# 版权信息

COPYRIGHT

书名：Git从入门到精通

作者：高见龙

出版社：北京大学出版社

出版时间：2019年9月

ISBN：9787301305874

字数：133千字

# 第1章 Git入门

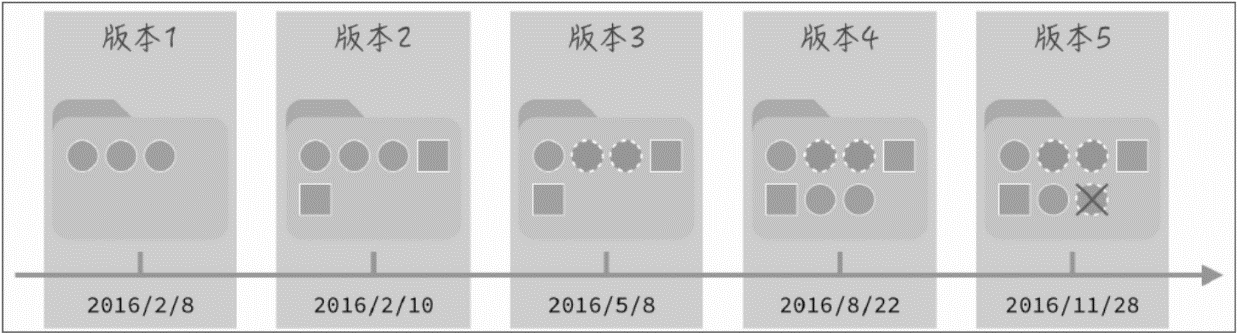
## 1.1 Git概述

### 1.1.1 什么是Git

如果你问那些正在使用Git工具的人“什么是Git”，他们大多可能会回答“Git是一种版本控制系统（Version Control System）”，专业一点的可能会说“Git是一种分布式版本的版本控制系统”。这样的解释对没接触过Git的新手来说是没有任何意义的。到底什么是“版本”？要“控制”什么东西？什么又是“分布式”？接下来就为你一一讲解。

不管你是不是程序员，只要你的日常工作离不开计算机，那么每天可能都要新建、编辑、改动很多的文件。例如，如果你是一名人力资源部门主管，就会创建一个resume目录，专门用来保存面试者的资料。

如图1-1所示，随着时间的变化，一开始resume目录中只有3个文件，过两天增加到5个；不久之后，其中的2个被修改了；过了3个月后又增加到7个；最后又删掉了1个，变成6个。每次resume目录的状态变化，不管是新建或删除文件，抑或是改动文件内容，都称为一个“版本”，如图1-1中的版本1 ～ 5。而所谓的“版本控制系统”，就是用来记录所有的这些状态变化的，使你可以像搭乘时光机一样，随时切换到过去某一“版本”的状态。

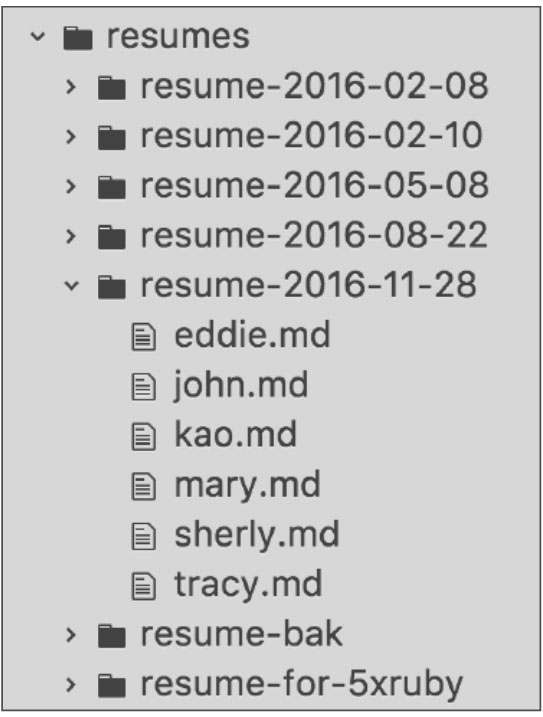


简单地说，Git就像玩游戏的时候可以存储进度一样。例如，为了避免打头目输了而损失装备，或者打倒头目却没有掉落期望的珍贵装备，可以在每次去打头目之前都记录一下，以便在发生状况的时候回到旧进度，再挑战一次。

### 1.1.2 为什么要学习Git

先问个问题，大家平常怎样整理或备份文件？

以图1-1为例，最传统也是最方便的方式，便是使用“复制+粘贴法”。这样操作之后，可能会出现图1-2所示的画面。



虽然一眼就可以看出每个“版本”的用途，但其他信息就不是那么明显了。例如，resume-2016-05-08目录中的那两个改动过的文件都改了什么内容？resume-2016-08-22和resume-2016-11-28这两个目录有什么不一样的地方？resume-bak与其他的目录有什么不同？最麻烦的是，如果这个目录是和其他人共享的，而文件被其他人覆盖了，该怎么处理呢？

如果你在乎这些问题的答案，那么使用“版本控制系统”就是一个很不错的选择。通过这样的系统，可以清楚地记录每个文件是谁在什么时候加进来的、什么时候被修改或删除的。Git就是这样一种版本控制系统，也是当前业界最流行的版本控制系统。

无论做任何工作，如果有Git帮你保留这些历史记录和证据，那么发生意外状况的时候你就能知道是从什么时候开始有问题的，以及该找谁负责，再也不用自己“背黑锅”了！

## 1.2 Git与其他版本控制系统的差异

### 1.2.1 Git的优点

那么Git到底有哪些厉害的地方，会让这么多人选择它呢？

1. 免费、开源

2005年，为了管理Linux内核程序代码，Linux内核的作者Linus Trovalds仅用了10天时间就开发出了Git。粗略算来，至今已有十几年的历史了。除了可免费使用外，整个Git的源代码也可以在互联网上获取（当然Git的源代码也是用Git做版本控制的）。

2. 速度快、文件体积小

如果使用前面提到的“复制+粘贴法”，那么这些备份的目录会占用大量空间。其他的版控系统大多是记录每个版本之间的差异，而不是完整地备份整个目录，所以整个目录的大小不会快速地增加。

Git的特别之处在于，它并不是记录版本的差异，而是记录文件内容的“快照”（snapshot），可以非常快速地切换版本。至于什么是“快照”，在后面的章节中会有更详细的介绍。

