'GIT分支及标签使用'

## GIT分支（Branch）

Git 的分支，其实本质上仅仅是指向提交对象的可变指针。 Git 的默认分支名字是 master。 在多次提交操作之后，你其实已经有一个指向最后那个提交对象的 master 分支。 它会在每次的提交操作中自动向前移动。

Git分支模型

Git分支模型

### 创建分支

想要新建一个分支并同时切换到那个分支上，你可以运行一个带有-b参数的git checkout命令

$ git checkout -b new-branch  
Switched to a new branch "new-branch"

### 切换分支

切换到现有分支，使用git checkout命令

$ git checkout master  
Switched to branch 'master'

### 远程分支

#### 推送本地分支到远程仓库

$ git push origin new-branch  
Counting objects: 20, done.  
Compressing objects: 100% (14/14), done.  
Writing objects: 100% (15/15), 1.74 KiB, done.  
Total 15 (delta 5), reused 0 (delta 0)  
To git@github.com:schacon/simplegit.git  
 \* [new branch] new-branch -> new-branch

#### 拉取远程分支

$ git fetch origin  
remote: Counting objects: 7, done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 (delta 0), reused 3 (delta 0)  
Unpacking objects: 100% (3/3), done.  
From https://github.com/schacon/simplegit  
 \* [new branch] new-branch -> origin/new-branch

要特别注意的一点是当抓取到新的远程跟踪分支时，本地不会自动生成一份可编辑的副本（拷贝）。 换一句话说，这种情况下，不会有一个新的 new-branch 分支 - 只有一个不可以修改的 origin/new-branch 指针。可以运行 git merge origin/new-branch 将这些工作合并到当前所在的分支。 如果想要在自己的 new-branch 分支上工作，可以将其建立在远程跟踪分支之上

#### 跟踪远程分支

$ git checkout -b serverfix origin/serverfix  
Branch serverfix set up to track remote branch serverfix from origin.  
Switched to a new branch 'serverfix'

### 合并分支

你可以使用 git merge 命令来达到目的

$ git checkout new-branch  
$ git merge master  
Updating f42c576..3a0874c  
Fast-forward  
 index.html | 2 ++  
 1 file changed, 2 insertions(+)

### 冲突解决

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。 如果你对 #53 问题的修改和有关 master 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：

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

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

$ 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")

任何因包含合并冲突而有待解决的文件，都会以未合并状态标识出来。 Git 会在有冲突的文件中加入标准的冲突解决标记，这样你可以打开这些包含冲突的文件然后手动解决冲突。 出现冲突的文件会包含一些特殊区段，看起来像下面这个样子：

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

这表示 HEAD 所指示的版本（也就是你的 master 分支所在的位置，因为你在运行 merge 命令的时候已经检出到了这个分支）在这个区段的上半部分（======= 的上半部分），而 new-branch 分支所指示的版本在 ======= 的下半部分。 为了解决冲突，你必须选择使用由 ======= 分割的两部分中的一个，或者你也可以自行合并这些内容。

#### cherry-pick

在本地 master 分支上做了一个commit ( 38361a68138140827b31b72f8bbfd88b3705d77a ) ， 如何把它放到本地new-branch分支上？ 办法之一是使用cherry-pick命令来完成 简单用法：

git cherry-pick <commit id>

例如：

$ git checkout old\_cc  
$ git cherry-pick 38361a68

如果顺利，就会正常提交,结果：

Finished one cherry-pick.  
# On branch old\_cc  
# Your branch is ahead of 'origin/old\_cc' by 3 commits.

如果在cherry-pick的过程中出现了冲突

Automatic cherry-pick failed. After resolving the conflicts,  
mark the corrected paths with 'git add <paths>' or 'git rm <paths>'  
and commit the result with:   
  
 git commit -c 15a2b6c61927e5aed6718de89ad9dafba939a90b

**就跟普通的冲突一样，手工解决**

#### 二进制文件冲突解决方法

对于二进制文件的冲突，你肯定不想通过编辑二进制文件来解决冲突，那是不可能完成的事情。 你要做的就是：要么选择对方的修改，要么选择自己的修改。 你可以用git checkout的–theirs或–ours选项。

git pull  
git checkout --theirs YOUR\_BINARY\_FILE  
// git checkout --ours YOUR\_BINARY\_FILE  
git add YOUR\_BINARY\_FILE  
git commit -m 'merged with the remote repos.'  
git push

## GIT标签（Tag）

像其他版本控制系统（VCS）一样，Git 可以给历史中的某一个提交打上标签，以示重要。 比较有代表性的是人们会使用这个功能来标记发布结点（v1.0 等等）。

### 列出标签

在 Git 中列出已有的标签是非常简单直观的,只需要输入git tag：

$ git tag  
v0.1  
v1.3

这个命令以字母顺序列出标签；但是它们出现的顺序并不重要。

你也可以使用特定的模式查找标签。 例如，Git 自身的源代码仓库包含标签的数量超过 500 个。 如果只对 1.8.5 系列感兴趣，可以运行：

$ git tag -l 'v1.8.5\*'  
v1.8.5  
v1.8.5-rc0  
v1.8.5-rc1  
v1.8.5-rc2  
v1.8.5-rc3  
v1.8.5.1  
v1.8.5.2  
v1.8.5.3  
v1.8.5.4  
v1.8.5.5

### 创建标签

Git 使用两种主要类型的标签：**轻量标签（lightweight）**与**附注标签（annotated）**。 > 一个轻量标签很像一个不会改变的分支 - 它只是一个特定提交的引用。 > 附注标签是存储在 Git 数据库中的一个完整对象。 它们是可以被校验的；其中包含打标签者的名字、电子邮件地址、日期时间；还有一个标签信息；并且可以使用 GNU Privacy Guard （GPG）签名与验证。 通常建议创建附注标签，这样你可以拥有以上所有信息；但是如果你只是想用一个临时的标签，或者因为某些原因不想要保存那些信息，轻量标签也是可用的。

