**Git分支 - 分支合并（快进合并）**

开发一个版本，采用的发布流程：

1. 从master分支的最新代码拉取一个开发分支，在上面进行开发（这里假设开发分支为dev）。
2. 在dev分支上不断地进行提交版本，期间master分支也会有因为其他版本上线，而不停有版本合并到master分支上。
3. dev分支要测试发布的时候，要把master分支的代码，先合并到dev分支上。
4. 最后发布完成后，再把dev分支的代码合并到master分支上。

**1、分支合并说明**

当dev分支上的工作完成后，需要将其合并到主分支master上。合并过程其实很快，只需要将master指针指向dev指针指向的节点，然后再将HEAD指针指向master指针指向的节点即可。即分支的合并就是修改了两个指针的指向而已。

|  |
| --- |
| 对于合并的较形象的理解是：合并就是将原来在dev分支上的节点，全部投射到master分支上。 |

合并分支命令：git merge。

对于分支的合并需要注意：如果需要将分支B合并到分支A上，首先要将当前分支切换到A分支上，然后再运行合并命令。

**2、综合示例**

让我们来看一个简单的分支新建与分支合并的例子，实际工作中你可能会用到类似的工作流程。

例如：

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

正在此时，你突然接到一个电话说有个很严重的问题需要紧急修补。

你将按照如下方式来处理：

1. 切换到主分支（production branch）或者开发分支上。
2. 为这个紧急任务新建一个分支，并在其中修复它。
3. 在测试通过之后，切换回线上分支，然后合并这个修补分支，最后将改动推送到线上分支。
4. 切换回你最初工作的分支上，继续工作。

**（1）正常工作情况说明**

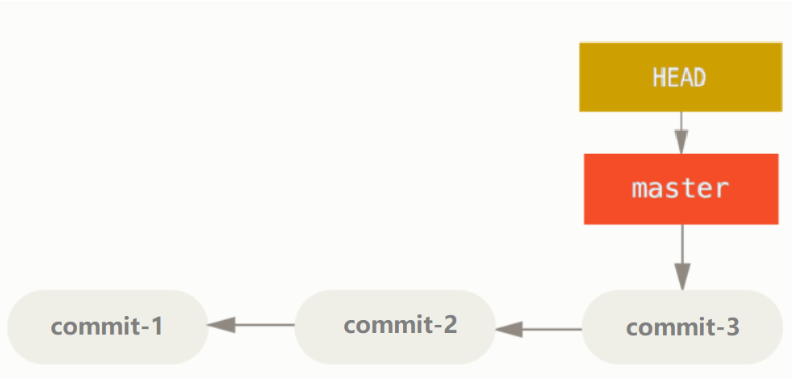
**1）主分支情况**

首先，你正在项目上工作，并且在 master 分支上已经有了一些提交。

如下：项目中已经有三次提交

|  |
| --- |
| Bash L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git log --oneline b97ccfd (HEAD -> master) 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 |

图解如下：

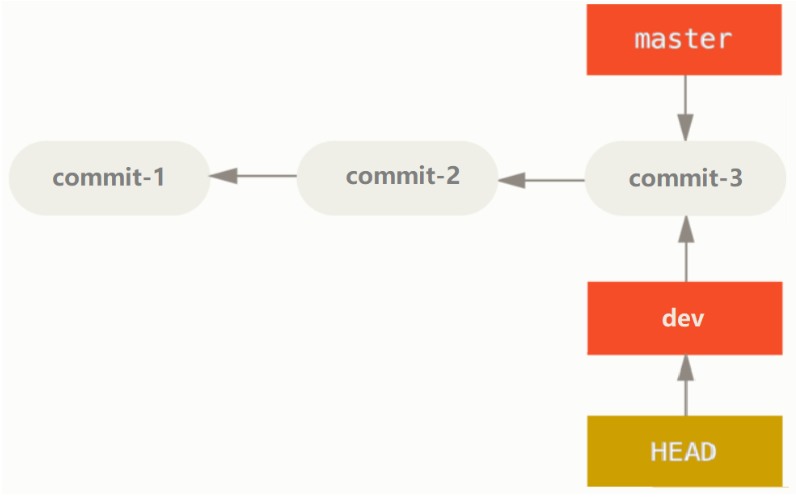


**2）我的开发分支情况**

为实现某个新的用户需求，创建一个分支，正在开发中，比如dev分支。

|  |
| --- |
| Bash # 1.创建dev分支 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git checkout -b dev Switched to a new branch 'dev'  # 2.进行开发工作 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ echo 'dev new file v1' > dev.txt |

