目录

[一、svn： 2](#_Toc524904973)

[简介 2](#_Toc524904974)

[安装 2](#_Toc524904975)

[确认不一样的 14](#_Toc524904976)

[查看冲突 15](#_Toc524904977)

[二、git： 15](#_Toc524904978)

[命令行 15](#_Toc524904979)

[GUI 18](#_Toc524904980)

[基本使用 18](#_Toc524904981)

[GitLab 28](#_Toc524904982)

[项目可见级别 28](#_Toc524904983)

[添加和配置SSH公钥 28](#_Toc524904984)

[如何忽略这些文件和文件夹的变更信息 28](#_Toc524904985)

[找回本地误删的文件 28](#_Toc524904986)

[完整上传一个新的项目 28](#_Toc524904987)

[本地删除文件如何提交 29](#_Toc524904988)

[本地文件冲突如何解决 30](#_Toc524904989)

[工作区、缓存区、本地库 31](#_Toc524904990)

[一个复杂操作的例子 31](#_Toc524904991)

[遇到冲突时的分支合并 39](#_Toc524904992)

[jekins自动构建 44](#_Toc524904993)

[常见问题解答 44](#_Toc524904994)

# 一、svn：

简介

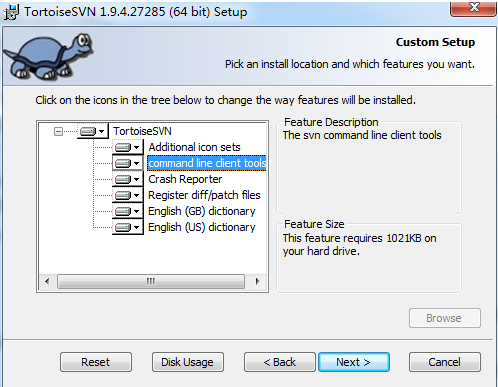
Apache Subversion 通常被缩写成 SVN，是一个开放源代码的版本控制系统。

安装

**服务器端安装：**待完成。

**客户端安装：**首先安装SVN客户端，windows一般选择乌龟客户端[https://tortoisesvn.net/downloads.html](https://tortoisesvn.net/downloads.html" \t "_blank)。根据系统位数选择相应客户端进行安装。

如果你喜欢用命令行操作，请务必记得勾选command line client tool为will be install on local hard driver,不用命令行的跳过这一步。

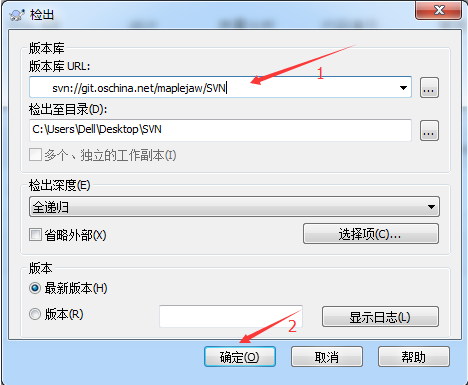


然后一路next即可安装。安装完毕后，在任意地方右键查看快捷菜单。发现TortoiseSVN即表示安装成功。如果勾选了安装命令行工具，那么输入命令SVN，有如下提示也表示安装成功。

**检出项目(Checkout)：**假如项目已经在服务器的仓库里，那么现在你要做的就是把它检出到本地。

首先创建一个空文件夹。在空文件夹内右键，选择SVN检出。

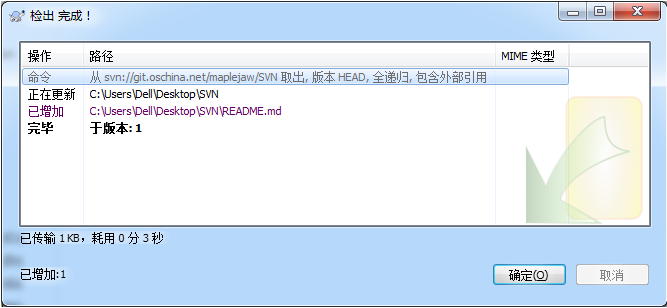
现在你看到应该是下方这个界面，填入版本库地址，选择确定。



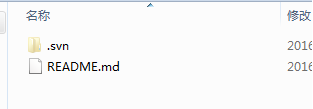
注：也可以选择版本签出。

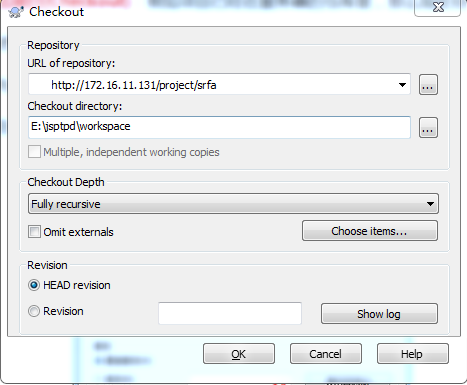
此时会弹出一个对话框让你输入账号密码，输入你的账号密码即可。记得勾选保存认证，不然每次操作都会让你输入。

等几分钟就可以检出完毕。



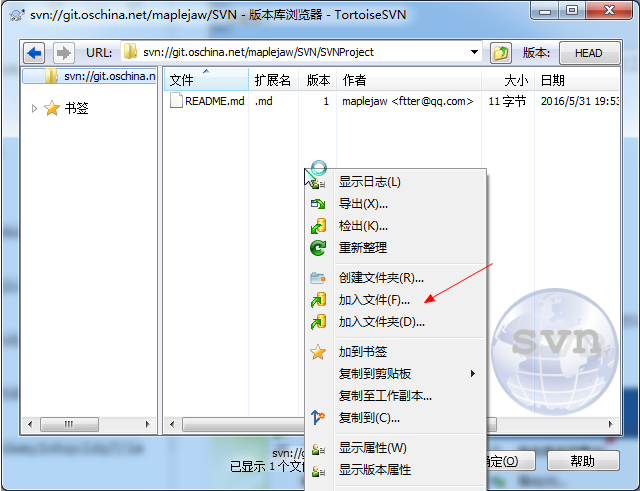
此时在你的目录下就能看到你的项目，现在可以开始愉快的工作了。



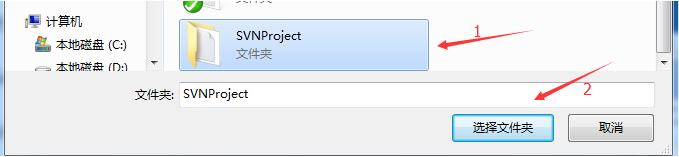


本公司的，Fully recursive是全递归的意思。

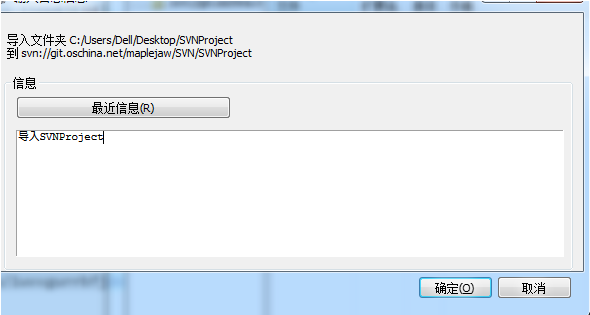
**导入项目：**有时候你已经在本地建立好了项目，需要把你项目推到SVN上，此时应怎么做呢？右键选择版本库浏览器。在相应目录下，右键，加入文件/加入文件夹，选择相应目录即可。



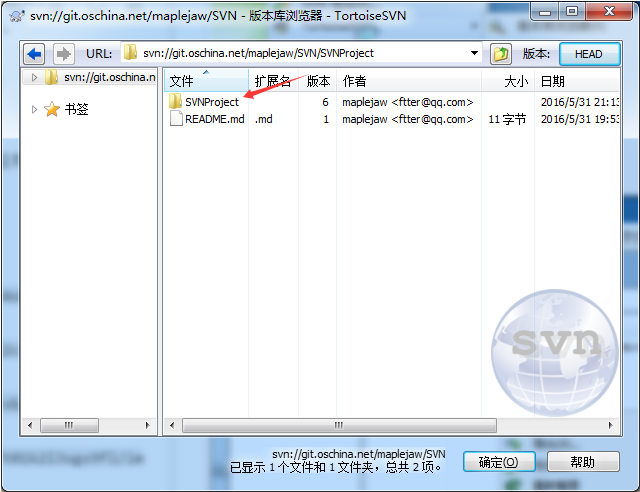
比如我现在有个项目叫SVNProject，我想把它传到SVN上。那么我只需选择加入文件夹即可。



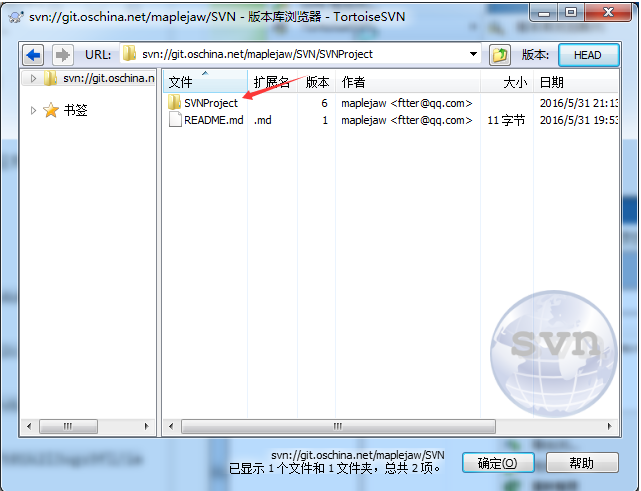
务必要输入提交信息。这样别人才能知道你干了什么。



导入成功就能看到目录。

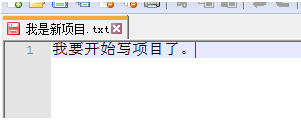


但是，不要以为导入成功就可以了。你还得重新检出，重新检出的项目才是受SVN控制的，务必记得检出。



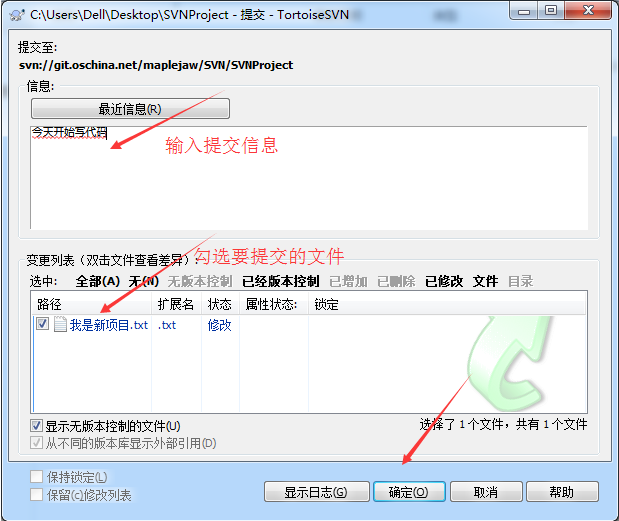
在SVNProject上右键检出到本地，然后在里面进行修改。现在就可以愉快的工作了。 检出过后的右键菜单变成了这样。

