本文件也已添加到github下，持续更新

安装

Git-2.6.4-64-bit.exe

Gvim74.exe 文本编辑器

vimcdoc-1.9.0-setup.exe文本编辑器中文说明文档

直接按照默认设置安装

初次运行 Git的配置与操作

教研室公用github帐号：

（邮箱[UESTC1430@163.com](mailto:UESTC1430@163.com)

邮箱密码14301430

github帐号名UESTC1430

github密码1430uestc1430

）

用共同工作帐号对项目进行操作。

用户信息设置

$ git config --global user.name "UESTC1430" #设置用户名

$ git config --global user.email [UESTC1430@163.com](mailto:UESTC1430@163.com)#设置邮箱地址(建议用注册giuhub的邮箱)

查看下面SSH访问GitHub方法关联电脑与GitHub帐号

$ git config --global core.editor gvim 设定你喜欢的文本编辑软件（gvim）

$ git config --global merge.tool vimdiff 设置默认的合并工具

$ git config --global http.postBuffer 524288000  上传上限设为500M

$ git config --list 查看自己的环境配置清单

$ git remote add origin [git@github.com:UESTC1430/AOID.git](mailto:git@github.com:UESTC1430/AOID.git) //在本地关联1430的远程库

远程库origin已经保存了AOID的最新工作版本，现在clone到本地

在保存目标目录下执行：$ git clone git@github.com:UESTC1430/AOID.git

这样AOID项目就从GitHub下载到本地了



基础操作

取得项目的git仓库repository

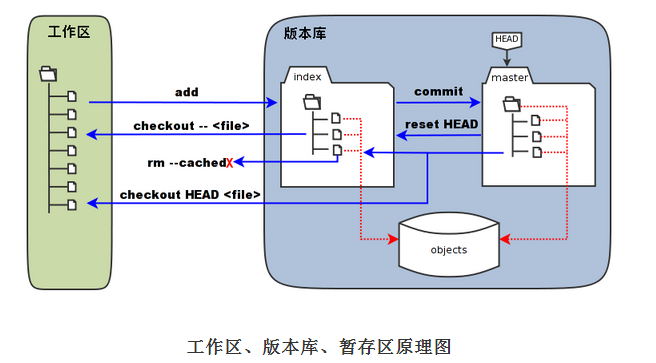
有两种取得 Git 项目仓库的方法。第一种是在现存的目录下，通过导入所有文件来创建新的 Git 仓库。第二种是从已有的 Git 仓库克隆出一个新的镜像仓库来。

1从当前目录初始化

只需到此项目所在的目录，执行：$ git init

2从现有仓库clone

在当前目录下，执行：$ git clone git://github.com/schacon/grit.git



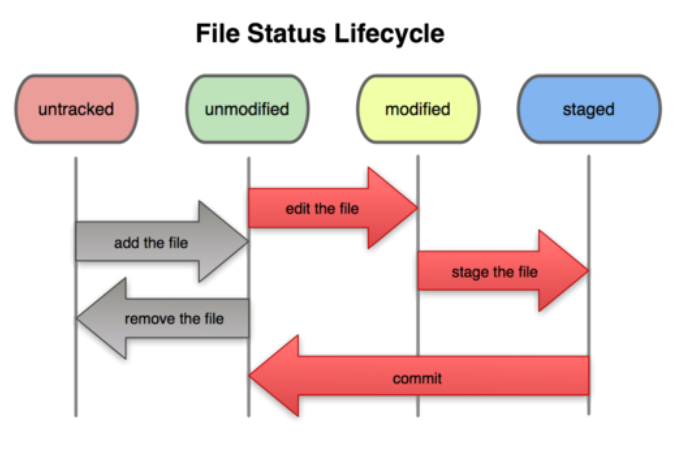
你的本地仓库由 git 维护的三棵“树”组成。

第一个是你的 工作目录，它持有实际文件；

第二个是 暂存区（Index），它像个缓存区域，临时保存你的改动；

最后是 HEAD，它指向你最后一次提交的结果。

仓库状态



查询repository的状态：git status

1跟踪新文件 $ git add \*.c （untracked——）unmodified）

2修改跟踪的文件（unmodified——）modified）

3暂存已修改文件（modified——）staged）

要暂存这次更新，需要运行 git add 命令（这是个多功能命令，根据目标文件的状态不同，此命令的效果也不同：可以用它开始跟踪新文件，或者把已跟踪的文件放到暂存区，还能用于合并时把有冲突的文件标记为已解决状态等）。

