# 安装

目前我只有两个平台，一个是windows平台一个是linux

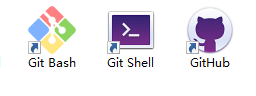
首先linux安装git比较简单直接一个安装命令就可以了，比如说在ubuntu环境下

sudo apt-get install git就可以了

然而我们的windows就不是很好了，我们需要在git官网上下载windows版本的git，由于中国需要翻墙，个人建议如果不能翻墙的话建议离线安装

<http://pan.baidu.com/s/1pLtLnoJ>这里是一个离线安装包的链接

安装就可以了，然后就会有三个程序



# 告诉git我们的名字以及邮箱

Git和github是连接上的，我们如果要使用需要让git绑定我们的github账号

我们需要执行

git config --global user.name "Your Name"

git config --global user.email "email@example.com"

--global这个参数表示这个机器上的所有仓库都使用这个账号

# 创建我们的个人仓库

我们现在就可以开始创建我们的仓库了，也可以说是版本库

我们首先要在我们指定的位置创建一个文件夹然后我们进去（windows路径尽量用英文不要带中文）

如果是windows就要多一步右键点Git Bash here就会出来一个终端，接下来我们就要输入命令了

首先需要在这个位置输入git init，然后终端就会在这个位置创建一个.git文件，这个文件一般是一个隐藏文件如果我们想看我们就得需要查看隐藏文件





我们尽量不要去动这个.git文件，因为我们的仓库最主要的文件就是它如果它数据丢失了，我们的库也就弄不起来了

然后我们可以在我们的仓库中创建一个readme.txt作为一个示例文件

Windows下需要注意需要使用UTF-8格式的文件，如果不是的话windows很可能会出现一些乱码

# 将编辑好的文件提交到仓库

然后我们可以编辑readme.txt了我们可以首先就写一句话



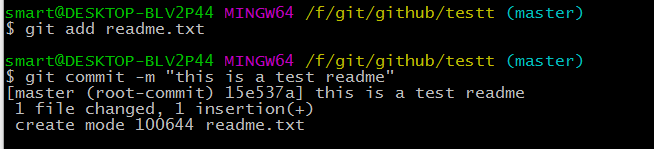
然后我们保存

我们现在就可以把readme上传到我们的仓库里面了

我们首先执行git add readme.txt命令，告诉仓库需要把这个文件放到仓库中，然后再执行git commit –m “this is a test file”

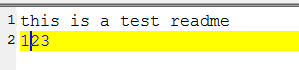
git commit命令是把文件提交到仓库，-m的意思就是后面的字符串是对这个文件的描述