**提交：**绿色表示当前文件没有被修改过（看不见颜色的重启下电脑就好了）。假如我现在在我是新项目.txt中加了一行字，然后保存。

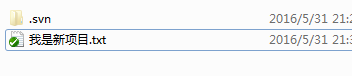


发现现在变成了红色，红色表示已修改。怎么提交修改？ 在根目录下，右键选择提交。

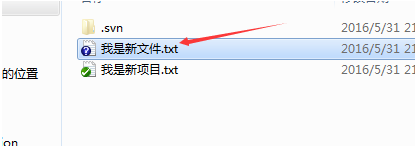
务必记得输入提交信息（虽然不输入也能提交），提交信息可以方便日后查看。



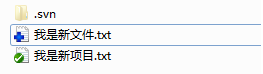
提交完毕后，可以发现又恢复到了绿色。



假如现在加入了一个新文件。可以看出是蓝色的。蓝色表示不属于版本库的未知文件，未知文件是不能提交的。



记住选择增加把它加入到版本库里面去。增加完毕后，变成了蓝色加号，表示新增加的版本库文件。



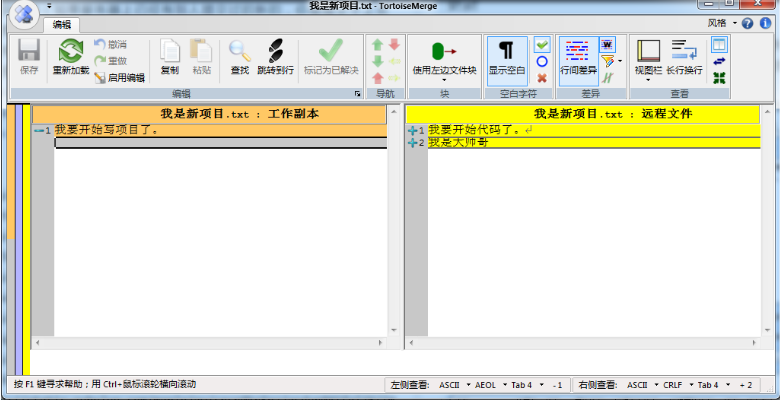
接下来，只需写代码，然后提交即可。删除文件也应该右键提交，如下。记得随时检查你的文件状态，如果没有添加到版本控制里要及时添加进去，不然你的文件提交不上去。

**更新：**假如你和B同学在协作。B同学写完代码提交到了SVN上，如果你想获取最新修改，就需要选择更新（如果服务器上已经有别人提交过的新的，你是提交不上去的，必须先更新再提交）。   
 怎么知道服务器有没有更新？你可以直接选择更新，有没有更新一下就知道。或者右键检查修改，然后检查版本库，就能看到服务器上改了哪些文件。

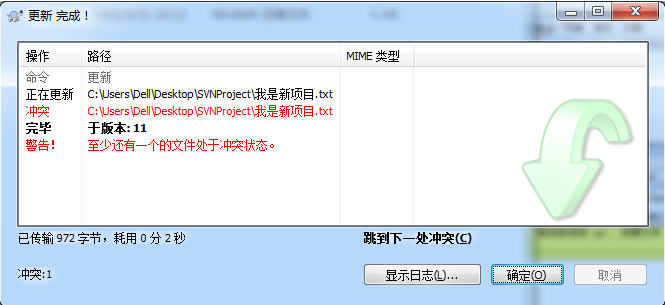


右键选择HEAD和BASE比较。

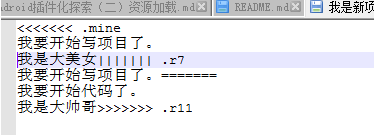
左边的表示你的代码，右边的表示服务器上的代码。



如果有修改记得及时更新到本地然后再继续工作。但是有时候更新会冲突，比如你和服务器上的改了同一个地方。 这时候你需要更新下来解决冲突。

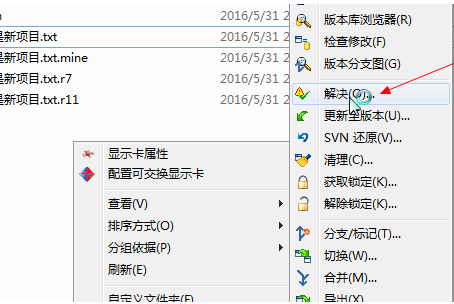


它会提示你哪个文件冲突，你只需打开那个文件，按照需求解决冲突即可。



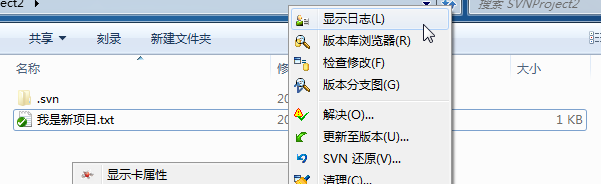
<<<<<<.mine到====表示你的代码，其他表示服务器的代码。你只需改成你想要的。

然后选择解决，告诉SVN我已经解决冲突了就行了。

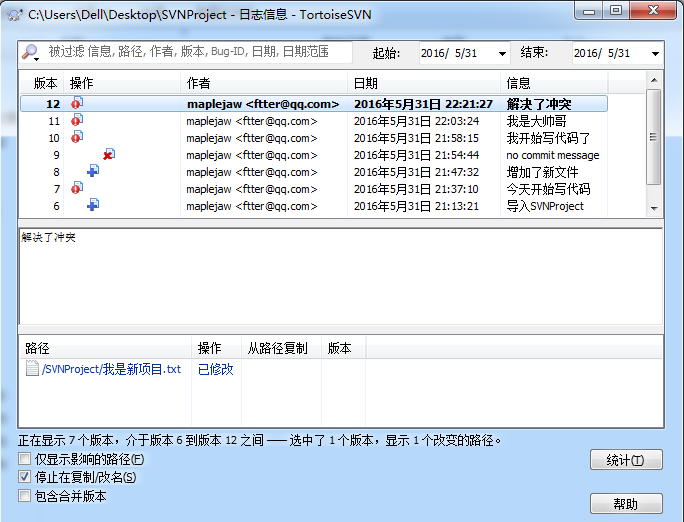


剩下的就是团队协作间的更新提交操作，这里不做赘述。

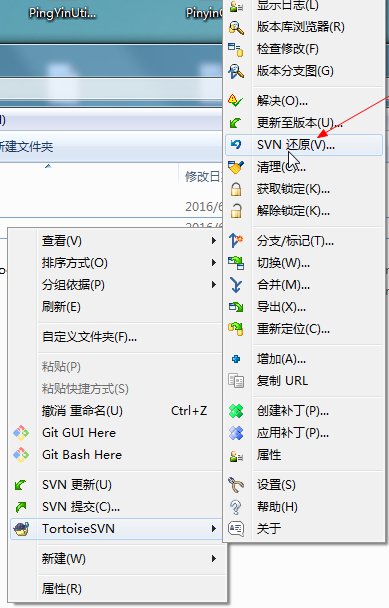
查看日志：选择显示日志，可以看出团队里面的人干了什么。



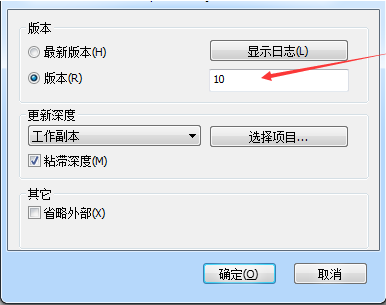
可以看出谁谁谁，什么时间，干了什么事。最后那一列信息是自己提交的时候写的。建议大家提交时务必要填写提交信息，这样别人一看就知道你干了什么。提交信息对于自己也是有好处的，时间长了也能看到当初做了什么。



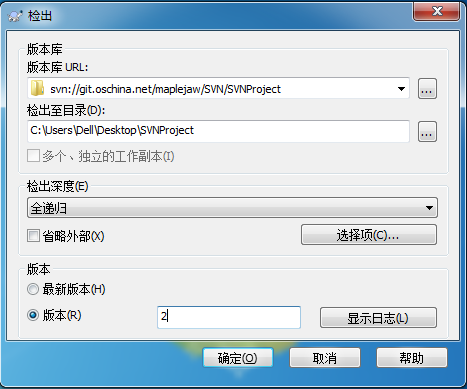
**版本回滚：**如果你改了东西，但是还没有提交，可以使用还原功能。



但是如果我们写错了东西并且提交了上去怎么办？通过版本回滚可以将文件恢复到以前的版本。右键更新至版本，通过查看日志来选择版本，然后回滚即可。



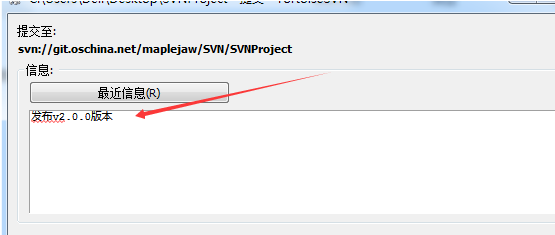
有时候我们需要查看以前版本的代码。此时我们可以新建个文件夹检出到指定版本。



注(网上不同意见，要确认解决)：如果要回滚到某个版本应该在本地库show log后选中要回退的某条记录右键选中 还原到这个版本，这样本地commit时才能识别到这些修改，右键更新至版本是无法提交到svn远程路径的。

