**API网址：http://avalonjs.coding.me/api.html**

**http://blog.csdn.net/vipzjyno1/article/details/22098621**

**user:=🡺huzi2840657988**

**password=🡺wd201100999**

**GitHub操作总结 : 总结看不明白就看下面的详细讲解.**

**GitHub操作流程** :

**第一次提交** :

**方案一** : **本地创建项目根目录, 然后与远程GitHub关联, 之后的操作一样**;

-- **初始化git仓库** :**git init** ;

-- **提交改变到缓存** :**git commit -m 'description'** ;

-- **本地git仓库关联GitHub仓库** : **git remote add origin git@github.com:han1202012/TabHost\_Test.git** ;

-- **提交到GitHub中** : **git push -u origin master** ;

**方案二**: **方案二就是不用关联GitHub仓库, 直接从GitHub冲克隆源码到本地, 项目根目录也不用创建**;

-- **从GitHub上克隆项目到本地** :**git clone git@github.com:han1202012/NDKHelloworld.git** , 注意克隆的时候直接在仓库根目录即可, 不用再创建项目根目录 ;

-- **添加文件** :**git add ./\*** , 将目录中所有文件添加;

-- **提交缓存** :**git commit -m '提交'**;

-- **提交到远程GitHub仓库** : **git push -u origin master** ;

**之后修改提交** :

-- **与GitHub远程仓库同步** :**git pull** ;

-- **查看文件变更** : **git status** ;

-- **提交代码到本地缓存** : **git commit -m 'description'**;

--**提交代码到远程GitHub仓库** :**git push** ;

**.gitignore用法** : 开放模式 注明忽略的文件 直接列出文件名, 保守模式 注明保留的文件 !文件名 ;

**Git标签操作** : 轻量级标签, 带注释标签;

--**查看标签** :**git tag** ;

--**添加标签** : 轻量级标签**git tag tagName** , 带注释标签**git tag -a tagName -m 'description'** ;

--**删除标签** :**git tag -d tagName** ;

--**提交标签到GitHub中** : **git push origin --tags** ;

**Git分支操作**: 创建分支后, 分支操作不会影响master分支, 但是master分支改变会影其它分支;

--**列出分支** :**git branch** ;

--**切换分支** :**git checkout master** ;

--**提交分支** : **git push origin branchName** ;

--**删除分支** : **git branch -d branchName** , 强制删除分支 **git branch -D branchName** ;

--**合并分支** : **git merge branchName** ;

.

**一. Git介绍**

**分布式** : Git版本控制系统是一个**分布式的系统**, 是用来保存工程源代码历史状态的命令行工具;

**保存点** : Git的保存点可以**追踪源码中的文件**, 并能得到某一个时间点上的整个工程项目额状态; 可以在该保存点将多人提交的源码合并, 也可以会退到某一个保存点上;

**Git离线操作性** :**Git可以离线进行代码提交**, 因此它称得上是完全的分布式处理, Git所有的操作**不需要在线进行**; 这意味着Git的速度要比SVN等工具快得多,  因为SVN等工具需要在线时才能操作, 如果网络环境不好, 提交代码会变得非常缓慢;

**Git基于快照** : SVN等老式版本控制工具是**将提交点保存成补丁文件**, Git提交是将**提交点指向提交时的项目快照**, 提交的东西包含一些元数据(作者, 日期, GPG等);

**Git的分支和合并** : 分支模型是Git最显著的特点, 因为这改变了开发者的开发模式, SVN等版本控制工具将每个分支都要放在不同的目录中, Git可以在同一个目录中切换不同的分支;

**分支即时性** : 创建和切换分支几乎是同时进行的, 用户可以上传一部分分支, 另外一部分分支可以隐藏在本地, 不必将所有的分支都上传到GitHub中去;

**分支灵活性** : 用户可以**随时 创建 合并 删除分支**, 多人实现不同的功能, 可以创建多个分支进行开发, 之后进行分支合并, 这种方式使开发变得快速, 简单, 安全;

**二. Git通用客户端(msysgit)**

**1. 下载Git客户端**

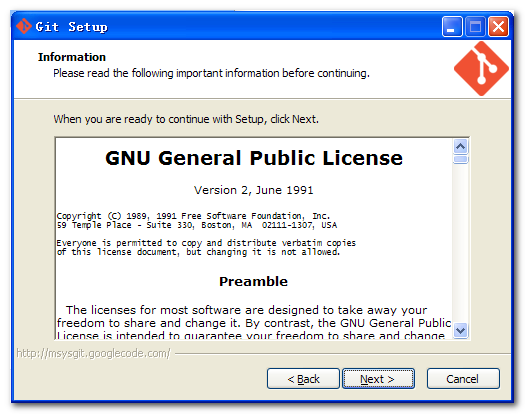
**Git客户端下载地址** :<https://code.google.com/p/msysgit/downloads/list> 将地址复制到浏览器栏即可下载.

