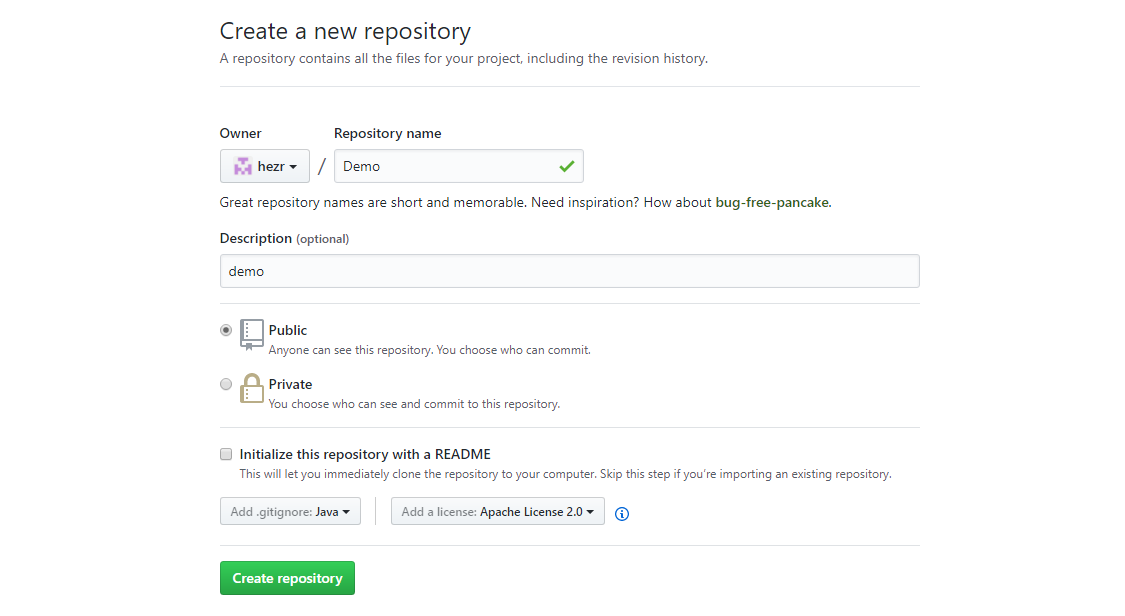
Git使用

1. 个人注册账户及Git配置
2. 在[www.github.com](http://www.github.com)上注册账户
3. 创建仓库，免费用户只能创建公共仓库。



1. 安装git

地址：<https://git-scm.com/downloads>

安装成功后，在桌面会生成Git Bush快捷方式。

1. 配置git
2. 生成SSH key

由于本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以必须要让github仓库认证你SSH key，在此之前，必须要生成SSH key，打开Git Bash，输入以下命令：

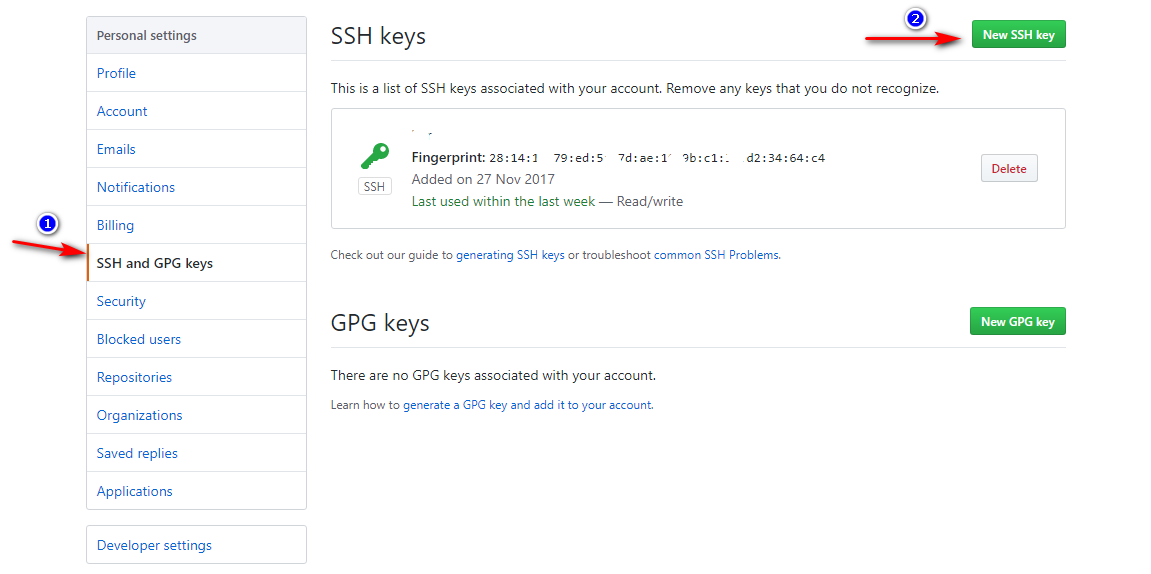
$ ssh -keygen -t rsa -C "youremail@example.com"；

