PastedGraphic-1

代码管理工具之Git

1. 什么是版本控制？

版本控制透过文档控制（documentation control）记录程序各个模组的改动，并为每次改动编上序号。这种方法是**工程图**（engineering drawings）维护（maintenance）的标准做法， 它伴随着工程图从图的诞生一直到图的定型。 一种简单的版本控制形式，例如，赋给图的初版一个版本等级“A”。当做了第一次改变后，版本等级改为“B”，以此类推等等

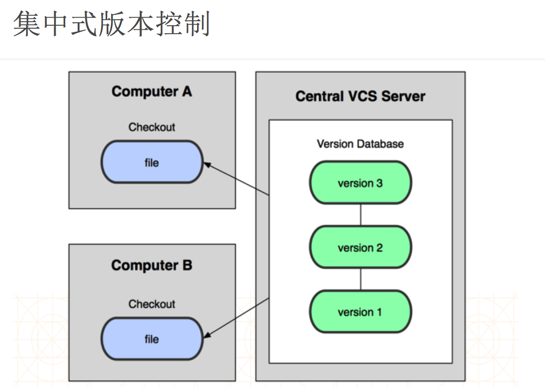
1. 什么是Git？

Git是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。（私有化项目，要收费）; Bitbucket可以：1，允许设置中文，2，在Bitbucket上可以得到免费的私人仓库，却在Github上为此付费。

具体不同：

https://www.oschina.net/translate/bitbucket-vs-github-its-more-than-just-features

1. 什么是集中式版本控制？什么又是分布式版本控制？
2. 集中式版本控制：



理解：

ComputerA想要服务器最新的代码直接通过checkout向服务器下载

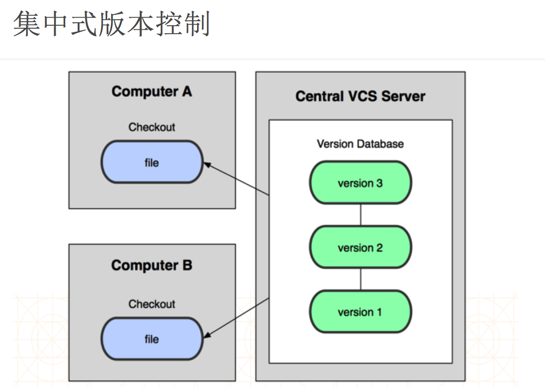
ComputerA在本地修改代码后,直接提交到服务器

ComputerB想要服务器最新的代码直接通过checkout向服务器下载

ComputerB在本地修改代码后,直接提交到服务器

总结 ：所有的内容统一交给服务器来进行管理

分布式版本控制：



理解：

服务器本地有个代码仓库,从服务器更新代码,上传代码

ComputerA想要服务器最新的代码由本地代码仓库将服务器的代码下载下来,再通过本地代码仓库的项目下载到ComputerA

ComputerA在本地修改完代码后先提交到本地的代码仓库,再由本地的代码仓库提交到服务器

ComputerB操作与ComputerA相似

总结：代码的提交与更新首先会通过本地代码仓库,本地代码仓库再通过服务器,并不是直接交给服务器来进行管理

1. **分布式和集中式的对比：**

1. 在速度上，git可以先提交给本地，然后再提交给代码仓库，而SVN只能直接提交给代码仓库，所以在网络出现状况时，git就表现出了绝对优势。

 2.SVN使用分支比较笨拙，需要先从tags中将代码拷贝过来,修改后备份到分支,分支再与主干的项目合并。

3.git可以轻松拥有无限个分支,轻松的在各个分支随意切换。

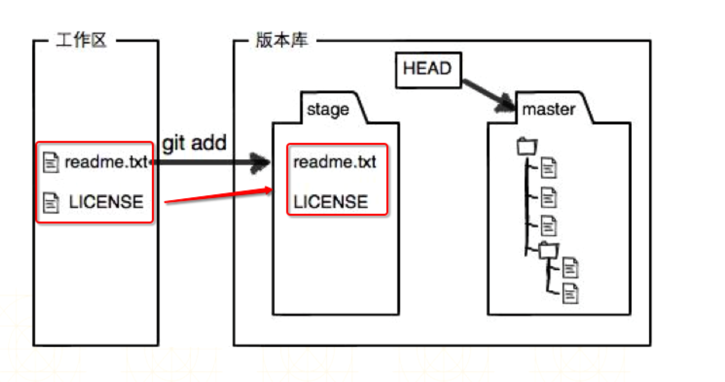
 4. 旧版本的SVN会在每一个目录置放一个.svn(目前svn1.7的只有一个.svn)，git只会在根目录拥有一个.git。