3. 分布式系统

对我来说，这可能是Git最大的优点了。其他的版本控制系统，比如CVS或SVN之类的集中式的版控系统（Centralize VersionControl），都需要有一台专用的服务器，所有的更新都要与这台服务器沟通。也就是说，一旦这台服务器坏了，或者处于没有网络连线的环境下，就无法使用了。

而Git是一款分布式的版控系统（Distributed Version Control），虽然通常也会有共同的服务器，但即使在没有服务器或在没有网络的环境下，仍然可以使用Git进行版控，待服务器恢复正常运行或移到有网络的环境后再进行同步，不会受到影响。事实上，在使用Git的过程中，大多数的Git操作在计算机本机上就可以完成。

### 1.2.2 Git的缺点

如果非要说Git的缺点，那大概就是易学难精。虽然Git的指令非常多，而且有的指令有点复杂，但平常会用到的指令并不多。根据“80/20法则”，大概20%的指令就足以应付80%的工作。

除了终端机（或命令提示符）环境下的Git指令外，还有很多实用的图形界面工具，让使用者不用输入复杂的指令就可以享用Git强大的功能。本书将使用终端机指令来解释概念，并以图形界面工具（如SourceTree）来辅助说明Git是怎样运行的。

## 1.3 常见问题

1. 我是设计师，我也可以用Git吗

基本上，Git只关心文件的“内容”，所以只要是文件，都可以使用Git来管理。只是设计生成的大多是Photoshop的PSD文件或Illustrator的AI文件，虽然Git也可以管理这些文件，但因为这些文件（二进制文件）不像常规文本文件那样可以一行一行地查看，也就无法那么精准地知道什么人在什么时候改了哪些字。但总体来说，Git还是帮得上忙的，至少当文件不小心被覆盖或删除的时候，还可以找回旧版本的文件。

2. Git就是我之前看到过的GitHub吗

这是很多新手容易有的误会，以为Git就是GitHub（或者认为GitHub就是Git），甚至有的公司在招聘启事上明确写着“会使用GitHub”。事实上，Git是一款版本控制软件，而GitHub是一个商业网站，其本体是一个Git服务器，但这个网站上的应用程序可以让大家通过Web操作来完成一些原本需要复杂的Git指令才能做到的事。

本书后面也会介绍如何使用GitHub与其他人进行协作。虽然GitHub很好用，但别忘了Git才是它的本体。

# 第2章 环境安装

## 2.4 图形界面工具

一开始接触Git时，很多人不太理解其运行原理，而且大部分人比较习惯图形界面工具（Graphic User Interface，GUI），对终端机（或命令提示符）及Git指令操作相对不熟悉。