4忽略某些文件

一般需要忽略自动生成文件

创建名为.gitigoire的文件，列出要忽略的文件模式，例如：

$ cat .gitignore

\*.[oa]  // Git 忽略所有以 .o 或 .a 结尾的文件

\*~ //Git 忽略所有以波浪符（ ~ ）结尾的文件

5查看暂存前后的变化

git diff此命令比较的是工作目录中当前文件和暂存区域快照之间的差异,也就是修改之后还没有暂存起来的变化内容.

6 提交更新（staged——）unmodified）

暂存区域已经准备妥当再运行提交命令：

$ git commit（这种方式会启动文本编辑器以便输入本次提交的说明。）

$ git commit -m "Story 182: Fix benchmarks for speed" 也可以使用 -m 参数后跟提交说明的方式，在一行命令中提交更新

7跳过使用暂存区域

$ git commit -a -m 'added new benchmarks'

Git 就会自动把所有已经跟踪过的文件暂存起来一并提交，从而跳过 git add 步骤

8移除文件

在工作目录移除文件，然后执行命令：

$ git rm grit.gemspec

9移动文件（文件改名）

$ git mv file\_from file\_to

10查看提交历史

$git log

11 撤消操作

$ git commit –amend //增补提交. 会使用与当前提交节点相同的父节点进行一次新的提交,旧的提交将会被取消

$ git reset HEAD benchmarks.rb //取消已经暂存的文件benchmarks.rb

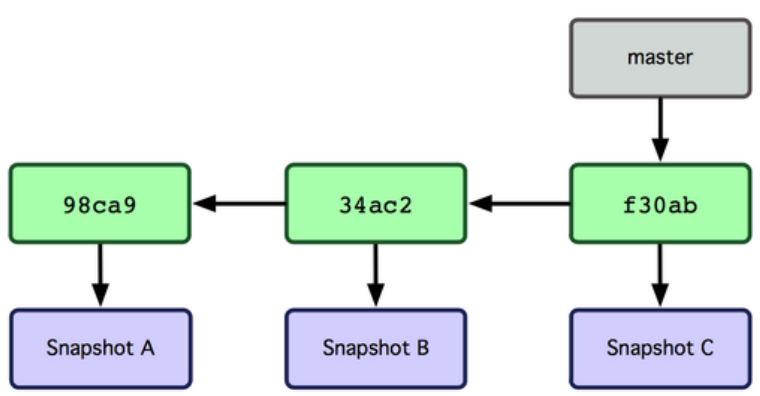
$ git checkout -- benchmarks.rb //取消对文件的修改，回到之前的状态（也就是修改之

