[在GitHub上分享和展示你的代码](http://serholiu.com/github-share-code)2011-10-14 22:34

最近大爱Web编程，于是寻找各种代码中，然后就发现了[GitHub](https://github.com/)这个网站，如果你知道Google Code，那么你就知道这个GitHub是做什么的了。不过GitHub主要是用作基于Git的分布式版本管理系统的库，可以保存和管理自己的代码，而且主要用作代码的合作开发。不过对于我来说，Git控制系统还比较难以掌握，或者开发小系统还不太用得着，因此我把GitHub当作分享和展示代码的网站。



注册GitHub后你就会有0.3G的免费空间，不过只能创建公开项目，这也满足代码分享的目的，我最喜欢的倒是它的代码展示方式，可以直接浏览你的代码，代码是经过高亮、添加行号处理过的，十分漂亮，体验一流，比如这个[Webpy托管](https://github.com/webpy/webpy)的地方。而作为想要了解你代码的人，可以选择直接在线浏览自己感兴趣的，也可以直接下载压缩包，或者直接使用Git clone到本地。

因为GitHub是基于Git版本控制系统，所以你上传修改代码什么的，都需要使用Git工具。我这里主要是用来分享和展示代码，所以不想在版本控制方面做过多的阐述，下面就简单讲解一下怎么在GitHub上新建一个项目，还有把自己的代码传上去。下面的前提是你已经注册了GitHub和下载安装了Git——[Git下载](http://git-scm.com/download)、[Windows版本下载](http://code.google.com/p/msysgit/downloads/list)。

上传分享代码

1.在GitHub上建立项目

登录GitHub后，你可以在右边靠中那里找到一个按钮“New Repository”，点击过后，填入项目名称、说明和网址过后就可以创建了，然后会出现一个提示页面，记下类似git@github.com:XXX/XXX.git的地址，这个就是你这个项目的地址了。

2.配置Git以及上传代码

安装Git成功后，如果是Windows下，选择Git Bash，在命令行中完成一切，可能开始有点麻烦，不过就那几条命令行，用几次就记住啦。首先初始设置Git：

1 git config --global user.name "Your Real Name"  
2 git config --global user.email you@email.address

然后开始进行最麻烦的一步了，你需要上传文件到GitHub的Git系统上，得需要一个SSH密匙来认证，下面就开始生成密钥和提交密钥。打开Git Bash,创建SSH key:

1 ssh-keygen -C 'your@email.address' -t rsa

然后要你输入SSH密匙的存放位置，可以不管，直接回车使用默认路径。再输入你想要的密码，SSH key就生成了。现在你需要将这个Key提交到GitHub，首先打开Key保存的位置，里面会有三个文件，找到id\_rsa.pub，用文本编辑器打开，复制里面的全部字符。到GitHub，在右上方工具栏里找到Account Settings。在这个页面上有一个SSH Public Keys标签，选择Add another public key。Title可以随便填一个，Key就粘贴刚才的字符，提交。

完成这些工作后，就可以上传自己的代码了。找到自己要分享上传的代码文件夹，右击选择Git Bash，或者在Git Bash中进入这个文件夹。建立一个仓库：

1 git init

选择要添加进仓库的文件：

1 git add .

一般如果你想分享这个文件夹里的所有代码，就在 add后面加“.”，上面的例子就是这样，如果传指定的，只需要把“.”改为文件名即可，现在只是选择了要加入仓库的文件，下面才是添加进入仓库：

1 git commit -m 'Test'

-m后面跟一个参数，表示说明，将代码提交到GitHub后，将会在代码文件信息上显示这个说明，如下图标记的地方。

搞了这么久，现在才开始把本地仓库上传到GitHub了，下面两行命令搞定问题：

首先需要pull下代码，才能push上传成功

（git pull origin master）

继续下面的操作

1. git remote add origin [git@github.com:XXX/XXX.git](mailto:git@github.com:XXX/XXX.git)
2. git pull origin master

3 git push -u origin master

这个git@github.com:XXX/XXX.git就是上面创建项目是生成的地址。现在打开你的项目网址，你就可以发现你的代码已经展示出来了。如果你要更新代码的话，就重复上面的吧。

如果提交了敏感信息，比如代码中设置的自己的密码什么的忘删除就上传上去了怎么办？重新修改过后上传依然有历史记录，而使用Git删除历史记录貌似很麻烦，于是就采用删除项目吧，删除了再重新上传。删除项目需要在GitHub网站上右上方找到admin按钮，进去后右边最下面有个删除的按钮，这样就可以删除了。

**二、 Git 常用命令  
  
1) 远程仓库相关命令** 

检出仓库：        $ git clone git://github.com/jquery/jquery.git

查看远程仓库：$ git remote -v

添加远程仓库：$ git remote add [name] [url]

