Git 与 GitHub 的使用

孙雪 郑海永

2013 年 08 月 11 日

目录

1 通过 SSH 代理使用 GitHub

Git 可以使用四种主要的协议来传输数据：本地传输，SSH 协议，Git 协议和 HTTP 协议。下面简单介绍一下 SSH 如何通过 SSH 代理使用 GitHub。

由于实验室需要，必须通过实验室代理来实现 GitHub 与本地仓库的连接。通过设置 SSH 代理可以实现这一需求，从而完成从本地仓库上传材料到 GitHub 及从 GitHub 克隆仓库到本地等操作。

SSH 为 Secure Shell 的缩写，是建立在应用层和传输层基础上的安全协议，专为远程登录会话和其他网络服务提供安全性的协议，利用 SSH 协议可以有效防止远程管理过程中的信息泄漏问题。

SSH 利用加密的方式可以有效的避免“中间人”截获你和服务器之间的数据传输。所谓“中间人”的攻击方式，就是“中间人”冒充真正的服务器接收你传给服务器的数据，然后再冒充你把数据传给真正的服务器。

SSH 提供了一种基于密匙的安全验证。也就是你必须为自己创建一对密匙，并把公用密匙放在需要访问的服务器上。如果你要连接到 SSH 服务器上，客户端软件就会向服务器发出请求，请求用你的密匙进行安全验证。服务器收到请求之后，先在该服务器上你的主目录下寻找你的公用密匙，然后把它和你发送过来的公用密匙进行比较。如果两个密匙一致，服务器就用公用密匙加密“质询”并把它发送给客户端软件。客户端软件收到“质询”之后就可以用你的私人密匙解密再把它发送给服务器。

下面就来说明 SSH 密匙的设置及如何通过 SSH 代理使用 GitHub。

1.1 设置 SSH 密钥

1. 生成密钥。到 ~ 目录下打开终端执行`ssh-keygen -t rsa`，然后一直按 Enter 键，此时在 ~ 目录下的`.ssh`目录下生成了`id_rsa.pub`。

2. 把公共密钥保存到 GitHub 网站上。到 GitHub 的 Account setting 上

选择 SSH Keys 选项，点击 Add SSH key，把 title 写为`your name@iplouc`。把`id_rsa.pub` 里面的内容复制到 key 中，保存，如图1。

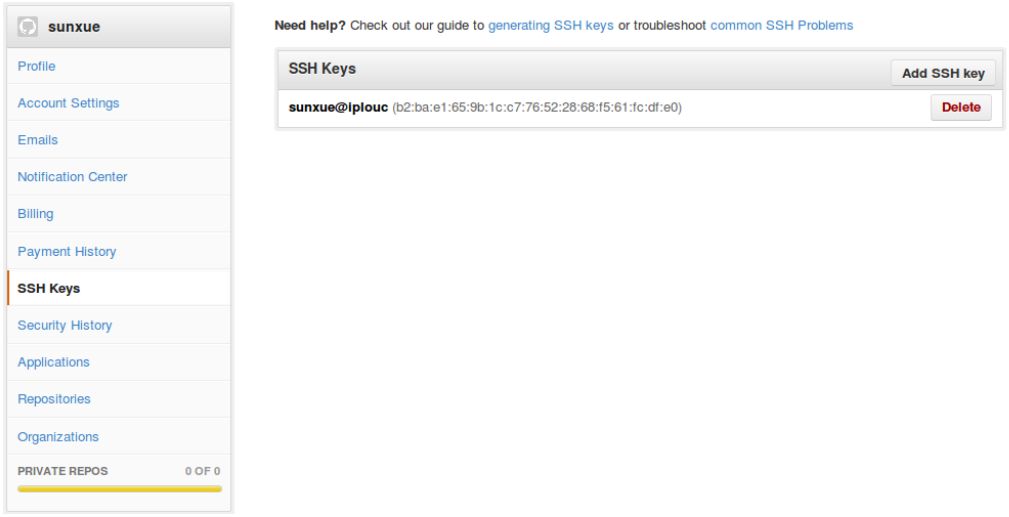

图 1: SSH Keys

3. 把密钥加载到 SSH 里。在终端输入`ssh-add`。

1.2 通过 SSH 代理使用 GitHub

1. ssh 配置文件。打开终端，执行：`vim ~/.ssh/config`，写入内容：

```
1 Host github.com
2 ProxyCommand ~/.ssh/ssh-https-tunnel %h %p
3 Port 443
4 Hostname ssh.github.com
```

