**Git分支 - 合并冲突（典型合并）**

**1、典型合并**

继续上面的前一篇文章的示例：

我需要继续开发dev分支上的功能。

需求：

1. 继续修改dev.txt文件。
2. 修改readme.txt文件，因为该文件和master分支都有的文件，修改一下为之后创造一个小冲突。

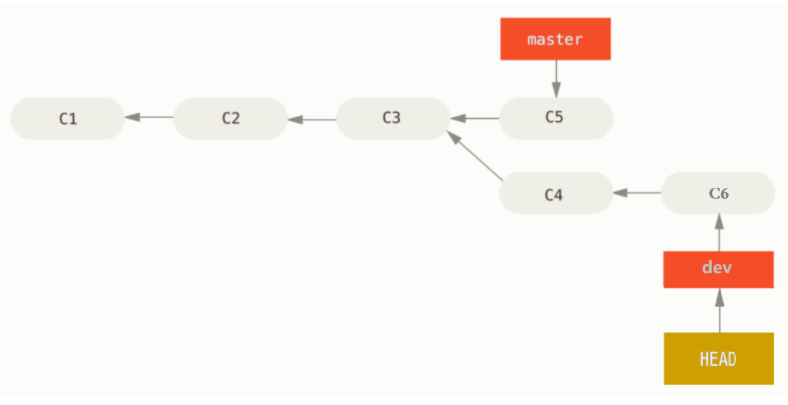
**1）在dev分支继续工作，并提交版本**

|  |
| --- |
| Bash # 1.查看工作目录中文件状态，非常干净 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ git status On branch dev nothing to commit, working tree clean  # 2.查看版本库提交历史 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ git log --oneline --graph --all \* 0690030 (master) 第5次提交，新增内容：branch test v4 | \* 9eb3224 (HEAD -> dev) 第4次提交，dev分支开发 dev.txt文件 |/ \* b97ccfd 第3次提交，新增内容：branch test v3 \* f72a9fe 第2次提交，新增内容：branch test v2 \* fa2439a 第1次提交，新增readme.txt文件  # 3.开始修改dev.txt文件和readme.txt文件 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ echo "dev new file v2 100%" >> dev.txt  L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ echo "branch test v4 dev 100%" >> readme.txt  # 4.提交到版本库中 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ git commit -a -m '第6次提交，修改dev和readme文件' warning: LF will be replaced by CRLF in dev.txt. The file will have its original line endings in your working directory warning: LF will be replaced by CRLF in readme.txt. The file will have its original line endings in your working directory [dev e94af28] 第6次提交，修改dev和readme文件  2 files changed, 2 insertions(+) |

**2）查看当前版本库的历史提交记录**

|  |
| --- |
| Bash # 1.查看历史提交记录 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ git log --oneline --graph --all \* e94af28 (HEAD -> dev) 第6次提交，修改dev和readme文件 \* 9eb3224 第4次提交，dev分支开发 dev.txt文件 | \* 0690030 (master) 第5次提交，新增内容：branch test v4 |/ \* b97ccfd 第3次提交，新增内容：branch test v3 \* f72a9fe 第2次提交，新增内容：branch test v2 \* fa2439a 第1次提交，新增readme.txt文件 |

此时工作目录的图解如下：



**3）把dev分支合并到master分支上**

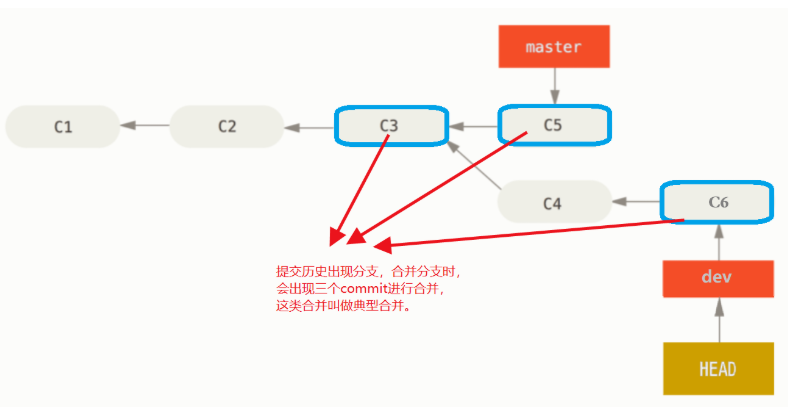
假设你已经开发完成新需求，并且打算将你的工作合并入 master 分支。

为此，你需要合并 dev 分支到 master 分支，这和之前你合并 hotfix 分支的时候，看起来有一点不一样。你会发现的提交历史从一个更早的地方开始分叉（diverged：发散）。

