的 `-d` 选项执行删除操作：

```
$ git branch -d hotfix
Deleted branch hotfix (was 3a0874c).
```

现在回到之前未完成的 #53 问题修复分支上继续工作（图 3-15）：

```
$ git checkout iss53
Switched to branch 'iss53'
$ vim index.html
$ git commit -a -m 'finished the new footer [issue 53]'
[iss53 ad82d7a] finished the new footer [issue 53]
 1 file changed, 1 insertion(+)
```

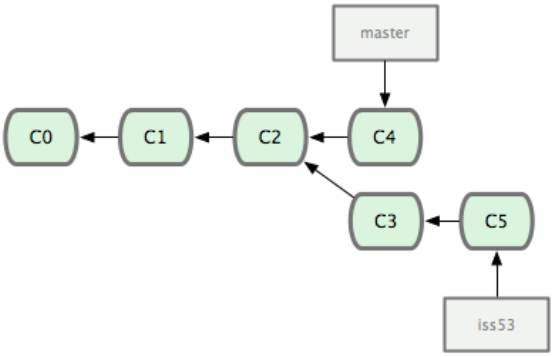

图 3-15. iss53 分支可以不受影响继续推进。

值得注意的是之前 `hotfix` 分支的修改内容尚未包含到 `iss53` 中来。如果需要纳入此次修补，可以用 `git merge master` 把 master 分支合并到 `iss53`；或者等 `iss53` 完成之后，再将 `iss53` 分支中的更新并入 `master`。

分支的合并

在问题 #53 相关的工作完成之后，可以合并回 `master` 分支。实际操作同前面合并 `hotfix` 分支差不多，只需回到 `master` 分支，运行 `git merge` 命令指定要合并进来的分支：

```
$ git checkout master
$ git merge iss53
Auto-merging README
Merge made by the 'recursive' strategy.
 README | 1 +
 1 file changed, 1 insertion(+)
```

请注意，这次合并操作的底层实现，并不同于之前 `hotfix` 的并入方式。因为这次你的开发历史是从更早的地方开始分叉的。由于当前

`master` 分支所指向的提交对象(C4)并不是 `iss53` 分支的直接祖先，Git 不得不进行一些额外处理。就此例而言，Git 会用两个分支的末端(C4 和 C5)以及它们的共同祖先(C2)进行一次简单的三方合并计算。图 3-16 用红框标出了 Git 用于合并的三个提交对象：

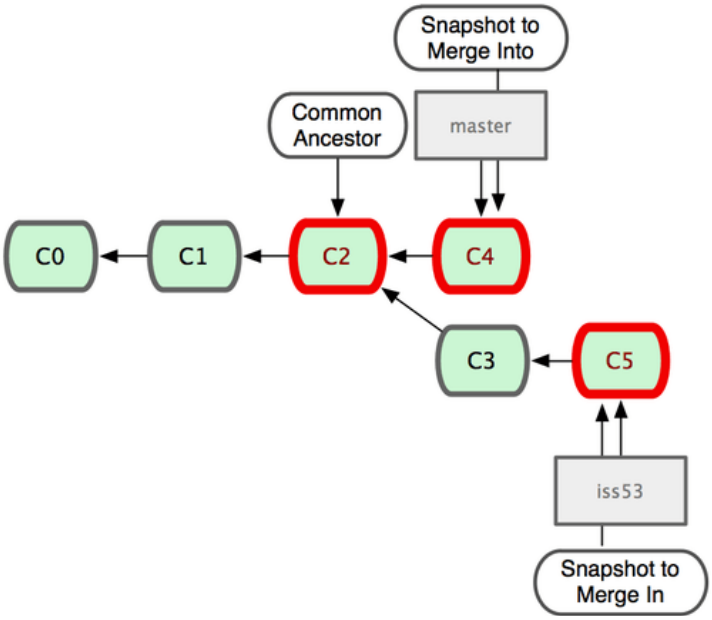

图 3-16. Git 为分支合并自动识别出最佳的同源合并点。

这次，Git 没有简单地把分支指针右移，而是对三方合并后的结果重新做一个新的快照，并自动创建一个指向它的提交对象(C6)(见图 3-17)。这个提交对象比较特殊，它有两个祖先(C4 和 C5)。

