**Git教案**

Git使用

<http://www.cnblogs.com/edisonchou/p/5990875.html>

<http://www.cnblogs.com/tugenhua0707/p/4050072.html>

<http://www.cnblogs.com/angeldevil/p/3238470.html>

<http://blog.csdn.net/wojilu/article/details/6976230>

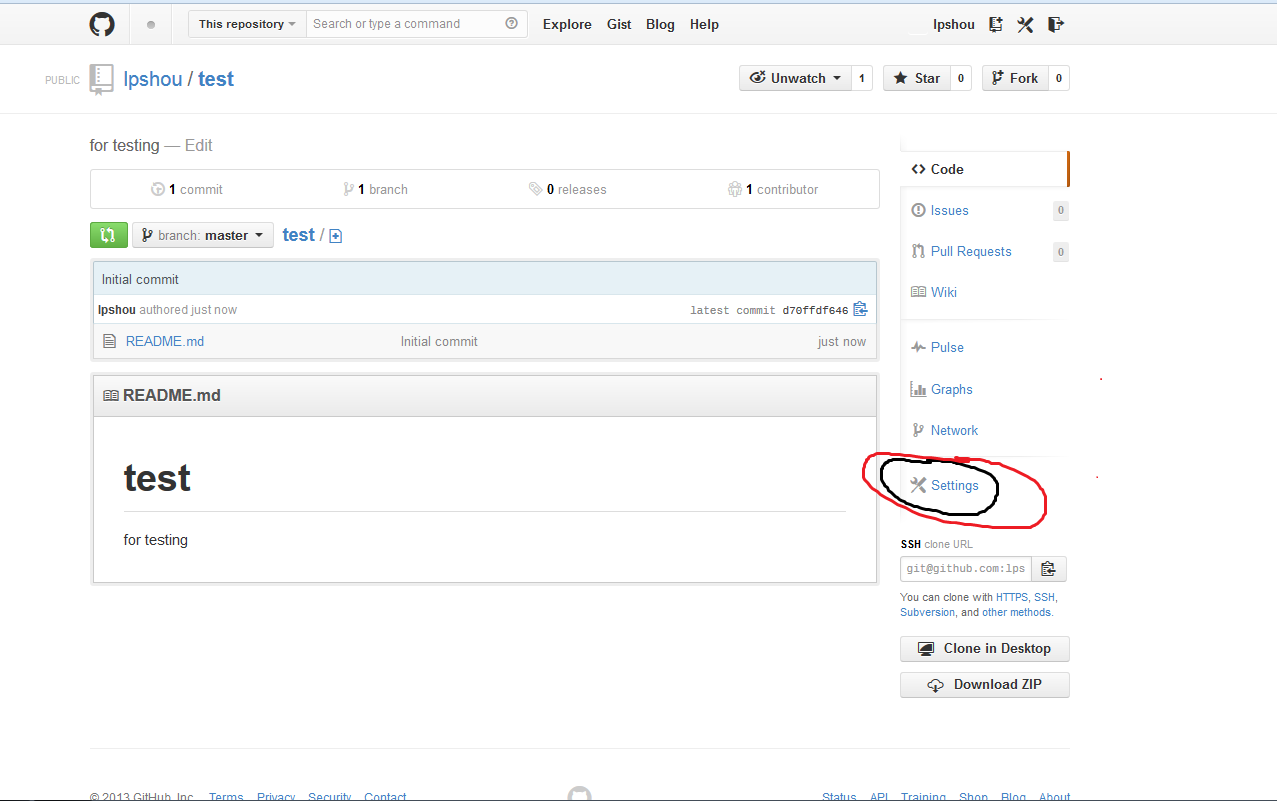
http://www.uml.org.cn/pzgl/201206211.asp