删除远程仓库：$ git remote rm [name]

修改远程仓库：$ git remote set-url --push [name] [newUrl]

拉取远程仓库：$ git pull [remoteName] [localBranchName]

推送远程仓库：$ git push [remoteName] [localBranchName]

\*如果想把本地的某个分支test提交到远程仓库，并作为远程仓库的master分支，或者作为另外一个名叫test的分支，如下： 

$git push origin test:master         // 提交本地test分支作为远程的master分支

$git push origin test:test              // 提交本地test分支作为远程的test分支

**2）分支(branch)操作相关命令**

查看本地分支：$ git branch

查看远程分支：$ git branch -r

创建本地分支：$ git branch [name] ----注意新分支创建后不会自动切换为当前分支

切换分支：$ git checkout [name]

创建新分支并立即切换到新分支：$ git checkout -b [name]

删除分支：$ git branch -d [name] ---- -d选项只能删除已经参与了合并的分支，对于未有合并的分支是无法删除的。如果想强制删除一个分支，可以使用-D选项

合并分支：$ git merge [name] ----将名称为[name]的分支与当前分支合并

创建远程分支(本地分支push到远程)：$ git push origin [name]

删除远程分支：$ git push origin :heads/[name] 或 $ gitpush origin :[name]

\*创建空的分支：(执行命令之前记得先提交你当前分支的修改，否则会被强制删干净没得后悔) 

$git symbolic-ref HEAD refs/heads/[name]

$rm .git/index

$git clean -fdx

**3）版本(tag)操作相关命令**

查看版本：$ git tag

创建版本：$ git tag [name]

删除版本：$ git tag -d [name]

查看远程版本：$ git tag -r

创建远程版本(本地版本push到远程)：$ git push origin [name]

删除远程版本：$ git push origin :refs/tags/[name]

合并远程仓库的tag到本地：$ git pull origin --tags

上传本地tag到远程仓库：$ git push origin --tags

创建带注释的tag：$ git tag -a [name] -m 'yourMessage'

**4) 子模块(submodule)相关操作命令**

添加子模块：$ git submodule add [url] [path]

如：$git submodule add git://github.com/soberh/ui-libs.git src/main/webapp/ui-libs

初始化子模块：$ git submodule init  ----只在首次检出仓库时运行一次就行

更新子模块：$ git submodule update ----每次更新或切换分支后都需要运行一下

删除子模块：（分4步走哦）

1) $ git rm --cached [path]

2) 编辑“.gitmodules”文件，将子模块的相关配置节点删除掉

3) 编辑“ .git/config”文件，将子模块的相关配置节点删除掉

4) 手动删除子模块残留的目录

**5）忽略一些文件、文件夹不提交**

在仓库根目录下创建名称为“.gitignore”的文件，写入不需要的文件夹名或文件，每个元素占一行即可，如

target

bin

\*.db

**三、 Git 命令详解**

现在我们有了本地和远程的版本库，让我们来试着用用Git的基本命令：

**git pull：**从其他的版本库（既可以是远程的也可以是本地的）将代码更新到本地，例如：'git pull origin master'就是将origin这个版本库的代码更新到本地的master主枝，该功能类似于SVN的**update**

**git add：**是将当前更改或者新增的文件加入到Git的索引中，加入到Git的索引中就表示记入了版本历史中，这也是提交之前所需要执行的一步，例如'git add app/model/user.rb'就会增加app/model/user.rb文件到Git的索引中，该功能类似于SVN的**add**

**git rm：**从当前的工作空间中和索引中删除文件，例如'git rm app/model/user.rb'，该功能类似于SVN的**rm、del**

**git commit：**提交当前工作空间的修改内容，类似于SVN的commit命令，例如'git commit -m story #3, add user model'，提交的时候必须用-m来输入一条提交信息，该功能类似于SVN的**commit**

**git push：**将本地commit的代码更新到远程版本库中，例如'git push origin'就会将本地的代码更新到名为orgin的远程版本库中

**git log：**查看历史日志，该功能类似于SVN的**log**

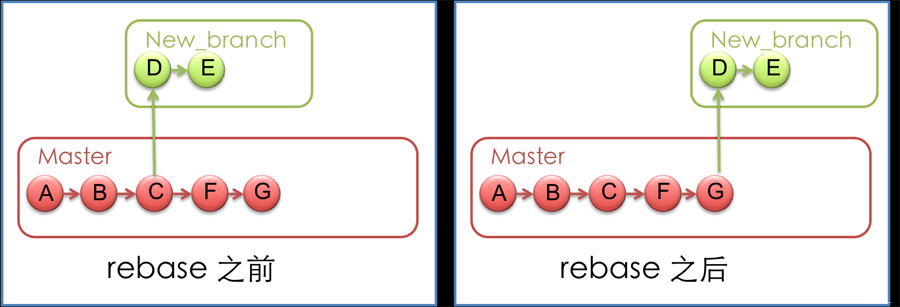
**git revert：**还原一个版本的修改，必须提供一个具体的Git版本号，例如'git revert bbaf6fb5060b4875b18ff9ff637ce118256d6f20'，Git的版本号都是生成的一个哈希值