#### 附注标签

在 Git 中创建一个附注标签是很简单的。 最简单的方式是当你在运行 tag 命令时指定 -a 选项：

$ git tag -a v1.4 -m 'my version 1.4'  
$ git tag  
v0.1  
v1.3  
v1.4

-m 选项指定了一条将会存储在标签中的信息。 如果没有为附注标签指定一条信息，Git 会运行编辑器要求你输入信息。

通过使用 git show 命令可以看到标签信息与对应的提交信息：

$ git show v1.4  
tag v1.4  
Tagger: Ben Straub <ben@straub.cc>  
Date: Sat May 3 20:19:12 2014 -0700  
  
my version 1.4  
  
commit ca82a6dff817ec66f44342007202690a93763949  
Author: Scott Chacon <schacon@gee-mail.com>  
Date: Mon Mar 17 21:52:11 2008 -0700  
  
 changed the version number

输出显示了打标签者的信息、打标签的日期时间、附注信息，然后显示具体的提交信息。

#### 轻量标签

另一种给提交打标签的方式是使用轻量标签。 轻量标签本质上是将提交校验和存储到一个文件中 - 没有保存任何其他信息。 创建轻量标签，不需要使用 -a、-s 或 -m 选项，只需要提供标签名字：

$ git tag v1.4-lw  
$ git tag  
v0.1  
v1.3  
v1.4  
v1.4-lw  
v1.5

这时，如果在标签上运行 git show，你不会看到额外的标签信息。 命令只会显示出提交信息：

$ git show v1.4-lw  
commit ca82a6dff817ec66f44342007202690a93763949  
Author: Scott Chacon <schacon@gee-mail.com>  
Date: Mon Mar 17 21:52:11 2008 -0700  
  
 changed the version number

#### 后期打标签

你也可以对过去的提交打标签。 假设提交历史是这样的：

$ git log --pretty=oneline  
15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'  
a6b4c97498bd301d84096da251c98a07c7723e65 beginning write support  
0d52aaab4479697da7686c15f77a3d64d9165190 one more thing  
6d52a271eda8725415634dd79daabbc4d9b6008e Merge branch 'experiment'  
0b7434d86859cc7b8c3d5e1dddfed66ff742fcbc added a commit function  
4682c3261057305bdd616e23b64b0857d832627b added a todo file  
166ae0c4d3f420721acbb115cc33848dfcc2121a started write support  
9fceb02d0ae598e95dc970b74767f19372d61af8 updated rakefile  
964f16d36dfccde844893cac5b347e7b3d44abbc commit the todo  
8a5cbc430f1a9c3d00faaeffd07798508422908a updated readme

现在，假设在 v1.2 时你忘记给项目打标签，也就是在 “updated rakefile” 提交。 你可以在之后补上标签。 要在那个提交上打标签，你需要在命令的末尾指定提交的校验和（或部分校验和）:

$ git tag -a v1.2 9fceb02

可以看到你已经在那次提交上打上标签了：

$ git tag  
v0.1  
v1.2  
v1.3  
v1.4  
v1.4-lw  
v1.5  
$ git show v1.2  
tag v1.2  
Tagger: Scott Chacon <schacon@gee-mail.com>  
Date: Mon Feb 9 15:32:16 2009 -0800  
  
version 1.2  
commit 9fceb02d0ae598e95dc970b74767f19372d61af8  
Author: Magnus Chacon <mchacon@gee-mail.com>  
Date: Sun Apr 27 20:43:35 2008 -0700  
  
 updated rakefile

### 共享标签

默认情况下，git push 命令并不会传送标签到远程仓库服务器上。 在创建完标签后你必须显式地推送标签到共享服务器上。 这个过程就像共享远程分支一样 - 你可以运行 git push origin [tagname]。

$ git push origin v1.5  
Counting objects: 14, done.  
Delta compression using up to 8 threads.  
Compressing objects: 100% (12/12), done.  
Writing objects: 100% (14/14), 2.05 KiB | 0 bytes/s, done.  
Total 14 (delta 3), reused 0 (delta 0)  
To git@github.com:schacon/simplegit.git  
 \* [new tag] v1.5 -> v1.5

如果想要一次性推送很多标签，也可以使用带有 –tags 选项的 git push 命令。 这将会把所有不在远程仓库服务器上的标签全部传送到那里。

$ git push origin --tags  
Counting objects: 1, done.  
Writing objects: 100% (1/1), 160 bytes | 0 bytes/s, done.  
Total 1 (delta 0), reused 0 (delta 0)  
To git@github.com:schacon/simplegit.git  
 \* [new tag] v1.4 -> v1.4  
 \* [new tag] v1.4-lw -> v1.4-lw

现在，当其他人从仓库中克隆或拉取，他们也能得到你的那些标签。

### 检出标签

在 Git 中你并不能真的检出一个标签，因为它们并不能像分支一样来回移动。 如果你想要工作目录与仓库中特定的标签版本完全一样，可以使用 git checkout -b [branchname] [tagname] 在特定的标签上创建一个新分支：

$ git checkout -b version2 v2.0.0  
Switched to a new branch 'version2'

当然，如果在这之后又进行了一次提交，version2 分支会因为改动向前移动了，那么 version2 分支就会和 v2.0.0 标签稍微有些不同，这时就应该当心了。

## 参考资料

* [Git Book: https://git-scm.com/book/zh/v2](https://git-scm.com/book/zh/v2)
* [Git教程:廖雪峰的官方网站](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000)