值得一提的是 Git 可以自己裁决哪个共同祖先才是最佳合并基础；这和 CVS 或 Subversion(1.5 以后的版本)不同，它们需要开发者手工指定合并基础。所以此特性让 Git 的合并操作比其他系统都要简单不少。

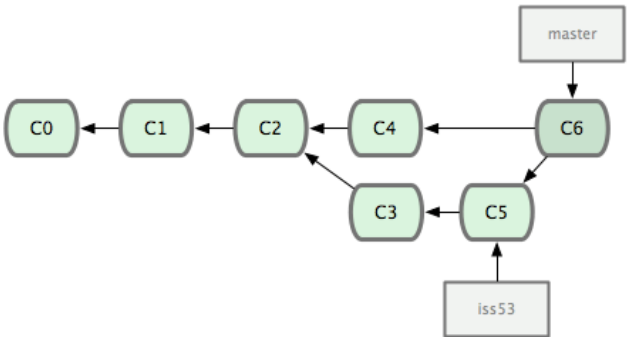

图 3-17. Git 自动创建了一个包含了合并结果的提交对象。

既然之前的工作成果已经合并到 `master` 了，那么 `iss53` 也就没用了。你可以就此删除它，并在问题追踪系统里关闭该问题。

```
$ git branch -d iss53
```

遇到冲突**时**的分支合并

有时候合并操作并不会如此顺利。如果在不同的分支中都修改了同一个文件的同一部分，Git 就无法干净地把两者合到一起(译注：逻辑上说，这种问题只能由人来裁决。)。如果你在解决问题 #53 的过程中修改了 `hotfix` 中修改的部分，将得到类似下面的结果：

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Git 作了合并，但没有提交，它会停下来等你解决冲突。要看看哪些文件在合并时发生冲突，可以用 `git status` 查阅：

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

任何包含未解决冲突的文件都会以未合并（unmerged）的状态列出。Git 会在有冲突的文件里加入标准的冲突解决标记，可以通过它们来手工定位并解决这些冲突。可以看到此文件包含类似下面这样的部分：

```
<<<<<<< HEAD
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> iss53
```

可以看到 `=======` 隔开的上半部分，是 `HEAD`（即 `master` 分支，在运行 `merge` 命令时所切换到的分支）中的内容，下半部分是在 `iss53` 分支中的内容。解决冲突的办法无非是二者选其一或者由你亲自整合到一起。比如你可以通过把这段内容替换为下面这样来解决：

```
<div id="footer">
please contact us at email.support@github.com
</div>
```

这个解决方案各采纳了两个分支中的一部分内容，而且我还删除了 `<<<<<<<`，`=======` 和 `>>>>>>>` 这些行。在解决了所有文件里的所有冲突后，运行 `git add` 将把它们标记为已解决状态（译注：实际上就是来一次快照保存到暂存区域。）。因为一旦暂存，就表示冲突已经解决。如果你想用一个有图形界面的工具来解决这些问题，不妨运行 `git mergetool`，它会调用一个可视化的合并工具并引导你解决所有冲突：

```
$ git mergetool

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecmerge p4merge araxis bc3 codecompare vimdiff emerge
Merging:
index.html

Normal merge conflict for 'index.html':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (opendiff):
```

如果不想用默认的合并工具（Git 为我默认选择了 `opendiff`，因为我在 Mac 上运行了该命令），你可以在上方"merge tool candidates"里找到可用的合并工具列表，输入你想用的工具名。我们将在第七章讨论怎样改变环境中的默认值。

退出合并工具以后，Git 会询问你合并是否成功。如果回答是，它会为你把相关文件暂存起来，以表明状态为已解决。

再运行一次 `git status` 来确认所有冲突都已解决：

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.html
```

如果觉得满意了，并且确认所有冲突都已解决，也就是进入了暂存区，就可以用 `git commit` 来完成这次合并提交。提交的记录差不多是这样：

```
Merge branch 'iss53'

Conflicts:
  index.html
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
```

```
#       .git/MERGE_HEAD
# and try again.
#
```

如果想给将来看这次合并的人一些方便，可以修改该信息，提供更多合并细节。比如你都作了哪些改动，以及这么做的原因。有时候裁决冲突的理由并不直接或明显，有必要略加注解。

prev | next
About this site
Patches, suggestions, and comments are welcome.
Git is a member of Software Freedom Conservancy