GUI工具的确比较方便，但在不懂原理的前提下，恐怕能做的只是机械地单击GUI工具中的按钮，并不清楚内部到底发生了什么，遇到问题也不知道该如何处理。

因此，本书将以指令为主，但会使用GUI工具加以辅助说明。

在Git官方网站（https://git-scm.com/downloads/guis）上有多款GUI工具，有的是商业软件，有的是免费软件，如图2-9所示。

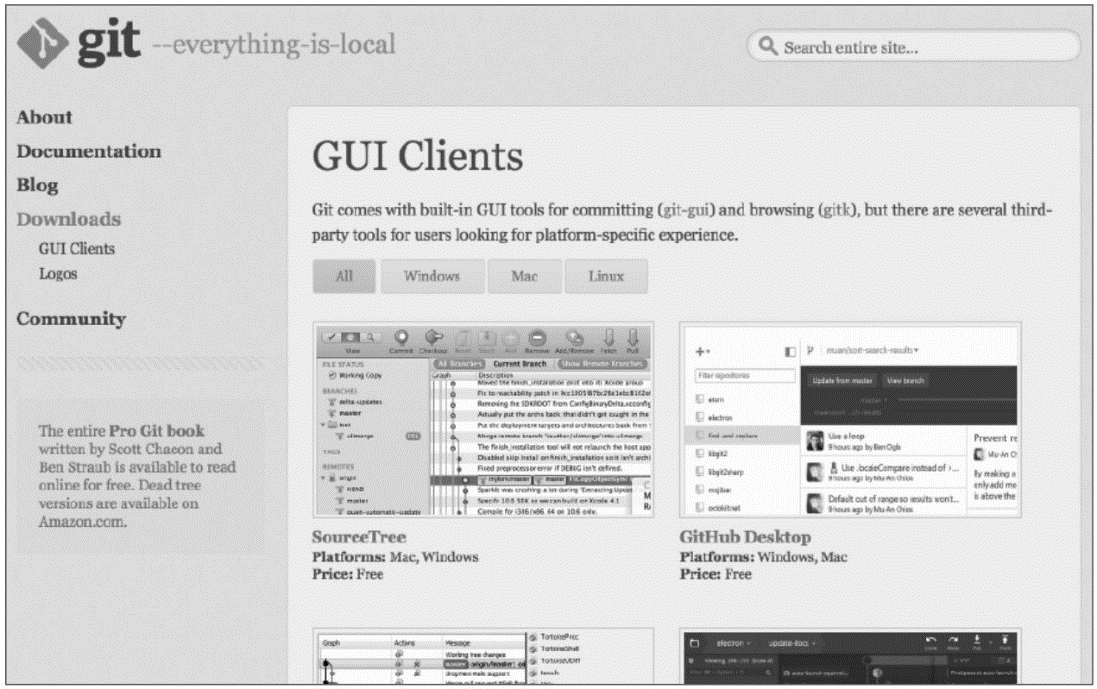


图2-9

其中，SourceTree和GitHub Desktop这两款工具，适用于macOX和Windows操作系统，而且功能都很完整，也都可免费使用，这里推荐给大家。

本书将以SourceTree软件为例进行说明。

在Linux/Ubuntu操作系统中，没有SourceTree可供安装。不过没关系，有gitk软件也可以。其安装只需一行代码：

$ sudo apt-get install gitk

gitk软件实际运行的界面如图2-10所示。

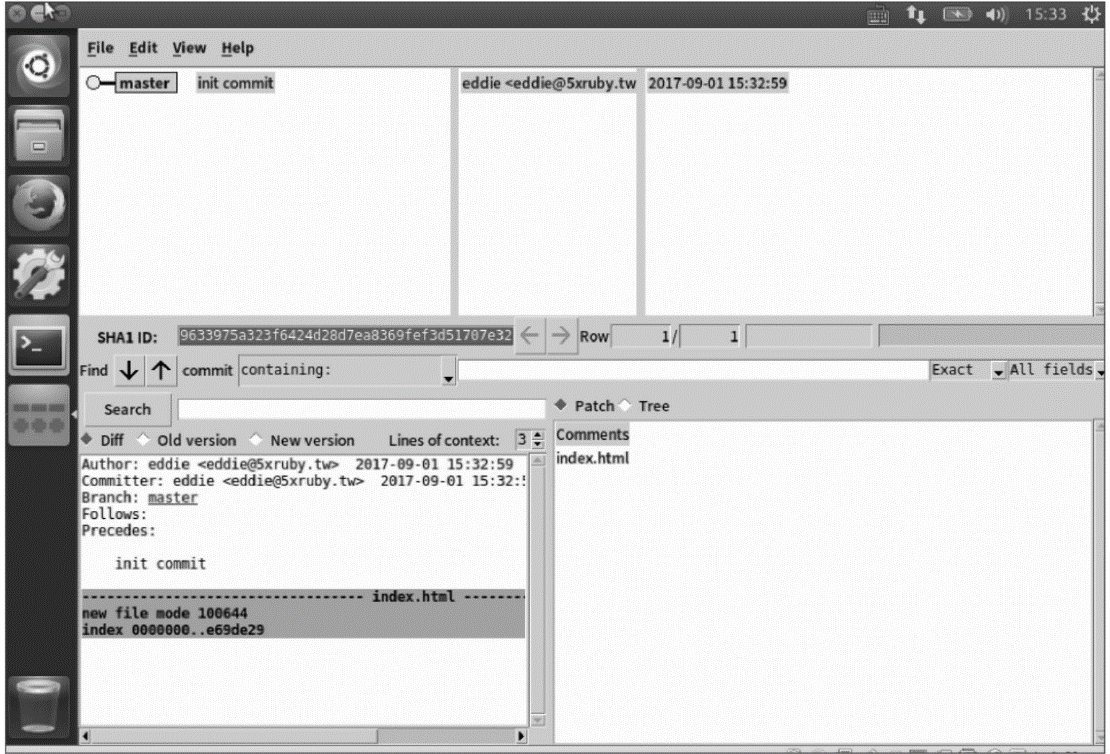


图2-10

该界面与SourceTree相比虽然有点简单，但基本功能一应俱全。

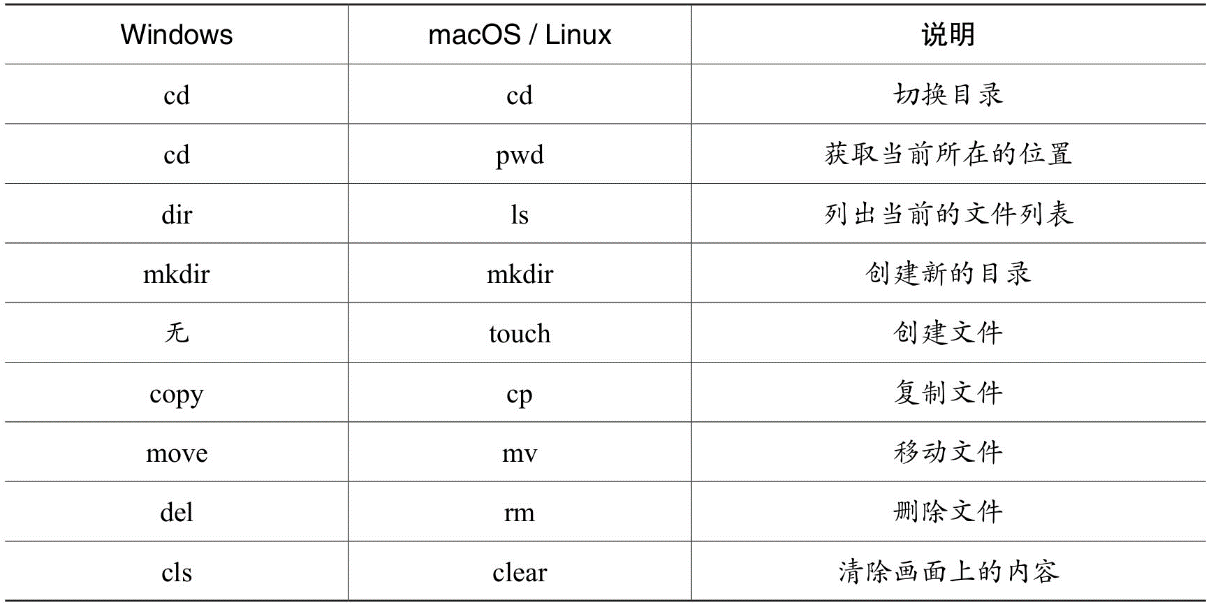
# 第3章 终端机/命令提示符

## 3.1 终端机及常用命令介绍

### 3.1.2 常用命令

在学习Git的过程中，很多命令都是在终端机环境下操作的。大部分初学者习惯使用图形界面工具，而不熟悉命令该如何输入，或者输入的命令是什么意思，这是让新手觉得困难的地方。表3-1中介绍了几个在终端机环境下常用的系统命令。

表3-1 终端机环境下常用的命令



不同的操作系统，命令也会不太一样。下面在macOs/Linux操作系统中使用这些命令。

#### 1. 目录切换及显示

在使用Git时，命令要在正确的目录下才能正常运行，所以学会目录的切换是很重要的。

# 切换到 /tmp目录（绝对路径）

$ cd /tmp

# 切换到my\_project目录（相对路径）

$ cd my\_project

# 往上一层目录移动

$ cd ..

# 切换到使用者的home / project 下的namecards 目录

# 这个 "~" 符号表示home 目录

$ cd ~/project/namecards/

# 显示当前所在目录

$ pwd/tmp

如果是在Windows操作系统中，命令如下。

# 切换到D 槽的5xruby 目录（绝对路径）

C:\> cd D:\5xruby

# 切换到5xruby 目录（相对路径）

D:\> cd 5xruby

# 往上一层目录移动

D:\5xruby> cd ..

