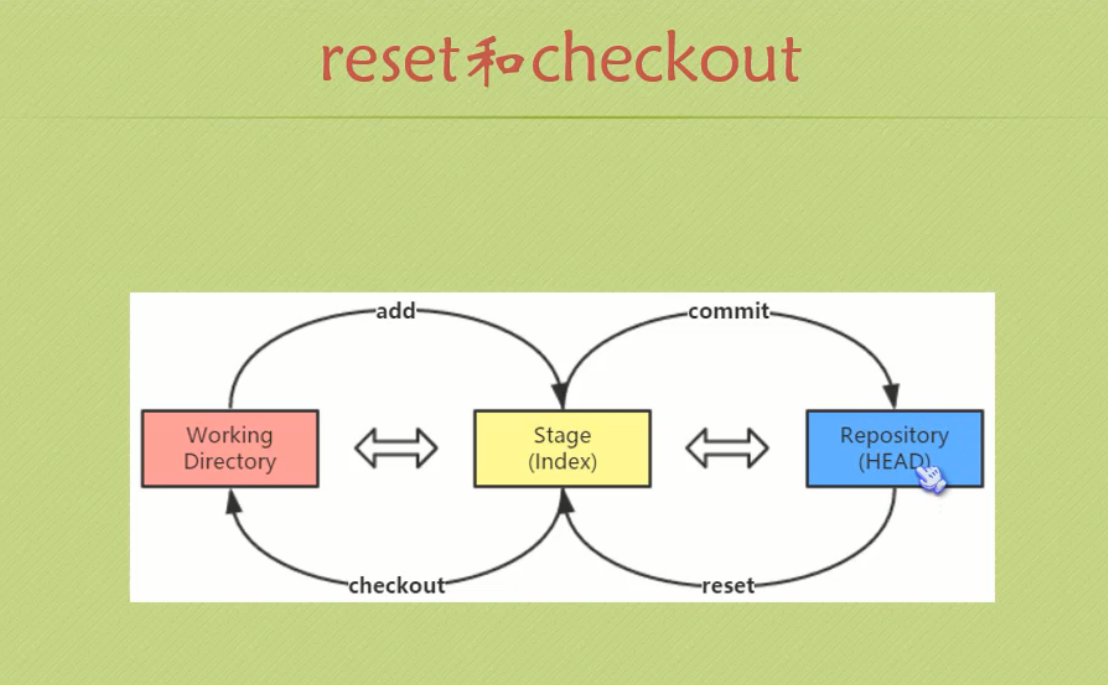
Git笔记

1. 初次使用git配置（添加自己的标识）

* git config --global user.name “” (don’t use Chinese)
* git config --global user.email “”
* git config --list验证，出现刚才配置的值即为成功

1. 理论基础
   1. Svn记录的是每一次版本变动的内容；Git则是将每个版本独立保存
   2. 三棵树：工作区域（Working Directory） 、暂存区（Stage（index））、Git仓库（Repository（HEAD））
   3. 工作流程
      1. 在工作目录中添加、修改文件；
      2. 将需要进行版本管理的文件放入暂存区域；
      3. 将暂存区域的文件提交到Git仓库；
   4. Git管理的文件三种状态
      1. 已修改（modified）
      2. 已暂存（staged）
      3. 已提交（committed）
2. 实战
   1. 创建并进入文件夹，git init 初始化仓库（Initialized empty Git repository）
   2. 在仓库中创建了或者修改了文件：
      1. git add filename 提交创建或者修改，对应2.3.2
      2. git commit –m “comment” 提交并添加说明，对应2.3.3(如果想要记述得更加详细,可以不加-m,执行后编辑器就会启动。第一行用文字简述提交的更改内容;第二行为空行;第三行记述更改的原因和详细内容。关闭即停止提交。)
3. 查看状态
   1. git status
   2. git checkout -- <file>... 将暂存区文件覆盖至工作区（即修改了一个文件之后未add，暂存区存在刚才add的该文件，注意--两边的空格）



1. 回退历史
   1. 查看历史提交 git log后面还有这个的一些log的相关选项参数介绍

* 后面如果加上目录名，便会显示该目录下的日志，接文件名，显示与该文件相关的日志。
* 如果要查看提交带来的变动，可以加上-p选项，文件的前后差别就会显示在提交信息之后。
  1. git reset HEAD 恢复到最新的一个快照，后面加~代表最新的上一个，依次类推。若有多个~，后面可以加上数字
* git reset –mixed HEAD~