图解如下：



因为你已经切换到该分支进行开发了，也就是说，你的 HEAD 指针指向了 dev 分支。

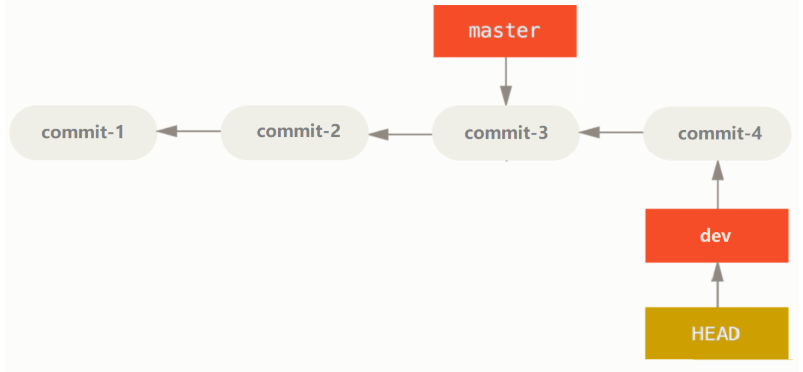
**（2）突然要处理新任务**

现在你接到那个电话，有个紧急问题等待你来解决。

**1）结束正在开发的任务，并提交到版本库中**

|  |
| --- |
| Bash # 1.把dev分支中开发的文件提交到暂存区中 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ git add ./ warning: LF will be replaced by CRLF in dev.txt. The file will have its original line endings in your working directory  # 2.然后提交到本地版本库中 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ git commit -m '第4次提交，dev分支开发 dev.txt文件' [dev 9eb3224] 第4次提交，dev分支开发 dev.txt文件  1 file changed, 1 insertion(+)  create mode 100644 dev.txt  # 3.查看历史提交记录 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ git log --oneline --graph \* 9eb3224 (HEAD -> dev) 第4次提交，dev分支开发 dev.txt文件 \* b97ccfd (master) 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 |

此时工作目录中图解如下：



注意：在你这么做之前进行切换分支之前， 最好的方法是保持好一个干净的状态工作目录。

否则可能出现：

* 切换分支失败。
* 或者分支污染。

**2）切换到master分支上**

|  |
| --- |
| Bash # 1.查看dev分支工作目录文件状态，干净的。 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分支上 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ git checkout master Switched to branch 'master' |

**3）在master分支上创建修复分支**

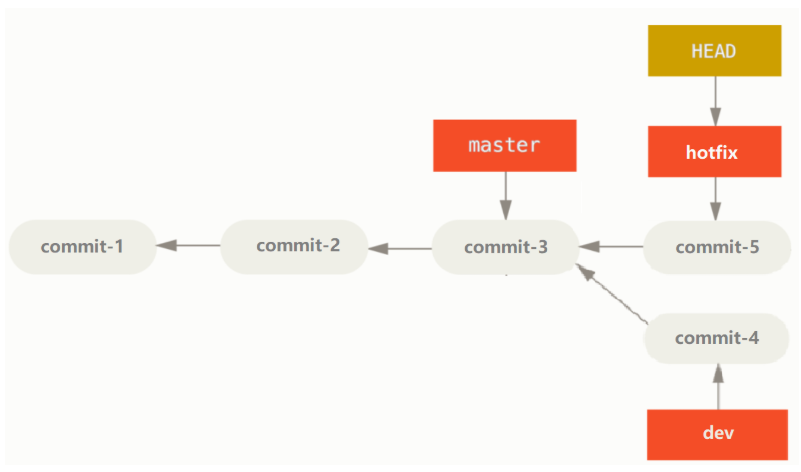
这个时候，你的工作目录将恢复到你在开发dev分支之前的模样，现在你可以专心修复紧急问题了。（那份代码出现问题了，就从哪个代码上开分支）

|  |
| --- |
| 提示：当你切换分支的时候，Git 会重置你的工作目录， Git 会自动添加、删除、修改文件，以确保此时你的工作目录和这个分支最后一次提交时的样子一模一样。 |

接下来，你要修复这个紧急问题。 我们来建立一个 hotfix 分支，在该分支上工作直到问题解决：

|  |
| --- |
| Bash # 1.创建修复分支hotfix L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (dev) $ git checkout -b hotfix Switched to a new branch 'hotfix'  # 2.进行修复，增加V4版本 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (hotfix) $ echo 'branch test v4' >> readme.txt  # 3.提交到本地版本库 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (hotfix) $ git commit -a -m '第5次提交，新增内容：branch test v4' [hotfix 0690030] 第5次提交，新增内容：branch test v4  Date: Sat Apr 17 18:29:35 2021 +0800  1 file changed, 1 insertion(+)  # 4.查看本地库克历史提交信息 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (hotfix) $ git log --oneline --graph --all \* 0690030 (HEAD -> hotfix) 第5次提交，新增内容：branch test v4 | \* 9eb3224 (dev) 第4次提交，dev分支开发 dev.txt文件 |/ \* b97ccfd (master) 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 |