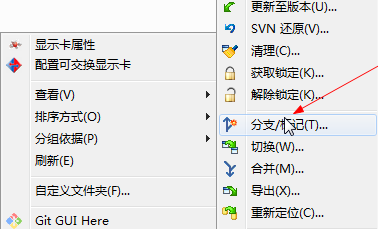
**版本控制：**版本控制有好几种方法，如下：

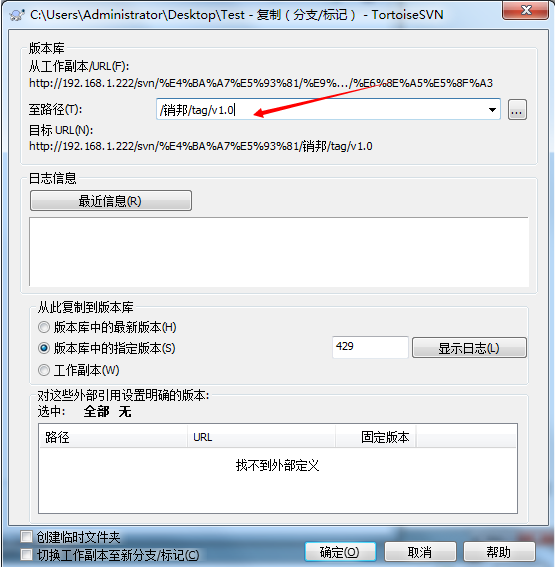
1.在提交发布版本时添加版本信息，这是最简单的一种方法。



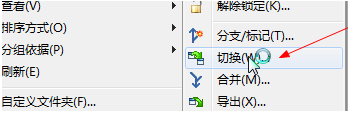
2. 打标签

每次发布版本时应该打标签。右键选择分支/标记。在至路径以版本号打上标签即可 。





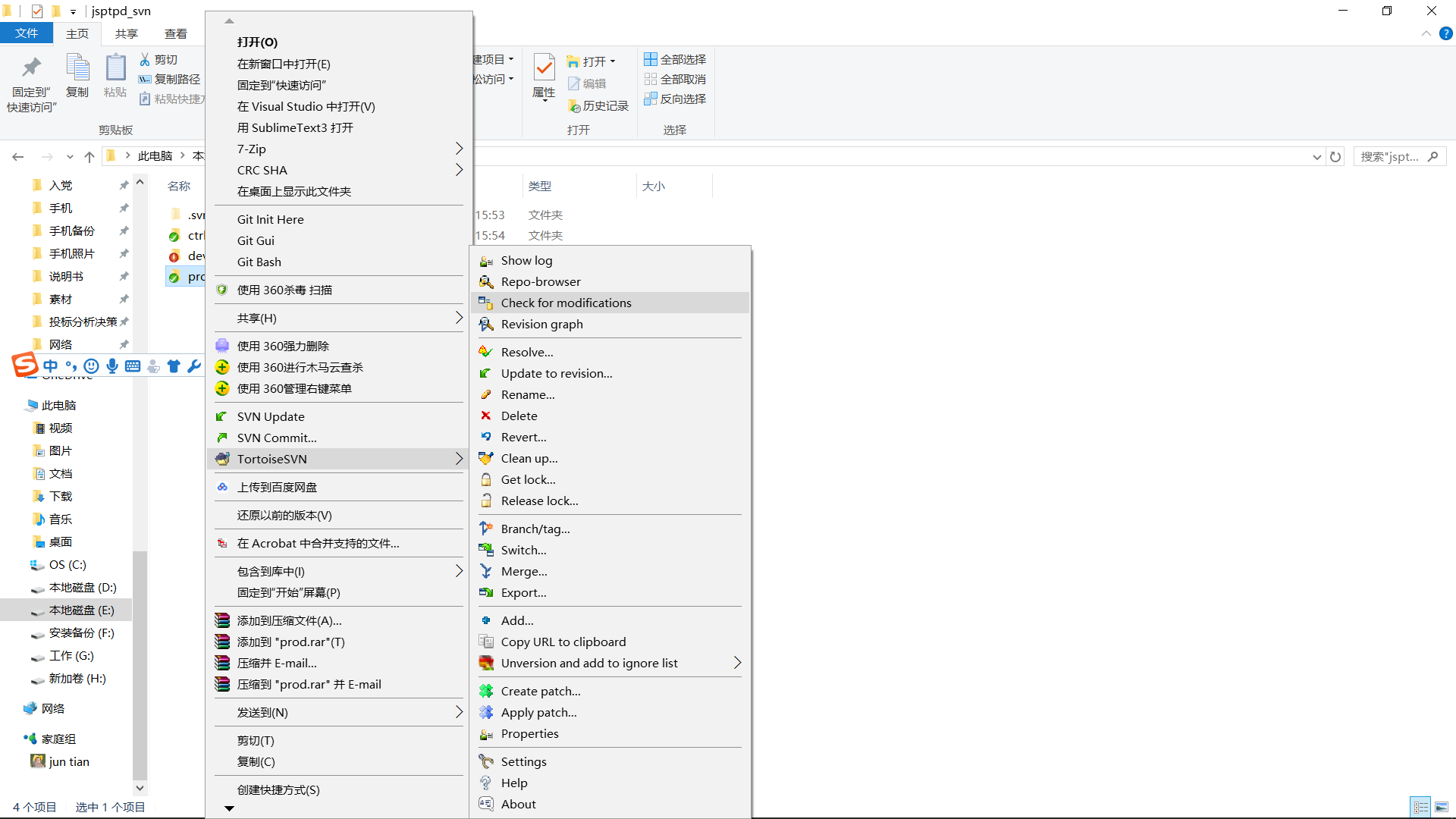
这样你就有了一个v1.0版本的标签。以后如果你想查看某个版本的代码，只需切换过去就行 。



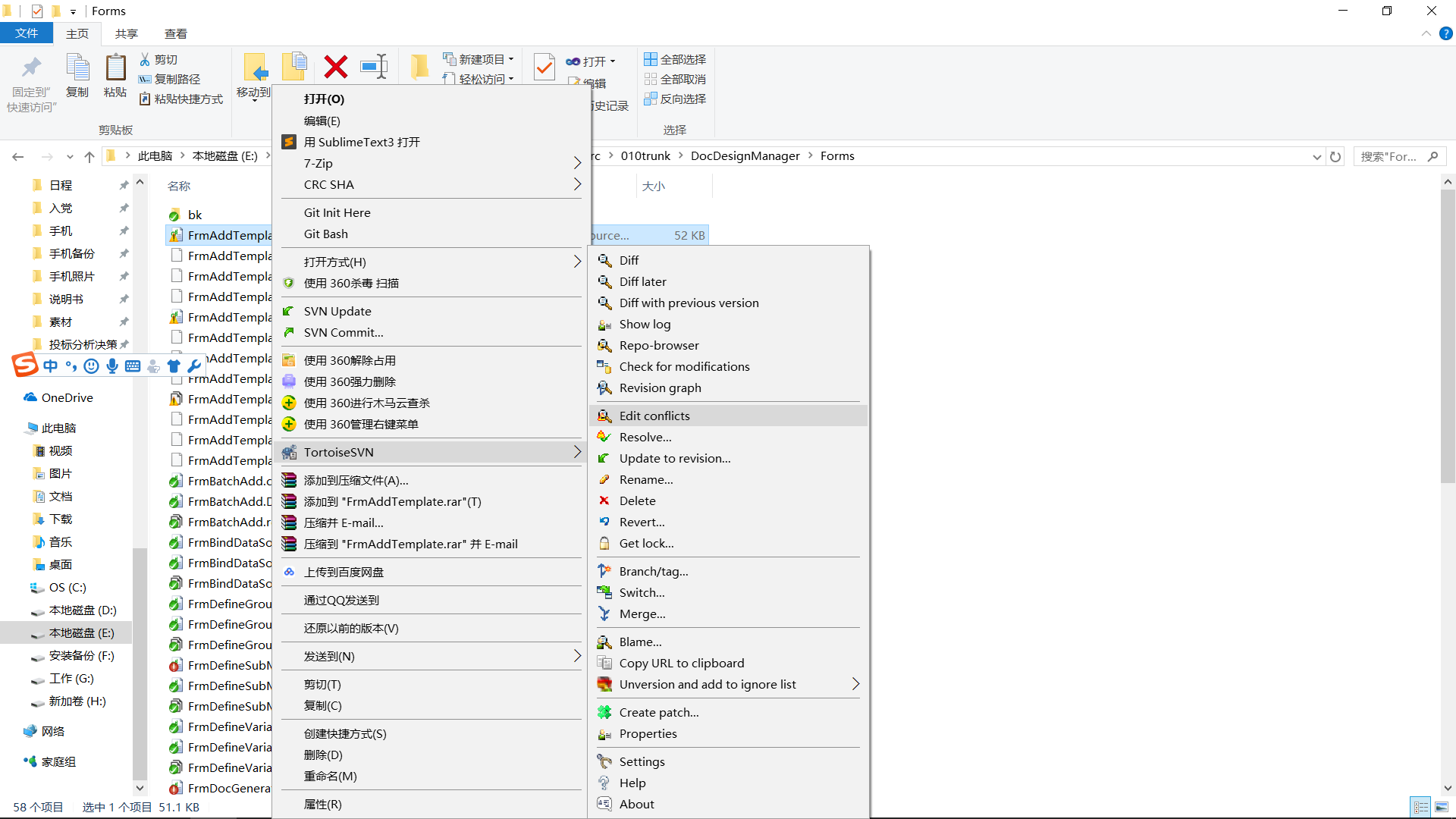
教程可见：<http://blog.csdn.net/maplejaw_/article/details/52874348>。

**删除文件：**选中要删除的文件，右键TortoiseSVN-->Delete,最后在执行一下commit(所在文件夹)。

确认不一样的



查看冲突



# 二、git：

## 使用的基本方法

1、

git status

git add .

git commit -m 'committest'

git push origin develop

通过.ignore可以忽略某个目录

.ignore文件添加:

.vs/

2、如果误传上去了，就删除目录：删除.vs目录

git rm -r --cached .vs

git commit -m '删除.vs'

git push origin master

3、处理冲突：

一般先commit，然后再pull，发现冲突。

回到项目里修改，修改完成后，

再commit，然后再pull。

4、创建release:

<https://www.cnblogs.com/kingBook/p/8117861.html>

5、常识

发布的版本release。

如果后续开发，发现现有版本有bug。

则从release出新开一个分支，修改，最后再和master合并。

可以下载对应工具进行管理，如VS就自带git管理。先选更改，全部提交或提交缓存，记得先把未修改的窗口关掉，然后拉取最新版本，如有冲突，处理冲突。没有冲突，就提交合并版本，就是merge一长传url的那个。

## 命令行

目录右键“Git Init Here”，该目录根下会多出一个.git的目录，用来存放git的一些信息，比如HEAD是版本的指针。

基本推送：

git add .

git commit -m "add text"

git push

尽量在分支上做，推送到master。

主要常见的命令：

git status

git add text.txt //添加文件到暂存区

git status

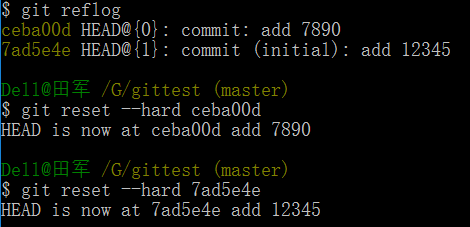
git commit -m "add text"     //提交到本地仓库

git push   //本次仓库推送到远程

git config --global user.name "13951601173"

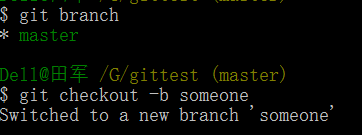
git config --global user.email "13951601173@139.com"

git reflog //查看日志



从gitlab上下载代码：

git clone http://172.16.11.71:31584/jsptpd/simpleapi.git



git checkout -b someone  新建一个分支并切换到它

git checkout someone  切换到someone分支

# 列出所有本地分支

**$ git branch**

# 列出所有远程分支

$ git branch -r

# 列出所有本地分支和远程分支

**$ git branch** **-a**

# 新建一个分支，但依然停留在当前分支

**$ git branch** **[branch-name]**

# 新建一个分支，并切换到该分支

**$ git checkout** **-b** **[branch]**

# 新建一个分支，指向指定commit

$ git branch [branch] [commit]

# 新建一个分支，与指定的远程分支建立追踪关系

$ git branch --track [branch] [remote-branch]

# 切换到指定分支，并更新工作区

**$ git checkout** **[branch-name]**

# 切换到上一个分支

**$ git checkout** **-**

# 建立追踪关系，在现有分支与指定的远程分支之间

$ git branch --set-upstream [branch] [remote-branch]

# 合并指定分支到当前分支

$ git merge [branch]

# 选择一个commit，合并进当前分支

$ git cherry-pick [commit]

# 删除分支

**$ git branch** **-d** **[branch-name]**

# 删除远程分支

$ git push origin --delete [branch-name]

$ git branch -dr [remote/branch]

## GUI

### 基本使用

☆如何在本地创建版本库呢？

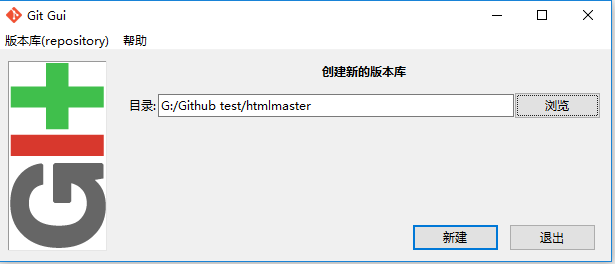
(1)右键”Git Init Here”:在目录中创建.git隐藏目录。



(2)上步也可以不做，直接”Git Gui”。



(3)点击“创建新的版本库”。



(4)选择具体的目录，该目录下就会多一个.git隐藏目录，和(1)一样。

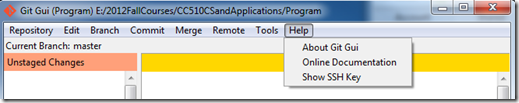


☆那么本地项目怎么和Github进行连接呢？

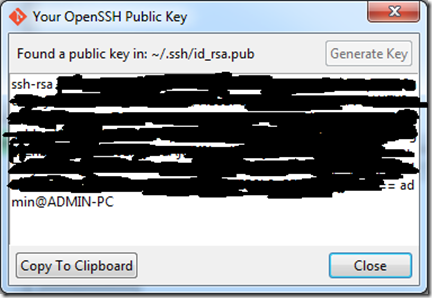
可以在有.git隐藏目录的文件夹中右键选择”Git Gui”。

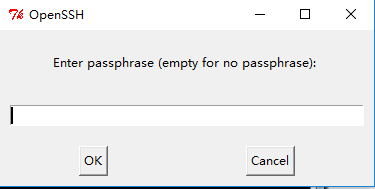
我们第一步就先让Github知道我们本地主机的存在。Github通过SSH key来对上传者进行认证。我们首先就要生成本地的SSH key。

在Git Gui中，点击help选择show SSH key项，

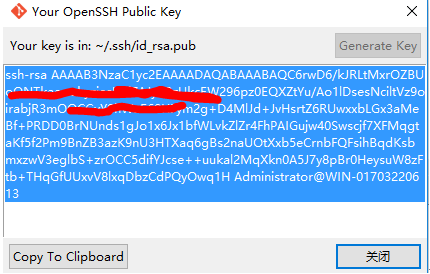
[](http://images.cnblogs.com/cnblogs_com/ResearchRecord/201210/201210231217361630.png)

然后单击Generate Key生成本机的SSH Key，并复制该key值。

[](http://images.cnblogs.com/cnblogs_com/ResearchRecord/201210/201210231217385892.png)

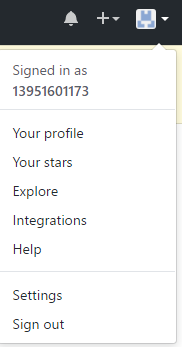


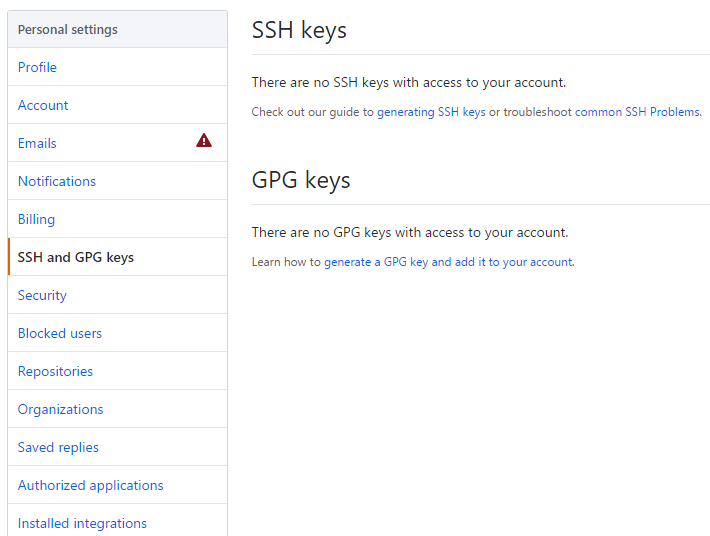
我这边默认：123456

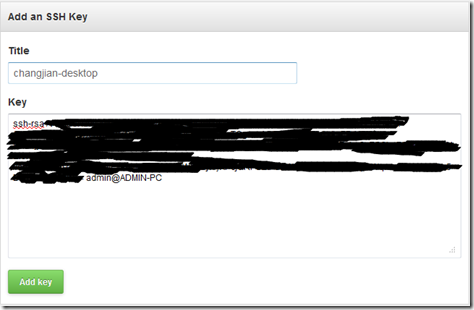


生成成功。

登录网页版git,选择“Settings”。

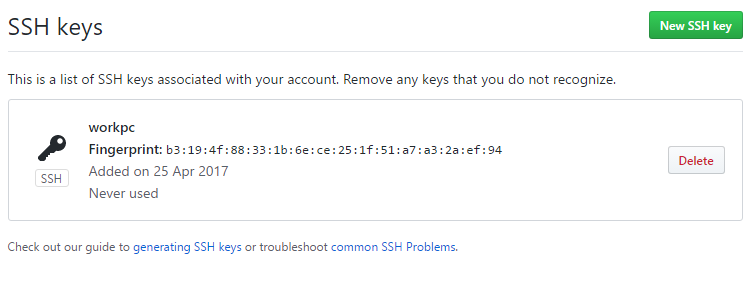


 单击Add SSH key进行Key值的添加。在出现的对话框中，添加该Key值的title（可以任意输入），在key值一栏中，输入刚刚生成的SSH Key值。然后单击Add key，该SSH Key值就被添加到你的github帐号中了。

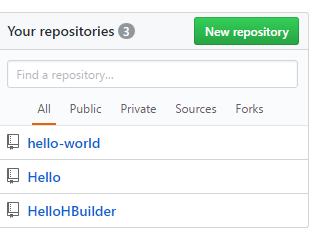
[](http://images.cnblogs.com/cnblogs_com/ResearchRecord/201210/201210231217422672.png)

这时SSH Keys下面就会显示刚刚添加的Key值，如果多台电脑同时进行项目的操作，那么每个电脑都要添加自己的key值，以便能够让github能够接受相应机器提交的代码。