1. 默认选项
2. 移动HEAD的指向，将其指向上一个快照；
3. 将HEAD移动后指向的快照回滚到暂存区域；
4. 影响了两棵树，Stage和Respository都变上快照记录的状态，而Working Directory还是最新的状态；

* git reset --soft HEAD~

1. 移动HEAD的指向，将其指向上一个快照；
2. 影响了Respository；

* git reset --hard HEAD~

1. 移动HEAD的指向，将其指向上一个快照；
2. 将HEAD移动后指向的快照回滚到暂存区域；
3. 将暂存区区域的文件还原到工作目录；
4. 影响了三棵树（危险，只会覆盖你工作目录的文件,但是不会删除你的新增文件）；

* git reset 版本快照的ID号（一般写四五位即可唯一确定）

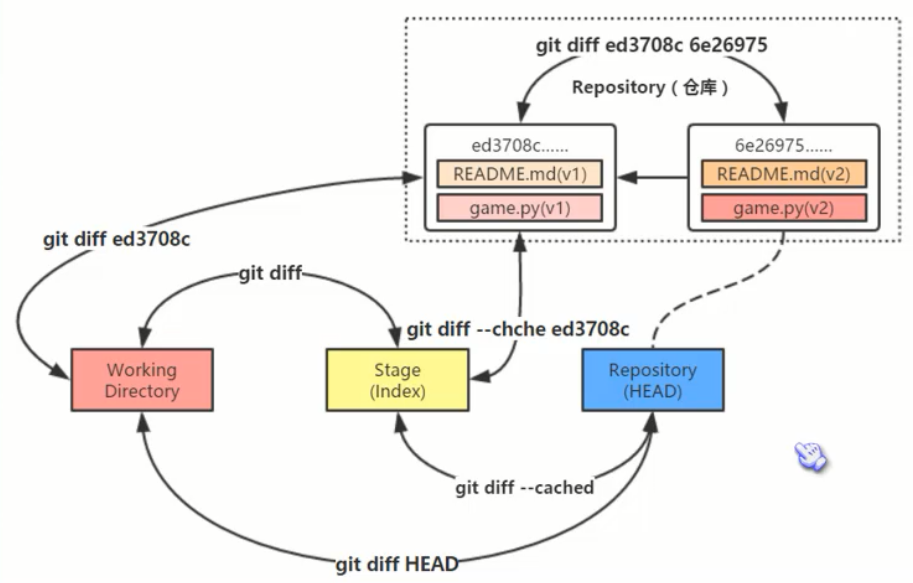
1. 回滚指定快照

* git reset 版本快照 文件名/路径

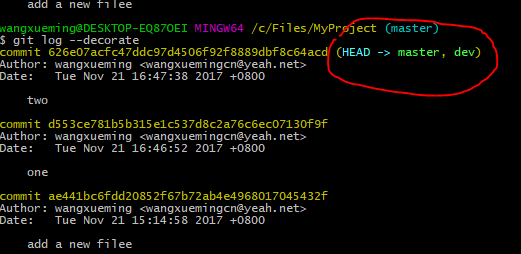
1. 回滚个别文件
2. 注意回滚的个别文件，HEAD指针并不会改变

* git reflog

1. git log 命令只能查看以当前状态为终点的历史日志，用来查看仓库的操作历史日志，通过日志可以找出回溯历史之前的哈希值，以便确定要回到未来的哪个版本
2. 版本对比
   1. git diff #比较暂存区域与工作目录
   2. git diff 快照1ID 快照2ID #比较两个历史快照
   3. git diff 快照ID #比较当前工作目录和Git仓库中的快照
   4. git diff --cached / git diff --cached 快照ID#比较暂存区域和Git仓库快照