前的版本）

这条操作不保留修改内容，如果只是想回退版本，同时保留刚才的修改以便将来继续工作，可以用stashing 和分支来处理，应该会更好些。

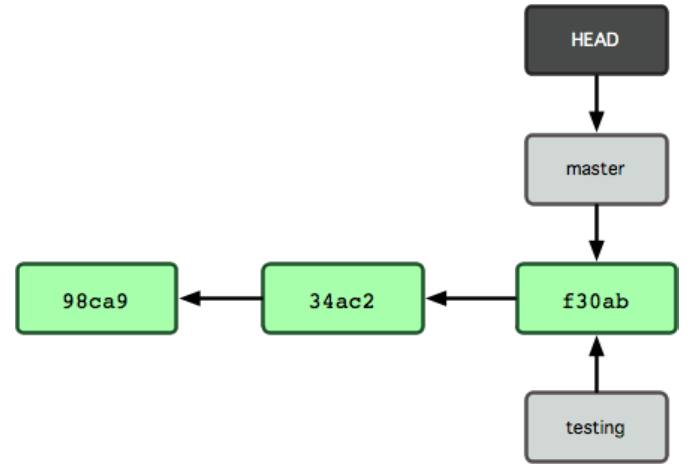
GIT分支

Git 中的分支，其实本质上仅仅是个指向 commit 对象的可变指针。Git 会使用 master 作为分支的默认名字。在若干次提交后，你其实已经有了一个指向最后一次提交对象的 master 分支，它在每次提交的时候都会自动向前移动。

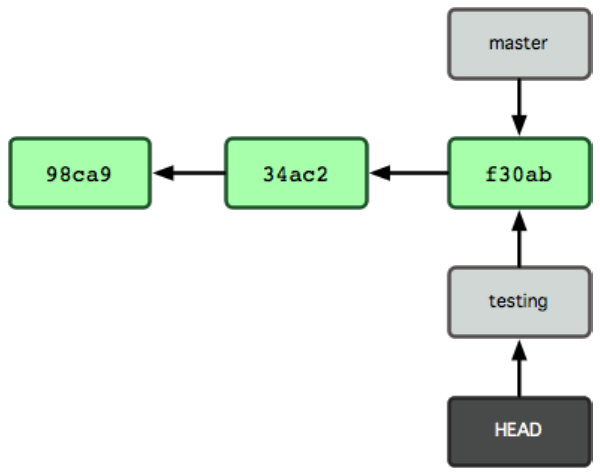


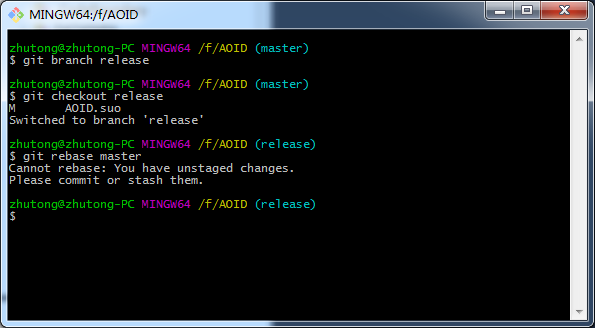
$ git branch testing //在当前 commit 对象上新建一个分支指针。

Git 中，HEAD是一个指向你正在工作中的本地分支的指针。运行 git branch 命令，仅仅是建立了一个新的分支，但不会自动切换到这个分支中去，所以在这个例子中，我们依然还在 master 分支里工作。