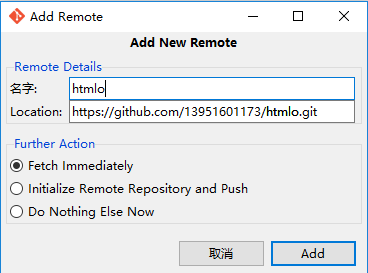
如下图：

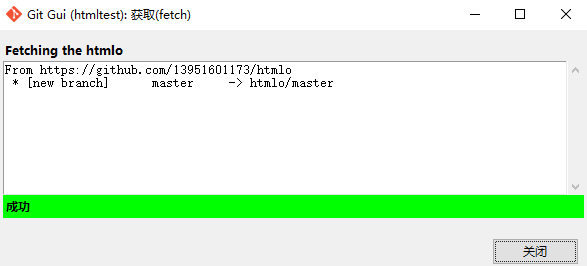


☆现在github已经能够接受本地主机提交的代码了，那么怎么让本地主机将代码正确的提交给github呢？



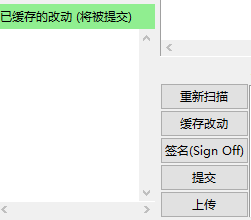
在网页新建一个代码仓库。这里是htmlo.git。



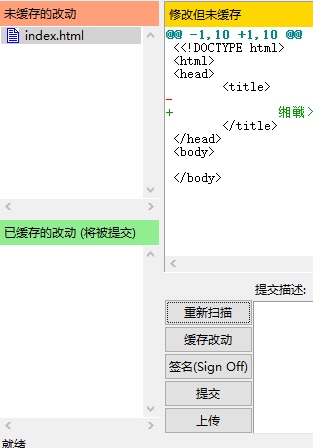


在本地目录中创建index.html,也就是说对本地文件进行修改，

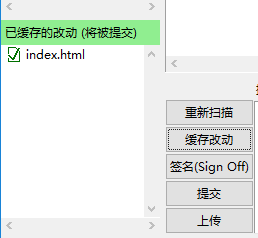
点击“重新扫描”，



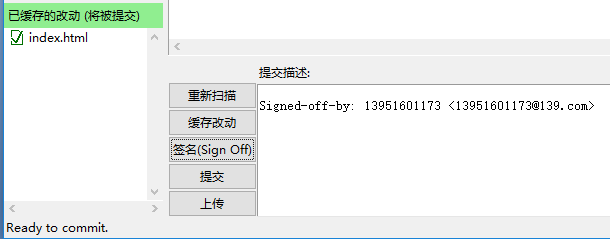
扫描的结果如下：



发现有文件发生变化了，点击“缓存改动”，结果如下：

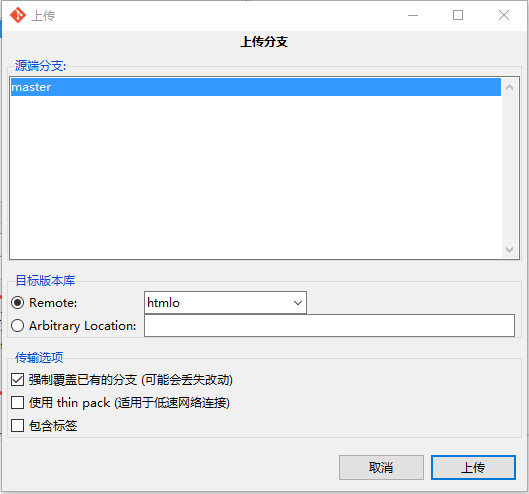


点击“签名”，

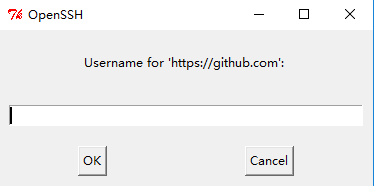


点击“提交”，即将修改的文件提交到本地仓库了，注意是本地仓库。

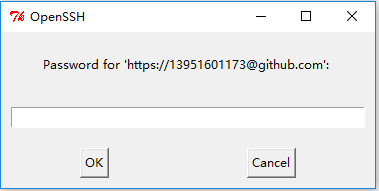
最后这一步是必须联网的，点击“上传”，



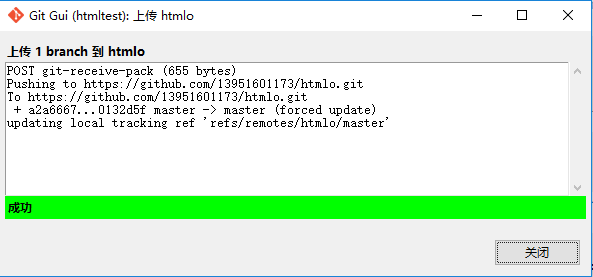
“传输选项”中至少选择一项，



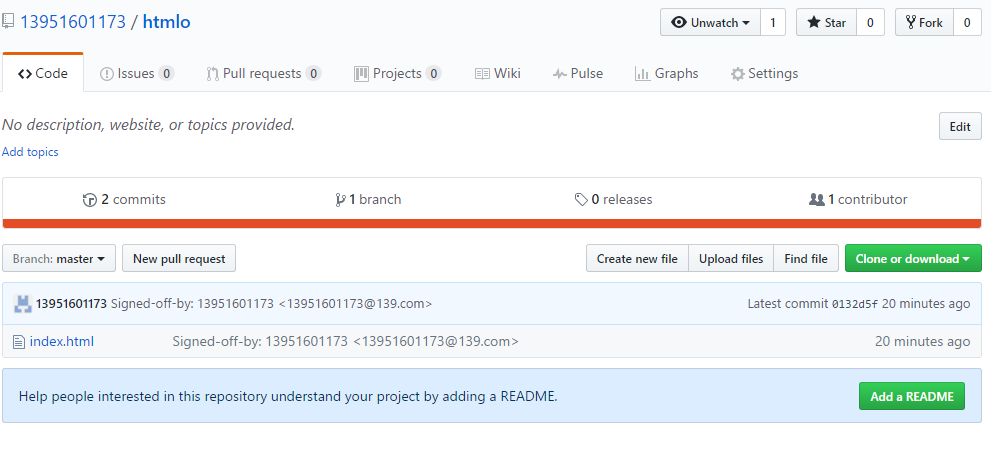
输入git的用户名，



输入git的密码，



上传成功。



## 命令行-切换当前用户

git config --global user.name "tianjun"

git config --global user.email "[tianjun@jsptpd.com](mailto:tianjun@jsptpd.com)"

## 命令行-查看当前用户

git config user.name

git config user.email

## 命令行-删除当前用户

git config --global --unset user.name zhang1

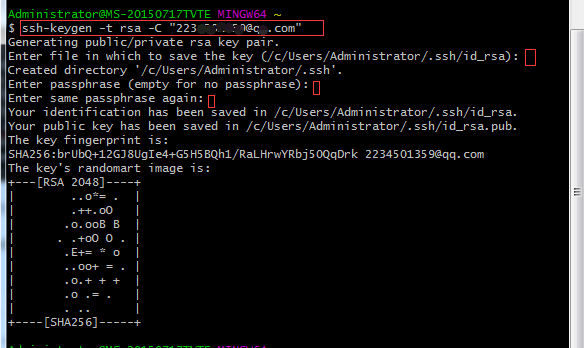
## SSH Key

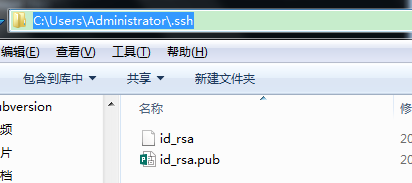
1）创建SSH Key

windows默认是在C:\Users\Dell\.ssh下，git bash后输入：

ssh-keygen -t rsa -C "your\_email@youremail.com"

执行这条命令会如上图提示文件保存路径，可以直接按Enter，然后提示输入密码，输入两次（可以不输直接两次Enter），执行完之后会在 .ssh 目录生产两个文件：id\_rsa和id\_rsa.pub





windows是打开id\_rsa.pub

用记事本打开.ssh目录下的id\_rsa.pub文件，复制里面的内容，到github和码云主页上找到的SSH key的信息，添加即可。

## 解决Enter passphrase for key

设置时，回车默认是空。有时密码会忘掉。cicdi的id\_rsa是nj02585890465

重新生成即可解决该问题。

## gitlab上已有项目，本地没有项目

gitlab上已有项目，但本地没有。

git clone <ssh://git@172.16.11.71:30022/ZKZS-Web-Platform/AutoDesignTool.git>

cd AutoDesignTool

touch README.md

git add README.md

git commit -m "add README"

git push -u origin master

## 本地已有项目目录但不是git项目，但gitlab是空项目

gitlab上创建一个新项目。

cd existing\_folder

git init

git remote add origin <ssh://git@172.16.11.71:30022/ZKZS-Web-Platform/AutoDesignTool.git>

git add .

git commit -m "new-project"

git push -u origin master

## 本地已有git项目目录

cd existing\_repo

git remote add origin <ssh://git@172.16.11.71:30022/ZKZS-Web-Platform/AutoDesignTool.git>

git push -u origin --all

git push -u origin --tags

## 多传文件/文件夹上去，需要删除

git一般都是先拉再删再推。

在github上只能删除仓库,却无法删除文件夹或文件, 所以只能通过命令来解决

$ git pull origin master                    # 将远程仓库里面的项目拉下来

$ git rm -r --cached .idea              # 删除.idea文件夹

$ git rm -r --cached 素材              # 删除素材文件夹

$ git rm test.txt              # 删除文件

$ git commit -m '删除.idea和素材'        # 提交,添加操作说明

$ git push -u origin master               # 将本次更改更新到github项目上去

## 配置多用户

1、通常，我们通过ssh的方式连接到Github/Gitee/GitLab服务，ssh连接需要一对秘钥。因此，如果想在同一台电脑上绑定不同的Github/GitLab/Gitee帐号，需要满足以下条件:

能够为每个账号生成私钥/公钥对；

在提交或更新代码时，可以区分不同账户，推送代码到相应的仓库或从相应的仓库拉去代码；

具体方案如下：

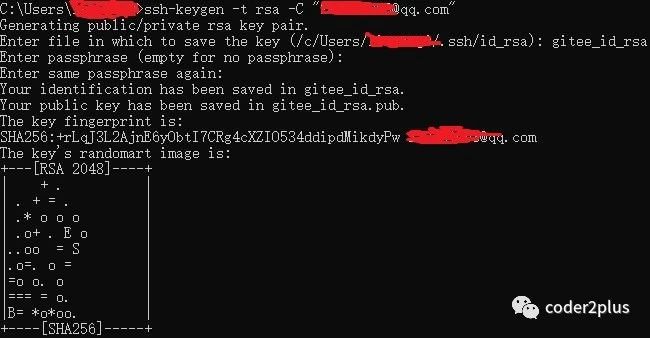
生成私钥/公钥时，密钥文件命名避免重复；

设置不同Host对应同一HostName，但密钥不同；

取消 git全局用户名/邮箱设置，为每个仓库独立设置用户名/邮箱；

Git使用的是SSH加密协议，Gitee与Github网站里，用户设置中需要配置相应的SSH keys。通常，需要管理多少个账户，一般就需要生成多少套SSH keys。