1. 修改最后一次提交
   1. 在实际开发中，你可能会遇到以下两种情景：
      * 情景一：版本刚一提交（commit）到仓库，突然想起漏掉两个文件还没有添加（add）；
      * 情景二：版本刚一提交（commit）到仓库，突然想起版本说明写的不够全面，无法彰显你本次修改的重大意义；
      * git commit --amend 文件名 #执行带--amend选项的commit提交命令，Git就会“更正”最近的一次提交；
2. 删除文件
   1. git rm 文件名
      * 该命令删除的只是工作目录和暂存区域的文件，也就是取消跟踪，在下次提交不纳入版本管理（用在commit之后，并且删除之后HEAD指针不变，也就是说仓库的快照不会删除，如果需要删除需要修改HEAD指针指向）；
      * 若工作目录与暂存区域的文件不同，不能删除，除非加上-f选项强制删除两个地方的文件；
      * git rm --cached 文件名 #如果只删除暂存区域的文件（保留工作目录的），那么你可以执行这条命令实现目的；
3. 重命名文件
   1. git mv/ren 旧文件名 新文件名
      * 如果自己重命名，git status会出现过删除与新增的记录
4. Git分支
   1. git branch 显示分支一览表，带星的为当前分支，添加-a，显示本地仓库和远程仓库的分支信息
   2. git branch 分支名 #创建分支，可以使用git log --decorate查看（--oneline可以让日志只显示一行）

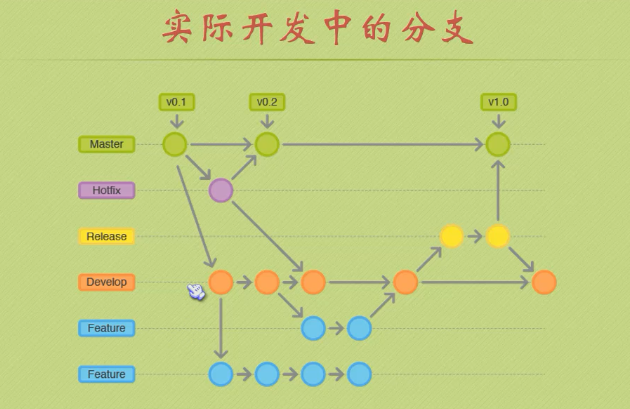




* 1. git checkout –b 分支名 #创建并切换到该分支
  2. git checkout –b 分支名 远程remote/远程分支名 #以远程仓库的某一分支开创建本地仓库的某一分支
  3. git checkout 分支名 #切换分支，将HEAD指针指到另一个分支

git log --decorate --oneline --graph --all 查看所有的分支情况（git log默认只显示自己的提交日志情况，选项各有意义，可以单用）

