# 前言

这个文章是3年前学习Git时整理的，很是粗糙。这篇整理的Git教程后期只会修改里面的明显错误，不再添加新内容。仅作为个人学习Git的小结。

主要是参考了以下两篇系列的文章：

[廖雪峰的git教程](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000)和[阮一峰的git教程中远程仓库Clone章节](http://www.ruanyifeng.com/blog/2014/06/git_remote.html)

# 一、安装Git

## 1.在Linux上安装Git

若是Ubuntu 或Debian Linux

$ sudo apt-get install git

老一点的Debian或Ubuntu Linux，要把命令改为sudo apt-get install git-core，因为以前有个软件也叫GIT（GNU Interactive Tools），结果Git就只能叫git-core了。由于Git名气实在太大，后来就把GNU Interactive Tools改成gnuit，git-core正式改为git

如果是其他Linux版本，可以直接通过源码安装。先从Git官网下载源码，然后解压，依次输入：./config，make，sudo make install这几个命令安装就好了

## 2.在Windows上安装Git

msysgit是Windows版的Git，从[http://msysgit.github.io/](http://msysgit.github.io/" \t "_blank)下载，然后按默认选项安装即可。补充[20171104]：下载git for windows

<https://www.git-scm.com/download/win>

或者

<https://git-for-windows.github.io/>（即以前的msysgit.github.io）

安装完成后，在开始菜单里找到“Git”->“Git Bash”，蹦出一个类似命令行窗口的东西，就说明Git安装成功！

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

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

注意git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

# 二、创建版本库

首先，选择一个合适的地方，创建一个空目录：



第二步，通过git init命令把这个目录变成Git可以管理的仓库：



# 三、Git基本操作

## 3.1提交文件到仓库

1. git add <filename> //文件加入暂存区（Stage）

2. git commit -m “提交说明” //文件从暂存区提交仓库

3. git status //查看状态

4. git diff <filename> //比较版本库和工作区的file变化

我们编写一个readme.txt 放在studygit目录下（子目录也可以）

内容如下：

Git is a version control system.

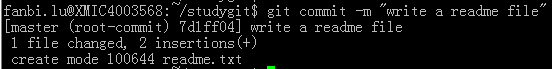
Git is free software.

把一个文件放到git仓库。需要两个步骤。

第一步，用命令git add 告诉Git，把文件添加到仓库：



第二步，用命令git commit 告诉Git，把文件提交到仓库：

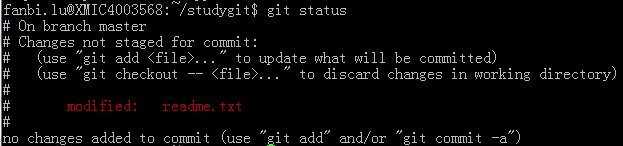


我们已经成功地添加并提交了一个readme.txt文件。然而，我们继续修改readme.txt文件，改成如下：

Git is a distributed version control system.

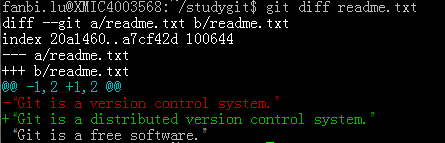
Git is free software.

现在，运行git status命令效果如下：



git status命令可以让我们时刻掌握仓库当前的状态，上面的命令告诉我们，readme.txt被修改过了，但还没有准备提交的修改。

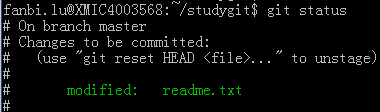
如果你忘记了修改了什么内容，可以用git diff这个命令查看



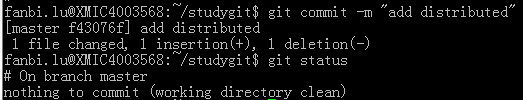
知道对readme.txt文件修改了什么。那就提交到仓库。还是分两步。

第一步是git add:





第二步是git commit



## 3.2版本回退

5. git reset --hard commit\_id //回到commit\_id版本的仓库

6. git log //查看提交历史

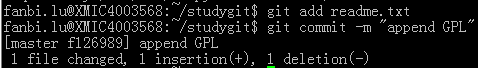
7. git reflog //查看命令历史

现在，我们再一次修改readme.txt如下：

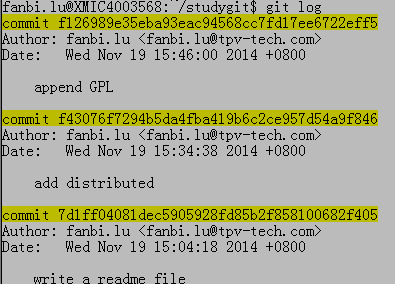
Git is a distributed version control system.

