Basic

December 15, 2018

1 Preparation

1.1 初始设置

对本地计算机里安装的 Git 进行设置

设置使用 Git 时的姓名和邮箱地址

$ git config −−global user.name “Firstname Lastname”

$ git config −−global user.email “your_email@example.com”

这个命令，会在 “∼ /.gitconfig” 中以如下形式输出设置文件

[user]

name = Firstname Lastname

email = your_email@example.com

想更改这些信息时，可以直接编辑这个设置文件。这里设置的姓名和邮箱地址会用在 Git 的提交日志中。由于在 GitHub 上公开仓库时，这里的姓名和邮箱地址也会随着提交日志一同被公开，所以请不要使用不便公开的隐私信息。

将 color.ui 设置为 auto 可以让命令的输出拥有更高的可读性

$ git config −−global color.ui auto

“∼ /.gitconfig” 中会增加下面一行。

[color]

ui = auto

1.2 设置 SSH Key

GitHub 上连接已有仓库时的认证，是通过使用了 SSH 的公开密钥认证方式进行的。创建公开密钥认证所需的 SSH Key，并将其添加至 GitHub。运行下面的命令创建 SSH Key

$ ssh-keygen -t rsa -C “your_email@example.com”

Generating public/private rsa key pair.

Enter file in which to save the key

(/Users/your_user_directory/.ssh/id_rsa): 按回车键

Enter passphrase (empty for no passphrase): 输入密码

Enter same passphrase again: 再次输入密码

“your_email@example.com” 的部分请改成您在创建账户时用的邮箱地址。密码需要在认证时输入，请选择复杂度高并且容易记忆的组合。输入密码后会出现以下结果。

Your identification has been saved in /Users/your_user_directory/.ssh/id_rsa.

Your public key has been saved in /Users/your_user_directory/.ssh/id_rsa.pub.

The key fingerprint is:

fingerprint 值 your_email@example.com

The key’s randomart image is:

id_rsa 文件是私有密钥，id_rsa.pub 是公开密钥。

在 GitHub 中添加公开密钥，今后就可以用私有密钥进行认证了。点击右上角的账户设定按钮（Account Settings），选择 SSH Keys 菜单。点击 Add SSH Key 之后，会出现输入框。在 Title 中输入适当的密钥名称。Key 部分请粘贴 id_rsa.pub 文件里的内容。

id_rsa.pub 的内容可以用如下方法查看。

$ cat ~ /.ssh/id_rsa.pub

ssh-rsa 公开密钥的内容 your_email@example.com

添加成功之后，创建账户时所用的邮箱会接到一封提示“公共密钥添加完成”的邮件。

完成以上设置后，就可以用手中的私人密钥与 GitHub 进行认证和通信了。

$ ssh -T git@github.com

The authenticity of host 'github.com (207.97.227.239)' can't be established.

RSA key fingerprint is fingerprint 值.

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

出现如下结果即为成功。

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

创建一个公开的仓库。点击右上角工具栏里的 New repository 图标，创建新的仓库。

在 Initialize this repository with a README 选项上打钩，随后 GitHub 会自动初始化仓库并设置 README 文件，让用户可以立刻 clone 这个仓库。如果想向 GitHub 添加手中已有的 Git 仓库，建议不要勾选，直接手动 push。

下方左侧的下拉菜单非常方便，通过它可以在初始化时自动生成.gitignore 文件[1]。这个

[1]该文件用来描述 Git 仓库中不需管理的文件与目录。

设定会帮我们把不需要在 Git 仓库中进行版本管理的文件记录在.gitignore 文件中，省去了每次根据框架进行设置的麻烦。下拉菜单中包含了主要的语言及框架，选择今后将要使用的即可。

右侧的下拉菜单可以选择要添加的许可协议文件。如果这个仓库中包含的代码已经确定了许可协议，那么请在这里进行选择。随后将自动生成包含许可协议内容的 LICENSE 文件，用来表明该仓库内容的许可协议。

输入选择都完成后，点击 Create repository 按钮，完成仓库的创建。

下面这个 URL 便是刚刚创建的仓库的页面。

https://github.com/用户名/Hello-Word

README.md 在初始化时已经生成好了。README.md 文件的内容会自动显示在仓库的首页当中。因此，人们一般会在这个文件中标明本仓库所包含的软件的概要、使用流程、许可协议等信息。如果使用 Markdown 语法进行描述，还可以添加标记，提高可读性。

在 GitHub 上进行交流时用到的 Issue、评论、Wiki，都可以用 Markdown 语法表述，从而进行标记。准确地说应该是 GitHub Flavored Markdown（GFM）语法。该语法虽然是 GitHub 在 Markdown 语法基础上扩充而来的，但一般情况下只要按照原本的 Markdown 语法进行描述就可以。使用 GitHub 后，很多文档都需要用 Markdown 来书写。

将已有仓库 clone 到身边的开发环境中。

```
$ git clone git@github.com:hirocastest/Hello-World.git
Cloning into 'Hello-World'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
```

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

这里会要求输入 GitHub 上设置的公开密钥的密码。认证成功后，仓库便会被 clone 至仓库名后的目录中。将想要公开的代码提交至这个仓库再 push 到 GitHub 的仓库中，代码便会被公开。