2、在.ssh文件夹下打开Git Bash或CMD窗口，输入下面指令生成第一个SSH key（以gitee环境举例，github环境类似）：

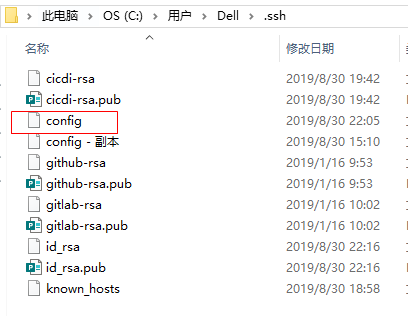


公钥可命名为gitee\_id\_rsa（公钥名需根据环境进行区分），密码为空或者输入，为空直接点击两次回车。

我一般rsa文件的密码设置为nj02584050729

按上面的方法生成rsa文件。

3、配置config文件，记住：文件名+后缀名就是config，其他什么也没有。



# gitlab

Host gitlab.com

HostName 172.16.11.71:30022

PreferredAuthentications publickey

IdentityFile ~/.ssh/cicdi-rsa

User tianjun

# github

Host github.com

HostName github.com

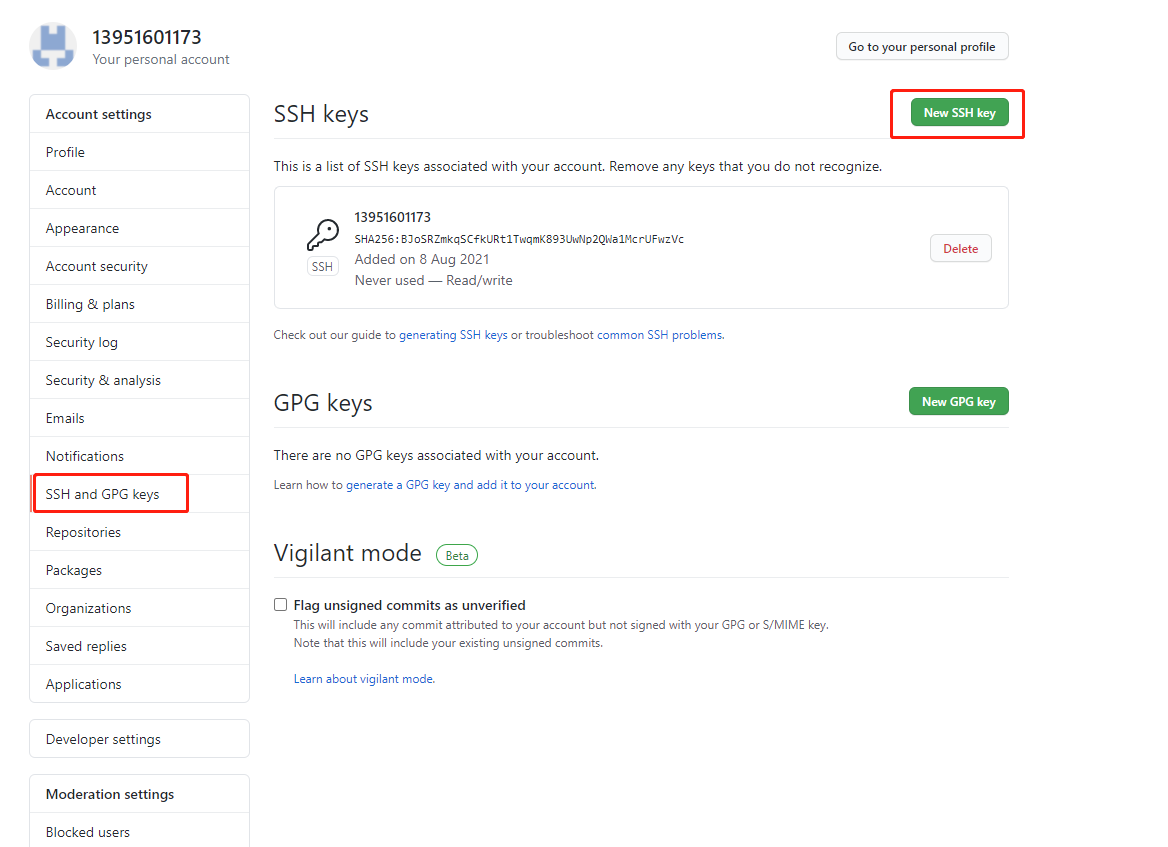
PreferredAuthentications publickey

IdentityFile ~/.ssh/github-rsa

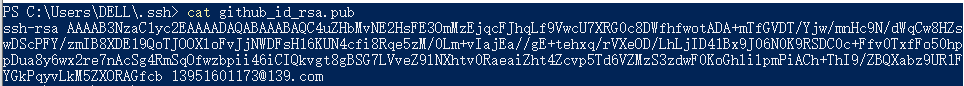
User [13951601173@139.com](mailto:13951601173@139.com)

HostName请使用域名，使用IP+端口目前貌似不行。

4、把github与gitee对应的公钥对应的公钥上传到服务器



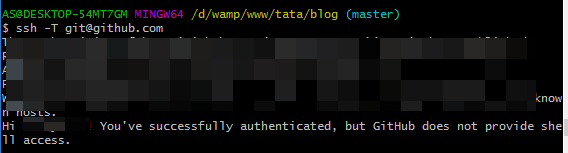
拷贝的就是对应id\_rsa.pub文件里的内容：

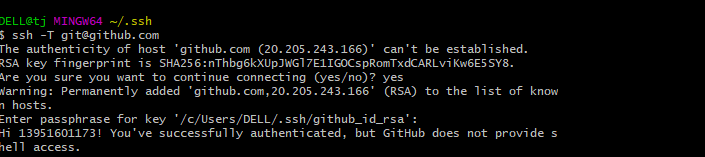


5、然后需要检测密钥对是否可以使用,github.com就是HostName：

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

如果出现以下信息，代表已经可以使用此密钥对：





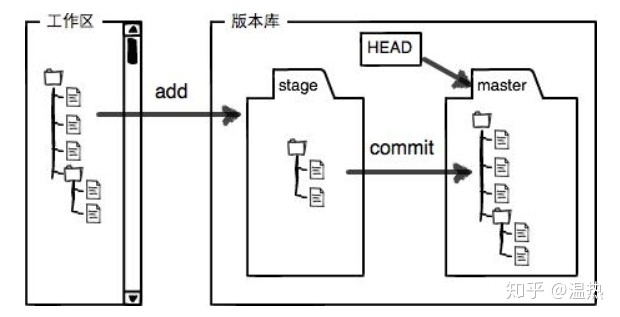
## 查看当前的remote url

git remote -v

## 暂存区和工作区

Git和其他版本控制系统如SVN的一个不同之处就是有暂存区的概念。工作区就是你在电脑里能看到的目录。

工作区有一个隐藏目录，这个不算工作区，而是Git的版本库。Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。工作区有一个隐藏目录，这个不算工作区，而是Git的版本库。Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。

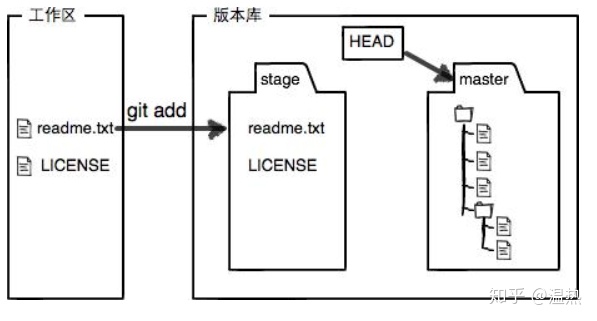


第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；

第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。

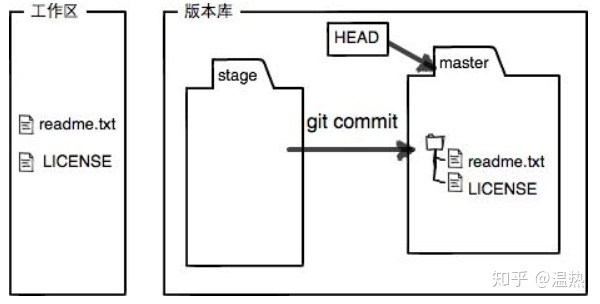
因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

当我们新增一个文件在目录里面时，就是在工作区增加了一个文档，使用命令git add，把readme.txt和新增文档添加后，暂存区就变成这样：



所以，git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行git commit就可以一次性把暂存区的所有修改提交到分支。

现在版本库变成了这样，暂存区就没有任何内容了：



## stash

正在开发一个项目（此时在分支A上），但是要临时切换到别的项目 (分支B)，但是A的工作还没有全部完成，不想提交，在分支A上做的修改如果没有add或者commit，切换分支就会不允许，此时需要将刚刚修改的文件保存到缓存区，完成B再回来A上开发的时候，使用git apply应用你缓存的内容。

git stash #将文件添加到暂存区，此时工作区干净了，新增的文件需要 git add才能加入stash list中，否则git stash无效。

git stash list

想切换到A，并且应用缓存内容的话：

git stash apply stash@{0} #将最新一次暂存的内容，应用到A分支上，stash{0}表示最新的一次暂存，stash{1}表示次新，以此类推。

## 代码回滚

想要让Git回退历史，有以下步骤：

使用git log命令，查看分支提交历史，确认需要回退的版本

使用git reset --hard commit\_id命令，进行版本回退

使用git push origin命令，推送至远程分支

快捷命令：

回退上个版本：git reset --hard HEAD^

注：HEAD是指向当前版本的指针，HEAD^表示上个版本,HEAD^^表示上上个版本。

如果修改到的文件比较少，我们可以不通过命令回滚的方式，手动删除之前的修改，再进行提交。

为了验证Bug，我们可能会直接在测试服务器上打断点调试。如果忘记去掉调试内容，在执行git pull更新时，Git会提示你提交修改。此时，你可能已经不记得修改什么了，这个时候，我们可以用git checkout -- file命令，来清空工作区的修改。是的，git checkout命令不仅可以用来切换分支，还能撤销文件修改。