同时在密钥中有一个注释字段，用-C来指定所指定的注释，可以方便用户标识这个密钥，指出密钥的用途或其他有用的信息。所以在这里输入自己的邮箱或者其他都行。

输入完毕后程序同时要求输入一个密语字符串(passphrase)，空表示没有密语。接着会让输入2次口令(password)，空表示没有口令。3次回车即可完成当前步骤，此时c盘/用户/自己的用户名/.ssh/id\_rsa.pub目录下已经生成好了。

1. 登陆github

点击头像->setting中，添加SSH key, title可随便填写。key只要复制id\_rsa.pub中的内容粘贴即可。



1. 测试连接github服务器
2. 设置用户信息

git要求使用者必须提供自己的身份标识，为此我们需要在git bash中执行以下命令（当然不设置也行，每次提交都是unknown... macOS 貌似会提交系统用户名）：

在Git Bash中输入命令:

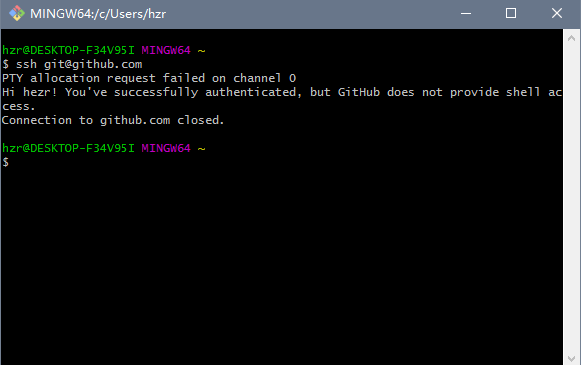
$ git config --global user.name "hanzichi" //给自己起个用户名

$ git config --global user.email "abc@gmail.com" //填写自己的邮箱

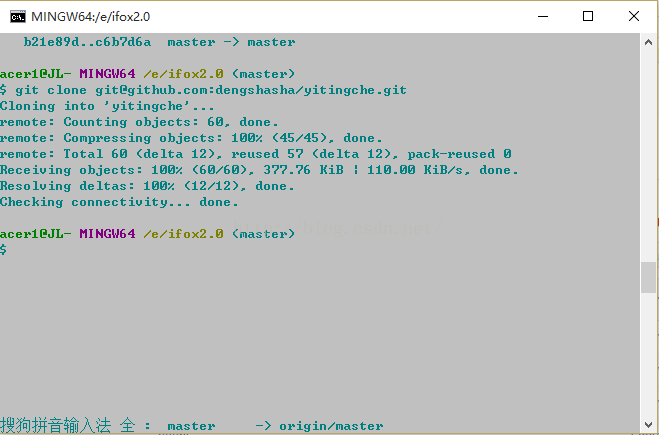
(2) 测试连接

ssh -T [git@github.com](mailto:git@github.com)

提示像这样：Hi hanzichi! You've successfully authenticated, but GitHub does not provide shell access. 那就说明连接成功了。