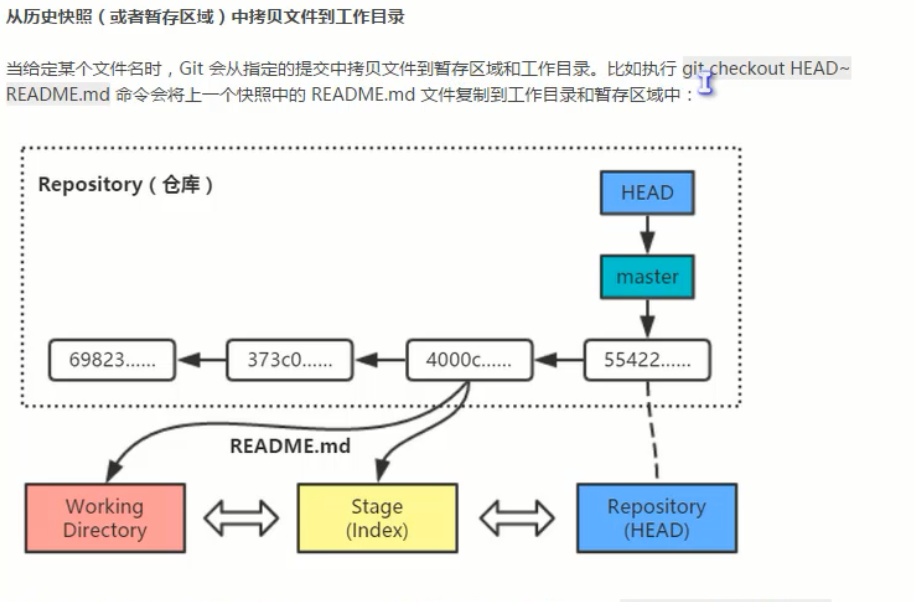
* 1. 实际开发中的分支



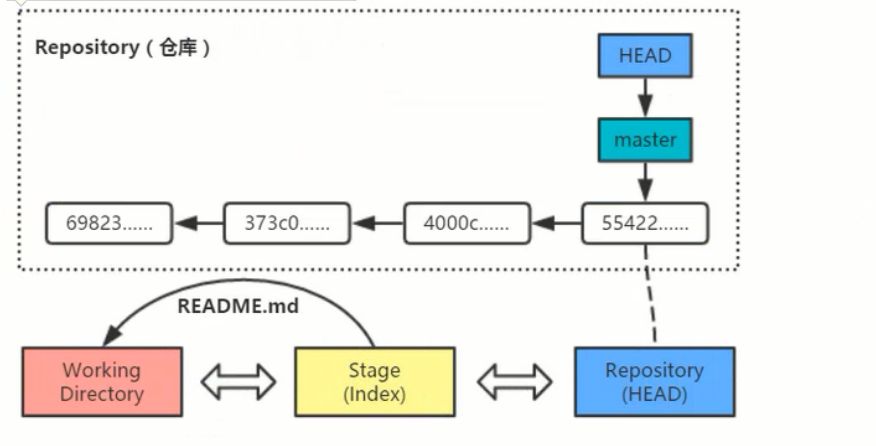
* 1. git merge 分支名
* 合并分支，例：合并dev到master，先切换到master分支，然后git merge dev即可；
* 有冲突会在一个冲突的模式中，可以使用git status查看冲突文件，需要处理了冲突，然后在提交合并。
* --no-ff参数可以启动编辑器录入合并提交的信息
  1. git branch –d 分支名 #删除分支
  2. git checkout HEAD~ #匿名分支

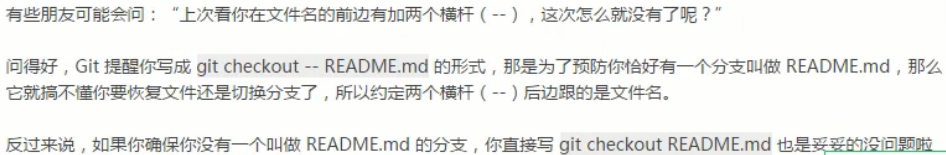
checkout后面接了一个快照而不是分支名，此时会创建一个匿名分支，匿名分支上所做的记录等到切换到其他分支后都会消失，除非使用“git branch 新分支名 最新的快照ID”来创建分支；

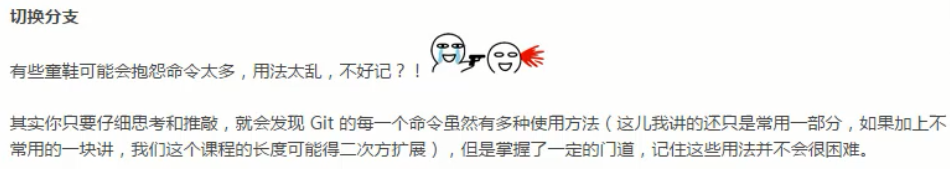
1. checkout命令
   1. 两种功能：
      * 从历史快照（或者暂存区域）中拷贝文件到工作目录；
      * 切换分支；
   2. 如下图：