# 显示当前所在目录

D:\5xruby> cd D:\5xruby

#### 2. 文件列表

ls命令可列出当前目录下的所有文件及目录。后面接的-al参数中，a是指以小数点开头的文件（如.gitignore）也会显示，l则是完整文件的权限、所有者，以及创建、修改的时间。

$ ls -al

total 56

drwxr-xr-x 18 user wheel 612 Dec 18 02:20 .

drwxrwxrwt 24 root wheel 816 Dec 18 02:19 ..

-rw-r--r-- 1 user wheel 543 Dec 18 02:19 .gitignore

-rw-r--r-- 1 user wheel 1729 Dec 18 02:19 Gemfile

-rw-r--r-- 1 user wheel 4331 Dec 18 02:20 Gemfile.lock

-rw-r--r-- 1 user wheel 374 Dec 18 02:19 README.md

-rw-r--r-- 1 user wheel 227 Dec 18 02:19 Rakefile

drwxr-xr-x 10 user wheel 340 Dec 18 02:19 app

drwxr-xr-x 8 user wheel 272 Dec 18 02:20 bin

drwxr-xr-x 14 user wheel 476 Dec 18 02:19 config

-rw-r--r-- 1 user wheel 130 Dec 18 02:19 config.ru

drwxr-xr-x 4 user wheel 136 Dec 18 02:41 db

drwxr-xr-x 4 user wheel 136 Dec 18 02:19 lib

drwxr-xr-x 4 user wheel 136 Dec 18 02:23 log

drwxr-xr-x 9 user wheel 306 Dec 18 02:19 public

drwxr-xr-x 9 user wheel 306 Dec 18 02:19 test

drwxr-xr-x 7 user wheel 238 Dec 18 02:23 tmp

drwxr-xr-x 3 user wheel 102 Dec 18 02:19 vendor

#### 3. 创建文件、目录

$ touch index.html

如果index.html文件本来不存在，touch命令会创建一个名为index.html的空白文件；如果index.html文件本来就已经存在，则touch命令只会改变该文件的最后修改时间，不会改动原本的内容。

$ mkdir demo

mkdir命令会在当前所在目录下创建一个名为demo的目录。

#### 4. 文件操作

复制文件index.html，并将副本命名为about.html：

$ cp index.html about.html

把文件index.html更名为info.html：

$ mv index.html info.html

删除文件index.html：

$ rm index.html

删除这个目录中所有的.html文档：

$ rm \*.html

在Windows操作系统下，则需把cp、mv及rm命令分别替换为copy、move及del命令。

这些命令看起来很难，但不用担心，在学习过程中用到的Git命令其实都不会太复杂，多试几次就能上手，不要因为命令输入错误就有失败感。

另外，在运行命令后，不管成功或失败，通常都会有消息显示在界面上。对于这些消息，切勿一带而过，务必要仔细阅读（最好把它念出来）。很多新手一看到信息就以为命令运行成功了，但事实上可能是错误信息。

不要害怕输入命令，不要害怕错误信息，加油！

## 3.2 超简明的Vim操作介绍

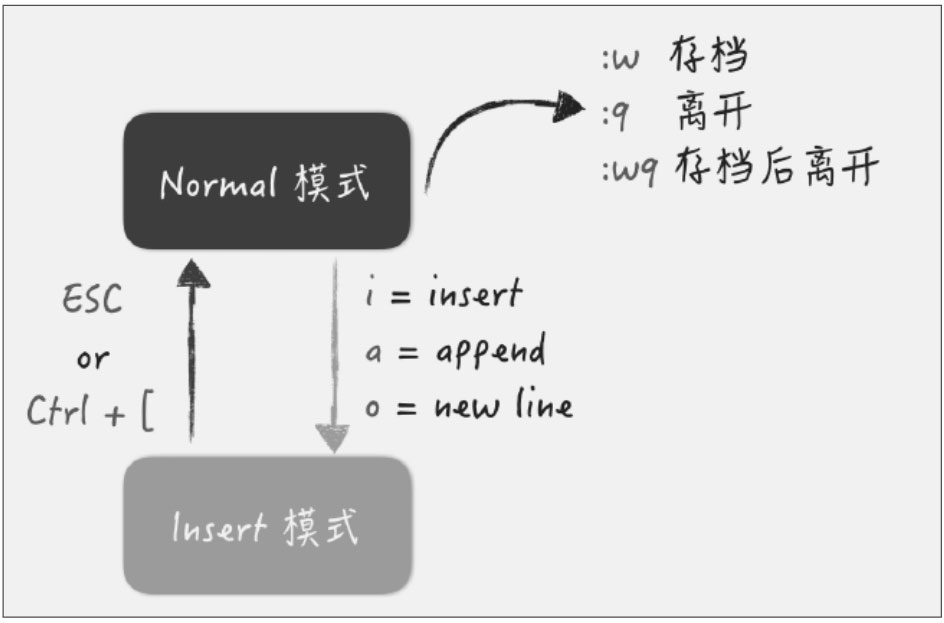
诞生于1976年的Vi软件至今已有四十多年的发展历程，人们现在用的大多是强化过的版本Vim（Vi IMproved），而第一个Vim版本迄今也已超过25年了。

很多人打开Vim后，会发现不知道如何打字、输入；好不容易开始输入了，也不知道该怎样存储；好不容易会存储了，又不知道该如何退出……简直是数字版的“密室逃脱”。

由于Vim是Git的默认编辑器，不少人在使用Git的过程中意外进入Vim编辑器之后，都不知道该如何编辑内容，甚至在知名的StackOverflow网站上关于“如何退出Vim”的问题有超过百万人浏览。

喜欢用Vim而且用习惯的人会觉得它非常好用，但这款软件的入门门槛有点高。限于篇幅，这里仅介绍它的基本使用方法，以便可以顺利地输入、存储、退出。

在Vim中，主要是通过模式的切换来进行输入、移动光标、选取、复制及粘贴等操作。常用模式有两种，即Normal模式和Insert模式，如图3-3所示。



模式说明如下。

（1）Normal模式又称命令模式，在该模式下无法输入文本，仅能进行复制、粘贴、存储或离开操作。

（2）输入文本前，需要先按下“i”“a”或“o”中的一个键进入Insert模式。其中，i表示insert，a表示append，而o则表示新建一行并开始输入。

（3）在Insert模式下，按下“Esc”键或“Ctrl+[”组合键，可退回至Normal模式。

（4）在Normal模式下，按下“:w”键将对文件进行存储，按下“:q”键将关闭文件（若未存储会提示先存储再离开），而按下“:wq”键则是存储完成后直接关闭文件。