2. 设置代理地址。执行`vim ~/.ssh/ssh-https-tunnel`，内容已在`https://github.com/zhenglab/LaTeX_Git-GitHub-howto`中公布，注意其中的`my $host` 和`my $port` 改为实验室代理地址和端口)。

3. 测试。在终端输入：`ssh -T git@github.com`，此时，若出现权限问题，如图2，则说明 ssh-https-tunnel 文件不可执行，此时，可以用命令：`chmod +x ~/.ssh/ssh-https-tunnel`来加上执行权限，用命令：`ls -al ~/.ssh/ssh-https-tunnel`来查看是否添加成功。

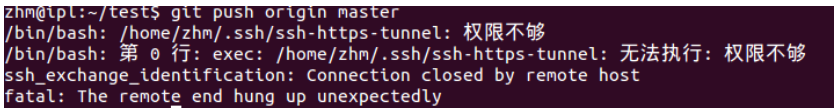

图 2: 权限问题

若出现如下密匙问题，如图3，则选择 yes，或参考网页 [7] 来解决。

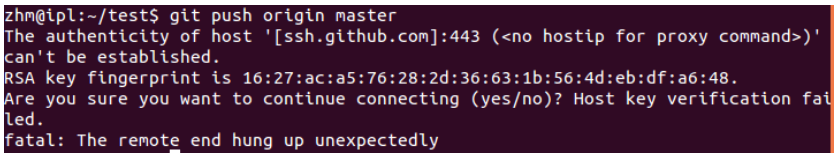

图 3: 密匙问题

若出现图4，则说明设置成功。

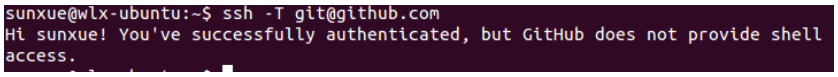

图 4: 设置成功

2 Git 的安装与配置

1. 下载与安装。在 Ubuntu 11.10 环境下: `sudo apt-get install git`。

2. 初次运行 Git 前的配置。Git 提供了一个叫做`git config`的工具，用来配置相应的工作环境变量，这些变量可以存放在三个地方 [2]:

   - `/etc/gitconfig` 文件：系统中对所有用户都普遍适用的配置。若使用`git config --system`，读写的就是这个文件。

- ~/.gitconfig 文件：用户目录下的配置文件, 只适用于该用户。若使用git config --global，读写的就是这个文件。
- 当前项目的git 目录中的配置文件 (也就是工作目录中的.git/config 文件)：这里的配置仅仅针对当前项目有效。

初次运行 Git 前一般需要用git config --global设置用户名和电子邮件地址 [5]。打开终端，输入

```
git config --global user.name "your name here"
git config --global user.email "your email here"
```

此时，用git config --list命令就可以查看是否设置成功。设置成功后，之后每次提交文档或代码时都会引用这两条信息说明是谁提交了更新以及他的电子邮件地址是什么。

3 Git 的使用

3.1 初始化仓库

1. 在本地创建Test 目录: mkdir ~/Test
2. 转到Test 目录: cd ~/Test
3. 初始化仓库: git init

初始化后，在当前目录下会出现一个名为.git 的隐藏目录，可以通过ls -a来查看，所有 Git 需要的数据和资源都存放在这个目录中。不过目前，仅仅是按照既有的结构框架初始化好了里边所有的文件和目录，但我们还没有开始跟踪管理项目中的任何一个文件 [1]。

3.2 跟踪文件与取消跟踪

对任何一个文件，Git 都有三个状态：已提交（committed），已修改（modified），已暂存（staged）。

已提交 表示该文件已经被安全地保存在本地数据库中了。

已修改 表示修改了某个文件，但还没有提交保存。

已暂存 表示把已修改的文件放在下次提交时要保存的清单中。

可以通过`git status`命令来查看文件的状态。

1. 新建`hello` 文档: `touch hello`。
2. 写入内容：`vim hello`，然后输入`Hello!` 保存。
3. 查看文件的状态：`git status`，会发现有提示：`Untracked files`，这说明 Git 没有跟踪`hello` 文档。
4. 开始跟踪文件：`git add hello`。对于已经跟踪的文件，`git add`的意义是把已跟踪的文件放在了暂存区。
5. 查看状态：`git status`，会发现有提示：`Changes to be committed`，这说明`hello` 文档已经被跟踪了，即放在了暂存区，属于已暂存状态。
6. 若是再修改`hello` 文档，比如说将`Hello!` 改为`hello world`，此时查看状态时你会发现`hello` 有两个状态，一个是未暂存，一个是已暂存。实际上，如果你此时提交，提交的是没有修改的`hello` 文档，所以每次对文档做了修改，都应该重新用`add`把文档放入暂存区已备提交。
7. 取消跟踪：`git rm --cached hello`，此时再查看状态你会发现又回到了原来的状态。