这样我们就将我们的readme已经提交到仓库里面了

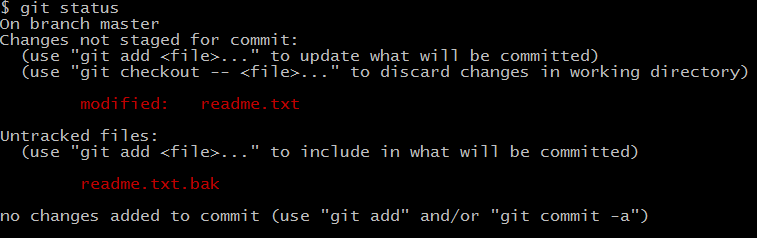


git status是查看现在库的状态的，如果我们没有改变文件的话它就会显示no changes，如果改变了它就会告诉你哪个文件被改变了但是还没提交

比如我们可以改一下readme文件

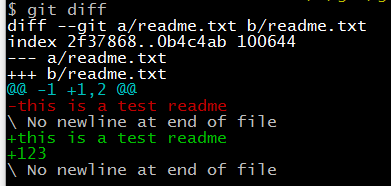


然后我们不提交看看



这就显示哪个文件被改变了，但是看不出来哪里被改变了，我们还有一个命令能查看哪里被改变了

git diff就能看到哪里被改变了

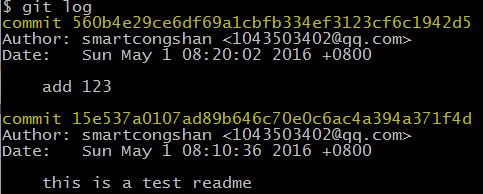


# 倒退回原来的版本

当我们写程序写的非常乱或者删错了东西的时候我们想回到原来的版本的时候git是有办法的

我们先来看看我们现在git已经给我们存储了多少个版本了

git log这个命令可以查看我们已经有多少个版本了



我们现在有两个版本，也就是说我们现在已经提交了两次

如果我们感觉很乱我们还有log还有一个选项让这些信息简化只是显示重要部分的东西git log - - pretty=oneline这样它只是显示很简洁的信息



现在我们可以退回我们原来的代码了，git reset命令是用来倒退回原来版本的

Git是通过一个HEAD指针来指向现在的版本的，所以我们HEAD^就代表退回上一个版本，如果HEAD^^就是退回到倒数第二个版本，如果很多退回到100个的话就可以写成HEAD-100

所以当我们写git reset –hard HEAD^就是退回到上一个版本

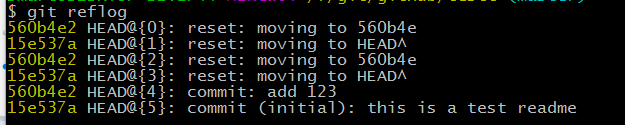


这时候我们的代码就是原来的代码了，但是我们又想推倒现在的版本了，git还是会有办法的i在我们之前看到的版本上面每个版本前面都会有一大堆数据，那个是每个版本的版本号，我们只需要写前几位就可以了



这样我们又回到之前的版本了

如果我们把终端关了然后忘了id怎么办，git reflog就可以帮我们找到之前的命令都干了什么



# 工作区和暂存区

Git的工作原理我们其实是需要知道的，当我们创建一个仓库git会为我们创建一个master分支，以及一个暂存区stage，工作区其实就是我们写代码的部分，也就是我们的文件夹，我们之前提交是用了两部分第一次是git add，然后git commit

第一次add其实就是把代码放到了暂存区，然后commit把暂存区的所有的代码放到主分支去，所以我们如果不add只是commit的话仓库的东西是不会变的

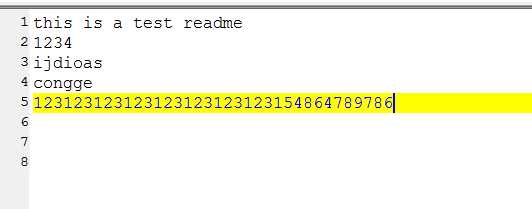
如果我们想查看现在工作区的代码跟现在主分区的最新版本的代码的区别我们也有命令

git diff HEAD -- file就可以了

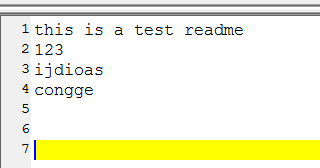
# 撤销

当我们在工作区的操作失误并且保存了之后我们也是有可能撤销的，在终端上输入git checkout – file

比如说我在readme上面写一个东西然后保存



然后我在终端上输入git checkout -- readme.txt，然后readme就变回原来的样子了



如果我们想撤销在暂存区

使用git reset HEAD file就可以了

Git checkout其实就是将版本库里面的东西拿到工作区而已

删除版本库中的东西用git rm就可以了

# 远程仓库

## 建立远程仓库

首先我们需要在github上面注册一个账号，我们现在有一个本地库，我们现在可以把我们本地库的内容推送上去

我们注册完账号之后我们首先需要创建SSH秘钥

我们需要执行ssh-keygen -t rsa -C "youremail@example.com"

然后一路回车就会在我们的C:\Users\你的用户名\.ssh\目录下就会有两个文件，打开那个.pub文件，我们把那里的东西复制到粘贴板，然后打开github网站