而master 分支所在提交节点，并不是 dev 分支所在提交点的直接祖先，所以 Git 不得不做一些额外的工作。

出现这种情况的时候，Git 会使用两个分支的末端提交的快照（C5 和 C6）以及这两个分支的公共祖先（C3），做一个简单的三方合并。

如下图：

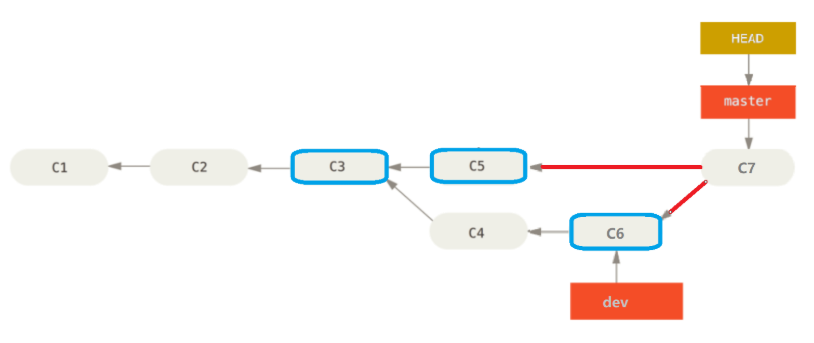


你只需要切换到你想合并入的分支上，然后运行 git merge 命令：

|  |
| --- |
| Bash # 1.查看工作目录中文件状态，非常干净 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ git status On branch dev nothing to commit, working tree clean  # 2.切换到master分支 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ git checkout master Switched to branch 'master' |

但这里要注意，和之前合并 hotfix 分支的不同的是，指针向前推进所不同。Git 将此次三方合并的结果，做了一个新的快照，并且自动创建一个新的提交指向它。 这被称作一次合并提交（典型提交），它的特别之处在于他有不止一个父提交。

如下图所示：



既然你的修改已经合并进来了，就不再需要 dev 分支了, 现在你可以删除这个分支。

|  |
| --- |
| Bash $ git branch -d dev |

**2、遇到冲突时的分支合并**

有时候合并操作不会如此顺利。 如果你在两个不同的分支中，对同一个文件的同一个部分进行了不同的修改，Git 就没法干净的合并它们。

如果你在dev分支的开发时，也进行了和 hotfix 分支有关的修改，都涉及到同一个文件的同一处，在合并它们的时候就会产生合并冲突：

接上面练习：

**1）合并dev分支到master分支上**

|  |
| --- |
| Bash # 直接合并分支dev到master分支上 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git merge dev Auto-merging readme.txt CONFLICT (content): Merge conflict in readme.txt Automatic merge failed; fix conflicts and then commit the result. |

说明：

* Auto-merging readme.txt：在自动合并readme.txt文件的过程中，
* CONFLICT (content): Merge conflict in readme.txt：冲突（内容），在readme.txt文件中发生冲突。
* Automatic merge failed; fix conflicts and then commit the result.：自动合并失败，你需要手动解决冲突，然后提交结果。

此时 Git 做了合并，但是没有自动地创建一个新的合并提交。 出现冲突时，Git 会暂停下来，等待你去解决合并产生的冲突。

你可以在发生合并冲突后的任意时刻，使用 git status 命令来查看，那些因包含合并冲突，而处于未合并（**unmerged**）状态的文件。

|  |
| --- |
| Bash L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master|MERGING) $ git status On branch master You have unmerged paths.  (fix conflicts and run "git commit")  (use "git merge --abort" to abort the merge)  Changes to be committed:  new file: dev.txt  Unmerged paths:  (use "git add <file>..." to mark resolution)  both modified: readme.txt |

**2）解决readme.txt文件中的冲突内容**

在合并分支时，产生冲突的文件都会被标识成**未合并状态（Unmerged）**，你需要打开这些文件，手动解决冲突。

出现冲突的文件会包含一些特殊区段，

打开冲突文件readme.txt如下：

|  |
| --- |
| Bash # 编辑readme.txt文件 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master|MERGING) $ vim readme.txt  # 内容如下： branch test v1 branch test v2 branch test v3 <<<<<<< HEAD branch test v4 ======= branch test v4 dev 100% >>>>>>> dev |

说明：

* HEAD 到 ======= ：表示所在分支的内容，也就是master分支的内容。
* ======= 到 dev ：表示被合并的分支，也就是dev分支的内容

解决冲突方式：

* 你选择使用由 ======= 分割的两部分中任何的一个。
* 或者你也可以自行合并这些内容。
* 并且 <<<<<<< , ======= , 和 >>>>>>> 这些标识要删除。

本例修改冲突内容如下：

|  |
| --- |
| Bash branch test v1 branch test v2 branch test v3 branch test v4 for master branch test v4 for dev |