3.3 提交文件

使用`git commit -m 'message'`命令可以把暂存区的文件提交，此时历史记录里就有此时的文件状态。每次提交实际上就是对项目做了一次快照。message 是对这一次的提交给出说明。.git 里的`COMMIT-EDITMSG` 文件，是最后一次提交时的说明。

可以通过`git commit -a -m 'message'`来跳过暂存，直接提交。

3.4 查看提交历史

用`git log`命令可以查看提交历史。默认不输入参数的话，`git log`会按提交时间列出所有的更新，最近的更新排在最上面，同时给出作者的名字和电子邮件地址。

用`git log -p`可以查看每次提交的内容差异，用`git log -2`则显示最近两次的更新。查看更多关于`git log`的参数设置可以通过`git help log`。

3.5 恢复历史版本

3.5.1 恢复项目的历史版本

在进行一个项目时，有时会发现你项目中后几次所做的修改完全是错误的，若想重置到之前某个版本，也就是说删除历史记录中某个版本之后的所有的版本的记录，Git 也可以帮你做到。

首先，你可以使用`git log`命令查看你想要恢复的版本的哈希值 (hash)。Git 使用 SHA-1 算法计算数据的校验和，通过对文件的内容或目录的结构计算出一个 SHA-1 哈希值，作为指纹字符串。该字串由 40 个十六进制字符 (0−9 及 a-f) 组成，看起来像是：f829d445fa8fdec258e58c61f730bd769e31bdbd。

然后就可以用`git reset`命令来恢复到你选中的版本，`git reset`命令有很多的参数选项，下面就简单的介绍其中两个，其它选项可以用`git help reset`命令查看。

hard 选用 hard 选项来恢复某个版本，会删掉这个版本之后所做的所有的提交，此时项目中的文件也回到了这个版本的状态。查看历史记录（`git log`）你会发现，记录中只是记载了这个版本之前的修改。具体做法是，用`git log`命令查看想要恢复的版本的哈希值（hash），然后用`git reset --hard hash`命令恢复。此时这个版本之后的所有的提交都会被删除。

soft 选用 soft 选项来恢复某个版本，删除了历史记录中的这个版本之后的所有版本，但是项目中的最新版本的文件并没有做相应的更改，仍是现在的状态。也就是说，用 soft 选项后的结果虽然删除了指定版本之后的所有的版本的历史记录，但是，因为项目中保留了最新的版本，所以若是出现了误删，仍可以重新用`git add`和`git commit`命令重新提交，从而把最新版本加入到提交记录中。

注意，若是在本地恢复了项目了历史版本，也就是说删除了一些历史记录后，若想更新到远程 GitHub 上，必须使用`git push -f origin master`命令（如果远程 GitHub 中包含你所删除的记录）。f 参数是强制执行的意思。因为若不强制执行，就会提示让用`git pull`命令把远程项目合并到本地来，这时就会把删掉的记录又重新合并到本地项目的历史记录中，造成记录的混乱。

3.5.2 恢复单个文件的历史版本

上面讲到的是恢复整个项目的某个版本，若是想恢复其中某个文件的版本，可以用`git log filename`来查看想回到版本的哈希值，然后用`git reset --hard hash`来恢复。

3.6 修改提交说明

修改最新提交的说明比较简单，只需一条命令即可：`git commit --amend`，此时你就可以修改 # 号之上的说明，并用 Ctrl+X 和 Y 键进行保存。

若是想修改历史记录中的提交说明，例如，倒数第三次提交说明，则需要几个步骤：

1. 输入`git rebase -i HEAD~2`命令，这句话的意思是衍合到倒数第二次提交，关于衍合的详细解释，可以用`git help rebase`查看。此时，你会看到如图5的界面。把第一行中的 pick 改为 edit，然后保存，退出(Ctrl+X 键和 Y 键)。注意，此时的这个列表与用`git log`查看历史记录时的列表顺序正好相反，所以修改时要注意。