git命令

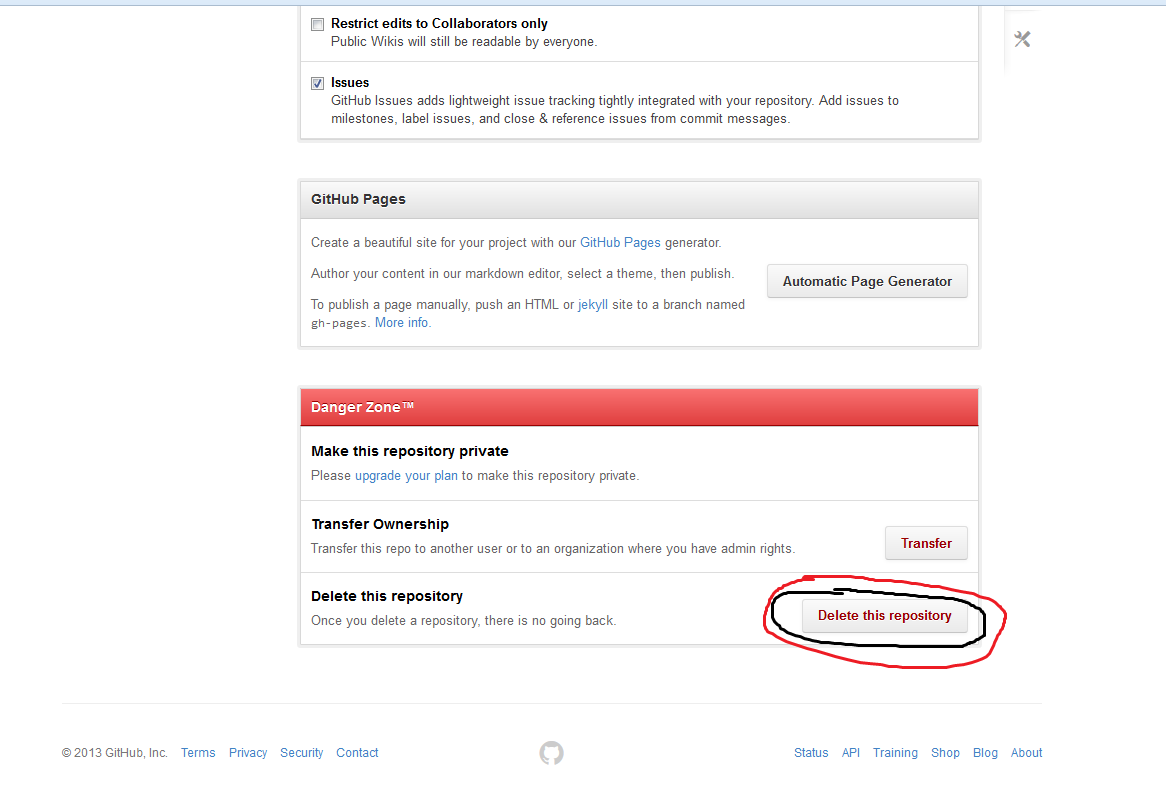
http://www.cnblogs.com/fanfan259/p/4810517.html