Git is free software distributed under the GPL.

我们再一次提交到仓库。



我们想查看了我们提交历史记录，用git log



如果觉得log信息输出太多。可以用git log –-pretty=oneline.



此时我们想回到上一个版本，我们可以用git reset --hard HEAD^，如果是上上版本就是HEAD^^,如果上上30个版本，HEAD~30就可以了。



我们可以查看下readme.txt 是否回到上一个版本。



此时我又想回到 “append GPL” 那个版本,怎么办？

还好我们知道它的commit\_id。所以可以用命令git reset –hard f12698



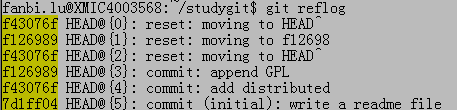
我们再次看下readme.txt。果然又回到”append GPL”那个版本了



但是如果我们返回到”add distributed”那个版本，关了电脑回家了。第二天，又想回到之前最新的版本，查下git log 如下：



找不到”append GPL”这个版本的ID了，怎么办，请用git reflog 命令



就可以看到”append GPL”提交的ID是f126989。

git reset –hard 含义如下：



## 3.3工作区和暂存区

工作区（Working Directory）：就是你在电脑里能看到的目录，比如我的studygit文件夹就是一个工作区：

版本库（Repository）：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库



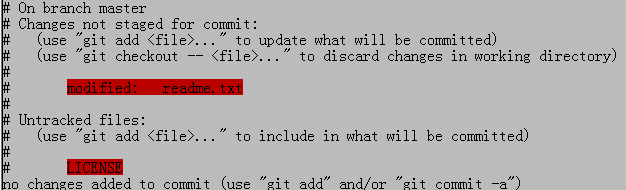
前面讲了我们把文件往Git版本库里添加的时候，是分两步执行的：

第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；

第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支

因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。

我们新建一文件LICENSE，修改readme.txt 查看下状态

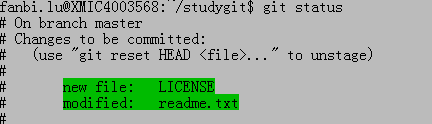


说明readme.txt被修改，LICENSE文件未被跟踪。

此时状态如下图：



git add 那两个文件，用git status查看如下：





我们执行git commit 一次性提交暂存区（stage）的所有修改到分支。



一旦提交后，如果你对工作区文件不做任何修改，此时工作区就是”干净”的。



版本库就变成如下：

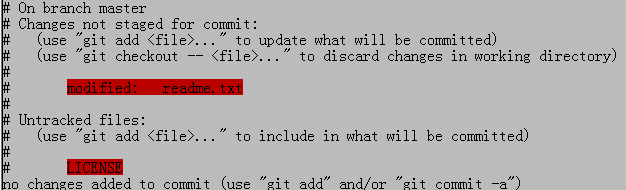


## 3.4管理和撤销修改

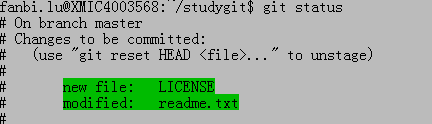
8. git checkout -- <file> //放弃工作区文件的修改

9. git reset HEAD <file> //撤销暂存区文件到文件区

当你修改工作区的readme.txt 但是还没有提交，你想回到没修改前的状态。当然你可以打开文件，慢慢删，但是手动操作，总让人感觉不舒服。如下图所示，你可以发现，Git会告诉你，git checkout -- file可以丢弃工作区的修改：



当你修改工作区的readme.txt，并且git add，但是还没有commit。那就用git reset HEAD readme.txt。 可以把暂存区的修改撤销掉（unstage）,重新放回工作区。



小结：

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。

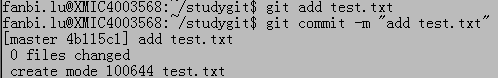
场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD file，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000)一节，不过前提是没有推送到远程库。