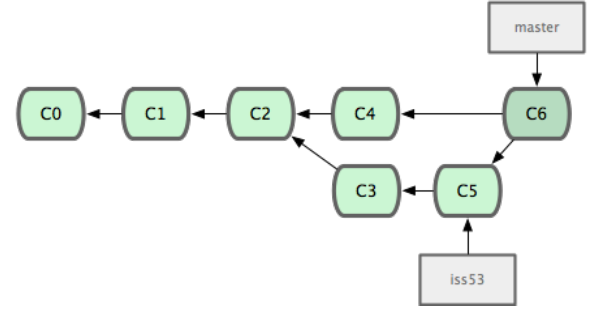


$ git checkout testing //转换HEAD指向新建的 testing 分支





我们可以不同分支里反复切换，并在时机成熟时把它们合并到一起。而所有这些工作，仅仅需要 branch 和 checkout 这两条命令就可以完成。

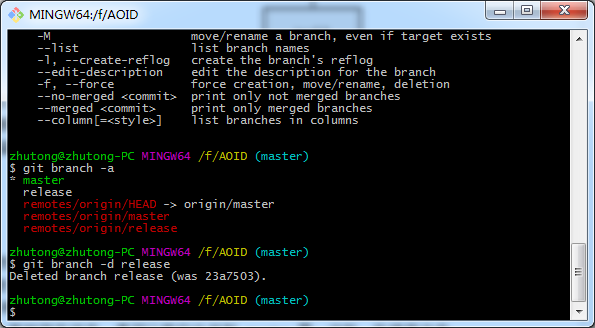


$ git checkout master

$ git merge iss53 //合并分支iss53到master（当前分支）自动创建一个指向它的commit C6

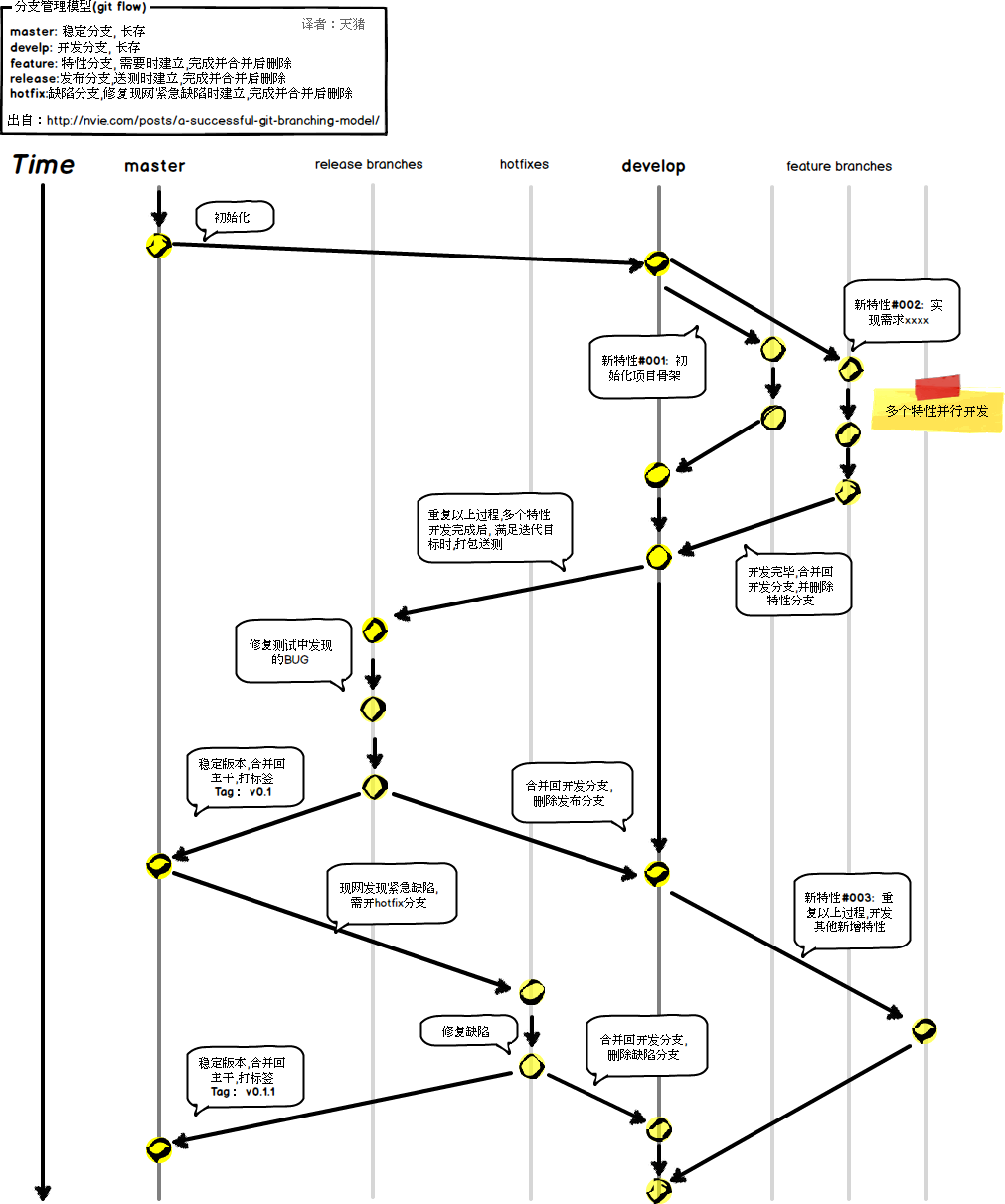
$ git branch -d iss53 //删除分支iss53

$ git branch –a //查看分支

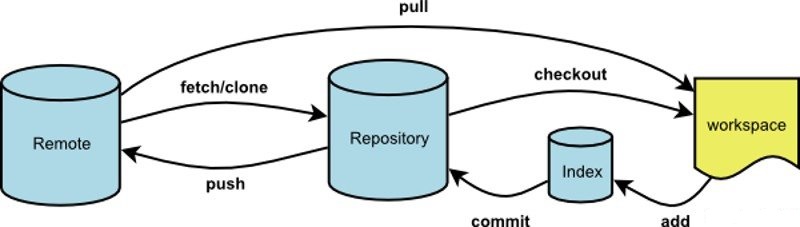


Git 流

许多使用 Git 的开发者都喜欢以这种方式来开展工作，比如仅在 master 分支中保留完全稳定的代码，即已经发布或即将发布的代码。与此同时，他们还有一个名为 develop 或 next 的平行分支，专门用于后续的开发，或仅用于稳定性测试 —— 当然并不是说一定要绝对稳定，不过一旦进入某种稳定状态，便可以把它合并到 master 里。这样，在确保这些已完成的特性分支（短期分支feature，如前例的 iss53 ）能够通过所有测试，并且不会引入更多错误之后，就可以并到主干分支中，等待下一次的发布。



服务器上的 Git （常用操作）



$ git remote add origin [git@github.com:UESTC1430/AOID.git](mailto:git@github.com:UESTC1430/AOID.git) //在本地关联1430的远程库

//远程库的默认名字就是origin  
 //在设置别名后[git@github.com](mailto:git@github.com)应改为别名github2

$ git remote rm origin //取消本地关联远程库origin

$ git push origin master //执行如下命令以将这些改动提交到远端仓库

//可以把 master 换成你想要推送的任何分支

//命令中origin代表的真实地址在.git/config中定义

$ git push <远程主机名> <本地分支名>:<远程分支名> //上边命令的完整版