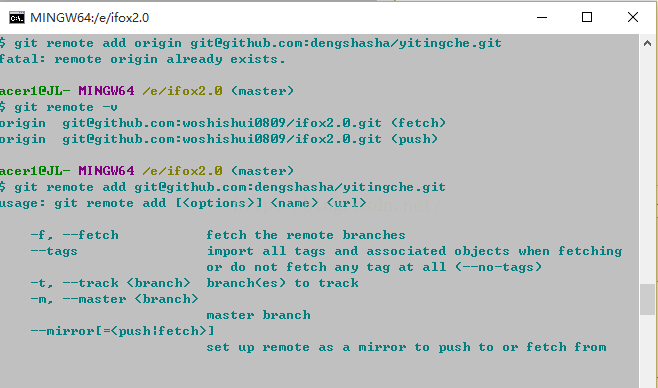
1. 使用github和git进行团队合作开发
2. 项目都上传到项目经理的github下，项目经理需要做以下事情：
3. 在github上创建一个组织，new organization，在这里邀请成员，注意（注意team的成员是要先成为organization的成员）。
4. 在该组织下创建一个team，然后邀请你的团队成员。
5. 新建一个repository。
6. 很重要的一步哈，在github上，如果你的项是public，那么任何人都可以clone你的项目到他的电脑，但是别人不能push自己的代码到你的仓库。你的队友肯定是要能修改这个工程的，我之前一直不知道怎么把队友和自己的仓库连接起来，很多人说的太模糊了，搞的我研究了半天。这里就要用SSH Keys了，队友需要在自己的电脑里生成一个ssh key，将公钥（id\_rsa.pub）里的内容拷出来owner，让owner新建一个ssh key保存你的公钥，队友自己不要新建一个ssh key去保存这个公钥。这一步做完了以后，队友就可以把自己的代码push到你们团队的这个repository里了。到这里基本连接工作就做完了哈。
7. 团队成员与远程仓库连接
8. 首先要在本机克隆这个项目（没有连接也是可以的）：git clone仓库的地址。



然后初始化项目：git init

1. 将你的本地仓库与远程仓库连起来

命令：git remote add 远程仓库地址

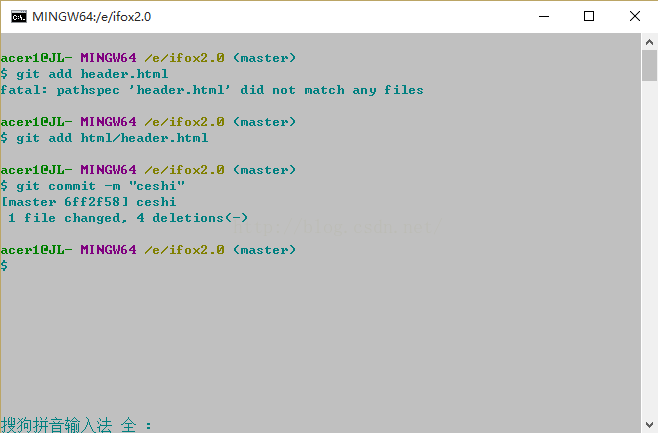


然后可以使用git remote -v查看你连接上的仓库。

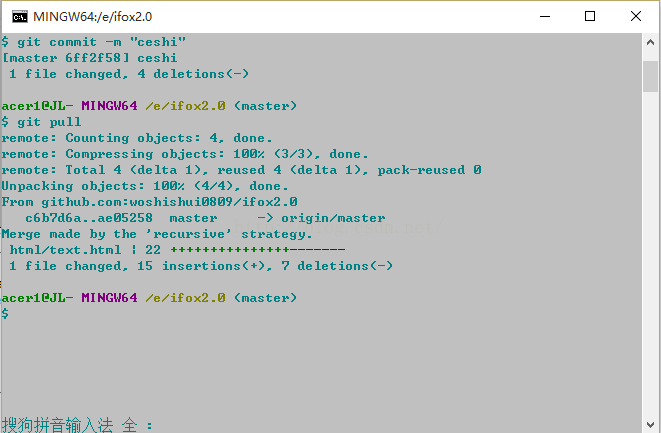
1. 连接仓库成功后，就可以添加要上传的文件了 git add . (.表示添加所有文件，也可以直接跟文件名)，一般这步不会出错。

注意如果是提交某个文件，格式跟你编写html插入某个文件或图片是一样的，比如你想添加html文件夹下的test.html文件，则命令：git add html/test.html

1. 本地提交 git commit -m "注释"，这一步好像会经常出错,（我现在提交没有问题，一般提交不成功可能是你提交的文件没有更改过)