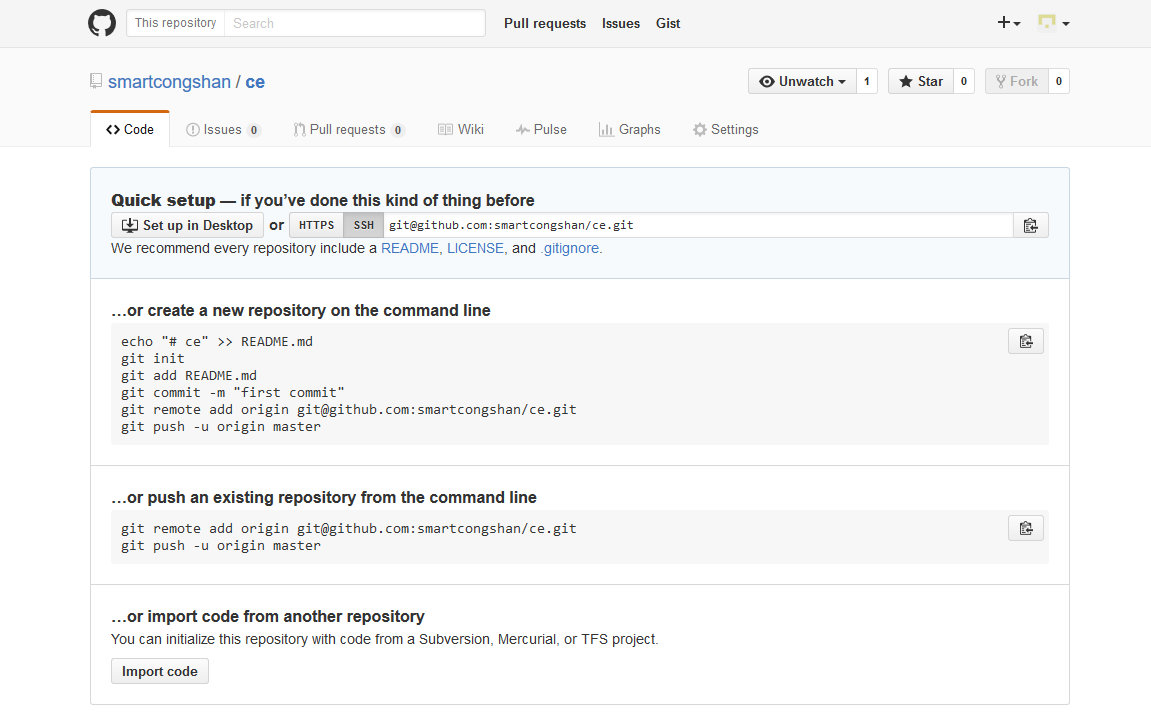




我们就按照图上面所示的弄就好了

## 将本地库推送到网上

我们首先在github上创建一个空的仓库



这里面有一个推送代码的命令

我们就按照它的写就行了

git remote add origin git@github.com:smartcongshan/ce.git

git push -u origin master

第二个指令的这个 –u是只有第一次推送到空的仓库里面使用的参数

这个就是不光把本地仓库的master分支推送到了github的master上而且还把它们两个关联起来

等到我们以后推送的时候直接就不用加 –u 这个选项了

只要我们本地仓库git commit完就可以直接就推送使用git push origin master指令就可以了

## 将网上的仓库克隆过来

我们回到我们的主目录下然后执行

git clone [git@github.com:username/repositoriesname.git](mailto:git@github.com:username/repositoriesname.git)