2. 输入`git commit --amend`命令，此时就可以修改提交说明，修改后保存。

3. 输入`git rebase --continue`命令，完成修改，此时查看历史记录会发现提交说明已经修改成功。

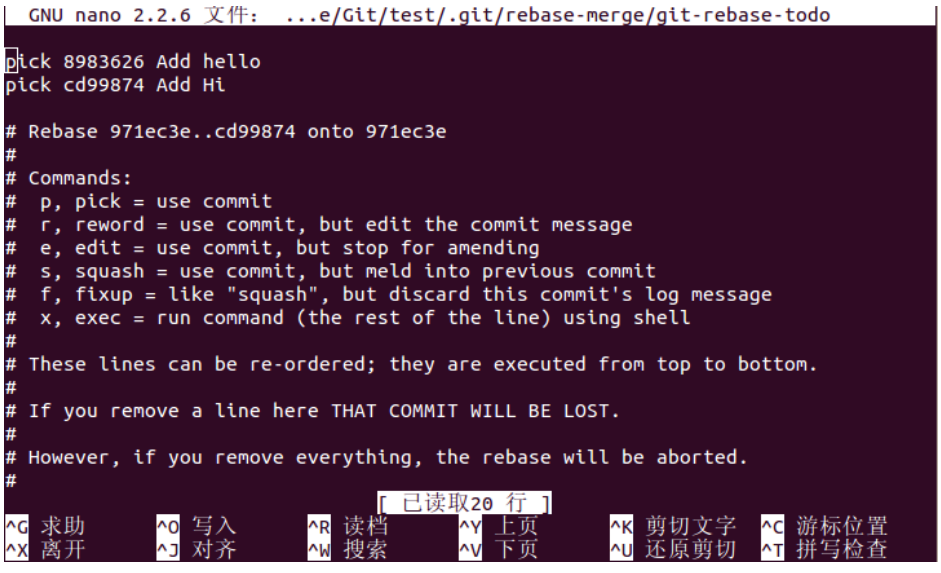

图 5:

跟上面提到的恢复历史版本一样，修改以后若想提交到远程 GitHub 上，而 GitHub 中包含修改之前的版本时，必须使用`git push -f origin master`命令。

3.7 查看某个历史版本的文件

Git 可以通过打标签的方式查看历史版本的文件，也可以不用打标签。

打标签 Git 可以对某一时间点上的版本打上标签。若是你觉得当前的版本相对较为成熟，你可以给当前版本打上标签，方便以后查看这个版本。

比如，我们提交了某个项目中的所有文件，然后我们新建标签：

```
1 git tag -a v1.0 -m 'message'
```

此时，我们就可以查看这个版本中的文件内容：

```
1 git show v1.0:./
```

冒号后为项目的路径，./是当前路径的意思，若是想查看这个项目中的某个文件的内容，可以在/后加上文件名。

不打标签 若想直接查看某个文件的历史版本，Git 也可以做到。首先应通过`git log`指令找到你想查看的版本的哈希值，比如说：d26673579fa5a8a328a7e36336724548cd7b210f。然后就可以用以下指令来查看了，当然，filename 要改成你想查看的文件的文件名。

```
1 git show :d26673579fa5a8a328a7e36336724548cd7b210f:./
      filename
```

3.8 忽略文件

有时编译时产生一些中间文件，如果不想跟踪这些文件，也可以进行设置 [4]。在新建一个项目时，养成一个写忽略文件的好习惯，可以有助于减少以后的麻烦，使 Git 管理更出色。

设置忽略文件有几种不同的方式，这里简单介绍三种。一种是建立.gitignore 文件，另一种是修改exclude 文件，还可以设置全局的忽略文件。

.gitignore 首先应确保你的忽略文件处于为跟踪状态，若是已跟踪，则用命令`git rm --cached filename`来取消跟踪。然后在你的项目仓库下创建.gitignore 文件：`touch .gitignore`。写入你想忽略的文件的扩展名，例如，想忽略所有.log 的文件和所有以 ~ 结尾的文件，则写入内容：

```
1 *.log
2 *~
```

此时，将不会再跟踪这两类文件。每个项目中的.gitignore 文件内容可以是不相同的，根据实际情况可以设置想忽略的文件。

exclude 你也可以在你的项目中的.git/info/exclude 中设置你的忽略文件，但是这个设置只是当前用户的设置，其他人无法共享。也就是说它无法被提交到 GitHub 网站上以供别人克隆和使用。

