----------编辑区模板-------------------------------

二级标题模板

啊

----------GitHub尝试（诸多问题）-------------------------------

配置Git

1. 说在前面：

这个SSH的配置和连接可能和协同开发的安全机制有着密切联系

1. 使用快捷Finder查找键入 ~/.ssh 进入用户目录的隐藏文件夹下，将里面的3个文件放入新建的备份文件夹中
2. $ cd ~/.ssh 终端进入隐藏文件夹下，进行剩下的操作
3. $ ssh-keygen -t rsa -C "609731056@qq.com" 来生成SSH

中间让你选择存放Key的位置（建议位置）

Generating public/private rsa key pair.

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

还让你键入什么密码不知道干嘛用的（回车可以跳过的，我还是键入了“大号密码去符号”）

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

生成结果

Your identification has been saved in /Users/Davie/.ssh/id\_rsa.

Your public key has been saved in /Users/Davie/.ssh/id\_rsa.pub.

剩下的就看不懂了

The key fingerprint is:

SHA256:sHj5mwL9cJpDN2XZ4mRZ7qiUfIm8z5O7zgOSWr5aPX8 609731056@qq.com

The key's randomart image is:

+---[RSA 2048]----+

| |

| . |

| . \* |

| . + O o |

| ..++SO = |

| ..Bo@ = . |

| \*.X+\* . |

| ..\* +B+ E |

| ...+ooO\* |

+----[SHA256]-----+

1. 在github中添加ssh：

登陆github，点击右上角头像，下拉列表中选择Setting

选择SSH and GPA keys —> delete —> new SSH key

Title：xxxxx@gmail.com

Key：打开你生成的id\_rsa.pub文件，将其中内容拷贝至此。

然后保存

1. 打开终端，先测试一下你的帐号跟github连上没有：ssh -T git@github.com 如果出现如下提示，表示你连

已经连上了.(因为有了第一步，所以不用自己做过多的连接github的操作了，另外，下一次要连接github的时候记得打开第一步的工具).

与网站给出的结果稍有不同（可能是因为我设置了密码）

The authenticity of host 'github.com (192.30.252.121)' can't be established.

RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.

Are you sure you want to continue connecting (yes/no)? yes

弹出窗口输入“大号密码去符号”后

Warning: Permanently added 'github.com,192.30.252.121' (RSA) to the list of known hosts.

Identity added: /Users/Davie/.ssh/id\_rsa ((null))

过了一会才出现网站一样的提示：

Hi Davie-Billincon! You've successfully authenticated, but GitHub does not provide shell access.

使用Git

在github网站下建自己的Repository

1. 右上角+号 new Repository
2. 一般Name就是你的项目名
3. 项目描述不要太长
4. 是否自动添加ReadMe文档（会帮你在文档中生成“license”，自动生成配置文档 .gitignore，都选None）
5. 点击create就生成一个仓库了

开始上传文件

将终端切换到工程目录下

这里好多不明白的操

touch README.md //(刚才仓库已经自动生成了)新建一个记录提交操作的文档

git init //初始化本地仓库

打印结果： Initialized empty Git repository in /Users/Davie/Desktop/学习总资料/手心iOS项目

/MyWavePullPro/.git/

在打开显示隐藏文件夹后，可以.git文件夹，里面并没有README.md，说明这是个配置文件

git add README.md //添加刚才touch出的README.md到 ？？？

打印结果：无

可能就是将其他文件添加到当前文件夹的意思，手动操作也行

git commit -m "first commit"//提交到本地仓库，并写一些注释

打印结果：[master (root-commit) c3be343] 我的第一次提交，好激动

1 file changed, 4 insertions(+)

create mode 100755 README.md

git remote add origin [git@github.com:youname/Test.git](mailto:git@github.com:youname/Test.git) //连接远程仓库并建了一个名叫origin的别名

打印结果：无

怀疑是新建分支，但反复刷新网站并没有看见分支被创建

当你再次使用该命令时，会提示已经存在：

fatal: remote whatname already exists.

$ git pull origin master

首先出现一堆打印，然后弹出一个vim让你输入东西

vim文件内容：

Merge branch 'master' of github.com:Davie-Billincon/MyWavePullPro

Please enter a commit message to explain why this merge is necessary

打印结果：

Warning: Permanently added the RSA host key for IP address '192.30.252.120' to the list of

known hosts.

warning: no common commits

remote: Counting objects: 5, done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (5/5), done.

From github.com:Davie-Billincon/MyWavePullPro

\* branch master -> FETCH\_HEAD

\* [new branch] master -> whatname/master

Already up-to-date!

Merge made by the 'recursive' strategy.

git push -u origin master //将本地仓库的东西提交到地址是origin的地址，master分支下

打印结果：Counting objects: 4, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (4/4), 612 bytes | 0 bytes/s, done.