添加至 Git 仓库的文件显示为 Untracked files。通过 git add 命令将文件加入暂存区 A，再通过 git commit 命令提交。添加成功后，可以通过 git log 命令查看提交日志。之后只要执行 git push，GitHub 上的仓库就会被更新。

2 基本操作

要使用 Git 进行版本管理，必须先初始化仓库。Git 是使用 **git init** 命令进行初始化的。建立一个目录并初始化仓库。如果初始化成功，执行了 git init 命令的目录下就会生成 .git 目录。这个.git 目录里存储着管理当前目录内容所需的仓库数据。

在 Git 中，将这个目录的内容称为 “附属于该仓库的工作树”。文件的编辑等操作在工作树中进行，然后记录到仓库中，以此管理文件的历史快照。如果想将文件恢复到原先的状态，可以从仓库中调取之前的快照，在工作树中打开。开发者可以通过这种方式获取以往的文件。

git status 命令用于显示 Git 仓库的状态。工作树和仓库在被操作的过程中，状态会不断发生变化。在 Git 操作过程中时常用 git status 命令查看当前状态。

$ git status

On branch master

#

Initial commit

#

nothing to commit (create/copy files and use “git add” to track)

结果显示了当前正处于 master 分支下。接着还显示了没有可提交的内容。所谓提交(Commit)，是指 “记录工作树中所有文件的当前状态”。尚没有可提交的内容，就是说当前建立的这个仓库中还没有记录任何文件的任何状态。只要对 Git 的工作树或仓库进行操作，git status 命令的显示结果就会发生变化。

如果只是用 Git 仓库的工作树创建了文件，那么该文件并不会被记入 Git 仓库的版本管理对象当中。要想让文件成为 Git 仓库的管理对象，就需要用 **git add** 命令将其加入暂存区(Stage 或者 Index) 中。暂存区是提交之前的一个临时区域。

$ git add README.md

$ git status

On branch master

#

Initial commit

#

Changes to be committed:

(use “git rm −−cached <file>...” to unstage)

#

new file: README.md

README.md 文件显示在 Changes to be committed 中了。

git commit 命令可以将当前暂存区中的文件实际保存到仓库的历史记录中。通过这些记录，可以在工作树中复原文件。

$ git commit −m “First commit”

[master(root − commit)9f129ba] First commit

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 README.md

-m 参数后的 “First commit” 称作提交信息，是对这个提交的概述。想要记述得更加详细，不加 -m，直接执行 git commit 命令。执行后编辑器就会启动，并显示如下结果。

Please enter the commit message for your changes. Lines starting

with ’#’ will be ignored, and an empty message aborts the commit.

On branch master

#

Initial commit

#

Changes to be committed:

(use “git rm −−cached <file>...” to unstage)

#

new file: README.md

在编辑器中记述提交信息的格式如下：

第一行：用一行文字简述提交的更改内容

第二行：空行

第三行以后：记述更改的原因和详细内容

只要按照上面的格式输入，今后可以通过确认日志的命令或工具看到这些记录。在以 #(井号) 标为注释的 Changes to be committed（要提交的更改）栏中，可以查看本次提交中包含的文件。将提交信息按格式记述完毕后，请保存并关闭编辑器，以 #(井号) 标

为注释的行不必删除。随后，刚才记述的提交信息就会被提交。

如果在编辑器启动后想中止提交，请将提交信息留空并直接关闭编辑器，随后提交就会被中止。

当前工作树处于刚刚完成提交的最新状态，所以结果显示没有更改。

git log 命令可以查看以往仓库中提交的日志。包括可以查看什么人在什么时候进行了提交或合并，以及操作前后有怎样的差别。

$ git log

commit 9f129bae19b2c82fb4e98cde5890e52a6c546922

Author: hirocaster <hohtsuka@gmail.com>

Date: Sun May 5 16:06:49 2013 +0900

First commit

commit 栏旁边显示的 “9f129b……” 是指向这个提交的哈希值。Git 的其他命令中，在指向提交时会用到这个哈希值。Author 栏中显示 Git 设置的用户名和邮箱地址。Date 栏中显示提交执行的日期和时间。再往下就是该提交的提交信息。

如果只想让程序显示第一行简述信息，可以在 git log 命令后加上 −−pretty=short。

Git commands :

(master) $: git commandname parameter1 parameter2 −−option

The command name (commandname in the example) is one of over 100 individual functions that Git can perform. Behind the scenes, each of these commands is a separate program responsible for its own specific job.

Options are special parameters that are denoted by at least one leading dash character. Many options have both a long form, like −−global, and a shortcut form, like −g. There are also options that take values, like git commit −−message=“hello world”.

There are two that it absolutely needs in order to function: your name and email address. Git adds an Author attribute to every commit you make that includes both your name and email address, so that your collaborators on a project can know who made a given change. The name you enter will be used to identify you in change logs and any other place where Git shows who made a particular change, while your email address not only tells people how to reach you, but also tells a hosted service like GitHub who you are on their service.

Use the git config command to tell Git who you are. Unlike most Git commands, which only work inside of a Git project, these can be run from any directory.