这样冲突文件就解决完成了。

|  |
| --- |
| 提示： 自己和自己的分支合并时，所产生冲突还好解决，关键是自己和同事的分支合并时产生冲突，就比较难解决（基本靠吼，哈哈）。 |

**3）把修改完的冲突文件加入暂存区并提交**

在你解决了所有文件里的冲突之后，对每个文件使用 git add 命令，将其添加到暂存区中。

一旦暂存这些原本有冲突的文件，Git 就会将它们标记为冲突已解决。

如果你对结果感到满意，并且确定之前有冲突的的文件都已经暂存了，这时你可以输入 git commit 来完成合并提交。

|  |
| --- |
| Bash # 1.查看工作目录中文件状态 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master|MERGING) $ git status On branch master You have unmerged paths.  (fix conflicts and run "git commit")  (use "git merge --abort" to abort the merge)  Changes to be committed:  new file: dev.txt  Unmerged paths:  (use "git add <file>..." to mark resolution)  both modified: readme.txt  # 2.添加到暂存区中 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master|MERGING) $ git add ./  # 3.再次查看工作目录中文件状态 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master|MERGING) $ git status On branch master All conflicts fixed but you are still merging.  (use "git commit" to conclude merge)  Changes to be committed:  new file: dev.txt  modified: readme.txt  # 4.提交 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master|MERGING) $ git commit -m '第7次提交，完成dev分支的新功能' [master 3110083] 第7次提交，完成dev分支的新功能 |

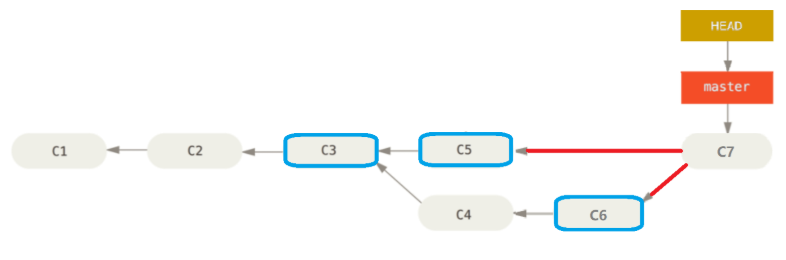
**4）查看版本库中的提交历史**

|  |
| --- |
| Bash # 查看提交历史 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git log --oneline --graph --all \* 3110083 (HEAD -> master) 第7次提交，完成dev分支的新功能 |\ | \* e94af28 (dev) 第6次提交，修改dev和readme文件 | \* 9eb3224 第4次提交，dev分支开发 dev.txt文件 \* | 0690030 第5次提交，新增内容：branch test v4 |/ \* b97ccfd 第3次提交，新增内容：branch test v3 \* f72a9fe 第2次提交，新增内容：branch test v2 \* fa2439a 第1次提交，新增readme.txt文件 |

此时dev分支已经完成了自己的使命，可以删除了，留着没有任何的意义。

|  |
| --- |
| Bash # 1.删除dev分支 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git branch -d dev Deleted branch dev (was e94af28).  # 2.查看版本库历史提交信息 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git log --oneline --graph --all \* 3110083 (HEAD -> master) 第7次提交，完成dev分支的新功能 |\ | \* e94af28 第6次提交，修改dev和readme文件 | \* 9eb3224 第4次提交，dev分支开发 dev.txt文件 \* | 0690030 第5次提交，新增内容：branch test v4 |/ \* b97ccfd 第3次提交，新增内容：branch test v3 \* f72a9fe 第2次提交，新增内容：branch test v2 \* fa2439a 第1次提交，新增readme.txt文件 # 可以看到信息中，没有dev分支的标识了。 |

此时工作目录的图解如下：



如上所示，就是解决一个冲突的全部流程。

**3、补充：Git与SVN文件冲突对比**

Git的冲突检测单位是文件，即当不同分支对同一个文件进行修改后进行合并，就会产生冲突。

这是与SVN是不同的。SVN检测冲突是以文件中的列为单位，什么意思？

* 若两个分支修改的不是同一个文件，合并时肯定不会产生冲突；
* 但若修改的是同一个文件上的不同行中的内容，合并时也不会产生冲突；
* 若修改的是同一个文件上同一行上的不同列，在合并的时候也不会产生冲突；
* 只有两个分支修改的是同一个文件同一行中同列数据的时候，在合并时才会产生冲突。

|  |
| --- |
| Git保存的不是文件的变化或者差异，而是一系列不同时刻的 **快照** 。 |

作者：繁华似锦Fighting 链接：https://www.jianshu.com/p/9af7f15b981e 来源：简书 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。