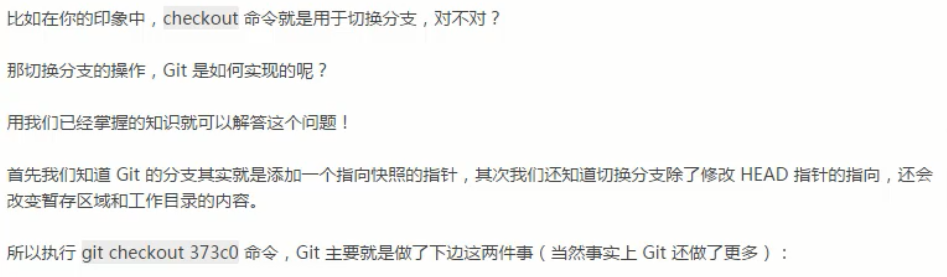


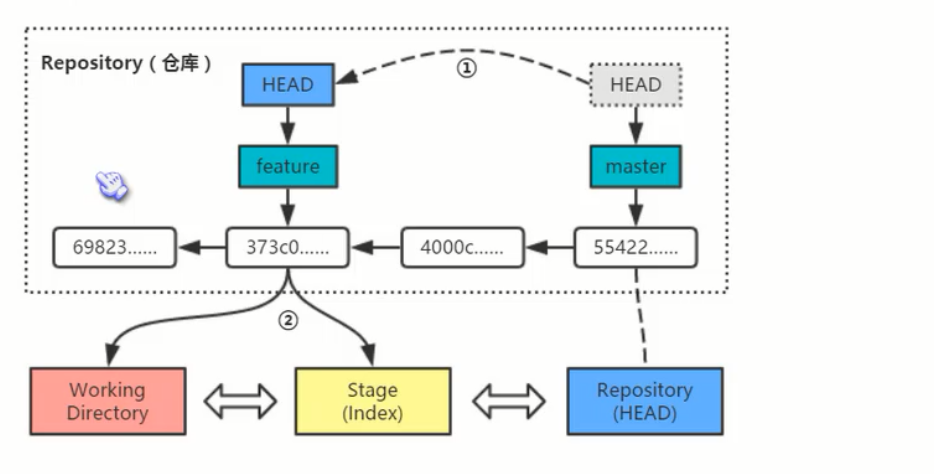


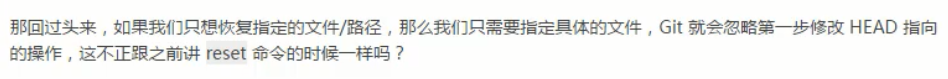






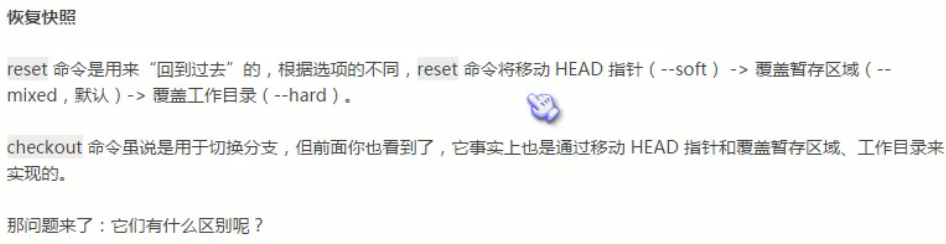


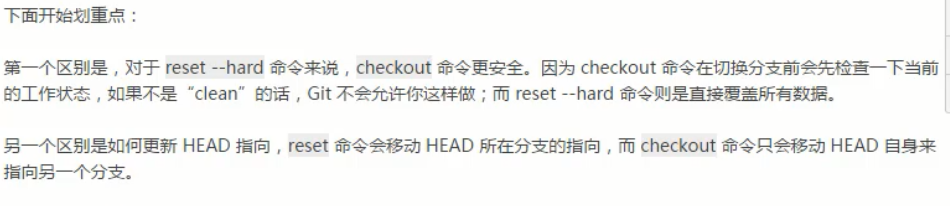




* 1. checkout与reset命令的区别









1. 远程仓库
   1. git remote add 标识符 仓库路径
   2. git push 标识符 分支名 #将本地仓库中的内容推送给远程仓库，-u参数可以在推送的同时，经标识符对应仓库的master分支设置为本地仓库当前分支的upstream（上游），添加了这个参数，将来运行git pull命令从远程仓库获取内容时，本地仓库的这个分支就可以直接从标识符的分支名分支获取内容，省去了另外添加参数的麻烦。
   3. git remote –v 查看远程仓库设置
2. Git中从远程的分支获取最新的版本到本地有这样2个命令：
   1. git fetch：相当于是从远程获取最新版本到本地，不会自动merge

* git fetch origin master
* git log -p master origin/master
* git merge origin/master #将远程的主分支合并到本地分支

以上命令的含义：

* 首先从远程的origin的master主分支下载最新的版本到origin/master分支上
* 然后比较本地的master分支和origin/master分支的差别
* 最后进行合并

上述过程其实可以用以下更清晰的方式来进行：

* git fetch origin master:tmp
* git diff tmp
* git merge tmp

从远程获取最新的版本到本地的test分支上

之后再进行比较合并

* 1. git pull：相当于是从远程获取最新版本并merge到本地

git pull origin master

上述命令其实相当于git fetch 和 git merge

在实际使用中，git fetch更安全一些

因为在merge前，我们可以查看更新情况，然后再决定是否合并

结束

1. Git tag 打标签

同大多数 VCS 一样，Git 也可以对某一时间点上的版本打上标签。人们在发布某个软件版本（比如 v1.0 等等）的时候，经常这么做。本节我们一起来学习如何列出所有可用的标签，如何新建标签，以及各种不同类型标签之间的差别。