$ git push origin :serverfix // 特殊方式删除远程分支（本地分支处用空格表示，

等于是在说“在这里提取空白然后把它变成[远程分支]”）

$ git push --all origin //第一次操作可以使用此指令推送所有分支

$ git clone [git@github.com:UESTC1430/AOID.git](mailto:git@github.com:UESTC1430/AOID.git) //从远程主机克隆一个版本库

$ git fetch origin master //取回origin主机的master分支

所取回的更新，在本地主机上要用”远程主机名/分支名”的形式读取。比如origin主机的master，就要用origin/master读取。

git branch命令的-r选项，可以用来查看远程分支，-a选项查看所有分支。

git pull命令的作用是，取回远程主机某个分支的更新，再与本地的指定分支合并。

$ git pull origin next:master //取回origin主机的next分支，与本地的master分支合并

$ git remote –v //查看当前的远程库

$ git remote add pb git://github.com/paulboone/ticgit.git //添加远程仓库

$ git remote add [shortname] [url] //添加远程仓库 可以用字串pb指代对应的仓库地址了

$ git fetch [remote-name] //此命令会到远程仓库中拉取所有你本地仓库中还没有的数据

#$ git push origin master //推送数据到远程仓库

$ git remote show origin //查看远程库的信息

$ git remote rename pb paul //修改某个远程仓库的简短名称

$ git remote rm paul //移除远端仓库

[git fetch和git pull之间的区别](http://blog.csdn.net/a19881029/article/details/42245955)



git fetch和git pull都可以用来更新本地库，git pull的问题是它把过程的细节都隐藏了起来，以至于你不用去了解git中各种类型分支的区别和使用方法。当然，多数时候这是没问题的，但一旦代码有问题，你很难找到出错的地方。