Vim的命令非常多，但就在Git中会遇到的状况（主要是编辑Commit信息）来说，上述这些命令已经足够使用。

# 第4章 设置Git

## 4.1 用户设置

使用Git时，首先要做的就是设置用户的E-mail信箱及用户名。打开终端机窗口，输入下面两行命令：

$ git config --global user.name "Eddie Kao"

$ git config --global user.email "eddie@5xruby.tw"

输入完成后，可以再检查一下当前的设置：

$ git config --list

user.name=Eddie Kao

user.email=eddie@5xruby.tw

如果安装过Git相关的图形界面工具，那么git config --list命令可能还会输出其他额外的设置，而刚才的这两行设置至少会新增user.name和user.email两个配置。

如果不喜欢或不习惯输入终端机命令，也可通过SourceTree来设置，效果是一样的，如图4-1所示。

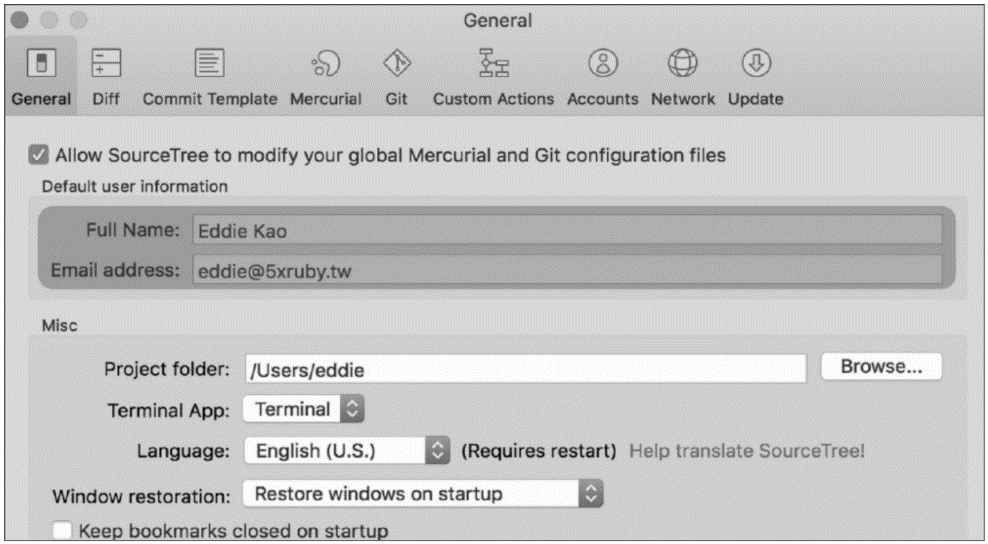


图4-1

设置文件的保存位置

不管是通过终端机命令还是图形界面工具完成的设置，所有Git相关的设置都默认保存在自己账号下的.gitconfig文件中，所以使用一般的文字编辑器直接手动修改该文件，也会有一样的效果：

[user]

name = Eddie Kao

email = eddie@digik.com.tw

[core]

excludesfile = /Users/eddie/.gitignore\_global # ... 略 ...

## 4.2 可以给每个项目设置不同的作者吗

可能有读者注意到了，前面在设置的时候多加了一个--global参数，其含义是要进行全局（Global）设置。但偶尔也会遇到一些特殊状况，例如，针对特定的项目设置不同的作者（不要问我什么时候需要做这件事），怎么办呢？可以在对该项目目录进行Git设置的时候，加上--local参数：

$ git config --local user.name Sherly

$ git config --local user.email sherly@5xruby.tw

## 4.3 其他方便的设置

程序员大多讨厌重复的工作，所以通常会设置一些方便的配置，以下是我自己在Git上通常会做的方便设置。

1. 更换编辑器

一般来说，Vim是Git默认的编辑器，所以对新手来说，在终端机下使用Git，Vim编辑器恐怕是个绕不过去的坎儿。虽然在3.2节介绍了这个神奇的编辑器如何使用，但对于平常不用，或者用不习惯的人来说还是会觉得很麻烦。

不过还好，不一定非要用Vim，可以在终端机执行以下命令：

$ git config --global core.editor emacs

其实，除了Vim和Emacs之外，还可以使用Sublime Text、Atom或Visual Studio Code等比较现代的文字编辑器。只需先搜索一下怎样从终端机启动这些应用程式，然后就可以用同样的方法把Vim换掉了。

如果在操作Git的过程中弹出Vim这件事情让你很困扰，那么建议搭配图形界面软件使用。

2. 设置缩写

虽然Git命令不长，但有时懒得打那么多字（如checkout命令就有8个字母），或者有些命令经常会打错（如status命令可能会打成state）。遇到这种状况，可以在Git中设置一些“缩写”，这样就可以少打几个字。只需在终端机执行以下命令：

$ git config --global alias.co checkout

$ git config --global alias.br branch

$ git config --global alias.st status

设置之后，只需输入git co命令，就可以实现与输入gitcheckout命令一样的效果；输入git st命令，就可以实现与输入git status命令一样的效果。

此外，还可以再加入一些参数。例如，每次在查看log时，为了看到比较精简的信息，都要输入git log --oneline --graph这么长的命令，而改用Alias设置（如下所示）：

$ git config --global alias.l "log --oneline --graph"

以后只需输入git l命令，就可以实现与原来的 --oneline --graph命令一样的效果了：

# 第5章 开始使用Git

## 5.1 新增、初始Repository

1. 如果是全新的开始

如果这是你第一次使用Git，那么就先从创建一个全新的目录开始吧。打开终端机窗口，并试着操作以下命令（命令后面的#是说明，不需要输入）：

$ cd /tmp # 切换至/tmp 目录

$ mkdir git-practice # 创建git-practice目录

$ cd git-practice # 切换至git-practice目录

$ git init # 初始化这个目录，让Git 对这个目录开始进行版控

Initialized empty Git repository in /private/tmp/git-practice/.git/

在上面的示例中主要做了以下几件事。

（1）使用mkdir命令创建了git-practice目录。

（2）使用cd命令切换到刚刚创建的git-practice目录。

（3）使用git init命令初始化git-practice目录，主要目的是让Git开始对这个目录进行版本控制。

其中，git init命令会在git-practice目录中创建一个.git目录，整个Git的精华都集中在这个目录中了。如果读者有兴趣，可以先看一下这个目录中的内容，不过现在并不打算介绍里面的细节，只是想让读者先体会一下使用Git的感觉，在后面的章节中再详细介绍。