1. Git的工作原理：

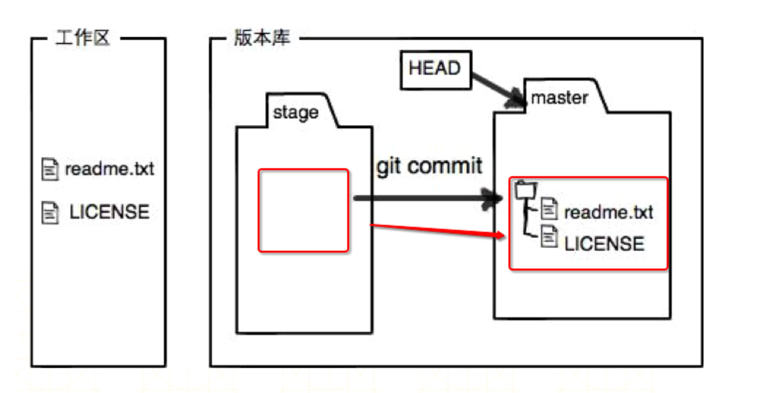


1.svn开发工作在主干进行,git开发工作在分支中进行

2.通过add命令可以将工作目录没有被添加到暂缓区的文件添加到暂缓区



3.通过commit命令将暂缓区的所有内容上传到当前分支,提交成功后清空暂缓区内容



1. Git的简单介绍：

6.1: **GIT的使用的2种方法：**

1.命令行:常用的命令简单且不多,直接使用命令行就行

2.图形化界面工具

  SourceTree GitHub（SourceTree GitHub 3.xcode xcode对git的集成非常非常好,一般情况下,   直接使用xcode就行）

6.2: **GIT的常用指令**

git help：git指令帮助手册        查看其他指令的做法：githelp其他指令

git config：git的配置信息相关（修改的是.git/config文件）

                配置用户名：git config “user.name”用户名（用于跟踪修改记录）

                配置邮箱：git config “user.email”邮箱（用于多人开发间的沟通）

                设置指令的别名：git config alias.别名原指令名称

                将此设置应用到整个系统中：gitconfig––gloabal

                查看配置信息：gitconfig–l

                编辑配置信息：gitconfig–e（用vim编辑，:wq是退出vim编辑器）

git status：查文件的状态

                查看某个文件的状态：git status文件名

                查看当前路径所有文件的状态：git status

git log：查看文件的修改日志

                查看某个文件的修改日志：git log文件名

                查看当前路径所有文件的修改日志：git log

                用一行的方式查看简单的日志信息：git log – –pretty=oneline

                查看最近的N次修改：git log –N（N是一个整数）

git diff：查看文件最新改动的地方

               查看某个文件的最新改动的地方：git diff 文件名

               查看当前路径所有文件最新改动的地方：git diff

git init：初始化一个空的本地仓库，生成一个.git目录，用于维护版本信息

               在当前路径初始化仓库：git init

               在其他路径初始化仓库：git init 仓库路径

git add：将工作区的文件保存到暂缓区

               保存某个文件到暂缓区：git add 文件名

               保存当前路径的所有文件到暂缓区：git add .（注意，最后是一个点.）

git commit：将暂缓区的文件提交到当前分支

               提交某个文件到分支：git commit -m ”注释”文件名

               保存当前路径的所有文件到分支：git commit -m ”注释”