**2. 安装Git客户端**

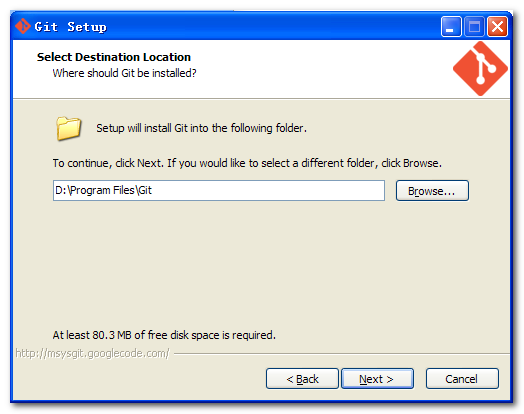
**欢迎界面** : 直接下一步;



**协议** : 必须接受;



**安装位置** : 预留100M空间, 自定义安装位置;



**选择安装组件** :**也可以默认选择**;

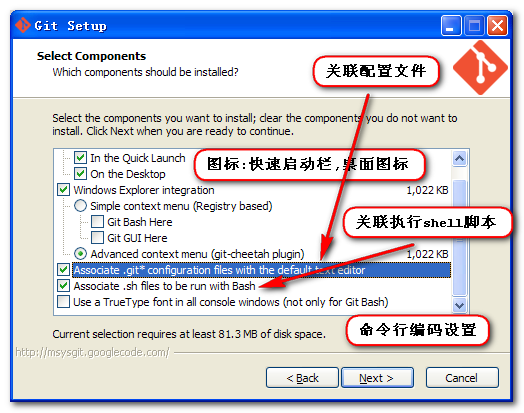
-- **图标组件**(Addition icons) : 选择是否创建快速启动栏图标 或者 是否创建桌面快捷方式;

-- **桌面浏览**(Windows Explorer integration) : 浏览源码的方法, 单独的上下文浏览 只使用bash 或者 只用Git GUI工具; 高级的上下文浏览方法 使用git-cheetah plugin插件;

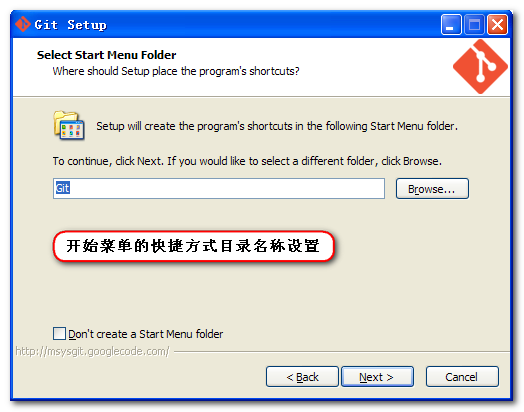
-- **关联配置文件** : 是否关联git配置文件, 该配置文件主要显示文本编辑器的样式;

-- **关联shell脚本文件** : 是否关联Bash命令行执行的脚本文件;

-- **使用TrueType编码** : 在命令行中是否使用TruthType编码, 该编码是微软和苹果公司制定的通用编码;



**开始菜单快捷方式目录** : 设置开始菜单中快捷方式的目录名称, 也可以选择**不再开始菜单中创建快捷方式**;

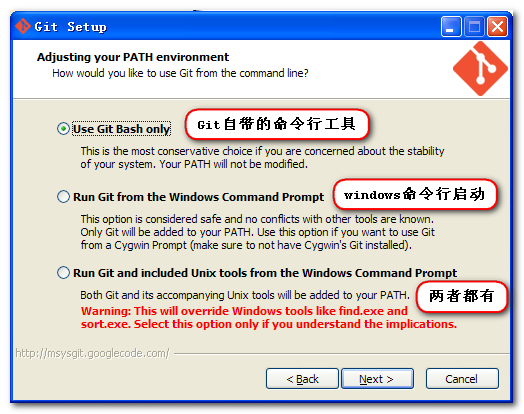


**设置环境变量** : 选择使用什么样的命令行工具, 一般情况下我们**默认使用Git Bash即可, 默认选择**;

-- **Git自带** : 使用Git自带的Git Bash命令行工具;

-- **系统自带CMD** : 使用Windows系统的命令行工具;