不要用git pull，用git fetch和git merge代替它。

1.git fetch git fetch只会将本地库所关联的远程库的commit id更新至最新

一旦远程主机的版本库有了更新（Git术语叫做commit），需要将这些更新取回本地，这时就要用到git fetch命令。

$ git fetch <远程主机名>

上面命令将某个远程主机的更新，全部取回本地。

git fetch命令通常用来查看其他人的进程，因为它取回的代码对你本地的开发代码没有影响。

默认情况下，git fetch取回所有分支（branch）的更新。如果只想取回特定分支的更新，可以指定分支名。

$ git fetch <远程主机名> <分支名>

比如，取回origin主机的master分支。

$ git fetch origin master

所取回的更新，在本地主机上要用"远程主机名/分支名"的形式读取。比如origin主机的master，就要用origin/master读取。

git branch命令的-r选项，可以用来查看远程分支，-a选项查看所有分支。

$ git branch -r

origin/master

$ git branch -a

\* master

remotes/origin/master

上面命令表示，本地主机的当前分支是master，远程分支是origin/master。

取回远程主机的更新以后，可以在它的基础上，使用git checkout命令创建一个新的分支。

$ git checkout -b newBrach origin/master

上面命令表示，在origin/master的基础上，创建一个新分支。

此外，也可以使用git merge命令或者git rebase命令，在本地分支上合并远程分支。

$ git merge origin/master

# 或者

$ git rebase origin/master

上面命令表示在当前分支上，合并origin/master。

2.git pull git pull会将本地库更新至远程库的最新状态

git pull命令的作用是，取回远程主机某个分支的更新，再与本地的指定分支合并。它的完整格式稍稍有点复杂。

$ git pull <远程主机名> <远程分支名>:<本地分支名>

比如，取回origin主机的next分支，与本地的master分支合并，需要写成下面这样。

$ git pull origin next:master

如果远程分支是与当前分支合并，则冒号后面的部分可以省略。

$ git pull origin next

上面命令表示，取回origin/next分支，再与当前分支合并。实质上，这等同于先做git fetch，再做git merge。

$ git fetch origin

$ git merge origin/next

在某些场合，Git会自动在本地分支与远程分支之间，建立一种追踪关系（tracking）。比如，在git clone的时候，所有本地分支默认与远程主机的同名分支，建立追踪关系，也就是说，本地的master分支自动"追踪"origin/master分支。

Git也允许手动建立追踪关系。

git branch --set-upstream master origin/next

上面命令指定master分支追踪origin/next分支。

如果当前分支与远程分支存在追踪关系，git pull就可以省略远程分支名。

$ git pull origin

上面命令表示，本地的当前分支自动与对应的origin主机"追踪分支"（remote-tracking branch）进行合并。

如果当前分支只有一个追踪分支，连远程主机名都可以省略。

$ git pull

上面命令表示，当前分支自动与唯一一个追踪分支进行合并。

如果合并需要采用rebase模式，可以使用--rebase选项。

$ git pull --rebase <远程主机名> <远程分支名>:<本地分支名>

如果远程主机删除了某个分支，默认情况下，git pull 不会在拉取远程分支的时候，删除对应的本地分支。这是为了防止，由于其他人操作了远程主机，导致git pull不知不觉删除了本地分支。