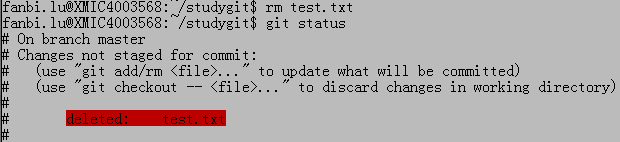
## 3.5删除文件

10. git rm <filename> //删除文件

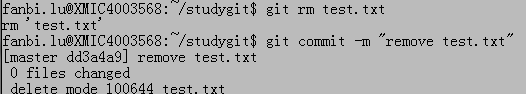
我们新添加一个test.txt文件并提交



用rm test.txt 删除文件后，Git 知道用户删除了文件，因此，工作区和版本库不一致了。用git status 看下



若要从版本库中删除文件，需要用git rm删掉文件，再提交。这样工作区和版本库就一致了。



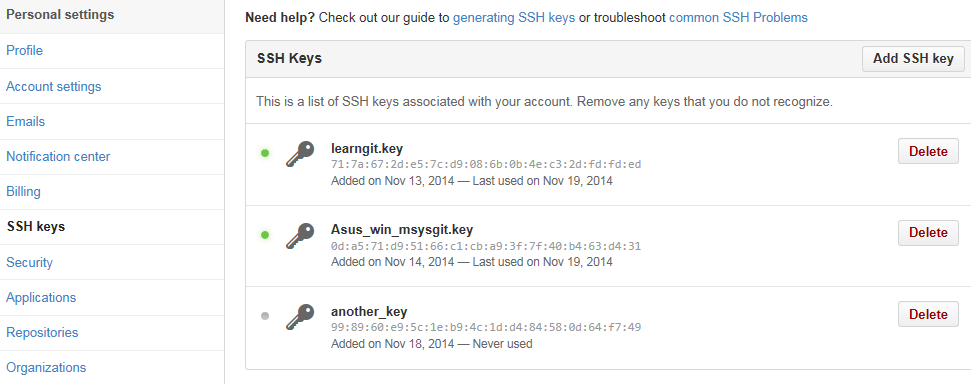
若刚才的git rm是删错了文件，版本库还有该文件。可以用git checkout -- <file> 撤销刚才的删除操作。

git checkout -- <file>其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。

# 四、远程仓库

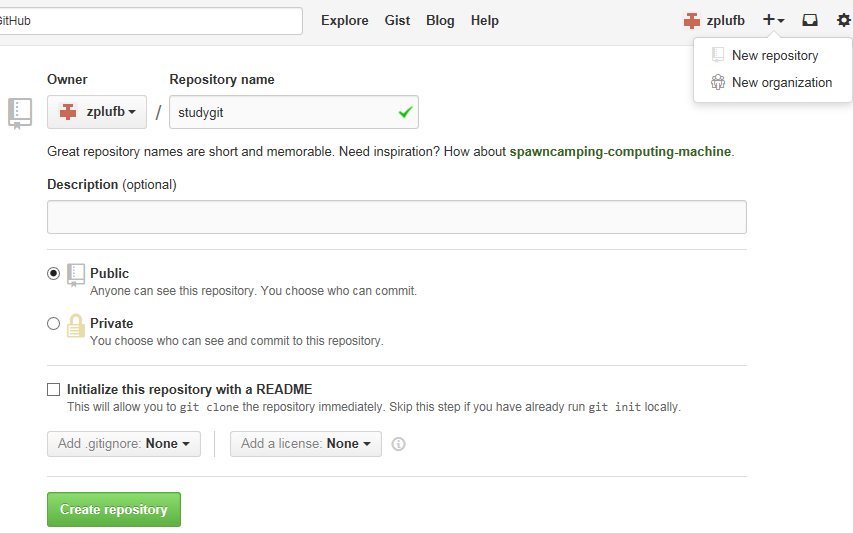
服务器端就是一个远程仓库。那怎么让客户端和服务端仓库关联起来呢？GitHub仓库也算是一种远程仓库（免费），由于你的本地Git仓库和服务器（GitHub仓库）之间的传输是通过SSH加密的，所以，需要一点设置：

$ ssh-keygen -t rsa -C [youremail@example.com](mailto:youremail@example.com) 生成密钥对。用户主目录下会生成.ssh文件夹。里面id\_rsa是私钥，id\_rsa.pub是公钥，这个要告诉服务器。例如：GitHub网站，在个人设置里添加ssh key