此时工作目录中图解如下：



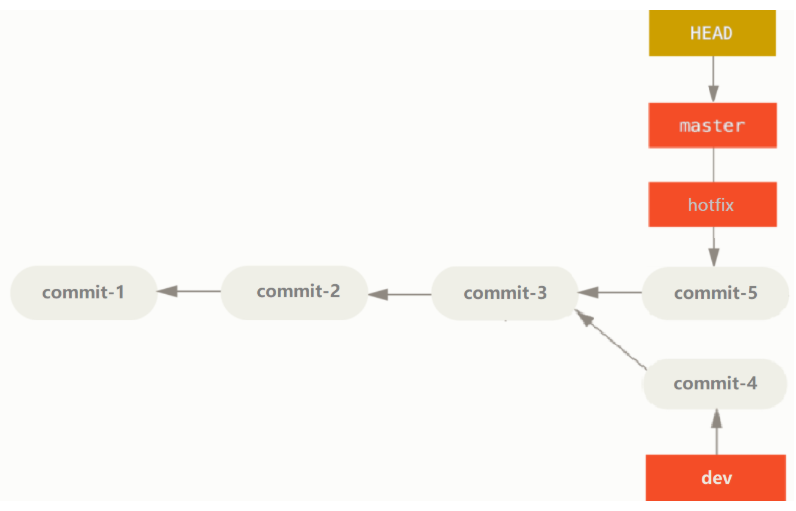
接下来可以运行测试你的修复，确保修改是正确的后，将 hotfix 分支合并回 master 分支，并部署到线上。

**4）将 hotfix 分支合并到 master 分支上**

你可以使用 git merge 命令来达到上述目的：

|  |
| --- |
| 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 (hotfix) $ git status On branch hotfix nothing to commit, working tree clean  # 2.切换到master分支 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (hotfix) $ git checkout master Switched to branch 'master'  # 3.查看是否切换到主分支上 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git branch  dev  hotfix \* master  # 4.合并hotfix分支到master分支 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git merge hotfix Updating b97ccfd..0690030 Fast-forward  readme.txt | 1 +  1 file changed, 1 insertion(+)  # 5.查看版本库历史提交记录 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git log --oneline --graph --all \* 0690030 (HEAD -> master, hotfix) 第5次提交，新增内容：branch test v4 | \* 9eb3224 (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 # 发现HEAD -> master, hotfix,说明此时master, hotfix分支合并了。 |

此时工作目录中图解如下：



在合并的时候，你应该注意到了“快进（fast-forward）”这个词。 由于你想要合并的分支 hotfix 所指向的提交 C5是你所在的提交 C3 的直接后继， 因此 Git 会直接将指针向前移动。

换句话说，当你试图合并两个分支时， 如果顺着一个分支走下去能够到达另一个分支，那么 Git 在合并两者的时候， 只会简单的将指针向前推进（指针右移），因为这种情况下的合并操作，没有需要解决的分歧的地方，这就叫做 “快进（fast-forward）”合并。（**快进合并是不会产生冲突的**）

现在，最新的修改已经在 master 分支所指向的提交快照中，你可以着手发布该修复了。

**5）合并完成后，删除hotfix分支**

关于这个紧急问题的解决方案发布之后，你需要准备回到被打断之前时的工作中。 然而，你应该先删除 hotfix分支，因为你已经不再需要它了， master 分支已经指向了同一个位置。

所以你可以使用带 -d 选项的 git branch 命令来删除分支：

|  |
| --- |
| 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 (master) $ git status On branch master nothing to commit, working tree clean  # 2.删除hotfix分支 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git branch -d hotfix Deleted branch hotfix (was 0690030).  # 3.查看版本库中中的分支 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git branch  dev \* master |

**6）之后我们就可以把修复的代码推送到远程仓库了**

这里先不讲解。

**7）切换到dev分支，继续自己的开发**

现在你可以切换回你刚刚在工作的分支，继续你的工作。

|  |
| --- |
| 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dev分支 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git checkout dev Switched to branch 'dev'  # 巴拉巴拉继续搬砖。 |

|  |
| --- |
| 提示：你在 hotfix 分支上所做的工作并没有包含到 dev 分支中。 如果你需要拉取 hotfix 所做的修改，你可以使用 git merge master 命令将 master 分支合并入 dev 分支，或者你也可以等到 dev 分支完成其使命，再将其合并回 master 分支。 |

**3、补充：分支选项**

--merged 与 --no-merged 这两个有用的选项，可以过滤这个列表中，已经合并或尚未合并到当前分支的其他分支。

**（1）--merged 选项**

如果要查看哪些分支已经合并到当前分支，可以运行 git branch --merged命令：

|  |
| --- |
| Bash $ git branch --merged  iss53 \* master |

因为之前已经合并了 iss53 分支，所以看到它在列表中。

在列表中，分支名字前没有 \* 号的分支，通常可以使用 git branch -d 删除掉；你已经将它们的工作整合到了另一个分支，所以并不会失去任何东西。

**（2）--no-merged 选项**

查看所有未合并到当前分支的其他分支，可以运行 git branch --no-merged命令：

|  |
| --- |
| Bash $ git branch --no-merged  testing |

这里显示了testing分支。 因为它包含了还未合并的工作（也就是该分支开发完还没有合并），尝试使用 git branch -d 命令删除它时会失败：

|  |
| --- |
| Bash $ git branch -d testing error: The branch 'testing' is not fully merged. If you are sure you want to delete it, run 'git branch -D testing'. |

如果真的想要删除该分支，并丢掉那些工作，可以使用 -D 选项强制删除它。

作者：繁华似锦Fighting 链接：https://www.jianshu.com/p/99d1841e2c44 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。