Git 入门教程

# 设置Git windows 客户端 msysgit

* 使用Git bash生成key
  + ssh-keygen -t rsa -C [nbnbnb.zhang@gmail.com](mailto:nbnbnb.zhang@gmail.com)，此时会出现设置制定生成密钥路径和访问密钥的密码，默认回车即可
* 复制由密钥id\_rsa生成的对应公钥id\_rsa.pub(以记事本打开即可)，添加到 github 的帐号 SSH Keys设置中即可
* 验证SSK
  + ssh -T [git@github.com](mailto:git@github.com)
* 添加全局设置(因为 github每次commit都会记录它们)
  + git config --global user.name "nbnbnb"
  + git config --global user.email "nbnbnb.zhang@gmail.com"

# 获取与提交代码

* 首先初始化本地的代码库
  + git init
    - 将会生成一个.git的隐藏文件夹
* 然后设置远程目标库的别名(可以手动在.git\config文件中配置)
  + git remote add python [git@github.com:nbnbnb/Python.git](mailto:git@github.com:nbnbnb/Python.git)
    - 注意，Python.git是区分大小写的，如果不一致，将会提示无法找到指定的库
    - 此处将远程库的别名设置为python (一般情况下应该用origin)
* 提交代码
  + 首先在本地创建一个hello.py的文件
  + 将代码添加到本地库
    - git add hello.py
    - git commit –m ‘print hello world!’
      * 填写注释的时候，最好用单引号，如果用双引号当注释且最后一个字符是”！”将会出现错误
  + 提交到远程库(github)
    - git push python master

# .gitignore文件

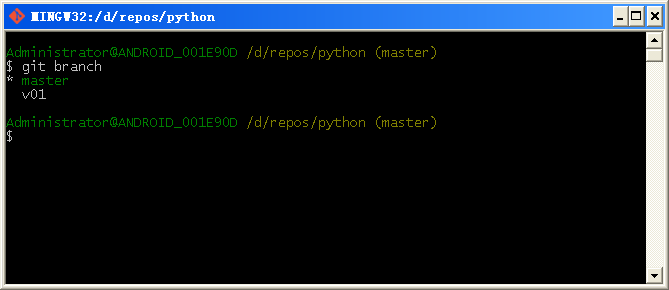
* 告诉git需要忽略的文件，一般我们写完代码执行完后会产生一些临时文件，这些都不是代码文件，是不需要git来管理的,使用git status命令时，可以查看到很多这样的文件，如果用 git add –A 命令就会将它们都添加进去，而手动一个个添加又太麻烦了；这时我们就需要要.gitignore了，默认在github中创建代码库时，可以指定各种语句的.gitignore模版，然后根据这个模版进行自定义就可以了

# tag

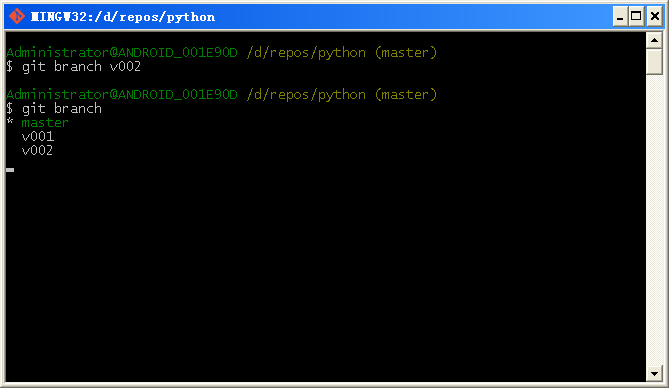
* 我们可以创建一个tag来指向软件开发中的一个关键时期，比如版本号更新的时候可以建立一个 “v2.0”，“v3.0”子类的标签，这样在以后回顾的时候会比较方便；tag的使用很简单，主要有：查看tag，创建tag，验证tag以及共享tag
  + 查看tag
    - 列出所有tag
      * git tag
    - 列出限定的tag
      * git tag –l v1.\*
  + 创建tag
    - 创建轻量级的tag
      * git tag v1.0
    - 创建带注释信息的tag
      * git tag –a v1.0 –m ‘first version’
    - 为指定的提交快照创建tag
      * git tag v3 7b1425709e
  + 删除tag(本地)
    - 指定tag名即可
      * git tag –d v1.0
  + 删除 tag(远端)
    - 使用—delete
      * git push origin –delete tag <tagname>
    - 或者推送一个空tag到远程tag
      * git tag –d <tagname>
      * git push origin :refs/tags/<tagname>
  + 共享tag至服务器端
    - 我们在执行git push 的时候，tag是不会上传到服务器的，比如现在的github，创建tag后git push，在github网页上是看不到tag的，为了共享这些tag，必须这样
      * git push python --tags

# git 命令

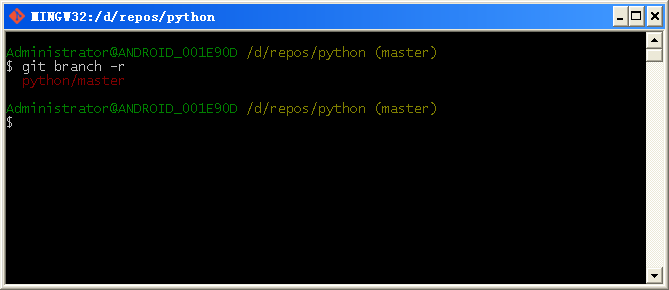
* git brance
  + 不带参数，列出本地已经存在的分支，并且在当前分支前面加上“\*”号