git reset：版本回退（建议加上––hard参数，git支持无限次后悔）

               回退到上一个版本：git reset – –hard HEAD^

               回退到上上一个版本：git reset – –hard HEAD^^

               回退到上N个版本：git reset – –hard HEAD~N（N是一个整数）

               回退到任意一个版本：git reset – –hard 版本号（版本号用7位即可）

git reflog：查看指令使用记录（能够查看所有的版本号）

git rm：删除文件（删完之后要进行commit操作，才能同步到版本库）

git clone：下载远程仓库到本地

               下载远程仓库到当前路径：git clone 仓库的URL

              下载远程仓库到特定路径：git clone 仓库的URL存放仓库的路径

git pull：下载远程仓库的最新信息到本地仓库

git push：将本地的仓库信息推送到远程仓库

git rm：删除文件（删完之后要进行commit操作，才能同步到版本库）

**6.3，远程仓库**

GIT服务器的搭建非常繁琐，如果是多人团队开发，最好还是搭建一个远程仓库

   搭建远程仓库的途径

   1.自己搭建一个git服务器：费时费力

   2.在GitHub上托管项目：公开项目免费、私有项目收费，很多第三方开源项目

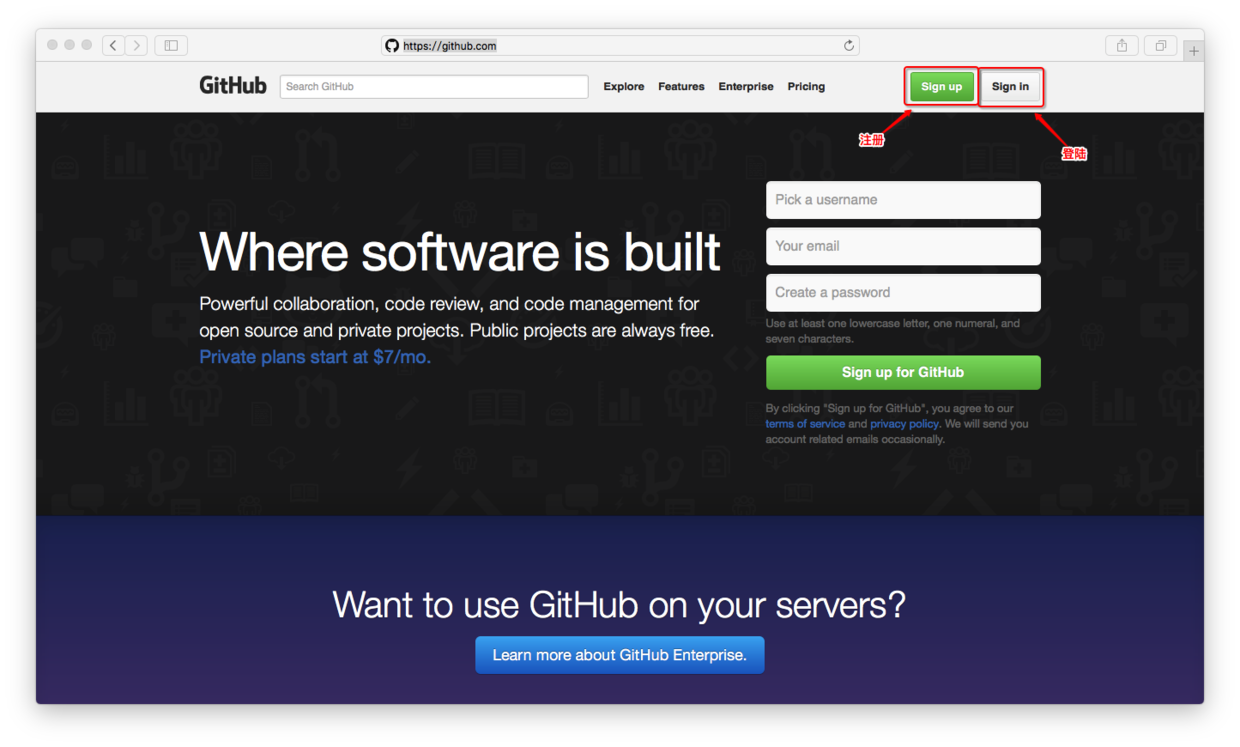
   3.在oschina上托管项目：完全免费，在国内访问速度快（推荐使用）

**6.4，创建Github代码仓库和HTTPs验证**

前提准备

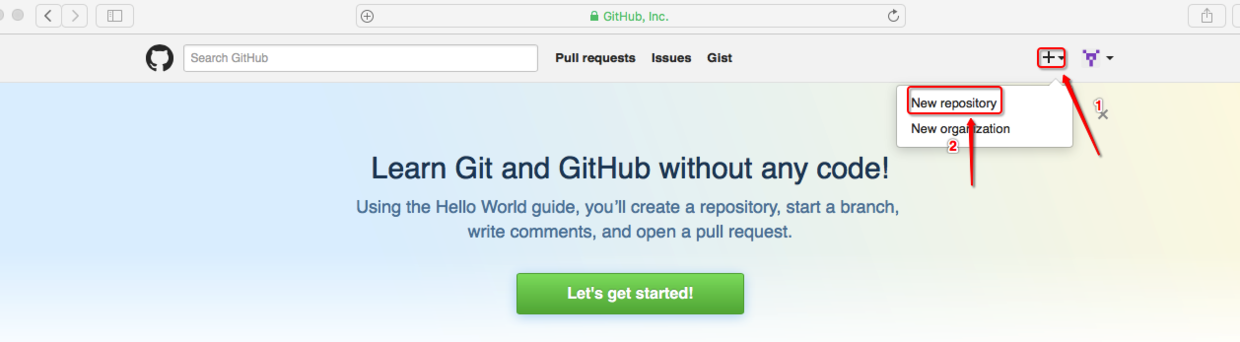