https://coding.net/help/doc/git/ssh-key.html#ssh-

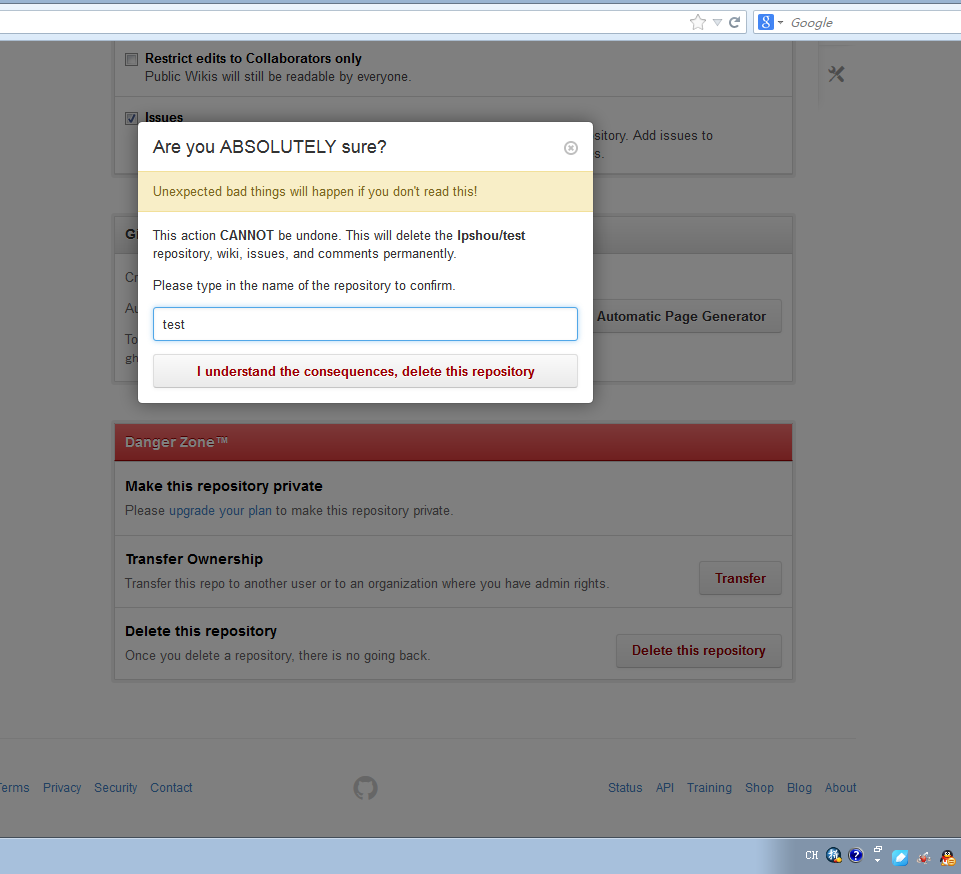
（1） 首先进入相应的repository，然后点击setting



2，点击 delete the repository



（3） 输入要删除的repository名字，即可删除



# [git 代码协同之回车问题 CRLF LF](http://blog.csdn.net/guodengh/article/details/8630888)

格式化

格式化是许多开发人员在协作时，特别是在跨平台情况下，遇到的令人头疼的细小问题。 由于编辑器的不同或者Windows程序员在跨平台项目中的文件行尾加入了回车换行符， 一些细微的空格变化会不经意地进入大家合作的工作或提交的补丁中。不用怕，[**Git**](http://lib.csdn.net/base/git)的一些配置选项会帮助你解决这些问题。

core.autocrlf

查看当前git的配置：**$ git config -list**   简写-l

假如你正在Windows上写程序，又或者你正在和其他人合作，他们在Windows上编程，而你却在其他系统上，在这些情况下，你可能会遇到行尾 结束符问题。 这是因为Windows使用回车和换行两个字符来结束一行，而Mac和[**Linux**](http://lib.csdn.net/base/linux)只使用换行一个字符。 虽然这是小问题，但它会极大地扰乱跨平台协作。

Git可以在你提交时自动地把行结束符CRLF转换成LF，而在签出代码时把LF转换成CRLF。用core.autocrlf来打开此项功能， 如果是在Windows系统上，把它设置成true，这样当签出代码时，LF会被转换成CRLF：

$ git config --global core.autocrlf true

Linux或Mac系统使用LF作为行结束符，因此你不想Git在签出文件时进行自动的转换；当一个以CRLF为行结束符的文件不小心被引入时你肯定想进行修正， 把core.autocrlf设置成input来告诉Git在提交时把CRLF转换成LF，签出时不转换：

$ git config --global core.autocrlf input

这样会在Windows系统上的签出文件中保留CRLF，会在Mac和Linux系统上，包括仓库中保留LF。

如果你是Windows程序员，且正在开发仅运行在Windows上的项目，可以设置false取消此功能，把回车符记录在库中：

**$ git config --global core.autocrlf false**

**----------------------------------------------------------------------**

比如我遇到的问题是“please remove carriage return's from...”，意思是消除回车，估计就是因为跨平台的原因，按上面把autocrlf设置为false就好了