如果想要撤销提交到暂存区后的文件内容怎么办呢(即执行git add之后)，我们可以使用git reset HEAD file命令撤销提交到暂存区的内容,再使用git checkout -- file命令来撤销工作区的修改，需要分两步进行操作。

## GitLab

### 项目可见级别

提供Private（私有的，只有你自己或者组内的成员能访问）/Internal（所有登录的用户）/Public(公开的，所有人都可以访问)三种选项。

### 添加和配置SSH公钥

SSH（Secure Shell）是一种安全协议，在你的电脑与GitLab服务器进行通信时，我们使用SSH密钥（SSH Keys）认证的方式来保证通信安全。

### 如何忽略这些文件和文件夹的变更信息

在git管理的项目文件夹中（严格的讲，应该叫做git的本地repository），创建一个文件名为“.gitignore”的纯文本文件。

### 找回本地误删的文件

不小心把本地的文件删除了一个？想从仓库git pull 下拉？对不起，这是不行的。

通过git status+git reset可恢复。

### 完整上传一个新的项目

登录gitlab，创建项目（注：删除项目是在项目的Setting里）。

填写项目名，如果需要添加其他人的权限，则Create Group，给Group起个好名字,将用户加入Group。

创建完成后，gitlab会提供具体的操作步骤，如下：

##### 步骤一：Git global setup

git config --global user.name "tianjun"

git config --global user.email "[tianjun@cicdi.com](mailto:tianjun@cicdi.com)"

##### 步骤二：Create a new repository 适用于远程仓库有项目，本地无的情况

git clone <http://tianjun@172.16.11.71:31584/SmartDesignWeb/SmartDesignWebBack.git>

cd SmartDesignWebBack

touch README.md

git add README.md

git commit -m "add README"

git push -u origin master

##### 步骤二：Existing folder 适用于项目在本地，添加到远程仓库的情况，如果项目是从别人git上取下，想消除之前的git记录，或者目录里压根没有.git，则git init。

cd existing\_folder

git init

git remote add origin <http://tianjun@172.16.11.71:31584/SmartDesignWeb/SmartDesignWebBack.git>

git add .

git commit

git push -u origin master

##### 步骤二：Existing Git repository 适用于项目在本地，添加到远程仓库的情况

cd existing\_repo

git remote add origin <http://tianjun@172.16.11.71:31584/SmartDesignWeb/SmartDesignWebBack.git>

git push -u origin --all

git push -u origin --tags

git push -u origin master后以后每次都只要git push

### 本地删除文件如何提交

git status

git add .

git commit -m "delete files"

git push

### 本地文件冲突如何解决

在服务器上git pull是出现以下错误：rror: Your local changes to the following files would be overwritten by merge:

说明代码冲突了。

可以使用git diff来查看代码不同之处。

如果希望用代码库中的文件完全覆盖本地工作版本. 方法如下:

git reset --hard

git pull



如果希望保留生产服务器上所做的改动,仅仅并入新配置项:

git stash

git pull

git stash pop

### 工作区、缓存区、本地库

实际上前三个名词对应到实体，都是你从远程仓库克隆下来的那个项目目录！  
只不过工作区就是你实际看到的目录和目录里的内容，当你修改/添加/删除了从远程仓库clone下来的目录中的文件后，工作区就变了。

而缓存区也是你实际看到的这个目录，但你要 git add . （注意还有个.表示所有文件）之后在工作空间中的修改才能保存到缓冲区中  
而本地库也是你实际看到的这个目录 ，但你要git commit -m "版本名" 缓冲区中的更改才能保存到本地库中

而远程库当然就是远程服务器上那个项目目录啦，你要git push -u origin master 本地库的修改才能保存到你下载这个本地库（本地项目）的远程库（远程项目）中！

### 一个复杂操作的例子

实际工作中大体也会用到这样的工作流程：

1）开发某个网站。

2）为实现某个新的需求，创建一个分支。

3）在这个分支上开展工作。

假设此时，你突然接到一个电话说有个很严重的问题需要紧急修补，那么可以按照下面的方式处理：

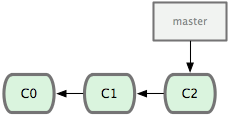
1）返回到原先已经发布到生产服务器上的分支。

2）为这次紧急修补建立一个新分支，并在其中修复问题。

3）通过测试后，回到生产服务器所在的分支，将修补分支合并进来，然后再推送到生产服务器上。

4）切换到之前实现新需求的分支，继续工作。

1、首先，我们假设你正在项目中愉快地工作，并且已经提交了几次更新（见下图）。



2、现在，你决定要修补问题追踪系统上的 #53 问题。顺带说明下，Git 并不同任何特定的问题追踪系统打交道。这里为了说明要解决的问题，才把新建的分支取名为iss53。要新建并切换到该分支，运行 git checkout 并加上 -b 参数：

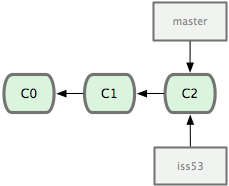
$ git checkout -b iss53

Switched to a new branch 'iss53'

这相当于执行下面这两条命令：

$ git branch iss53

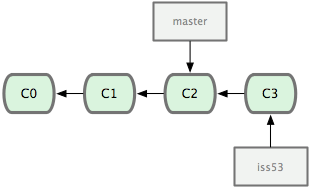
$ git checkout iss53



接着你开始尝试修复问题，在提交了若干次更新后，iss53 分支的指针也会随着向前推进，因为它就是当前分支（换句话说，当前的 HEAD 指针正指向 iss53，见下图）：

$ vim index.html

$ git commit -a -m 'added a new footer [issue 53]'



3、现在你就接到了那个网站问题的紧急电话，需要马上修补。有了 Git ，我们就不需要同时发布这个补丁和 iss53 里作出的修改，也不需要在创建和发布该补丁到服务器之前花费大力气来复原这些修改。唯一需要的仅仅是切换回 master 分支。

不过在此之前，留心你的暂存区或者工作目录里，那些还没有提交的修改，它会和你即将检出的分支产生冲突从而阻止 Git 为你切换分支。切换分支的时候最好保持一个清洁的工作区域。稍后会介绍几个绕过这种问题的办法（分别叫做 stashing 和 commit amending）。目前已经提交了所有的修改，所以接下来可以正常转换到 master 分支：

$ git checkout master

Switched to branch 'master'

此时工作目录中的内容和你在解决问题 #53 之前一模一样，你可以集中精力进行紧急修补。这一点值得牢记：Git 会把工作目录的内容恢复为检出某分支时它所指向的那个提交对象的快照。它会自动添加、删除和修改文件以确保目录的内容和你当时提交时完全一样。

3、接下来，你得进行紧急修补。我们创建一个紧急修补分支 hotfix 来开展工作，直到搞定（见下图）：

$ git checkout -b hotfix

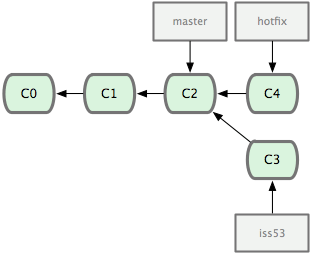
Switched to a new branch 'hotfix'

$ vim index.html

$ git commit -a -m 'fixed the broken email address'

[hotfix 3a0874c] fixed the broken email address

1 files changed, 1 deletion(-)



有必要作些测试，确保修补是成功的，然后回到 master 分支并把它合并进来，然后发布到生产服务器。用 git merge 命令来进行合并：

$ git checkout master

$ git merge hotfix

Updating f42c576..3a0874c

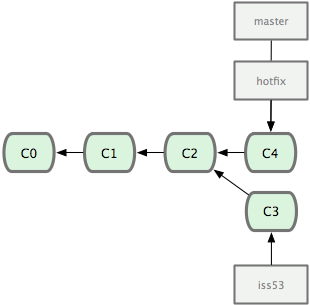
Fast-forward

README | 1 -

1 file changed, 1 deletion(-)

请注意，合并时出现了“Fast forward”的提示。由于当前 master 分支所在的提交对象是要并入的 hotfix 分支的直接上游，Git 只需把 master 分支指针直接右移。换句话说，如果顺着一个分支走下去可以到达另一个分支的话，那么 Git 在合并两者时，只会简单地把指针右移，因为这种单线的历史分支不存在任何需要解决的分歧，所以这种合并过程可以称为快进（Fast forward）。

现在最新的修改已经在当前 master 分支所指向的提交对象中了，可以部署到生产服务器上去了（见下图）。



合并之后，master 分支和 hotfix 分支指向同一位置。

在那个超级重要的修补发布以后，你想要回到被打扰之前的工作。由于当前 hotfix 分支和 master 都指向相同的提交对象，所以 hotfix 已经完成了历史使命，可以删掉了。使用 git branch 的 -d 选项执行删除操作：

$ git branch -d hotfix

Deleted branch hotfix (was 3a0874c).

现在回到之前未完成的 #53 问题修复分支上继续工作（图 3-15）：

$ git checkout iss53

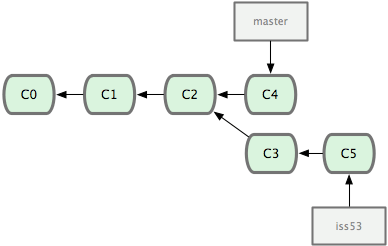
Switched to branch 'iss53'

$ vim index.html

$ git commit -a -m 'finished the new footer [issue 53]'

[iss53 ad82d7a] finished the new footer [issue 53]

1 file changed, 1 insertion(+)



iss53 分支可以不受影响继续推进。

值得注意的是之前 hotfix 分支的修改内容尚未包含到 iss53 中来。如果需要纳入此次修补，可以用 git merge master 把 master 分支合并到 iss53；或者等 iss53 完成之后，再将 iss53 分支中的更新并入 master。

在问题 #53 相关的工作完成之后，可以合并回 master 分支。实际操作同前面合并 hotfix 分支差不多，只需回到 master 分支，运行 git merge 命令指定要合并进来的分支：

$ git checkout master

$ git merge iss53

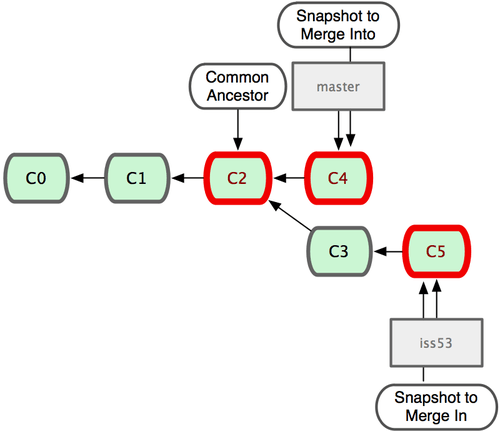
Auto-merging README

Merge made by the 'recursive' strategy.