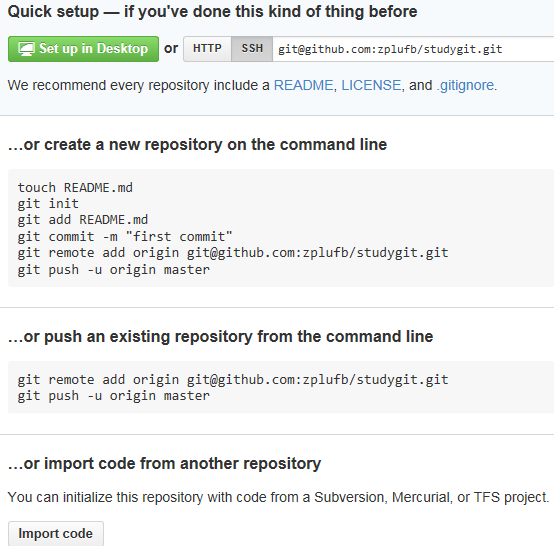


## 4.1添加远程仓库

这里是以GitHub作为例子。在GitHub新建一个空的仓库，如下

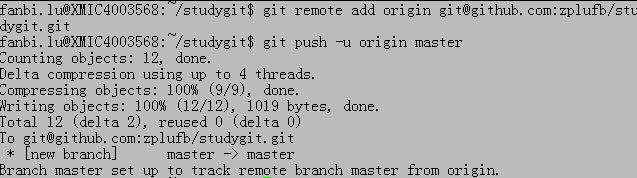


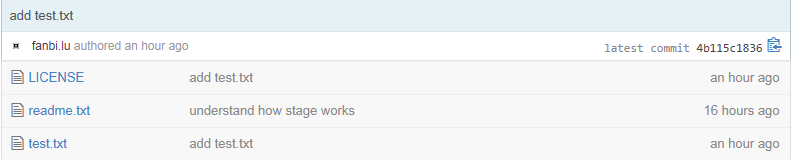
新建好了，如下：



根据提示，用git remote add origin [git@github.com:zlufb/studygit.git](mailto:git@github.com:zlufb/studygit.git) 关联远程仓库。

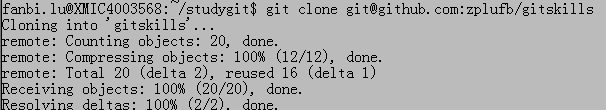
然后再git push –u origin master 命令，首次push时，待参数-u(意思是set-upstream for git pull/status)，后面push就不需要了。





## 4.2从远程仓库克隆

假定，我要clone 远程仓库。[那么用git@github.com:yourname/your\_project.git](mailto:那么用git@github.com:yourname/your_project.git) (ssh协议)



这样就在你当前目录下，clone了远程仓库。

Push

Pull

Fetch

Clone

Remote

详情见链接：<http://www.ruanyifeng.com/blog/2014/06/git_remote.html>

# 五、分支管理

## 5.1创建于合并分支

1.git branch //查看当前分支情况

2.git branch <branch\_name> //创建分支branch\_name

3.git checkout <branch\_name> //切换到分支branch\_name

4.git merge <branch\_name> //合并branch\_name分支到当前分支。

5.git branch –d <branch\_name> //删除分支branch\_name

6.git checkout –b <branch\_name> //创建分支并切换到该分支

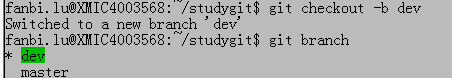
创建分支并切换到分支，可以用

git checkout –b dev

等价于

git branch dev

git checkout dev



我们在dev分支上修改readme.txt文件，添加内容如下：

Creating a new branch is quick.

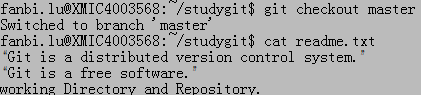
我们git add,git commit。



此时查看readme.txt



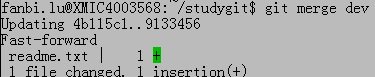
切换到master主分支上。在查看readme.txt如下：



发现readme.txt并没有变化。



那合并dev到master上。用git merge dev



此时master主分支上readme就和dev一致了。

此时dev分支使命完成了，那就删除它

Git branch –d dev

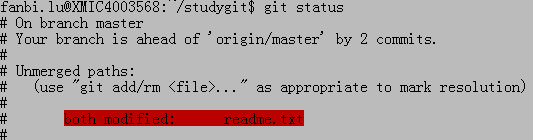


## 5.2冲突处理

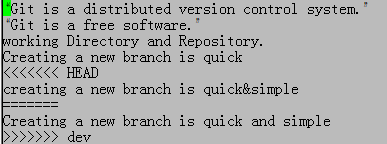
当dev和master分支都同时修改了readme.txt且都提交了。合并dev到master时就会发生冲突如下：