然后我们的目录下就出现了github上的仓库了

# 分支

Git的一大特色就是分支特别快相比于SVN来说快了非常多，因为git的分支其实就是一个指针

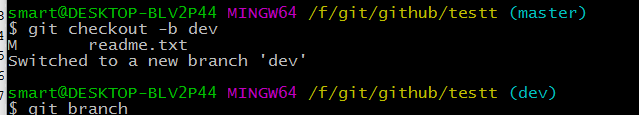
分支的具体原理看廖雪峰的教程里面写的很详细

我们创建一个分支用的是git branch，我们首先创建一个分支checkout有一个选项是-b，这个选项是既能创建一个分支又能切换到那个分支

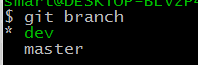
我们就先创建一个分支



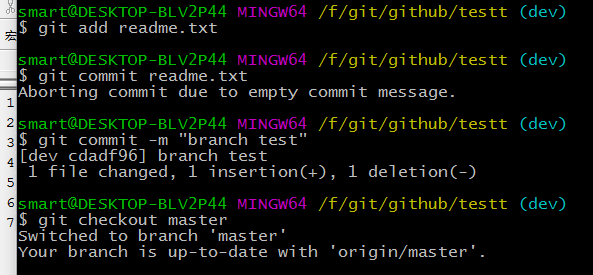
然后我们注意，在我们的windows下master会变成现在所在的分支



然后可以用git branch查看当前的在的是哪个分支



这时候我们改变readme文件然后并提交之后我们再回到master分支上再查看readme文件



切换是用的checkout指令

这时候我们就会看到readme还是跟我们原来的是一样的，我们刚才提交的是我们的分支所以我们的主分支是不会变的

然后我们用git merge dev融合分支

最后用git branch –d dev删除分支

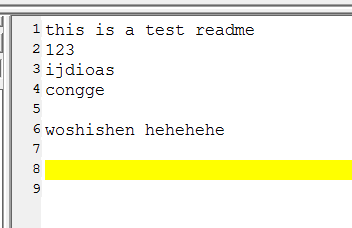
## 解决冲突

当我们创建分支之后在那里修改完了，但是我们的主分支却也改动了，改的地方跟分支改的地方相同但是内容却不同

如果说我们把两个分支都提交了之后然后融合分支就会出现冲突的错误

比如

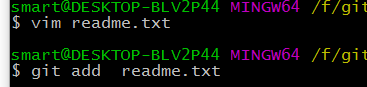
我们创建了一个ad分支，之后在ad分支的第6行加入了



然后我们提交，然后在主分支上再在那行修改成hhhhhhh

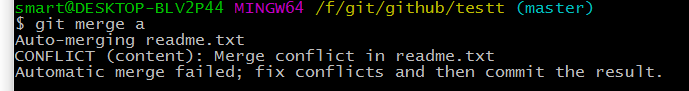
然后我们提交，然后融合分支就会提示我们错误

.