   1.在GIT中创建一个Github文件夹演练的github

   2.登陆github网站github

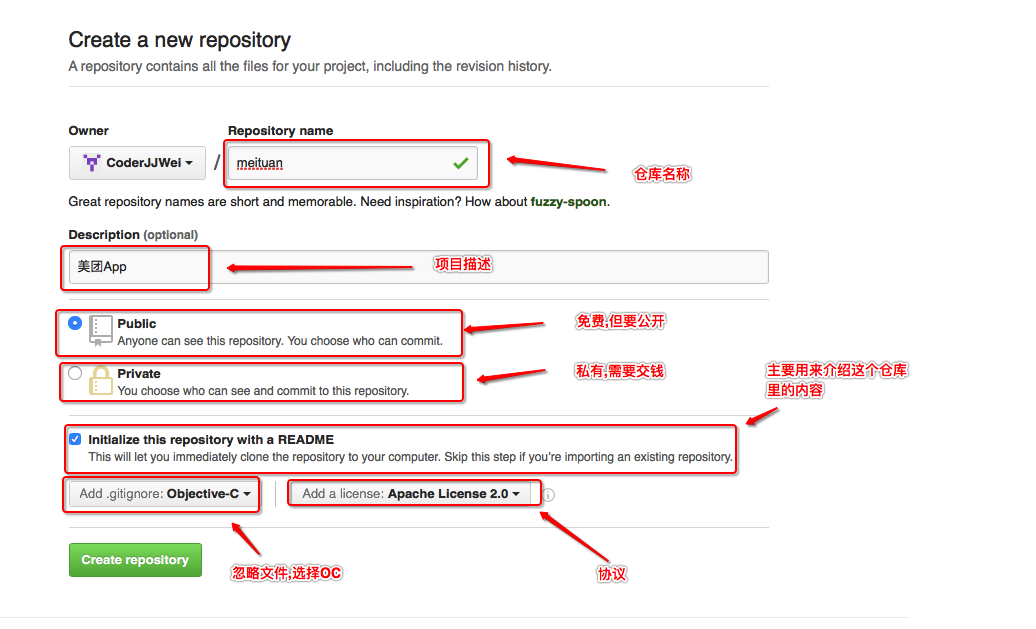


3.创建远程仓库

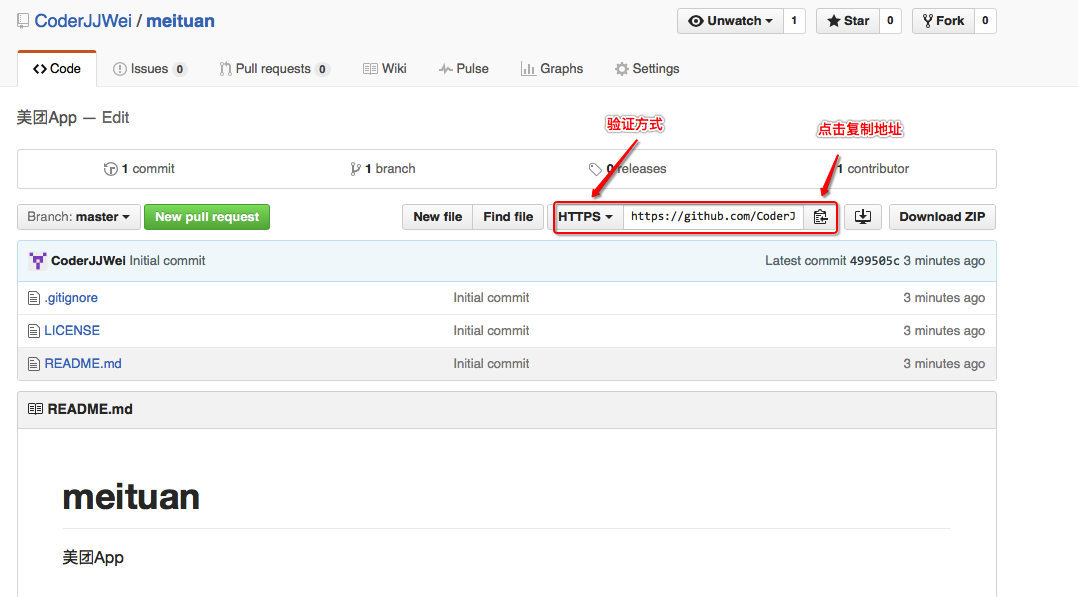
1.）点击+ ->New repository



2.）创建远程代码仓库



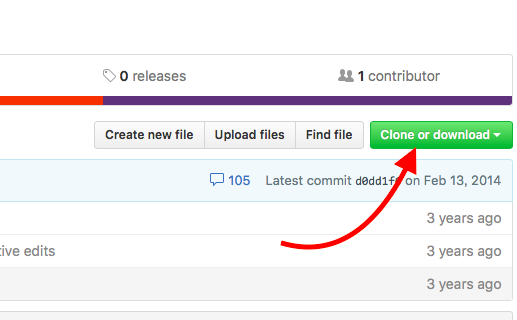
3.）创建后会来到此界面,将https的url复制下来



1. SourceTree工具简单介绍：

7.1：创建仓库

7.1.1:通过以上的操作我们的仓库已经创建成功。创建成功后，我们在页面的右下角找到链接，点击复制



7.1.2:打开我们的SourceTree，点击：“+新仓库”，选择：“从URL克隆”