----------------------------------------------------------------------

2015/12/3  再次编辑，对此问题的进一步解释

**CRLF -- Carriage-Return Line-Feed 回车换行**

这两个ACSII字符不会在屏幕有任何输出，但在Windows中广泛使用来标识一行的结束。而在Linux/UNIX系统中只有换行符。

　　CR和LF组合在一起即CRLF命令，它表示键盘上的Enter键(可以用来模拟回车键)。

　　CRLF注入就是说黑客能够将CRLF命令注入到系统中。它不是系统或服务器软件的漏洞，而是网站应用开发时，有些开发者没有意识到此类攻击存在的可能而造成的。

　　就算黑客发现网站存在CRLF注入，他们仍然受到应用结构和这个缺陷的严重程度的限制。

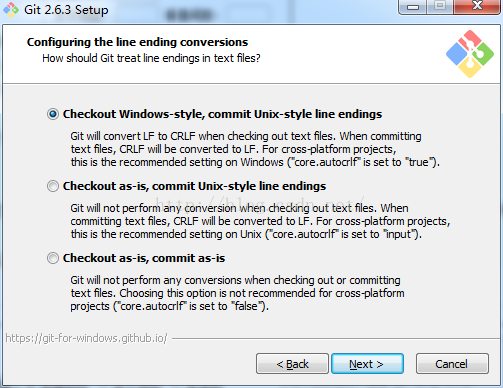
　　对有些站点它将非常严重，而有些站点它只是很小的bug。

　　HTTP Header CRLF Injection

　　许多网络协议，包括HTTP也使用CRLF来表示每一行的结束。这就意味着用户可以通过CRLF注入自定义HTTP header，导致用户可以不经过应用层直接与Server对话。

　　HTTP header的定义就是基于这样的Key: Value的结构，用CRLF命令表示一行的结尾。

**现在的git在安装时会询问换行风格**



**Start a new Git repository for an existing code base**

$ cd /path/to/my/codebase

$ git init **(1)**

$ git add . **(2)**

$ git commit **(3)**

1. Create a /path/to/my/codebase/.git directory.
2. Add all existing files to the index.
3. Record the pristine state as the first commit in the history.

**1.1 创建文件存储GIT用户名和密码**

在%HOME%目录中，一般为C:\users\Administrator，也可以是你自己创建的系统用户名目录，反正都在C:\users\中。文件名为.git-credentials,由于在Window中不允许直接创建以"."开头的文件，所以需要借助git bash进行，打开git bash客户端，进行%HOME%目录，然后用touch创建文件 .git-credentials, 用vim编辑此文件，输入内容格式：

touch .git-credentials

vim .git-credentials

https://{username}:{password}@git.coding.net       //我这里用的是coding.net域名

针对 SSH 协议的方式， 配置SSH 公钥，使用 SSH 协议操作，具体见 [SSH 公钥配置](https://coding.net/help/doc/git/ssh-key.html)

对于 Https 协议

1. 首先在全局配置保存你的密码， ~/.git-credentials （没有就创建）内添加 https://{username}:{passwd}@git.coding.net
2. 然后配置 Git 命令存储认证，

**1.2 添加Git Config 内容**

进入git bash终端， 输入如下命令：

git config --global credential.helper store

执行完后查看%HOME%目录下的.gitconfig文件，会多了一项：

[credential]

helper = store

重新开启git bash会发现git push时不用再输入用户名和密码

常用命令

**查看、添加、提交、删除、找回，重置修改文件**

git help <command> # 显示command的help

git show # 显示某次提交的内容 git show $id

git co -- <file> # 抛弃工作区修改

git co . # 抛弃工作区修改

git add <file> # 将工作文件修改提交到本地暂存区

git add . # 将所有修改过的工作文件提交暂存区

git rm <file> # 从版本库中删除文件

git rm <file> --cached # 从版本库中删除文件，但不删除文件

git reset <file> # 从暂存区恢复到工作文件

git reset -- . # 从暂存区恢复到工作文件

git reset --hard # 恢复最近一次提交过的状态，即放弃上次提交后的所有本次修改

git ci <file> git ci . git ci -a # 将git add, git rm和git ci等操作都合并在一起做

git ci -am "some comments"