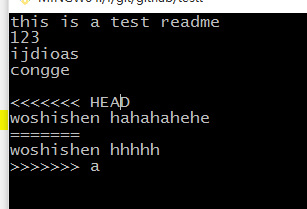




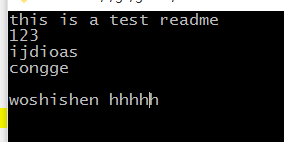
然后就会出冲突



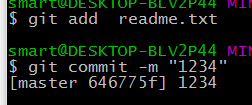
这时候如果我们再打开刚才编辑过的文件就会出现



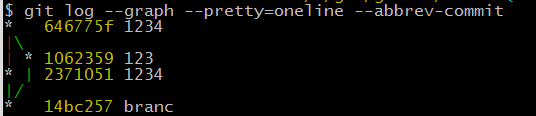
就会告诉你哪个是主分支的哪个是a分支的，然后我们就需要手动改成一样的



然后我们提交



这时候我们用git log看一下我们的版本



这时会我们就会看到分支冲突就是这么被解决的

然后我们要删除分支



--graph选项是可以看到分支合并图

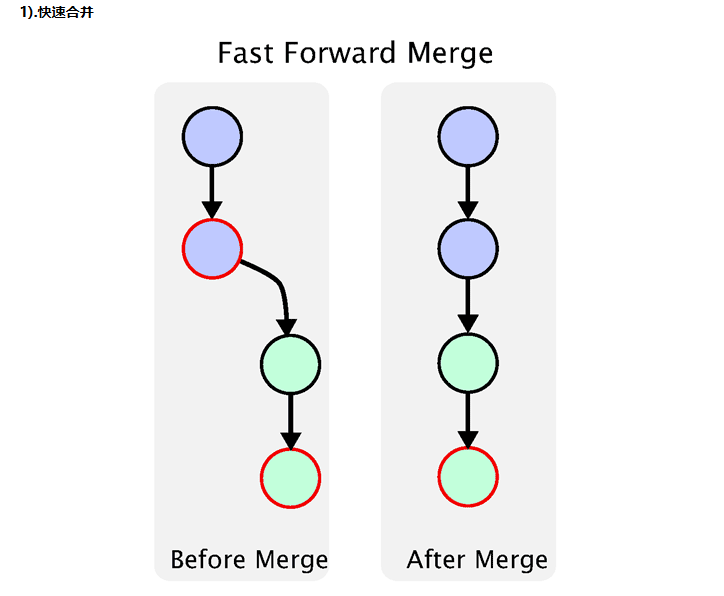
我们现在的分支就是下图的样子

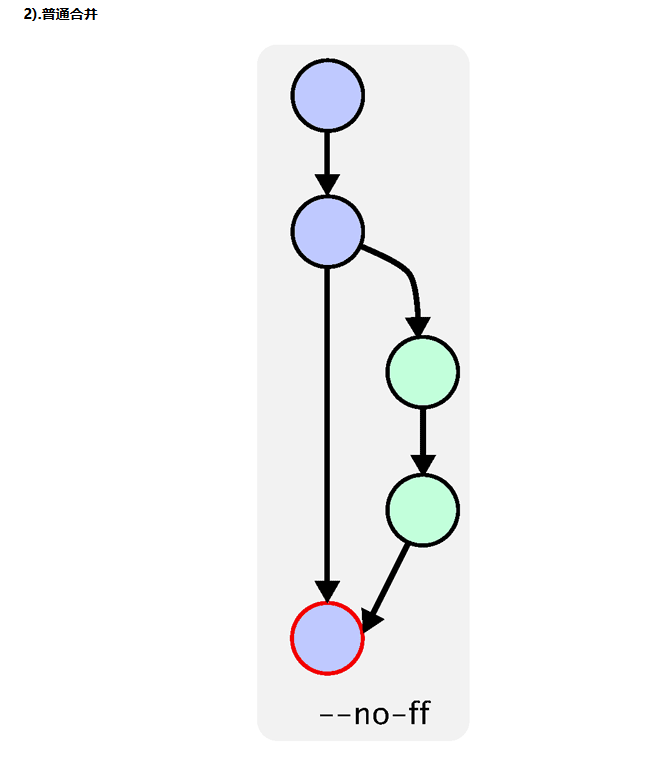


## 普通合并

Git的快速分支是将HEAD指向master 但是有很多情况用的不适合快速合并而是普通的合并

用几张图就可以很好的说明问题

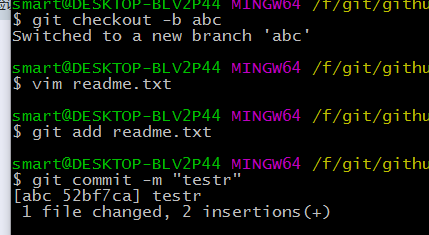




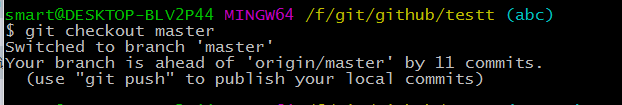
这就是两个的区别

我们来写一个

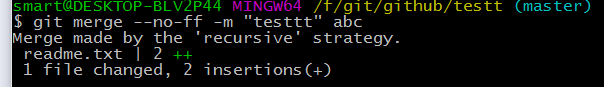
我们新创建个分支



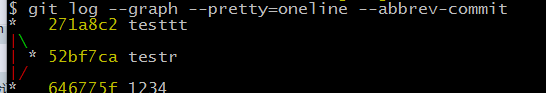
然后回到主分支



然后用 –no-ff参数，这个参数是将两个分支合并并且commit到主分支上的指令，也就是说我们需要加上 –m参数以及描述语句



我们可以用git log看看