但是，你可以改变这个行为，加上参数 -p 就会在本地删除远程已经删除的分支。

$ git pull -p

# 等同于下面的命令

$ git fetch --prune origin

$ git fetch -p

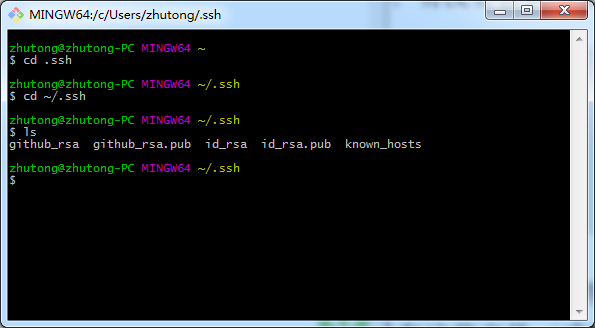
Git配置SSH访问GitHub

1.检查本机是否有ssh key设置

$ cd ~/.ssh 或cd .ssh

如果没有则提示： No such file or directory

如果有则进入~/.ssh路径下（ls查看当前路径文件，rm \* 删除所有文件）



关键是看有没有用文件名和文件名.pub 来命名的一对文件，这个文件名通常是id\_dsa或者 id\_rsa。.pub文件是公钥，另一个文件是密钥。

2.使用Git Bash生成新的ssh key

$ cd ~  #保证当前路径在”~”下

$ ssh-keygen -t rsa -C "xxxxxx@yy.com"  #建议填写自己真实有效的邮箱地址

Generating public/private rsa key pair.

Enter file in which to save the key (/c/Users/xxxx\_000/.ssh/id\_rsa):   #不填直接回车

Enter passphrase (empty for no passphrase):   #输入密码（可以为空）

Enter same passphrase again:   #再次确认密码（可以为空）

Your identification has been saved in /c/Users/xxxx\_000/.ssh/id\_rsa.   #生成的密钥

Your public key has been saved in /c/Users/xxxx\_000/.ssh/id\_rsa.pub.  #生成的公钥

The key fingerprint is:

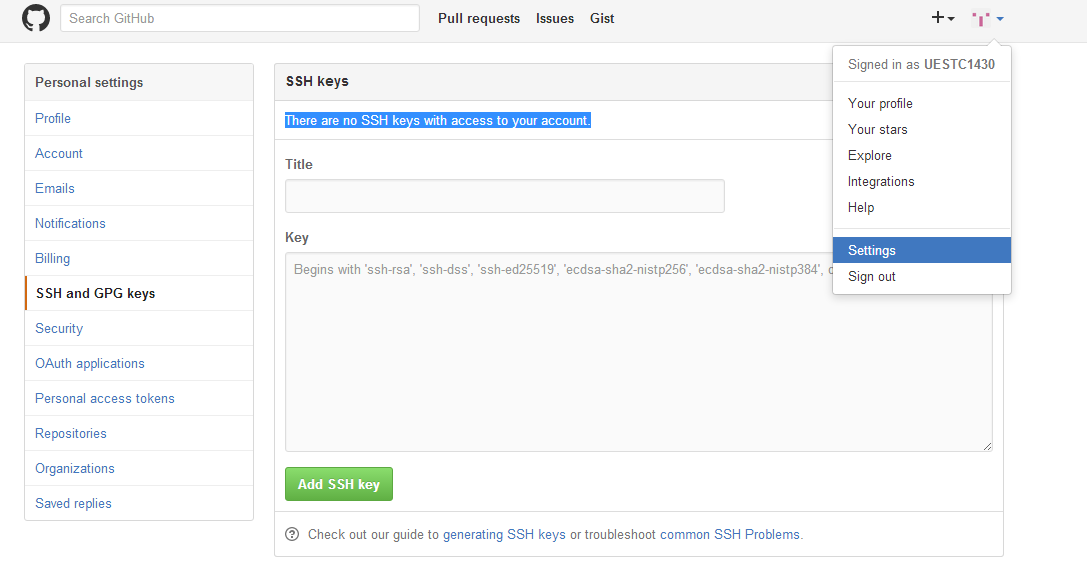
e3:51:33:xx:xx:xx:xx:xxx:61:28:83:e2:81 xxxxxx@yy.com