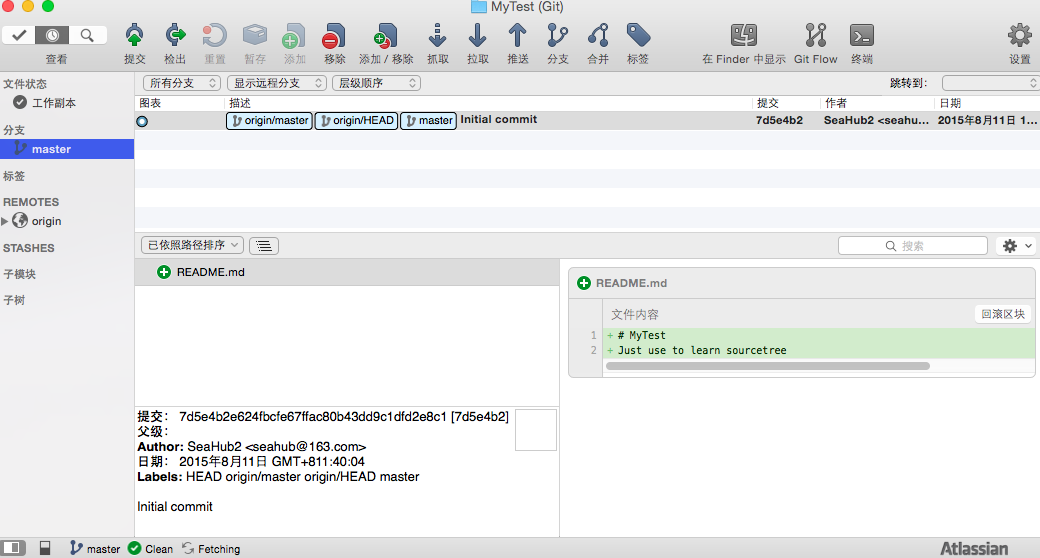
*Clone 1*

7.1.3: 粘贴我们的仓库链接至源URL，SourceTree会自动帮我们生成目标路径（本地仓库路径）以及名称，点击克隆



*Clone 2*

7.1.4: 等待数秒后，SourceTree会为我们自动打开我们刚才克隆的仓库，选择master选项，这里我们可以看到我们仓库里的所有文件



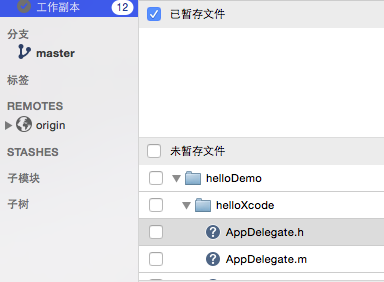
*MainPage*

7.1.5:接下来，上传一个项目到我们的远程Github仓库中。我们点击右上角在Finder”中显示。然后SourceTree会帮我们打开我们的本地仓库，我们将需要上传的项目复制到本地的Finder文件夹内，然后关闭文件夹，回到主页面。我们会发现工作副本出现了更改提示

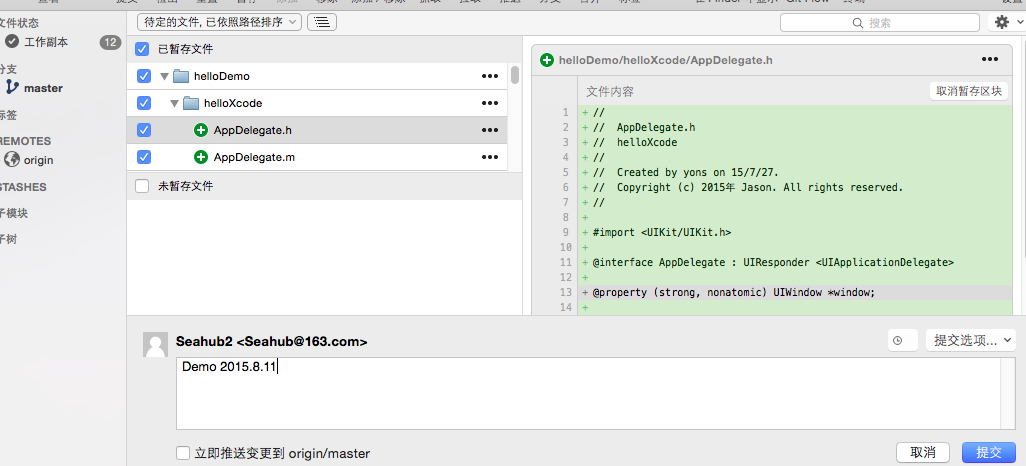


*Change Notification*

7.1.6:我们点击工作副本，然后我们发现我们刚才上传的文件都在未暂存文件当中，此时，我们勾选“未暂存文件”



7.1.7：发现我们的文件变成了已暂存文件。此时，我们可以输入更新信息，然后，点击提交按钮

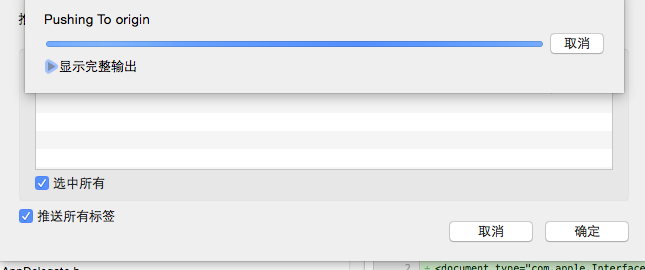


7.7.8:我们切换回master分支，会发现master分支以及推送（Push）按钮，都出现了更改提示。这表示SourceTree已经将我们刚才添加的文件成功提交到本地仓库，而本地仓库的内容则比远程仓库超前了一个版本。我们这个时候点击推送（Push）即可将本地仓库的内容同步至远程仓库。