Git告诉我们，readme.txt文件存在冲突，必须手动解决冲突后再提交。查看下;

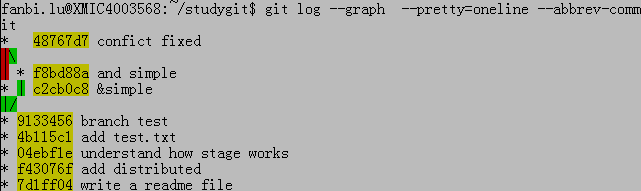


此时readme.txt 内容如下：



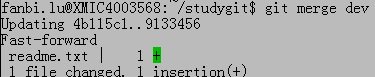
我们手动修改后，在add，commit就可以了。

我们用git log –graph –pretty=oneline –abbrev-commit查看下log.



## 5.3分支管理策略

之前我们是直接使用git merge dev,默认是使用了fast-forward模式，这种合并下，删除分支后，会丢掉分支信息。



我们合并时用git merge –-no-ff dev。则会保留分支信息。

合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

所以，团队合作的分支看起来就像这样：

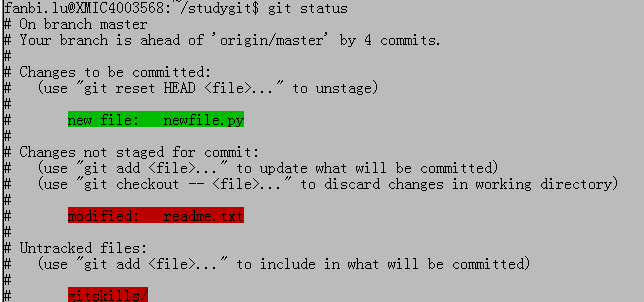


## 5.4 Bug分支

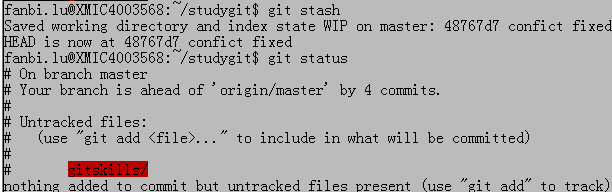
当你在接到一个代号101bug的任务时，你会想创建一个分支issue-101来修复它，但是你手头进行的工作还没有提交，怎么办?

Git stash 命令就是可以保存你现在的工作空间。

例子如下：newfile.py还没有被提交，readme.txt还没加到stage，此时又要修复一个bug任务。



此时用git stash命令。

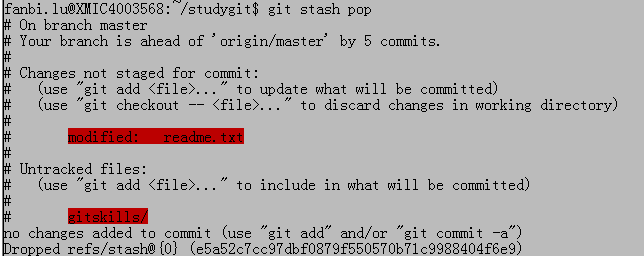


当你建立分支，修复完bug，切换到master,并完成合并。此时你要开始之前的工作，那用git stash list 来查看



此时可以用git stash apply 恢复，stash内容并不删除，得用命令git stash drop来删除。

或者用git stash pop,恢复同时，并同时把stash内容删除了。



当多次stash，恢复时，先用git stash list 查看，然后恢复指定的stash。



## 5.5 Feature分支

开发一个新功能的分支，并且你提交。现在要放弃这个新功能分支的开发，提示还没合并呢



提示我们用git branch –D feature来删除。



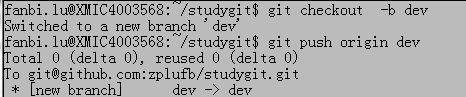
## 5.6 多人协助

•查看远程库信息，使用git remote -v；



•本地新建的分支如果不推送到远程，对其他人就是不可见的；

•从本地推送分支，使用git push origin branch-name.