我们在开发过程其实不是在master的分支上干的因为这个主分支最主要的目的就是推送版本的，我们研发会新建一个dev分支然后开始干，然后又别人的话会继续新建分支最后合并，用的并不是快速合并

## Bug分支

当我们的程序出现bug的时候我们就需要停止现在的工作去修复bug，但是如果我们现在正在工作的dev分支还没弄好不能提交的话我们就需要用到藏储命令了git stash，当我们使用这个命令的时候然后我们之前改的东西就被隐藏了，然后我们就可以去修复bug了，我们修复bug一般都是在master分支中进行的，也就是说我们回到master分支然后再次创建一个bug分支然后修复后合并分支就可以了

然后我们回到dev分支把我们原来隐藏的分支重新弄回来，这里面有两种方法

一种是用git stash pop是恢复了之后直接就把隐藏的内容给删了

还有一种就是git stash apply这个就是恢复内容但不删除，然后删除需要用git stash drop命令

## Feature分支

当我们需要增加新的功能的时候我们往往会在dev分支下新建一个feature分支来进行对程序新功能的开发

我们开发完之后就跟dev分支进行合并

合并完之后就可以删掉这个分支了，但是如果在没合并完的时候如果说不需要那个功能了，我们就可以删掉了但是这时候因为还没有合并普通删除是不好使的，我们得把-d换成-D进行强制删除就好了

# 多人合作

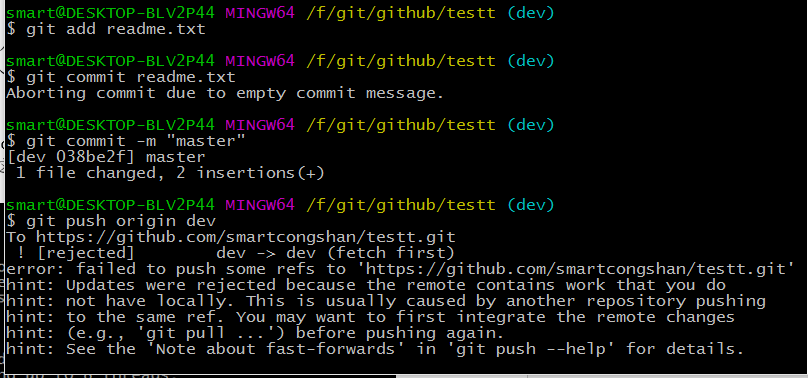
当我们好几个人一起完成一个项目的时候就需要注意一些东西了

就比如说如果我们创建这个项目的人上传了这个代码，然后有人从我们这里clone了这个工程

但是别人clone只能是clone master分支，别的分支也是不能clone，然后那个人必须要执行git checkout dev origin/dev这样就是建立了一个dev分支并且跟远程库关联起来了，然后我们在这里进行了修改，然后提交提交之后将dev分支推送到远程仓库中