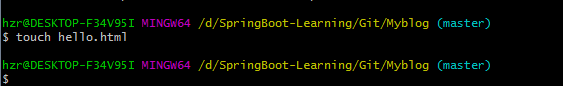


1. 然后pull一下，这一步主要是确认你本地有没有更新成最新的，如果不成功的话就很有可能是没有连接上仓库。git pull (这里有时候要跟远程仓库的地址，有时候直接git pull 也可以)



1. 现在可以上传你的代码了git push （这里有时候也可以不用仓库地址）
2. 本地项目上传至github
3. 首先要在github中新建一个仓库， （团队项目需要在organization新建仓库，然后在Team中添加），仓库名和项目名统一。
4. 本地创建.ignore文件，可以用Git Bash进入到项目下，用命令新建

touch filename.格式



在IDEA中，可以在Teminal中敲命令。 .ignore可以过滤不需要提交的文件。

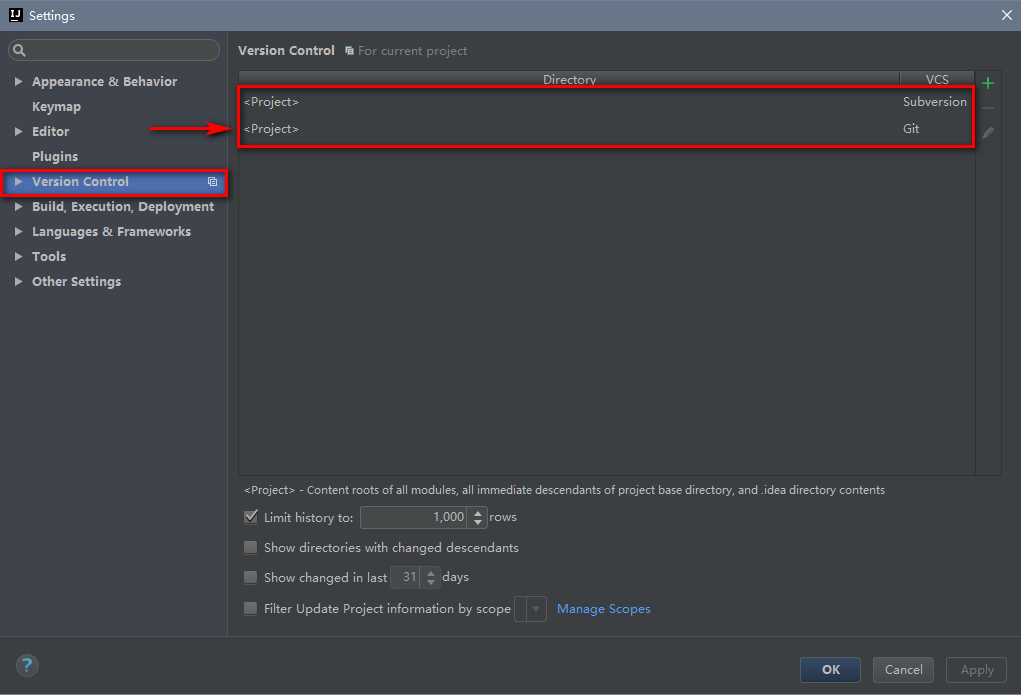
1. 创建README.md文件。
2. 添加需要提交的文件，git add 文件名或者git add .(所有)
3. 提交文件git commit -m ‘注释’
4. 上传到github.