git ci --amend # 修改最后一次提交记录

git revert <$id> # 恢复某次提交的状态，恢复动作本身也创建次提交对象

git revert HEAD # 恢复最后一次提交的状态

**查看文件diff**

git diff <file> # 比较当前文件和暂存区文件差异 git diff

git diff <id1><id1><id2> # 比较两次提交之间的差异

git diff <branch1>..<branch2> # 在两个分支之间比较

git diff --staged # 比较暂存区和版本库差异

git diff --cached # 比较暂存区和版本库差异

git diff --stat # 仅仅比较统计信息

**查看提交记录**

git log git log <file> # 查看该文件每次提交记录

git log -p <file> # 查看每次详细修改内容的diff

git log -p -2 # 查看最近两次详细修改内容的diff

git log --stat #查看提交统计信息

**tig**

Mac上可以使用tig代替diff和log，brew install tig

**Git 本地分支管理**

**查看、切换、创建和删除分支**

git br -r # 查看远程分支

git br <new\_branch> # 创建新的分支

git br -v # 查看各个分支最后提交信息

git br --merged # 查看已经被合并到当前分支的分支

git br --no-merged # 查看尚未被合并到当前分支的分支

git co <branch> # 切换到某个分支

git co -b <new\_branch> # 创建新的分支，并且切换过去

git co -b <new\_branch> <branch> # 基于branch创建新的new\_branch

git co $id # 把某次历史提交记录checkout出来，但无分支信息，切换到其他分支会自动删除

git co $id -b <new\_branch> # 把某次历史提交记录checkout出来，创建成一个分支

git br -d <branch> # 删除某个分支

git br -D <branch> # 强制删除某个分支 (未被合并的分支被删除的时候需要强制)

**分支合并和rebase**

git merge <branch> # 将branch分支合并到当前分支

git merge origin/master --no-ff # 不要Fast-Foward合并，这样可以生成merge提交

git rebase master <branch> # 将master rebase到branch，相当于： git co <branch> && git rebase master && git co master && git merge <branch>

**Git补丁管理(方便在多台机器上开发同步时用)**

git diff > ../sync.patch # 生成补丁

git apply ../sync.patch # 打补丁

git apply --check ../sync.patch #测试补丁能否成功

**Git暂存管理**

git stash # 暂存

git stash list # 列所有stash

git stash apply # 恢复暂存的内容

git stash drop # 删除暂存区

**Git远程分支管理**

git pull # 抓取远程仓库所有分支更新并合并到本地

git pull --no-ff # 抓取远程仓库所有分支更新并合并到本地，不要快进合并

git fetch origin # 抓取远程仓库更新

git merge origin/master # 将远程主分支合并到本地当前分支

git co --track origin/branch # 跟踪某个远程分支创建相应的本地分支

git co -b <local\_branch> origin/<remote\_branch> # 基于远程分支创建本地分支，功能同上

git push # push所有分支

git push origin master # 将本地主分支推到远程主分支

git push -u origin master # 将本地主分支推到远程(如无远程主分支则创建，用于初始化远程仓库)

git push origin <local\_branch> # 创建远程分支， origin是远程仓库名

git push origin <local\_branch>:<remote\_branch> # 创建远程分支

git push origin :<remote\_branch> #先删除本地分支(git br -d <branch>)，然后再push删除远程分支

**Git远程仓库管理**

[*GitHub*](http://blog.jobbole.com/6492/)

git remote -v # 查看远程服务器地址和仓库名称

git remote show origin # 查看远程服务器仓库状态

git remote add origin git@ github:robbin/robbin\_site.git # 添加远程仓库地址

git remote set-url origin git@ github.com:robbin/robbin\_site.git # 设置远程仓库地址(用于修改远程仓库地址) git remote rm <repository> # 删除远程仓库

**创建远程仓库**

**Git clone git@git.coding.net:zhangguo5/A3.git**

git clone --bare robbin\_site robbin\_site.git # 用带版本的项目创建纯版本仓库

scp -r my\_project.git git@ git.csdn.net:~ # 将纯仓库上传到服务器上

mkdir robbin\_site.git && cd robbin\_site.git && git --bare init # 在服务器创建纯仓库

git remote add origin git@ github.com:robbin/robbin\_site.git # 设置远程仓库地址

git push -u origin master # 客户端首次提交

git push -u origin develop # 首次将本地develop分支提交到远程develop分支，并且track

git remote set-head origin master # 设置远程仓库的HEAD指向master分支

也可以命令设置跟踪远程库和本地库

git branch --set-upstream master origin/master

git branch --set-upstream develop origin/develop

因为Git是分布式版本控制系统，所以，每个机器都必须自报家门。你的名字和Email地址。在Git Bash中输入以下命令：

1） $ git config --global user.name "名字"

牋牋? 在Git的全局配置中添加一个用户名

2）$ git config --global user.email "邮箱地址"

牋牋? 在Git的全局配置中添加一个邮箱