README | 1 +

1 file changed, 1 insertion(+)

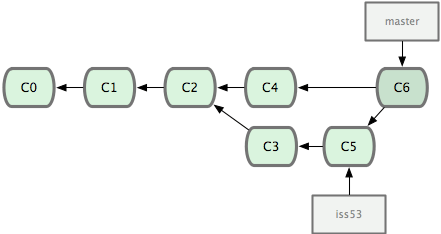
请注意，这次合并操作的底层实现，并不同于之前 hotfix 的并入方式。因为这次你的开发历史是从更早的地方开始分叉的。由于当前 master 分支所指向的提交对象（C4）并不是 iss53 分支的直接祖先，Git 不得不进行一些额外处理。就此例而言，Git 会用两个分支的末端（C4 和 C5）以及它们的共同祖先（C2）进行一次简单的三方合并计算。图 3-16 用红框标出了 Git 用于合并的三个提交对象：



Git 为分支合并自动识别出最佳的同源合并点。

这次，Git 没有简单地把分支指针右移，而是对三方合并后的结果重新做一个新的快照，并自动创建一个指向它的提交对象（C6）（见图 3-17）。这个提交对象比较特殊，它有两个祖先（C4 和 C5）。

值得一提的是 Git 可以自己裁决哪个共同祖先才是最佳合并基础；这和 CVS 或 Subversion（1.5 以后的版本）不同，它们需要开发者手工指定合并基础。所以此特性让 Git 的合并操作比其他系统都要简单不少。



Git 自动创建了一个包含了合并结果的提交对象。

既然之前的工作成果已经合并到 master 了，那么 iss53 也就没用了。你可以就此删除它，并在问题追踪系统里关闭该问题。

$ git branch -d iss53

### 遇到冲突时的分支合并

有时候合并操作并不会如此顺利。如果在不同的分支中都修改了同一个文件的同一部分，Git 就无法干净地把两者合到一起（译注：逻辑上说，这种问题只能由人来裁决。）。如果你在解决问题 #53 的过程中修改了 hotfix 中修改的部分，将得到类似下面的结果：

$ git merge iss53

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

Git 作了合并，但没有提交，它会停下来等你解决冲突。要看看哪些文件在合并时发生冲突，可以用 git status 查阅：

$ git status

On branch master

You have unmerged paths.

(fix conflicts and run "git commit")

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: index.html

no changes added to commit (use "git add" and/or "git commit -a")

任何包含未解决冲突的文件都会以未合并（unmerged）的状态列出。Git 会在有冲突的文件里加入标准的冲突解决标记，可以通过它们来手工定位并解决这些冲突。可以看到此文件包含类似下面这样的部分：

<<<<<<< HEAD

<div id="footer">contact : email.support@github.com</div>

=======

<div id="footer">

please contact us at support@github.com

</div>

>>>>>>> iss53

可以看到 ======= 隔开的上半部分，是 HEAD（即 master 分支，在运行 merge 命令时所切换到的分支）中的内容，下半部分是在 iss53 分支中的内容。解决冲突的办法无非是二者选其一或者由你亲自整合到一起。比如你可以通过把这段内容替换为下面这样来解决：

<div id="footer">

please contact us at email.support@github.com

</div>

这个解决方案各采纳了两个分支中的一部分内容，而且我还删除了 <<<<<<<，======= 和 >>>>>>> 这些行。在解决了所有文件里的所有冲突后，运行 git add 将把它们标记为已解决状态（译注：实际上就是来一次快照保存到暂存区域。）。因为一旦暂存，就表示冲突已经解决。如果你想用一个有图形界面的工具来解决这些问题，不妨运行 git mergetool，它会调用一个可视化的合并工具并引导你解决所有冲突：

$ git mergetool

This message is displayed because 'merge.tool' is not configured.

See 'git mergetool --tool-help' or 'git help config' for more details.

'git mergetool' will now attempt to use one of the following tools:

opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecmerge p4merge araxis bc3 codecompare vimdiff emerge

Merging:

index.html

Normal merge conflict for 'index.html':

{local}: modified file

{remote}: modified file

Hit return to start merge resolution tool (opendiff):

如果不想用默认的合并工具（Git 为我默认选择了 opendiff，因为我在 Mac 上运行了该命令），你可以在上方"merge tool candidates"里找到可用的合并工具列表，输入你想用的工具名。我们将在第七章讨论怎样改变环境中的默认值。

退出合并工具以后，Git 会询问你合并是否成功。如果回答是，它会为你把相关文件暂存起来，以表明状态为已解决。

再运行一次 git status 来确认所有冲突都已解决：

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: index.html

如果觉得满意了，并且确认所有冲突都已解决，也就是进入了暂存区，就可以用 git commit 来完成这次合并提交。提交的记录差不多是这样：

Merge branch 'iss53'

Conflicts:

index.html

#

# It looks like you may be committing a merge.

# If this is not correct, please remove the file

# .git/MERGE\_HEAD

# and try again.

#

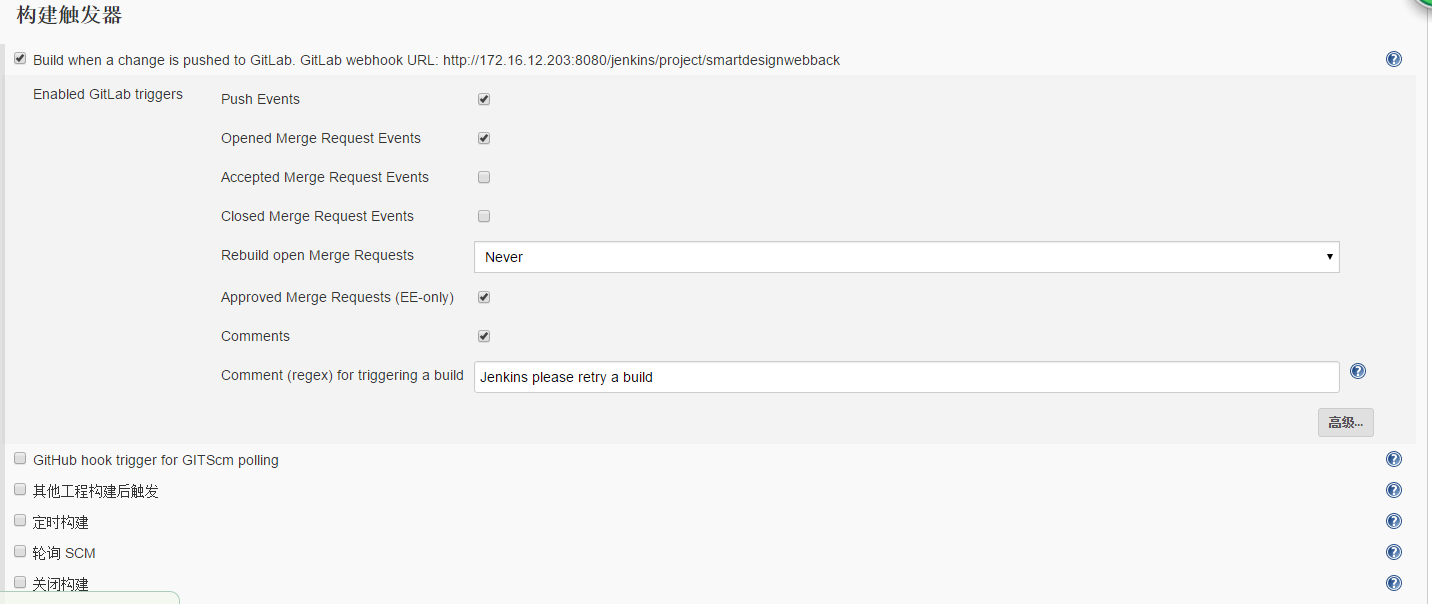
如果想给将来看这次合并的人一些方便，可以修改该信息，提供更多合并细节。比如你都作了哪些改动，以及这么做的原因。有时候裁决冲突的理由并不直接或明显，有必要略加注解。

参考：

https://git-scm.com/book/zh/v1/Git-分支-分支的新建与合并

https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/001375840038939c291467cc7c747b1810aab2fb8863508000

### jekins自动构建



### 常见问题解答

#### Lines starting # with '#' will be ignored

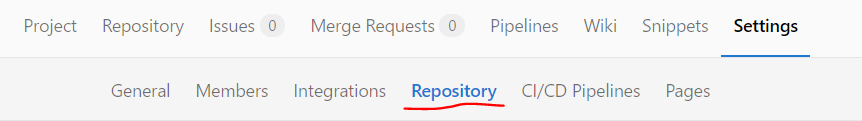
git提交的时候遇到# Please enter the commit message for your changes. Lines starting with '#' will be ignored, and an empty message aborts the commit.的问题，提示你说是因为代码中带有注释，需要去掉注释，但是完全不需要这样，在git commit文件时指令为git commit -m “文件名”，这样就可以将文件提交到github里面

然后git push -u origin master即可。

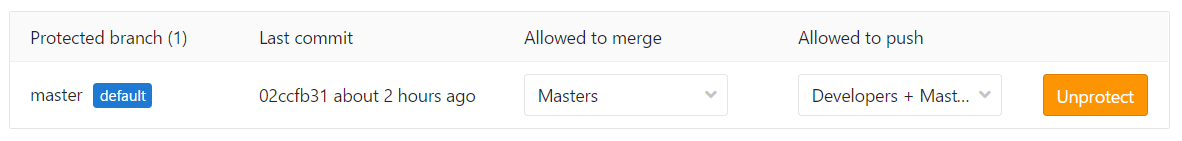
#### 除master外其他角色无法提交

权限的问题

进入gitlab中的对应项目，在Setting中的Repository中：



可以直接在下图的选择框里改，或者直接点击Unprotect。



这时没有任何保护项，可以再添加。