如果推送失败，先用git pull抓取远程的新提交；

•在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；

•建立本地分支和远程分支的关联，使用git branch --set-upstream branch-name origin/branch-name；

•从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。

详情见链接：<http://www.ruanyifeng.com/blog/2014/06/git_remote.html>

详情见链接：<http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013760174128707b935b0be6fc4fc6ace66c4f15618f8d000>

# 六、标签管理

1. git tag <tagname> //打一个新标签

2. git tag //查看标签

3. git tag <tagname> <commit\_id>//在commit\_i那次提交打标签

4. git show <tagname> //查看标签

5. git tag –a <tagname> -m “说明” <commit\_id> //-a指定标签

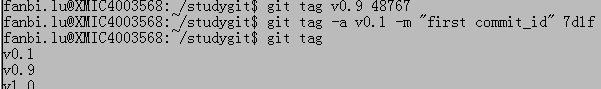
6. git tag –d <tagname>

7. git push origin <tagname> //tagname推送到远程仓库上。

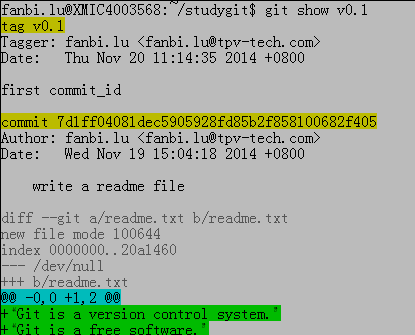
8. git push origin –tags //本地未推送的tags推送到远程仓库上

9. git push origin :refs/tags/tagname //删除远程仓库的某个tag

不同方式新建几个tag



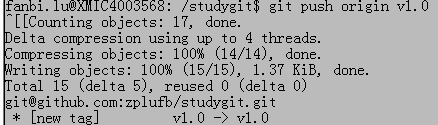
查看v0.1



Tag 可以创建，就可以删除。



推送某个标签(例如：v1.0)用git push origin v1.0。



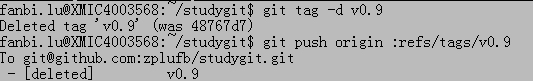
或者一次性把标签全部（未推送到）推送到远程仓库

Git push origin –-tags



当你需要修改之前的tag ，例如v0.9。

我们可以删除本地的v0.9，然而远程仓库还在。那我们需要用git push origin :refs/tags/v0.9



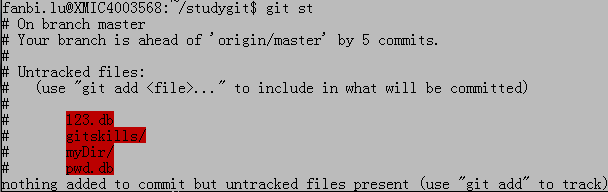
# 七、自定义Git

## 7.1 忽略特殊文件

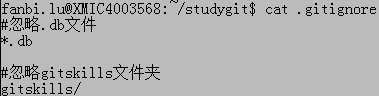
如果，我们不希望.db文件，debug文件夹，保留密码等文件，放入仓库

还有如果我们不打算把Untracked files的文件或文件夹加入仓库。

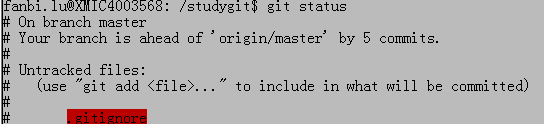
那就忽略它。



首先建一个.gitignore文件。在里面写上规则即可。支持正则表达式。



但是，我们要把.gitgnore文件加入到版本库中。



## 7.2 配置别名

git log –-pretty=oneline –abbrev-commit ，你可能觉得，命令好长 ,敲太累，或容易敲错了。git config --global alias.lg ‘log --pretty=oneline --abbrev-commit’

常见的别名如下：

$ git config --global alias.st status

$ git config --global alias.co checkout

$ git config --global alias.ci commit

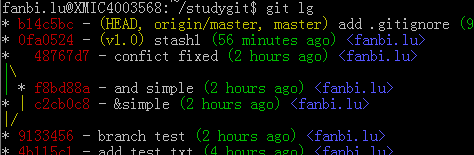
$ git config --global alias.br branch

$ git config --global alias.unstage 'reset HEAD'

$ git config --global alias.last 'log -1'

$ git config --global alias.lg

"log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset%s%Cgreen(%cr)%C(boldblue)<%an>%Creset' --abbrev-commit"



如果提示fatal: bad color value 'boldblue' for variable '--pretty format' ，则'boldblue'改成blue

配置文件，加上—-global的，当前用户下隐藏文件.gitconfig。否则，在各自仓库.git/config中。