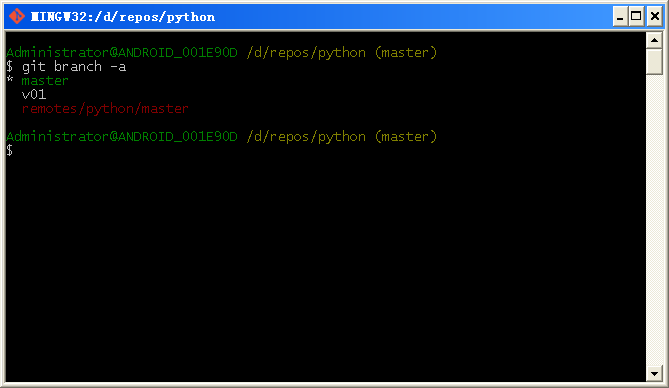
* git branch v002
  + 创建一个新的本地分支v002



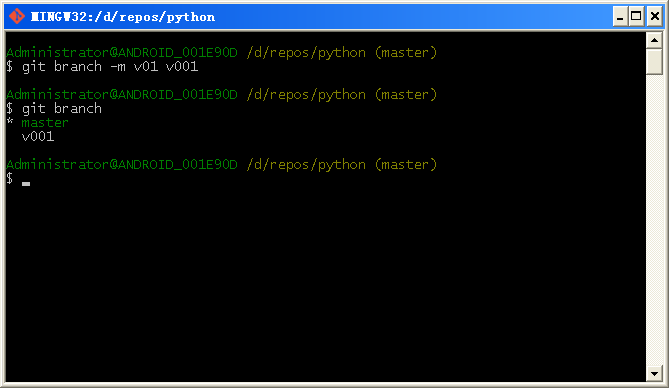
* git branch –r
  + 列出远程分支



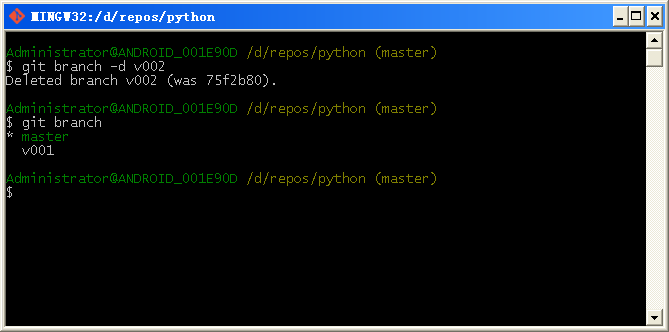
* git branch –a
  + 列出本地分支和远程分支(绿色的代表当前本地分支，白色代表本地分支，红色代表远程分支)



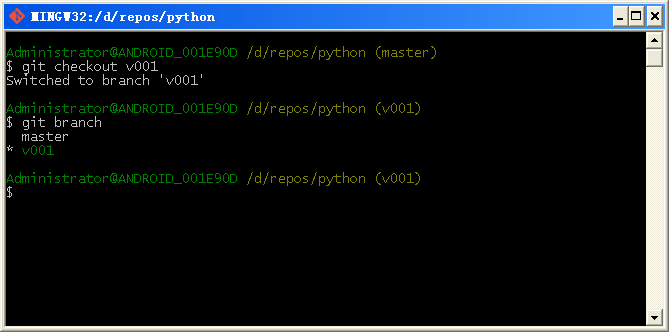
* git branch –m v01 v001
  + 将本地分支v01重命名为v001(如果 v001已存在需要使用-M强制重命名)



* git branch –d v002
  + 删除本地分支 v002



* git checkout v001
  + 切换分支



* git checkout –b b0 master
  + 通过指定master为基点，创建新分支b0,并且立即切换到b0分支上
* git push python b0
  + 将分支b0同步到远程库中
* git reset –hard b0
  + 将当前分支设置为以b0为基点的分支
* git pull python b2
  + 将远程库python中的b2分支合并到当前库的分支
* git rm Git.docx
  + 删除库中的文件
* git checkout –b b1 python/b1(此时可以在断网的情况下创建分支)
  + 从远程库的b1分支创建本地分支b1
* git fetch (Download objects and refs from another repository)
  + 更新所有的远程库分支为最新版本,此命令不会影响任何的本地库,因为它只会将更新后的数据存储在本地的缓存中
* git fetch python +b0\_local:b0\_remote b1\_local:b1\_remote
  + 分别基于远程库python中的b0\_remote和b1\_remote分支，更新或创建本地分支 b0\_local和b1\_local
  + 注意前面的+, 这用于标示是否是使用快速向前的更新操作；由于有些修改不是非线性的，如经常的更改分支基点，此时就需要对获取的数据进行手动的合并以解决冲突和失败
* git reset --merge
  + 用于取消一个有冲突的合并
* 删除远程分支
  + 使用--delete
    - git push origin –-delete b01
  + 或者推送一个空分支到远程扩
    - git push origin :<branchName>

# 配置文件功能

* HEAD文件
  + 用来记住当前的分支
  + 如果将切换分支到标签对象上，则此时HEAD文件显示为40位的编码

# 在Github上同步fork过来的库

* 当将其它项目的库fork到自己的库中时，默认将会添加一个叫做origin的别名，执行刚刚fork过来的库，如果需要保持对原始扩的追踪，需要添加另外一个别名，用来指向原始的库。
* 此处示例使用 upstream别名
  + git remote add upstream <https://github.com/senchalabs/connect>
* 然后获取最新版本到本地缓存(不会修改本地版本数据，仅仅存储在本地缓存)
  + git fetch upstream
* 最后将本地版本与最新库进行合并
  + git merge upstream/master
* 同时也以最原始版本创建一个本地分支
  + git checkout –b newmaster remotes/upstream/master
  + 再将本地分支上传到远程库中
    - git push origin newmaster