3. 如果这个目录不想再被Git控制

其实，Git的版控很单纯，全都是靠.git目录在做事。如果这个目录不想被版控，或者只想给客户不含版控记录的内容，只要把.git目录移除，Git就对这个目录失去控制权了。

小提示

整个项目目录中，无论哪个文件或目录被删除了都能找回来，但如果.git目录被删除了就没有办法找回来了。

## 5.2 把文件交给Git管控

上一节对目录进行了Git的初始化，让Git可以开始管理这个目录。接下来，我们就来看看Git是怎样操作的。

### 5.2.1 创建文件后交给Git

#### 1. 创建文件

$ git status

On branch master

Initial commit

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

现在在这个目录中，除了Git生成的那个.git隐藏目录外什么都没有，所以提示“nothing to commit”（现在没有内容可以提交）。接下来，在这个目录中通过系统命令创建一个内容为“hello,git”的文件，并命名为welcome.html：

$ echo "hello, git" > welcome.html

这个步骤使用一般的文本编辑器或由文件管理员来完成都可以，总之在这个目录中创建一个名为welcome.html的文件即可。接着，再次使用git status命令，然后来看一下这个目录的状态：

$ git status

On branch master

Initial commit

Untracked files:

(use "git add <file>..." to include in what will be committed)

welcome.html

nothing added to commit but untracked files present (use "git add" to rack)

这时的状态与一开始不太一样了。这个welcome.html文件当前的状态是Untracked files，即这个文件尚未被加到Git版控系统中，还未正式被Git“追踪”，只是刚刚加入这个目录而已。

#### 2. 把文件交给Git

既然文件当前的状态是Untracked，接下来就要把welcome.html文件交给Git，让Git开始“追踪”它。使用的命令是gitadd，后面加上文件名：

$ git add welcome.html

在终端机执行这个命令不会输出任何结果。如果使用SourceTree，可以在welcome.html文件上右击，然后选择Add toindex选项，如图5-6所示。

这样就可以把该文件交给Git管控了。再次使用git status命令查看当前的状态：

$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: welcome.html

从以上信息可以看出，文件状态已从Untracked变成newfile。这表示该文件已经被安置到暂存区（Staging Area），稍后将与其他文件一起被存储到存储库中。

如果觉得git add welcome.html这样一次只加一个文件有点麻烦，也可以使用万用字元：

$ git add \*.html

这样就可把所有后缀名是.html的文件全部加到暂存区。如果想要一口气把全部的文件都加到暂存区，可以直接使用--all参数：

$ git add --all

### 5.2.2 如果在git add之后又改动了那个文件的内容该怎么办

设想一下下面这样一种情境。

（1）新增了一个文件abc.txt。

（2）执行git add abc.txt命令，把文件加至暂存区。

（3）编辑abc.txt文件。

完成编辑后，可能想要进行Commit，把刚刚改动的内容保存下来。这是新手很容易犯的错误之一，以为执行Commit命令就能把所有的异动都保存下来，事实上这样的想法是不正确的。执行git status命令，看一下当前的状态：

$ git status

On branch master

Changes to be committed:

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

new file: abc.txt

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in workingdirectory)

modified: abc.txt

可以发现，abc.txt文件变成了两个，这是为什么？其实步骤（2）的确把abc.txt文件加入暂存区了，但在步骤（3）中又编辑了该文件。对Git来说，编辑的内容并没有再次被加入暂存区，所以此时暂存区中的数据还是步骤（2）中加进来的那个文件。

如果确定这个改动是你想要的，那就再次使用git addabc.txt命令，把abc.txt文件加入暂存区。

### 5.2.4 把暂存区的内容提交到存储库里存档

如果仅是通过git add命令把异动加到暂存区，还不算是完成整个流程。如果想让暂存区的内容永久保存下来，就要使用git commit命令：

$ git commit -m "init commit"

[master (root-commit) dfccf0c] init commit

1 file changed, 1 insertion(+)

create mode 100644 welcome.html

在后面加上“-m "init commit"”是要说明“这次的Commit做了什么事”，只要使用简单、清楚的文本说明即可，中英文都可以，重点是要说清楚，能让自己和别人很快明白就行。

完成了这个动作后，对Git来说就是“把暂存区的内容存放到存储库（Repository）中”了。换言之，就是“我完成一个存档（或备份）的动作了”，即创建了一个第1章中提到的“版本”。关于存储库，将在5.3节中详细介绍。

注意！

要完成Commit命令才算是完成整个流程！

#### 1. 到底Commit了哪些东西

首先要记住一个很重要的观念：Git每次的Commit都只会处理暂存区中的内容。也就是说，在执行git commit命令时，那些还没有被加到暂存区中的文件不会被Commit到存储库中。

例如，如果新增了一个文件，但没有执行git add命令把它加至暂存区，那么在执行git commit命令时，该文件就不会被加至存储库中。

#### 2. 输入的信息很重要吗

对，很重要！很重要！很重要！

在Commit时，如果没有输入这个信息，Git默认是不会完成Commit的。它最主要的目的就是告诉你自己及其他人“这次的改动做了什么”。以下是几点关于信息的建议。

（1）尽量不要使用太过情绪化的字眼，以避免不必要的问题。

（2）英文、中文都可以，重点是要简单、清楚。

（3）尽量不要使用类似bug fixed这样模糊的描述，因为没有人知道你修正了什么bug。但如果搭配其他的系统使用，则可使用类似#34 bug fixed这样的描述，因为这样可以知道这次的Commit修正了第34号bug。

#### 3. 等等，怎么弹出了一个奇怪的窗口

在执行git commit命令时，如果没有在后面加上信息参数，默认会弹出一个黑色的画面，也就是编辑器——Vim。该编辑器对新手不太友好，可参考3.2节，或者直接使用SourceTree之类的图形界面工具来处理输入及提交信息的问题。

#### 4. 一定要有内容才能Commit吗

只要加上--allow-empty参数，没有内容也是可以Commit的：

$ git commit --allow-empty -m " 空的 "

[master 76a5b84] 空的

$ git commit --allow-empty -m " 空的 "

[master f4f568c] 空的

$ git commit --allow-empty -m " 空的 "