全局忽略文件 设置全局的.gitignore 文件可以使你所有的项目都忽略掉你所设置的忽略文件。

参考文献 [3] 中包含很多种类的.gitignore 文件的模板，比如 LaTeX 等等，可以拷贝下来放在自己的项目中。

我们规定了实验室中使用忽略文件的方式：首先，郑老师在 GitHub 网站上新建项目的时候会选择忽略文件的类型，这样项目创建成功后，相应的忽略文件就会包含在项目中。我们克隆项目的时候忽略文件就会被克隆到本地。若是需要，可以修改项目中的`.gitignore` 文件。

我们可以设置全局的忽略文件来优化我们的设置。打开终端，转到 ~ 目录下，执行`git config --global core.excludesfile ~/Linux.gitignore`，然后编辑`Linux.gitignore` 文件：`vim Linux.gitignore`，写入内容：

```
1 .*
2 !.gitignore
3 *~
```

这样，所有的 Git 项目中除了`.gitignore` 文件之外的其他隐藏文件和以 ~ 结尾的文件在提交到 GitHub 上时都会被忽略。

3.9 本地仓库与远程仓库的冲突合并

有时候我们在用`git commit`命令在本地提交之后，想要用`git push`提交到远程仓库上时，会发现无法推送的问题6。这是因为远程仓库上的版本不是本地版本的上一版本。

```
sunxue@sunxue:~/Git/test$ git push origin master
To git@github.com:sunxue/test.git
 ! [rejected]        master -> master (non-fast-forward)
error: 无法推送一些引用到 'git@github.com:sunxue/test.git'
提示：更新被拒绝，因为您当前分支的最新提交落后于其对应的远程分支。
提示：再次推送前，先与远程变更合并（如 'git pull'）。详见
提示：'git push --help' 中的 'Note about fast-forwards' 小节。
```

图 6: push error

此时，根据提示，应该用`git pull`在终端执行一下，然后会看到提示7。

比如说，当前本地仓库有个名为`hello` 的文件，里面的内容是 hello kitty，hello kitty 的前一版本的内容是 hello。远程仓库的 hello 文件的内

图 7: pull error

容是 hello world。此时要把本地仓库的`hello` 文件提交到远程仓库，即执行`git push`和`git pull`时就会出现以上错误。此时打开本地仓库中的`hello` 文件，可以看到内容如图8。

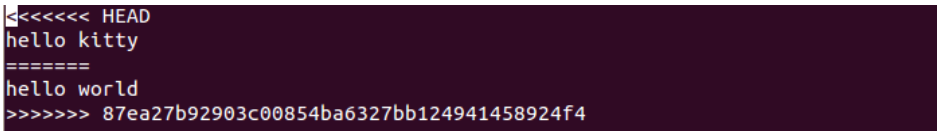

图 8: conflict

其中 ====== 隔开的上半部分是本地仓库中的内容，下半部分是远程仓库上的内容。解决冲突的办法是二者选其一或者整合到一起（====等符号及解释都应该删除)。修改完之后，再进行`git add`，`git commit`以及`git push`就没有问题了。

这种本地仓库与远程仓库的冲突一般是在合并同一个文件的同一部分时，其内容不同造成的。合并同一文件的不同部分或者合并不同文件时，不会产生冲突，此时使用`git pull`命令，就可以把 GitHub 上新更新的内容自动的拉取和合并到本地仓库中。

另外，若用`git pull`时出现图9情况，可以按 Ctrl+X 键和 Y 键保存这个合并信息。

图 9: pull problem

4 实验室中 Git 和 GitHub 的使用

学会上面的用法后，下面就从实验室的使用情况来做简单介绍。

4.1 克隆已有的仓库

实验室的 GitHub 网站为`https://github.com/zhenglab/`，通常情况下，若是需要你上传一些材料到这个网站，郑老师会新建一个项目仓库供你上传。下面就开始进行操作：

1. 首先要登陆网站，选择你要克隆的 Repositories。

2. 开始克隆。打开终端，转到你想要建立仓库的目录，比如说 Git 目录，执行`git clone git@github.com:zhenglab/ROC_PR.git`，此时在你 Git 目录下就会生成一个`ROC_PR` 的仓库。