****

*Branch Master*

7.7.9:点击推送（push），等待片刻即可，我们重新登录github网站，会发现我们刚才本地仓库的文件已经成功推送到远程仓库

****

*Push*

7.2：参与开源：

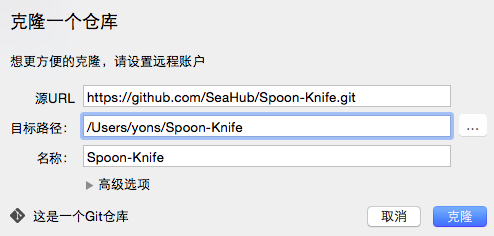
(以下以 [https://github.com/shenshanyikuchai/lihuitest] 举例)  
7.2.1: 首先打开上述页面，然后点击右上角“fork”按钮。fork意味着将他人的仓库复制到我们账号中。如果我们想要参与开源项目，首先要fork下别人的项目，然后在我们复制过来的仓库中，对别人的代码做修改。

Macintosh HD:Users:lihui:Desktop:李辉测试:674642-8f6c592ea0b0e8e5.png

*fork*

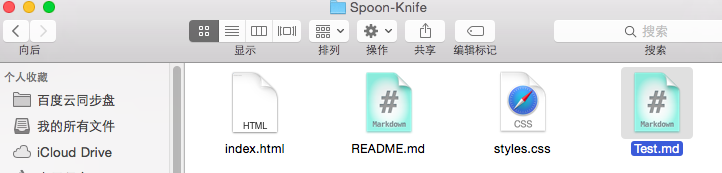
7.2.2:fork完之后，我们用上文提到的方法，将自己账号中的[Spoon-Knife]仓库克隆（clone）到本机SourceTree中，并在稍作更改后，推送（push）到自己账号的远程仓库