[master 7653117] 空的

这样就做了3个空的Commit出来，它们基本上没什么意义，但在上Git课的时候会很方便，可以不用新增文件就快速产生Commit来练习合并。

## 5.3 工作区、暂存区与存储库

5.2节介绍了可以使用git add命令把文件加至暂存区，然后再使用git commit命令把暂存区的内容移往存储库。

在Git中，针对工作目录、暂存区以及存储库3个主要区域，可以通过不同的Git命令，把文件移往不同的区域，如图5-9所示。

（1）git add命令可以把文件从工作目录移至暂存区（或下标）。

（2）git commit命令可以把暂存区的内容移至存储库。

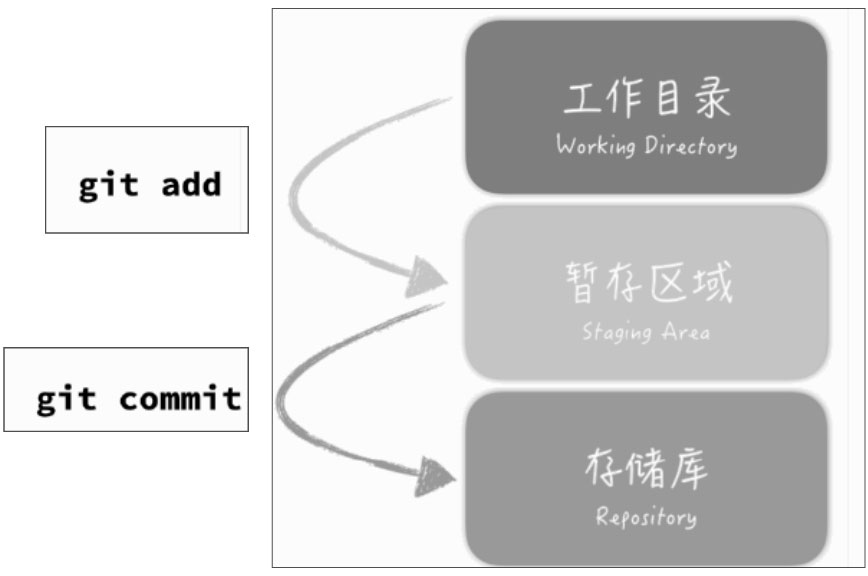


图5-9

注意，要执行Commit命令，也就是将文件存放到Repository区域，才算完成整个流程。基本上只要记得这3个区域怎样操作，在本机端的Git操作就没有太大的问题了。不管是用命令操作，还是用图形界面工具操作，都一定要亲自操作，熟悉这个流程。

### 1. 一定要二段式吗

如果觉得先add再commit有点烦琐，可以在Commit时多加一个-a参数，缩短这个流程：

$ git commit -a -m "update content"

这样即使没有先add，也可以完成Commit。但要注意的是，这个-a参数只对已经存在于Repository区域的文件有效，对新加入（也就是Untracked files）的文件是无效的。

### 2. 为什么要二段式这么麻烦

先add再commit，这样的二段式可能会让人觉得有点烦琐，但也是有好处的。假设有一个仓库，仓库门口有个小广场，这个广场就相当于暂存区。把要存放到仓库的货物先放到小广场（git add），等货物达到一定数量后打开仓库门，把小广场上的货物送进仓库中（git commit），并且记录下这批货的来源和用途。

当然，也可以每来一件货物就打开仓库门存一次、记录一次，但这样一来，开仓库的次数就会非常多。也就是说，这种操作方式会导致Commit太过零碎，在查阅记录时可能会不太方便。另外，过于零碎的Commit也可能给其他人带来一些困扰，例如，要进行Code Review的时候，比较有规律的Commit可以一次性看到比较完整的内容，而不需要一个一个Commit慢慢看。

### 3. 什么时候可以Commit

这个问题没有标准答案，可以将很多文件都改动好后一起Commit，也可只改一个字就Commit。常见的Commit时间点如下。

（1）完成一个任务时：大到完成一整个电子商务的金流系统，小到只加了一个页面甚至只改了几个字，都算是完成了一个任务。

（2）下班的时候：虽然可能还没有完全完成任务，但至少先Commit今天的进度，除了备份之外，也让公司知道你今天一直在努力工作。

（3）想要Commit的时候：只要想Commit，就可以Commit。

## 5.4 查看记录

### 5.4.1 查看记录的方法

接下来介绍如何查看之前Commit的记录。

因为当前只Commit了一次，不好比较，所以这里再次Commit，顺便也复习一下前文介绍的内容：

$ touch index.html # 创建文件index.html

$ git add index.html # 把index.html加至暂存区

$ git commit -m "create index page" # 进行Commit

#### 1. 使用Git命令

查看Git记录使用的是git log命令，其执行后的结果如下：

$ git log

commit cef6e4017eb1a16a7bb3434f12d9008ff83a821a (HEAD -> master)

Author: Eddie Kao <eddie@5xruby.tw>

Date: Wed Aug 2 03:02:37 2017 +0800

creat e index page

commit cc797cdb7c7a337824a25075e0dbe0bc7c703a1e

Author: Eddie Kao <eddie@5xruby.tw>

Date: Sun Jul 30 05:04:05 2017 +0800

init commit

越新的信息会显示在越上面。从上面这段信息中大致可以看出以下内容。

（1）Commit的作者是谁。

（2）什么时候Commit的。

（3）每次的Commit大概做了些什么事。

至于那个看起来像乱码的cef6e4017eb1a16a7bb3434f12d9008ff83a821a，其实是有特殊用意的。在Git中，这种看起来像乱码的文本，都是使用SHA-1（Secure Hash Algorithm 1）算法计算的结果。计算的方式会在5.17节进行详细的介绍，现在可以先把它当作一种重复概率非常低的文本。Git使用这样的字符串作为识别码。每个Commit都有一个这样的值，可以把它想象成每个Commit的身份证号。