Total 4 (delta 1), reused 0 (delta 0)

To git@github.com:Davie-Billincon/MyWavePullPro.git

f960d99..b461597 master -> master

刷新网站发现，我之前add并commit的README.md文件也出现在了网站中

然后可以看到一个master的commit，并且我在vim编辑的内容出现在了这里

我想上传整个文件夹，又该怎么办，总不能一个个add吧？

尝试上传整个项目

1. 我使用了 git add \* 然后后续操作不知道怎么回事没反应且报错……
2. 跑完步回来我使用了

Davie-Pro:MyWavePullPro Davie$ git add Podfile

Davie-Pro:MyWavePullPro Davie$ git commit -m "第二次Podfile的提交"

1. 结果commit后出现一大堆上传文件（本项目的所有文件）
2. 然后再使用命令：git push whatname master，发现成功上传
3. 而且我发现，whatname貌似表示一个连接的别名，而不是分支名，因为也可以将master换成分支名

尝试重头上传项目

1. 新建一个文件夹

出现的问题

执行git push -u origin master时出现以下打印

! [rejected] master -> master (fetch first)

error: failed to push some refs to 'git@github.com:Davie-Billincon/MyWavePullPro.git'

hint: Updates were rejected because the remote contains work that you do

hint: not have locally. This is usually caused by another repository pushing

hint: to the same ref. You may want to first integrate the remote changes

hint: (e.g., 'git pull ...') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.

翻译结果：

这次的更新被拒绝，因为远端已经包含你现在在做的工作了

没有本地化，这个问题一般是由于其他仓库的push操作造成的

相同的引用，你可能是想第一次整体化远程变化

最好是先pull在push

在 git push –help 查看 ‘fast-forwards手册’ 更详细的信息

解决办法如下：

1、先输入$ git pull origin master //先把远程服务器github上面的文件拉下来

2、再输入$ git push origin master

3、如果出现报错 fatal: Couldn't find remote ref master或者fatal: 'origin' does not appear to be a

git repository以及fatal: Could not read from remote repository.

4、则需要重新输入$ git remote add origingit@github.com:djqiang/gitdemo.git

.gitignore 配置（网址待学）

.gitignore 配置文件用于配置不需要加入版本管理的文件，配置好该文件可以为我们的版本管理带来很大的便利，

以下是个人对于配置 .gitignore 的一些心得。

http://www.cnblogs.com/haiq/archive/2012/12/26/2833746.html

应该还有其他学习网址自己搜

----------版本控制 & Git起步-------------------------------

自问题组：

1. 两个人都同时下载了master某一时刻的整个项目，第一个人修改自己模块A提交整个项目后，第二个人修改

模块B提交整个项目，那么master与第二人在 模块A 与 模块B 的不同如何处理？？

解决一：

第二个人在提交前，需要手动下载校对，或提交的时候系统帮你校对

无论是文件的增多，文件的缺失；同名文件同一行内容的增多，缺失，不同，都会处于校对工作中

你可以指定范围（模块B）校对，并对范围外的所有本次提交与master的不同（模块A）忽视校对更改

在你确认校对后（同意增删改），确认提交来提交并覆盖范围内（模块B）的内容

上述覆盖不一定是服务端文件删除然后提交，也许就是必要文件的上传和日记记录

解决二：

上传前就划定范围（模块B）

即使上传整个项目，日志Care的仅仅是范围内的更改

并且范围内的所有更改都默认上传替换，不进行任何校对

解决三：

在解决二的基础上，加入冲突校对

也就是对增加和同行的消失不做校对，而对同行的不同内容进行校对

版本控制是一种记录若干文件内容变化，以便将来查阅特定版本修订情况的系统。在本书所展示的例子中，我们仅对保存着软件源代码的文本文件作版本控制管理，但实际上，你可以对任何类型的文件进行版本控制。

采用版本控制系统 （VCS）是个明智的选择。有了它你就可以将某个文件回溯到之前的状态，甚至将整个项目都回退到过去某个时间点的状态。你可以比较文件的变化细节，查出最 后是谁修改了哪个地方，从而导致出现怪异问题，又是谁在何时报告了某个功能缺陷等等。

本地版本控制系统

许多人习惯用复制整个项目目录的方式来保存不同的版本，或许还会改名加上备份时间以示区别。

为了解决这个问题，人们很久以前就开发了许多种本地版本控制系统，大多都是采用某种简单的数据库来记录文件的历次更新差异

其中最流行的一种叫做 rcs， Mac OS X 系统上安装了开发者工具包之后，也可以使用 rcs 命令。它的工作原理基本上就是保存并管理文件补丁（patch）。文件补丁是一种特定格式的文本文件，记录着对应文件修订前后的内容变化。所以，根据每次 修订后的补丁，rcs 可以通过不断打补丁，计算出各个版本的文件内容。

集中化的版本控制系统

如何让在不同系统上的开发者协同工作？于是，集中化的版本控制系统（ Centralized Version Control Systems，简称 CVCS ）应运而生。这类系统，诸如 CVS，Subversion 以及 Perforce 等，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。多年以来，这 已成为版本控制系统的标准做法。

分布式版本控制系统

----------GitHub原理探究-------------------------------

网址：http://www.open-open.com/lib/view/open1328070620202.html

Git的优势

几乎每一种版本控制系统都以某种形式支持分支。使用分支意味着你可以从开发主线上分离开来，然后在不影响主线的同时继续工作。在很多版本控制系统中，这是个昂贵的过程，常常需要创建一个源代码目录的完整副本，对大型项目来说会花费很长时间。

Git 的分支可谓是难以置信的轻量级，它的新建操作几乎可以在瞬间完成，并且在不同分支间切换起来也差不多一样快。和许多其他版本控制系统不同，Git 鼓励在工作流程中频繁使用分支与合并，理解分支的概念并熟练运用后，你才会意识到为什么 Git 是一个如此强大而独特的工具，并从此真正改变你的开发方式。

这是开始