然后我们在原来的工作区也进行同样的操作

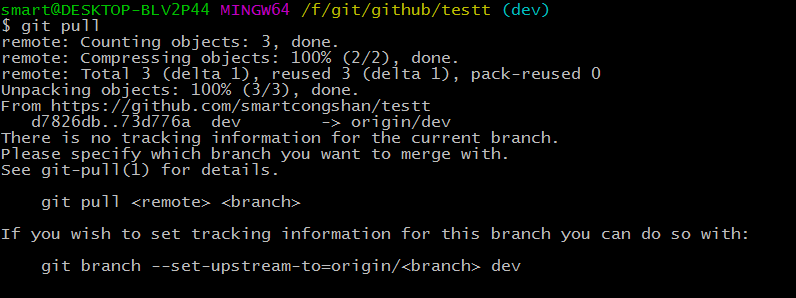


结果是失败的，因为这跟之前另外一个人提交的是有冲突的，这时候编辑的话跟之前的处理冲突是一样的，这代表现在这边的分支的版本是比较落后的

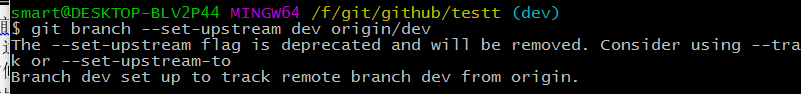
所以一般大家在写东西的时候每次都会更新一下分支

我们也可以看到在我们推送出错的时候就会有提示，说让我们用git pull试试

那么我们就用git pull试试

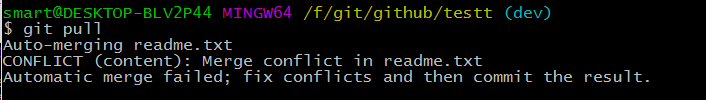


这也出错了，这是因为我们这边的dev分支没有跟远程库的关联上，我们用一个



这个命令就好了

--set-upstream这个选项就是将后面两个分支关联起来，这时候再去pull就会成功了



然后我们获得了最新版本的分支之后先解决冲突，然后再提交最后推送

# 标签

我们每次在主分支发布新版本的时候一般都会弄一个标签，也就是类似快照的一个东西，就是能够保存当时的提交的部分

我们用git tag命令来弄，我们如果只是用git tag 名字的话就是把最新的提交作为标签，我们也可以把之前的版本做成标签

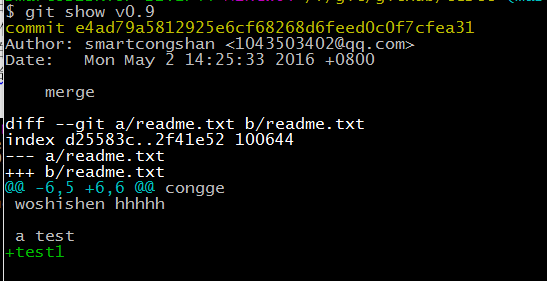
就是我们运用之前的git log命令看之前的序列号然后git tag v1.0 序列号 就可以了





注意必须要在主分支上

git show 标签名就可以看到这个标签的详细信息了

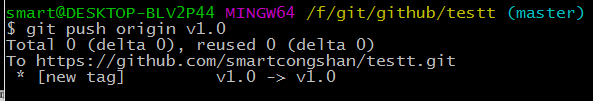




我们删除这个标签加上-d就可以了



我们如果想把标签推送到远程的话也可以用push



如果我们想在远程删除一个标签的话，我们用也用git push命令

但我们需要首先将本地的删除之后在用下面的命令

git push origin :refs/tags/v0.9

# 忽略文件

当我们在工作区中有不想让git检测到的文件的时候我们可以写一个.gitignore文件来忽略文件

这个文件的编写规则如下

# 此为注释 – 将被 Git 忽略

# 忽略所有 .a 结尾的文件

\*.a

# 但 lib.a 除外

!lib.a

# 仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO

/TODO

# 忽略 build/ 目录下的所有文件

build/

# 会忽略 doc/notes.txt 但不包括 doc/server/arch.txt

doc/\*.txt

abc.txt #文件名可以直接写

然后我们需要把.gitgnore提交一下就好了

# 配置别名

如果说我们感觉有些命令比较长的恶心我们可以用

git config --global alias.我们自己起的名字 我们要省略的名字

--global代表所有的仓库都用这个简写，如果不加只是目前所在的这个仓库用

我们可以在.git目录下的config文件可以看到我们配置的东西，我们总目录在users\用户名\下的.gitconfig文件中