**Git操作 - 标签tag（二）**

**5、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（手动）推送标签到远程服务器上。

需要将标签推送到远程版本库作为一个发行版本，可以通过以下两种方式：

**（1）推送本地的指定标签**

这个过程就像共享远程分支一样，你可以执行命令： git push origin <tagname>。

|  |
| --- |
| Bash $ 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 |

**（2）推送本地所有为推送的标签**

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推送过去。

|  |
| --- |
| Bash $ 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 |

当然其他人从仓库中克隆或拉取，他们也能得到你的那些标签。

|  |
| --- |
| 说明：   * git push 推送两种标签使用 git push <remote> --tags 命令（remote：远程仓库）。 * 推送标签并不会区分轻量标签和附注标签， 没有选项能够让你只选择一种标签进行推送。 |

**（3）查看结果**

登录GitHub并打开远程版本库页面，在release中可以查看推送到远程库中的标签，即发行版本。

其他用户在更新本地版本库时，同时会将标签一并更新，然后可以在本地指定标签版本上，作一个新的分支进行开发，开发完成后再合并到主要分支上，最后将该分支删除。

**6、删除标签**

**（1）删除本地标签**

要删除掉本地仓库上的标签，可以使用命令 git tag -d <tagname>。

例如，可以使用以下命令删除一个轻量标签：

|  |
| --- |
| Bash $ git tag -d v1.4-lw Deleted tag 'v1.4-lw' (was e7d5add) |

注意上述命令并不会从任何远程仓库中移除这个标签。

**（2）删除远程标签**

你必须用命令： git push <remote> :refs/tags/<tagname> 来更新你的远程仓库。

|  |
| --- |
| 提示：若要删除远程库中的标签，首先要删除本地库中的该标签，然后再运行上面的命令。 |

第一种方式是 git push <remote远程库> :refs/tags/<tagname> ：

|  |
| --- |
| Bash $ git push origin :refs/tags/v1.4-lw To /git@github.com:schacon/simplegit.git  - [deleted] v1.4-lw |

上面这种操作的含义是，将冒号前面的空值推送到远程标签名，从而高效地删除它。

第二种更直观的删除远程标签的方式是：

|  |
| --- |
| Bash $ git push origin远程库 --delete <tagname> |

**7、修改标签指定提交的代码**

软件版本一旦被指定，即标签一旦与某一commit-id绑定，那么这个版本（提交）的代码还能修改吗？

* 若将master分支回退到该commit-id，然后再修改代码，修改完成后再提交，我们会发现该提交代码修改过了，但该标签绑定的commit-id并没有发生变化，即该软件版本（标签）指向的代码仍未修改。（也就是标签指向的提交没有改变）
* 当然，此时我们可以将该标签删除，然后再定义一个同名标签，与修改过代码的提交进行绑定，这样也是可以的。

但是如上操作存在一个巨大的风险，我们修改过的代码是master主分支上的，一旦修改过的代码出现问题，将可以导致整个代码出问题。所以，我们一般不会修改master主分支上的代码。

那应该怎么办？

* Git将标签定义为与分支同级别的概念，它不仅是一个提交的别名。
* Git允许程序员使用分支切换命令git checkout ，将代码转向标签所指定的版本。

小示例：

|  |
| --- |
| Bash # 1.查看当前版本库分支 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git tag v1.0 v2.0  # 2.切换到v1.0标签上 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learning\_github (master) $ git checkout v1.0 Note: switching to 'v1.0'.  You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.  If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:   git switch -c <new-branch-name>  Or undo this operation with:   git switch -  Turn off this advice by setting config variable advice.detachedHead to false  HEAD is now at 69fc420 第二次提交，添加v2版内容  L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learning\_github ((v1.0)) $ |

如上，命令执行完毕，系统给出了很多的提示，该提示的总体意思为：当前处于“分离头指针”状态，在该状态下用户的任何修改与提交对任何的分支都没有影响（言外之意是：其修改将不会被保留）。若想要保留修改，则可以通过git checkout -b命令，创建一个新的分支。

这里特别要注意一点，就是最后一行命令提示符末尾的((v1.0))，说明此时HEAD指针，指向了v1.0标签。

继续执行命令：

|  |
| --- |
| Bash # 3.执行git checkout -b 命令，是新的提交有分支指向 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learning\_github ((v1.0)) $ git checkout -b newbranch Switched to a new branch 'newbranch'  L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learning\_github (newbranch) $ |

我们可以看到HEAD指针由(v1.0)标签指向了newbranch分支上。在该新分支上再进行修改提交，然后再合并到master分支，最后再将该分支删除，此时创建的分支名称可以随意。

当newbranch分支合并到master分支后，仍需要删除原标签，然后再与新的commit-id绑定。所以，生产环境下，一旦标签定义完成，就不会对标签进行删除再绑定。而是会再定义一个新的标签与新的提交绑定。

**8、标签在.git目录中的位置**

无论是轻量标签还是附注标签，他们都会存在在.git/refs/tags目录中。

|  |
| --- |
| Bash # 1.查看.git/refs/tags目录 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ ll .git/refs/tags/ total 2 -rw-r--r-- 1 L 197121 41 4月 18 17:12 v1.0 -rw-r--r-- 1 L 197121 41 4月 19 20:46 v2.0  # 2.查看v1.0标签的内容 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git cat-file tag v1.0 object b97ccfd5f3e98c12ab9c0fb4ca4b308e6805f7ed type commit tag v1.0 tagger sun\_wk <sun\_wk@126.com> 1618737173 +0800  v1.0 里程碑  # 3.查看v1.0标签内容中object的类型 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git cat-file -t b97ccfd5f3e98c12 commit # 我们可以看到object对象的类型是commit对象， # 是一个提交，所以索引的是一个提交。  # 4.查看V1.0标签文件中的内容，可以看到也是一个对象的索引 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ cat .git/refs/tags/v1.0 e086427dfe88d8cd370d2f94eaf8610c169a3333  # 5.查看这个对象的类型 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git cat-file -t e086427 tag # 我们可以查看该对象是一个标签对象。  # 6.我们也可以通过tag对象的索引值，查看内容， # 和上边git cat-file tag v1.0命令显示的内容是一样的。 L@DESKTOP-T2AI2SU MINGW64 /j/git-repository/learngit (master) $ git cat-file -p e086427 object b97ccfd5f3e98c12ab9c0fb4ca4b308e6805f7ed type commit tag v1.0 tagger sun\_wk <sun\_wk@126.com> 1618737173 +0800  v1.0 里程碑 |

总结来说：

1. Git中的标签，都会以文件的形式存储在.git/refs/tags目录中。
2. 文件中记录的是tag标签的校验和（索引值）。
3. 通过索引值或者标签名可以查看该tag对象的内容。
4. tag对象的内容就包括他指向的commit。其实tag对象就是对commit对象的一个封装。

**9、本文中所使用到的命令**

* git tag：查看版本库中的标签列表。
* git tag -l "v1.8.5\*"：查看版本库中，可匹配的标签列表。
* git tag -a 标签名 -m '说明信息'：当前分支最新一次提交打标签。
* git tag -a 标签名 commit-id -m '说明信息'：为之前的提交打标签。
* git tag 标签名：创建轻量标签。
* git push 远程仓库名 标签名：推送标签到远程仓库。
* git push 远程仓库名 --tags：推送所有标签到远程仓库。
* git tag -d 标签名 ：删除本地版本库中的标签。
* git show 标签名：擦看标签的具体信息。

作者：繁华似锦Fighting 链接：https://www.jianshu.com/p/2f17f273ba45 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。