$ git push origin master

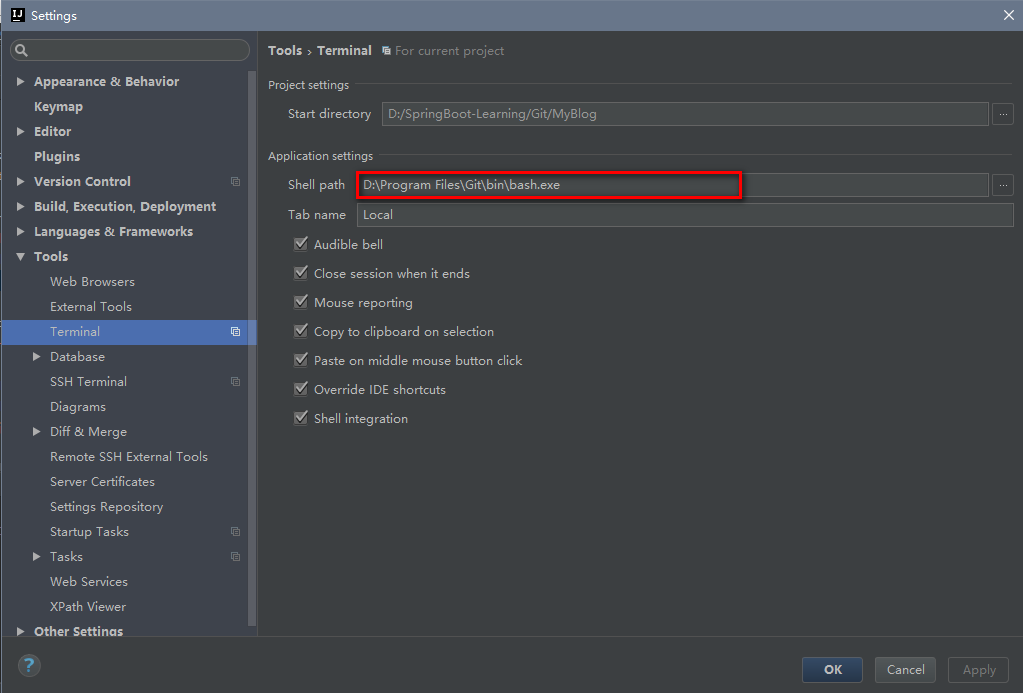
首次提交，先git pull下，修改代码后，使用git status可以查看文件的差别。

1. IDEA使用github
2. 设置git

file -> setting -> Version Control,添加git选项，这一步的目的是在右键项目的时候，会出现git这个选项。



在tools中设置git路径，这一步的目的是可以再teminal输入git命令，不需要git bash



IDEA配置好以后，使用方法与SVN大致相同。

# 附录

Git命令 （#后面是注释）

1. 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 $id  git checkout -- <file> # 抛弃工作区修改  git checkout . # 抛弃工作区修改  git add <file> # 将工作文件修改提交到本地暂存区  git add . # 将所有修改过的工作文件提交暂存区  git rm <file> # 从版本库中删除文件  git rm <file> --cached # 从版本库中删除文件，但不删除文件  git reset <file> # 从暂存区恢复到工作文件  git reset -- . # 从暂存区恢复到工作文件  git reset --hard # 恢复最近一次提交过的状态，即放弃上次提交后的所有本次修改  git commit <file> 提交指定文件  git commit . 提交所有修改文件  git commit -a # -a是代表add，把所有的change加到git index里然后再commit  　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　git commit -am "some comments" #将git add, git rm和git commit等操作都合并在一起做.  git commit --amend # 修改最后一次提交记录  git revert <$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 |

1. 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><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 |

1. 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 |

Git本地分支管理

1. 查看、切换、创建和删除分支

|  |
| --- |
| git branch -r # 查看远程分支  git branch <new\_branch> # 创建新的分支  git branch -v # 查看各个分支最后提交信息  git branch --merged # 查看已经被合并到当前分支的分支  git branch --no-merged # 查看尚未被合并到当前分支的分支  git checkout <branch> # 切换到某个分支  git checkout -b <new\_branch> # 创建新的分支，并且切换过去  git checkout -b <new\_branch> <branch> # 基于branch创建新的new\_branch  git checkout $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 checkout $id -b <new\_branch> # 把某次历史提交记录checkout出来，创建成一个分支  git branch -d <branch> # 删除某个分支  git branch -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) |

1. 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> |

1. 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 |

1. 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 |

1. 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 checkout --track origin/branch # 跟踪某个远程分支创建相应的本地分支  git checkout -b <local\_branch> origin/<remote\_branch> # 基于远程分支创建本地分支，功能同上 |

1. 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 |

1. Git远程仓库管理

|  |
| --- |
| 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 |

1. 创建远程仓库

|  |
| --- |
| 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 |

1. 代码回退

|  |
| --- |
| #回退所有内容到上一个版本  git reset HEAD^  #回退a.py这个文件的版本到上一个版本  git reset HEAD^ a.py  #向前回退到第3个版本  git reset –soft HEAD~3  #将本地的状态回退到和远程的一样  git reset –hard origin/master  #回退到某个版本  git reset 057d  #回退到上一次提交的状态，按照某一次的commit完全反向的进行一次commit  git revert HEAD |

更多信息：http://blog.csdn.net/zhang19871201/article/details/47748923