\*本机已完成ssh key设置，其存放路径为：c:/Users/xxxx\_000/.ssh/下。

*注释：可生成ssh key自定义名称的密钥，默认id\_rsa。*

3.添加ssh key到GitHub

登录https://github.com/UESTC1430/AOID （用户名密码见P1）

打开对应位置，在title下输入连接者信息或身份

复制id\_rsa.pub的公钥内容。

1) 进入c:/Users/xxxx\_000/.ssh/目录下，打开id\_rsa.pub文件，全选复制公钥内容。

2) Title自定义，将公钥粘贴到GitHub中Add an SSH key的key输入框，最后“Add Key”。

附录1 多github帐号的SSH key切换

如果个人和团队项目分别有github帐号，而且在一台电脑下操作，则可以用下边配置：

1.新建user2的SSH key

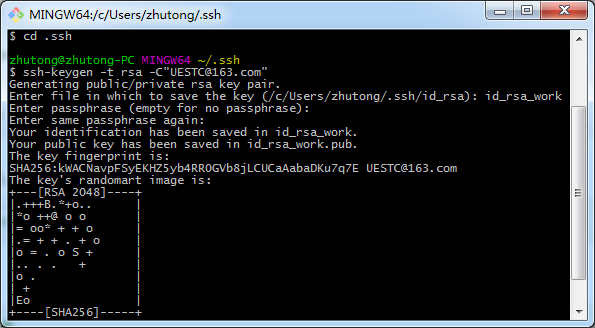
#新建SSH key：

$ cd ~/.ssh # 切换到C:\Users\Administrator\.ssh

ssh-keygen -t rsa -C "mywork@email.com" # 新建工作的SSH key

# 设置名称为id\_rsa\_work

Enter file in which to save the key (/c/Users/ zhutong /.ssh/id\_rsa): id\_rsa\_work



2.新密钥添加到SSH angent中

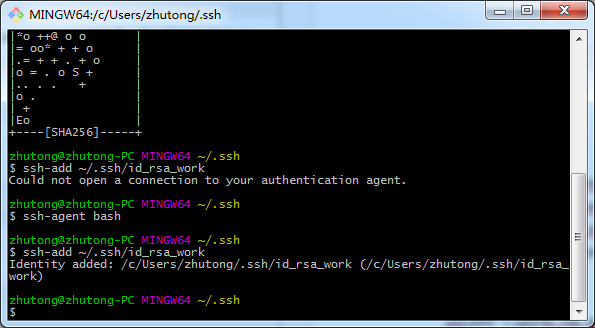
因为默认只读取id\_rsa，为了让SSH识别新的私钥，需将其添加到SSH agent中：

ssh-add ~/.ssh/id\_rsa\_work

如果出现Could not open a connection to your authentication agent的错误，就试着用以下命令：

ssh-agent bash

ssh-add ~/.ssh/id\_rsa\_work



3.修改config文件

在~/.ssh目录下找到config文件，如果没有就创建：

touch config # 创建config

然后修改如下：  
我的config配置如下：

# 该文件用于配置私钥对应的服务器

# Default github user(first@mail.com)

Host github.com

HostName github.com

User git

IdentityFile C:/Users/Administrator/.ssh/id\_rsa

# second user(second@mail.com)

# 建一个github别名，新建的帐号使用这个别名做克隆和更新

Host github2

HostName github.com

User git

IdentityFile C:/Users/Administrator/.ssh/id\_rsa\_work

如果存在的话，其实就是往这个config中添加一个Host：

#建一个github别名，新建的帐号使用这个别名做克隆和更新

Host github2

HostName github.com

User git

IdentityFile ~/.ssh/id\_rsa2

其规则就是：从上至下读取config的内容，在每个Host下寻找对应的私钥。这里将GitHub [SSH仓库地址中的git@github.com替换成新建的Host别名如](mailto:SSH%E4%BB%93%E5%BA%93%E5%9C%B0%E5%9D%80%E4%B8%AD%E7%9A%84git@github.com%E6%9B%BF%E6%8D%A2%E6%88%90%E6%96%B0%E5%BB%BA%E7%9A%84Host%E5%88%AB%E5%90%8D%E5%A6%82)：github2，那么原地址是：[git@github.com](mailto:git@github.com):funpeng/Mywork.git，替换后应该是：github2:funpeng/Mywork.git.

4. 打开新生成的~/.ssh/ id\_rsa\_work.pub文件，将里面的内容添加到GitHub后台。

5.测试



报出warning IP address’192.30.252.122’ 的host问题，在config里改host可解决