3. 从远程拷贝文件 [6]。

- 转到这个仓库：`cd ROC_PR`。
- 告诉 Git 远程库的名字和地址：

```
git remote add upstream git@github.com:zhenglab/
    ROC_PR.git
```

- 从远程抓取数据：`git fetch upstream`。

4. 提交文件。把你想要提交的文档（例如ROC_PR.pdf）复制到你的ROC_PR 文件夹下，执行

```
git add ROC_PR.pdf
git commit -m 'Add ROC_PR.pdf'
git push origin master
```

此时，https://github.com/zhenglab/上的ROC_PR 仓库中就有了你提交的ROC_PR.pdf 文档。

5. 要注意的是，每次修改过的文档，都必须执行git add，git commit，git push（推送当前分支到指定远程分支）才能使得 GitHub 上得到更新。

4.2 新建仓库

你也可以在你自己的 GitHub 上新建一个仓库，然后把本地的文件上传上去。

- 打开浏览器，转到 https://github.com/your name。
- 点击 Resposito ries 选项卡，然后点击 New，创建一个新的库。

- 填写库名 (比如 Test) 和描述，选择 Public 或者是 Private，在 Initialize this repository with a README 选项前打勾，可以选择添加.gitignore 文件和license，然后点击 Create repository，如图10。

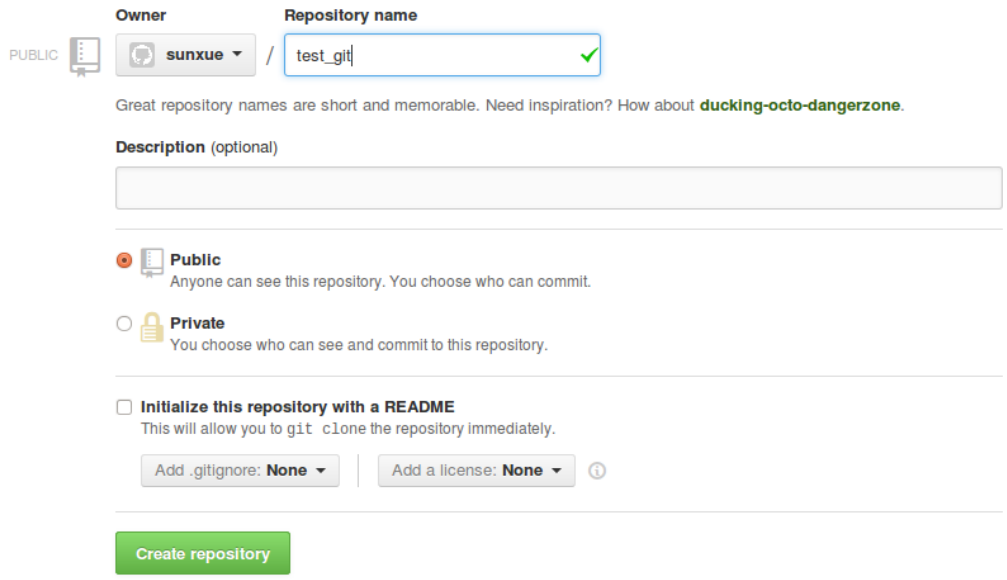

图 10: new repository

- 让 Git 记住远程库的名和地址（前提是你已经在本地仓库中用git commit提交了你的hello 文档，参考第3章)。

```
1 git remote add origin git@github.com:Your name/Test.git
```

- 推送到远程库：git push origin master。

以上就是对于 Git 和 GitHub 的基本使用，要想了解更多关于 Git 和 GitHub 的内容，可以参考 [5][2]。

参考文献

[1] Scott Chacon. Git 详解之一：git 起步（原文：pro git）, 2011. http://blog.jobbole.com/25775/.

[2] Scott Chacon. Git 详解之二：git 基础（原文：pro git）, 2011. http://blog.jobbole.com/25808/.

[3] GitHub. A collection of useful .gitignore templates. https://github.com/github/gitignore.

[4] GitHub Help. Git 忽略文件. https://help.github.com/articles/ignoring-files.

[5] GitHub Help. 官方帮助文档. https://help.github.com/.

[6] Xiang Ma. 使用 git 远程仓库, 2011. http://blog.maxiang.net/working-with-git-remote-repositories/274/.

[7] 笑遍世界. 错误：ssh 连接时提示“the authenticity of host xx can't be established”, 2012. http://smilejay.com/2012/12/ssh-config-host-key-checking/.