上面的命令几乎都是每个版本控制工具所公有的，下面就开始尝试一下Git独有的一些命令：

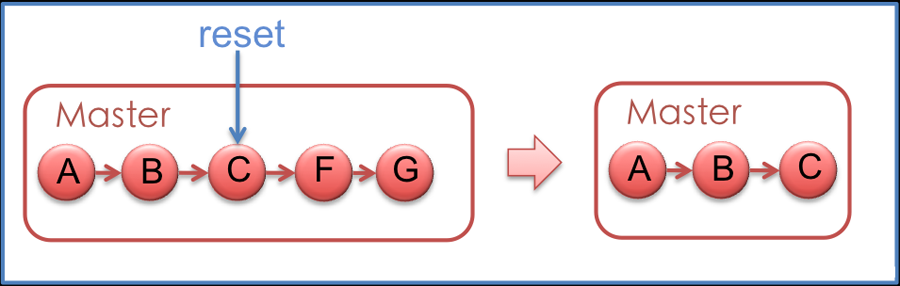
**git branch：**对分支的增、删、查等操作，例如'git branch new\_branch'会从当前的工作版本创建一个叫做new\_branch的新分支，'git branch -D new\_branch'就会强制删除叫做new\_branch的分支，'git branch'就会列出本地所有的分支

**git checkout：**Git的checkout有两个作用，其一是在不同的branch之间进行切换，例如'git checkout new\_branch'就会切换到new\_branch的分支上去；另一个功能是还原代码的作用，例如'git checkout app/model/user.rb'就会将user.rb文件从上一个已提交的版本中更新回来，未提交的内容全部会回滚

**git rebase：**用下面两幅图解释会比较清楚一些，rebase命令执行后，实际上是将分支点从C移到了G，这样分支也就具有了从C到G的功能

[](http://static.oschina.net/uploads/img/201301/06111738_OSuY.png)

**git reset：**将当前的工作目录完全回滚到指定的版本号，假设如下图，我们有A-G五次提交的版本，其中C的版本号是 bbaf6fb5060b4875b18ff9ff637ce118256d6f20(git log可以得到)，我们执行了'git reset bbaf6fb5060b4875b18ff9ff637ce118256d6f20'那么结果就只剩下了A-C三个提交的版本

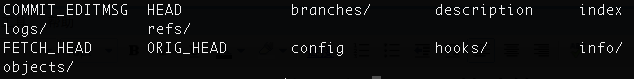


**git stash：**将当前未提交的工作存入Git工作栈中，时机成熟的时候再应用回来，这里暂时提一下这个命令的用法，后面在技巧篇会重点讲解

**git config：**利用这个命令可以新增、更改Git的各种设置，例如'git config branch.master.remote origin'就将master的远程版本库设置为别名叫做origin版本库，后面在技巧篇会利用这个命令个性化设置你的Git，为你打造独一无二的 Git

**git tag：**可以将某个具体的版本打上一个标签，这样你就不需要记忆复杂的版本号哈希值了，例如你可以使用'git tag revert\_version bbaf6fb5060b4875b18ff9ff637ce118256d6f20'来标记这个被你还原的版本，那么以后你想查看该版本时，就可以使用 revert\_version标签名，而不是哈希值了

Git 之所以能够提供方便的本地分支等特性，是与它的文件存储机制有关的。Git存储版本控制信息时使用它自己定义的一套文件系统存储机制，在代码根目录下有一个.git文件夹，会有如下这样的目录结构：

[](http://static.oschina.net/uploads/img/201301/06111739_Cqas.png)

有几个比较重要的文件和目录需要解释一下：HEAD文件存放根节点的信息，其实目录结构就表示一个树型结构，Git采用这种树形结构来存储版本信息，那么HEAD就表示根；refs目录存储了你在当前版本控制目录下的各种不同引用（引用指的是你本地和远程所用到的各个树分支的信息），它有heads、remotes、stash、tags四个子目录，分别存储对不同的根、远程版本库、Git栈和标签的四种引用，你可以通过命令'git show-ref'更清晰地查看引用信息；logs目录根据不同的引用存储了日志信息。因此，Git只需要代码根目录下的这一个.git目录就可以记录完整的版本控制信息，而不是像SVN那样根目录和子目录下都有.svn目录。那么下面就来看一下Git与SVN的区别吧