-- **二者都有** : 上面二者同时配置, 但是注意, 这样会将windows中的find.exe 和 sort.exe工具覆盖, 如果不懂这些尽量不要选择;

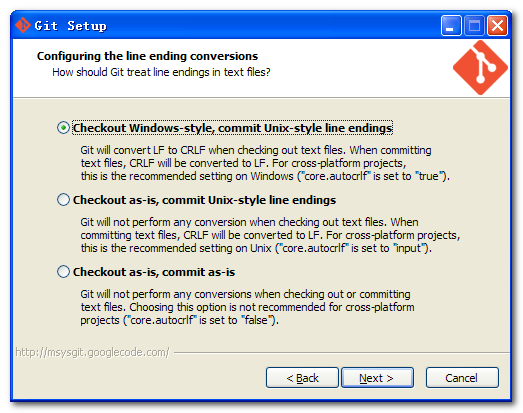


**选择换行格式** :

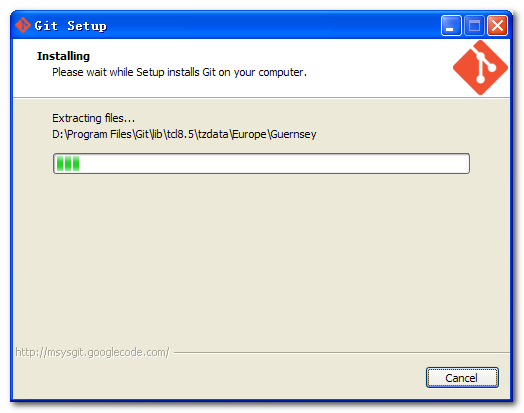
-- **检查出windows格式转换为unix格式** : 将windows格式的换行转为unix格式的换行在进行提交;

-- **检查出原来格式转为unix格式** : 不管什么格式的, 一律转为unix格式的换行在进行提交;

-- **不进行格式转换** : 不进行转换, 检查出什么, 就提交什么;



开始安装 :

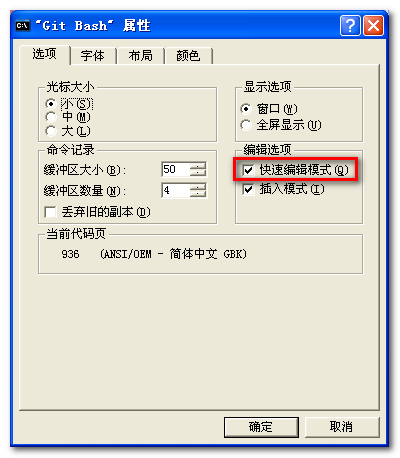


安装结束 : over;



**3. 配置GitHub**

**修改Git Bash的配置 :** 将Git Bash设置为**快速编辑模式**, 可以更好的使用该命令行工具 :



**(1) 创建本地ssh**

**使用命令** : 创建本地ssh

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

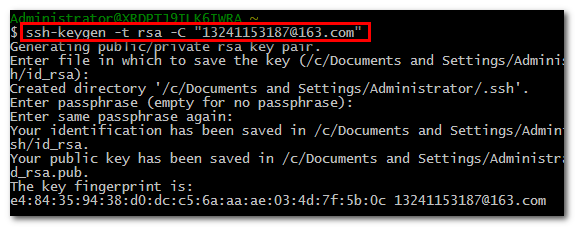
1. 这里以配置github的ssh key为例：
2. 1. 配置git用户名和邮箱
3. git config user.name "用户名"git config user.email "邮箱"
4. 在config后加上 --global 即可全局设置用户名和邮箱。
5. ssh-keygen -t rsa -C "13241153187@163.com"

连续回车三次密码为空

GitHub邮箱 : 该命令后面的邮箱就是GitHub的注册邮箱

**路径选择** : 使用该命令之后, 会出现提示选择ssh-key生成路径, 这里直接点回车默认即可, 生成的ssh-key在默认路径中;

**密码确认** : 这里我们不使用密码进行登录, 用密码太麻烦;



**(2) 将ssh配置到GitHub中**

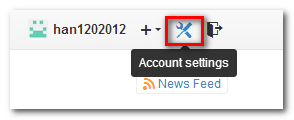
**进入生成的ssh目录** : C:\Documents and Settings\Administrator\.ssh 中, 使用记事本打开 id\_rsa.pub 文件, 将该文件中的内容复制;