7.2.3:在SourceTree中建立新仓库，并复制URL

****

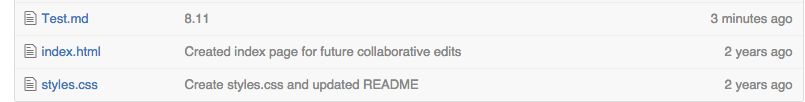
*clone Spoon-Knife*

7.2.4 在本地仓库中稍作更改（如图我新建了一个Test）

****

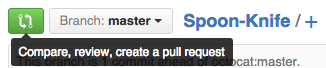
*Example*

7.2.5 通过上文方法推送（Push），登陆Github，进入Spoon-Knife仓库，发现test已经上传到我们账号的远程仓库Spoon-Knife当中

****

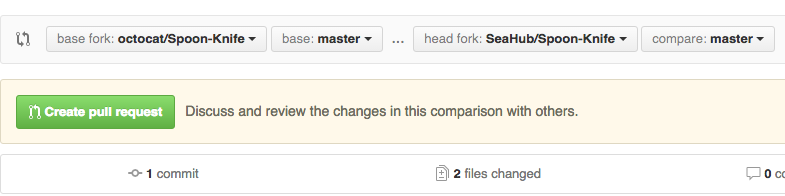
*Success*

7.2.6 ：上传完后，我们点击Github branch旁的绿色按钮

****

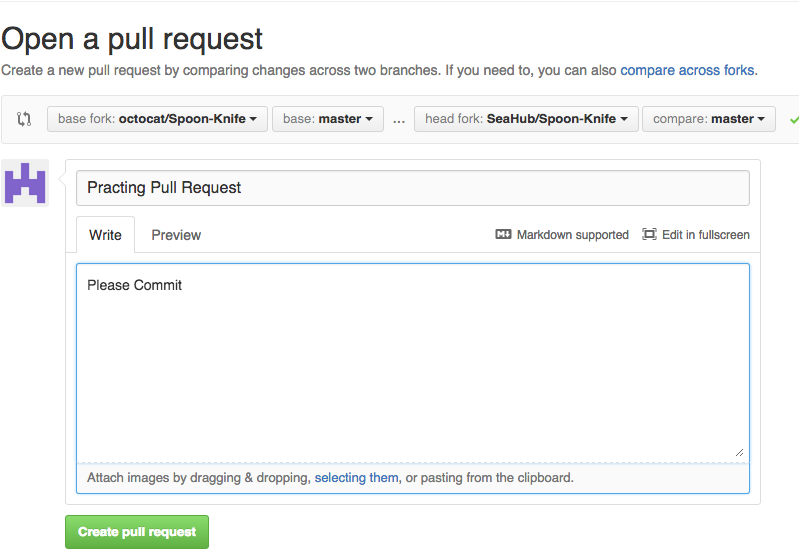
*Pull Request 1*

7.2.7: 之后我们会进入一个Compare页面，这个页面用于比较作者仓库与我们仓库的文件的不同。Base fork：指的是作者仓库目录地址；Head fork：指的是我们账号中fork后所产生的仓库地址。我们点击Create pull request即可

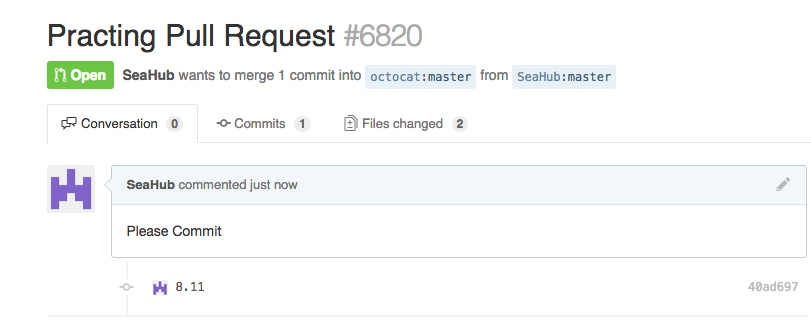
****

*Compare*

7.2.8. 然后我们会进入一个Pull Request界面，在这里，我们可以输入自己更改的原因/更改的内容。这里写的文字会显示给源代码作者，如果作者接受了我们的推送请求（pull request）后，我们的代码将会上传到源代码作者的仓库内，成功为开源做贡献。如果作者拒绝了我们的推送请求（pull request）后，我们的代码将不会上传到源代码作者的仓库内。

****

7.2.9:我们点击Create Pull Request即可，系统将自动跳转到等待回复的页面，这里会显示作者是否接受我们的代码更改。

****

1. **SourceTree&Git部分名词解**
2. 克隆(clone)：从远程仓库URL加载创建一个与远程仓库一样的本地仓库
3. 提交(commit)：将暂存文件上传到本地仓库（我们在Finder中对本地仓库做修改后一般都得先提交一次，再推送）
4. 检出(checkout)：切换不同分支
5. 添加（add）：添加文件到缓存区
6. 移除（remove）：移除文件至缓存区
7. 暂存(git stash)：保存工作现场
8. 重置(reset)：回到最近添加(add)/提交(commit)状态
9. 合并(merge)：将多个同名文件合并为一个文件，该文件包含多个同名文件的所有内容，相同内容抵消
10. 抓取(fetch)：从远程仓库获取信息并同步至本地仓库
11. 拉取(pull)：从远程仓库获取信息并同步至本地仓库，并且自动执行合并（merge）操作，即 ***pull=fetch+merge***
12. 推送(push)：将本地仓库同步至远程仓库，一般推送（push）前先拉取（pull）一次，确保一致
13. 分支(branch)：创建/修改/删除分枝
14. 标签(tag):给项目增添标签
15. 工作流(Git Flow):团队工作时，每个人创建属于自己的分枝（branch），确定无误后提交到master分枝
16. 终端(terminal):可以输入git命令行