* 1. [列显已有的标签](https://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%89%93%E6%A0%87%E7%AD%BE#列显已有的标签)：直接运行 git tag 即可：
* git tag
* 显示的标签按字母顺序排列，所以标签的先后并不表示重要程度的轻重。
  1. [新建标签](https://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%89%93%E6%A0%87%E7%AD%BE#新建标签)
* Git 使用的标签有两种类型：轻量级的（lightweight）和含附注的（annotated）。
* 轻量级标签就像是个不会变化的分支，实际上它就是个指向特定提交对象的引用。
* 附注标签，实际上是存储在仓库中的一个独立对象，它有自身的校验和信息，包含着标签的名字，电子邮件地址和日期，以及标签说明，标签本身也允许使用 GNU Privacy Guard (GPG) 来签署或验证。一般我们都建议使用含附注型的标签，以便保留相关信息；当然，如果只是临时性加注标签，或者不需要旁注额外信息，用轻量级标签也没问题。
* 创建一个含附注类型的标签非常简单，用 -a （译注：取 annotated 的首字母）指定标签名字即可：

git tag -a v1.0 -m 'version sh and sql'

而 -m 选项则指定了对应的标签说明，Git 会将此说明一同保存在标签对象中。如果没有给出该选项，Git 会启动文本编辑软件供你输入标签说明。

* 1. 可以使用 git show 命令查看相应标签的版本信息，并连同显示打标签时的提交对象。
* git show v1.0
* 我们可以看到在提交对象信息上面，列出了此标签的提交者和提交时间，以及相应的标签说明。
  1. [签署标签](https://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%89%93%E6%A0%87%E7%AD%BE#签署标签)
* 如果你有自己的私钥，还可以用 GPG 来签署标签，只需要把之前的 -a 改为 -s （译注： 取 signed 的首字母）即可：
* git tag -s v1.5 -m 'my signed 1.5 tag'
* 现在再运行 git show 会看到对应的 GPG 签名也附在其内：
  1. [轻量级标签](https://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%89%93%E6%A0%87%E7%AD%BE#轻量级标签)
* 轻量级标签实际上就是一个保存着对应提交对象的校验和信息的文件。要创建这样的标签，一个 -a，-s或 -m 选项都不用，直接给出标签名字即可
  1. [验证标签](https://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%89%93%E6%A0%87%E7%AD%BE#验证标签)
* 可以使用 git tag -v [tag-name] （译注：取 verify 的首字母）的方式验证已经签署的标签。此命令会调用 GPG 来验证签名，所以你需要有签署者的公钥，存放在 keyring 中，才能验证：

git tag -v v1.0

* 若是没有签署者的公钥，会报告类似下面这样的错误：

could not verify the tag 'v1.0'

* 1. [后期加注标签](https://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%89%93%E6%A0%87%E7%AD%BE#后期加注标签)
* $git log --pretty=oneline

15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'

a6b4c97498bd301d84096da251c98a07c7723e65 beginning write support

我们忘了在提交 “'experiment'” 后为此项目打上版本号 v1.2，没关系，现在也能做。只要在打标签的时候跟上对应提交对象的校验和（或前几位字符）即可：

git tag -a v1.2 1502795

* 1. [分享标签](https://git-scm.com/book/zh/v1/Git-%E5%9F%BA%E7%A1%80-%E6%89%93%E6%A0%87%E7%AD%BE#分享标签)
* 默认情况下，git push 并不会把标签传送到远端服务器上，只有通过显式命令才能分享标签到远端仓库。其命令格式如同推送分支，运行 git push [remotename] [tagname] 即可：

git push origin v1.5

* 如果要一次推送所有本地新增的标签上去，可以使用 --tags 选项：

git push origin –tags

* 1. 切换到标签
* 与切换分支命令相同，用git checkout [tagname]
  1. 删除标签
* 误打或需要修改标签时，需要先将标签删除，再打新标签。

git tag -d v0.1.2 # 删除标签

参数d即delete的缩写，意为删除其后指定的标签。