**id\_rsa.pub 文件内容** :

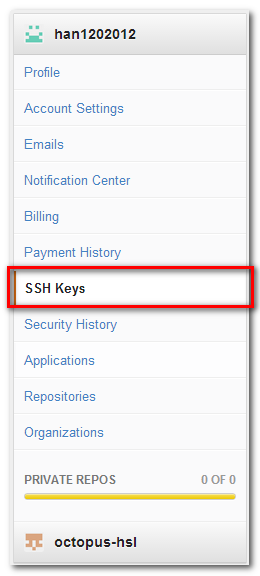
**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAtT1YCeaNulpfC+ARqAWrCdfpi6CpW3gkGT0hp6Q8by7NnEfy4dah9CwSrNbWJH5eS4tiqckE+bdbSVNvAboFD1MtGZjtzE4GDweG/6J/SDYV/ADFN/RLWGb+5rQ8wMCjc/fODgLJDFxk1Fwk/TTqTcbtLab1toLcts3zGIW5DstA3RQ0CCX/sPew5m7vh7DcKXluj2TBd9hw== 13241153187@163.com

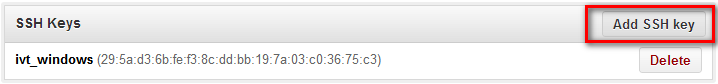
进入GitHub网站 : 登录GitHub, 选择Account Setting 用户设置 :



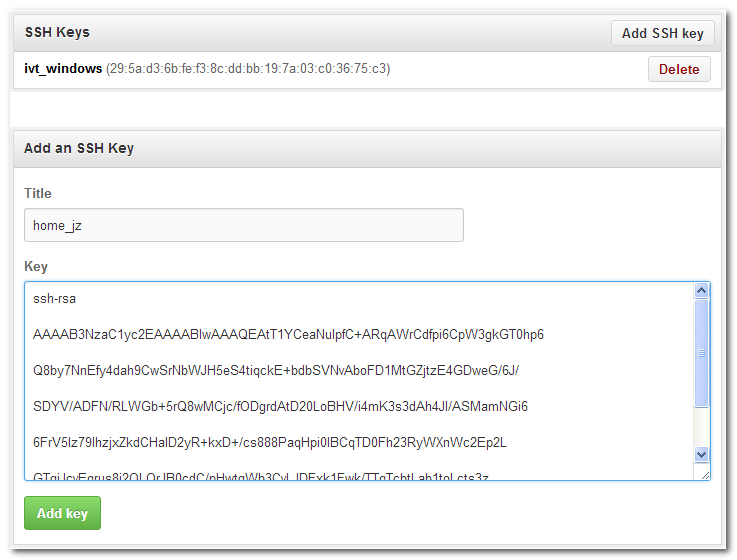
**选择左侧的SSH-KEY选项** :



**点击右侧的Add SSH key** :



**将上面复制好的ssh-key复制进去** :



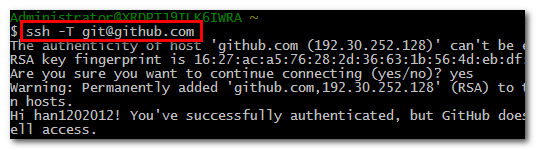
**验证是否配置成功** :

**使用命令** :

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. ssh -T git@github.com

**成功提示** : 如果出现Hi han1202012! You've successfully authenticated, but GitHub does not provide shell access. 就说明配置成功, 可以连接上GitHub;



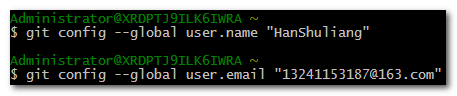
**(3) 配置本地用户和邮箱**

**用户名邮箱作用** : 我们需要设置一个用户名 和 邮箱, 这是用来上传本地仓库到GitHub中, 在GitHub中显示代码上传者;

**使用命令** :

**[java]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. git config --global user.name "HanShuliang" //设置用户名
2. git config --global user.email "13241153187@163.com"  //设置邮箱



**4. Git Bash提交源码到GitHub**

**(1) GitHub中创建一个工程**

**工程的https地址**: https://github.com/han1202012/TabHost\_Test.git .

**工程的SSH地址** : git@github.com:han1202012/TabHost\_Test.git .

**GitHub提示生成的命令** :

-- Create a new repository on the command line :

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. touch README.md
2. git init
3. git add README.md
4. git commit -m "first commit"
5. git remote add origin git@github.com:han1202012/TabHost\_Test.git
6. git push -u origin master

-- Push an existing repository from the command line :

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. git remote add origin git@github.com:han1202012/TabHost\_Test.git
2. git push -u origin master

**(2) 初始化git目录**

使用命令 :

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. git init

**如果不执行这条命令, 就会出现错误** : fatal: Not a git repository (or any of the parent directories): .git .

**(3) 添加文件**

使用命令 :

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. git add ./\*

可能会报出一些警告, 不用理会 :

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. warning: LF will be replaced by CRLF in AndroidManifest.xml.
2. The file will have its original line endings in your working directory.
3. warning: LF will be replaced by CRLF in bin/AndroidManifest.xml.
4. The file will have its original line endings in your working directory.
5. warning: LF will be replaced by CRLF in gen/shuliang/han/tabhost\_test/BuildConfig.java.

将仓库中的所有文件添加到缓存中;

**(4) 提交缓存**

使用命令 :

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. git remote add origin git@github.com:han1202012/TabHost\_Test.git

将添加或者改变的内容提交到缓存中;

**(5) 将Git本地缓存提交到GitHub中**

使用命令 :

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. git push -u origin master

执行该命令, 源码就被提交到了GitHub 中;

.

**作者** :**万境绝尘**

**转载请注明出处 :**[**http://blog.csdn.net/shulianghan/article/details/18812279**](http://blog.csdn.net/shulianghan/article/details/18812279)

.

**三. Git的一些用法**

**1. .gitignore文件**

**屏蔽文件** : .gitignore文件是告诉Git哪些目录或者文件需要忽略, 这些文件将不被提交;

**常用场景** : 写完代码后会执行变异调试等操作, 使用 .gitignore 文件将这些编译后的文件屏蔽, 这些文件不需要Git工具进行管理;

**Android中的.gitignore** : 在Android中 bin 和 gen 两个目录可以忽略;

**.gitignore位置** : 项目根目录下;

**过滤模式** : Git中的 .gitignore 中有两种模式,**开放模式** 和**保守模式**,**保守模式的优先级要高于开放模式**;

**开放模式** : 设置哪些文件 活 目录 被过滤, 凡是在文件中列出的文件或者目录都要被过滤掉;

-- **过滤目录** : /bin/ 就是将bin目录过滤, 该文件下的所有目录和文件都不被提交;

-- **过滤某个类型文件** : \*.zip \*.class 就是过滤zip 和 class 后缀的文件, 这些文件不被提交;

-- **过滤指定文件** : /gen/R.java, 过滤该文件, 该文件不被提交;

**保守模式** : 设置哪些文件不被过滤, 凡是列在其中的文件都要完整的提交上去;

-- **跟踪目录** : !/src , 该目录下的所有文件都要被提交;

-- **跟踪某类文件** : !\*.java , 凡是java文件都要保留;

-- **跟踪指定文件** : !/AndroidManifest.xml , 该文件需要保留, 提交上去;

**配置原则** : 一般情况下采用开放模式鱼保守模式**共同使用**;

eg : 一个目录下有很多目录和文件, 当我们只需要保留其中的一个文件的时候, 先用开放模式不保留这些文件, 然后用保守模式将这个文件留下来, 保守模式的优先级要高于开放模式;

**2. 标签使用(Tag)**

**标签作用**: 在开发的一些关键时期,**使用标签来记录这些关键时刻**, 例如发布版本, 有重大修改, 升级的时候, 会使用标签记录这些时刻, 来**永久标记项目中的关键历史时刻**;

**查看标签**: 列出的标签安装ASCII字母顺序确定, 排序没有很明确的意义;

-- **列出所有的标签** :

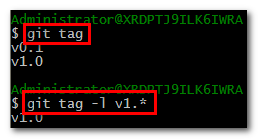
**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. git tag

-- 使用限定列出限定后的标签 :

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. git tag -l v1.\*



**标签分类** : Git中的标签分为 轻量级标签(lightweight) 和 带注释的标签(annotated), 一般情况下推荐使用带注释的标签, 如果标签是临时的可以采用轻量级标签;

-- **轻量级标签** : 轻量级标签中的信息含量很少, 这种标签只代表某时刻代码的提交, 相当于指向这个提交的指针;

-- **带注释标签** : 这种标签是一种校验和, 包含标签名, 邮箱, 日期, 标签信息, GPG签名 和 验证, 它相当于一个对象, 封装了这些信息;

**创建标签** :

-- **创建轻量级标签** : 这样的标签没有附带其它的信息;

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. git tag v2.0



-- 创建带注释标签 : -m 后跟的是注释信息, 当使用git show v2.1的时候, 会显示这个注释信息;

**[plain]** [view plaincopy](http://blog.csdn.net/shulianghan/article/details/18812279)[在CODE上查看代码片](https://code.csdn.net/snippets/174418)

1. git tag -a v2.1 -m 'first version'

  
-- 创建GPG私钥的注释标签 :