在使用git log命令时，如果加上额外参数，可以看到不一样的输出格式。例如，加上-- oneline和–graph参数：

$ git log --oneline --graph

\* cef6e40 create index page

\* cc797cd init commit

输出的结果就会更为精简，可以一次性看到更多的Commit。

### 5.4.2 使用Git查询历史记录时的常见问题

以下是一些在使用Git查询历史记录时可能会遇到的问题（当前的历史记录如图5-13所示）。

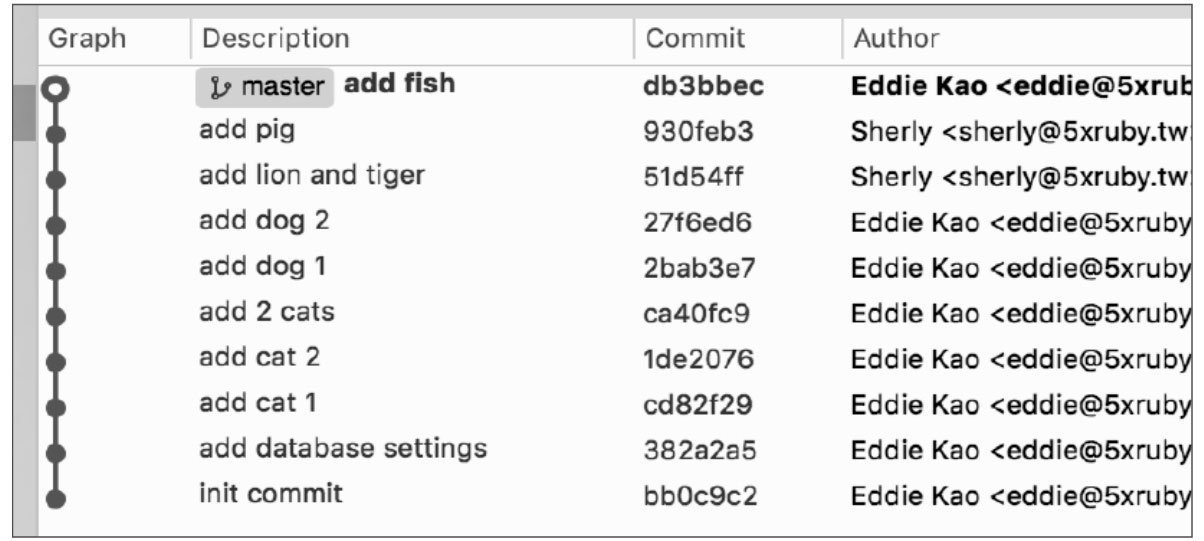


图5-13

#### 1. 想要找某个人或某些人的Commit

例如，想找一位名叫Sherly的作者的Commit，可以使用如下命令：

$ git log --oneline --author="Sherly"

930feb3 add pig

51d54ff add lion and tiger

此外，使用“\|”（“|”是“或者”的意思，前面需要加上“\”，否则会被判定为一般的文字而不是功能符号）可以查询Sherly以及Eddie这两个人的Commit记录：

$ git log --oneline --author="Sherly\|Eddie"

#### 2. 想要找Commit信息中是否含有某些关键字

使用--grep参数，可以从Commit信息中搜索符合关键字的内容，如搜索“LOL”：

$ git log --oneline --grep="LOL"

#### 3. 怎样在Commit文件中找到Ruby

使用-S参数，可以在所有的Commit文件中进行搜索，找到那些符合特定条件的内容：

$ git log -S "Ruby"

#### 4. 怎样查找某一时间段内的Commit

在查看历史记录时，可以搭配--since和--until参数查询：

$ git log --oneline --since="9am" --until="12am"

这样就可以找出“今天早上9点到12点之间所有的Commit”。还可以再加一个after：

$ git log --oneline --since="9am" --until="12am" --after="2017-01"

这样可以找到“从2017年1月之后，每天早上9点到12点之间的Commit”。

## 5.5 如何在Git中删除文件或变更文件名

### 5.5.1 删除文件

#### 1. 直接删除

可以使用系统命令rm或资源管理器之类的工具来删除文件。例如：

$ rm welcome.html # 删除文件welcome.html

然后看一下状态：

$ git status

On branch master

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in workingdirectory)

deleted: welcome.html

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

可以看到welcome.html文件当前的状态是deleted。如果确定这是你想做的，就可以把这次的“改动”加到暂存区：

$ git add welcome.html

再看一下当前的状态：

$ git status

On branch master

Changes to be committed:

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

deleted: welcome.html

它现在的状态是deleted，而且已被加至暂存区，所以接下来就可以进行Commit了。如果“把删除文件加到暂存区”让你觉得不好理解，就把“删除文件”也当作一种“改动”就行了。

#### 2. 请Git帮你删除

可以先执行rm命令删除，然后再执行git add命令加入暂存区的两段式动作，也可以直接使用git rm命令来完成：

$ git rm welcome.html

rm 'welcome.html'

这时候查看状态会发现：

$ git status

On branch master

Changes to be committed:

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

deleted: welcome.html

它就直接在暂存区了，不需要再add一次，可以少做一个步骤。

### 5.5.2 变更文件名

# 第6章 使用分支

## 6.1 使用分支的原因

在开发的过程中，一路往前Commit也没什么问题，但当越来越多的同伴加入到同一个项目中后，就不能这么随意地想Commit就Commit了。这时分支就派上了用场。例如，想要增加新功能，或者修正Bug，又或是想试试某些新的做法，都可以另外做一个分支来进行，在做完确认没问题之后再合并回来，不会影响正在运行的产品线。

在多人团队共同开发时，甚至也可以引入Git Flow之类的开发程序，让同一个团队的人都用相同的方式进行开发，减少不必要的沟通成本。

在Git中使用分支非常方便，成本也很低（原因在6.10节会详细说明），所以即使只有自己一个人开发，也推荐使用分支。

# 第10章 远端共同协作——使用GitHub

## 10.1 GitHub概述

### 1. GitHub是什么

GitHub是一个商业网站，是当前全球最大的Git服务器，如图10-1和图10-2所示。在这里，你可以跟一些优秀的开发者交朋友，为其他人的项目贡献自己的力量，或者为自己的项目寻求帮助。

同时，它也是开发者最好的履历展示平台，你曾经做过哪些专案和贡献、写过哪些Code，一目了然，想要造假非常难。

如果要上传Open Source项目，可以完全免费使用。如果要在上面开设私人项目，则需要交费，费用是$7/月。

### 2. Git与GitHub的区别

Git是工具，GitHub是网站。GitHub的本体是一个Git服务器，而GitHub这个网站则是使用Ruby on Rails开发的，别把这两个名词混淆了。

## 10.2 将内容Push到GitHub上

### 1. 在GitHub上创建新项目

要上传文件到GitHub，需要先在上面创建一个新的项目。首先在GitHub网站的右上角单击按钮，在弹出的下拉列表中选择New repository选项，如图10-3所示。

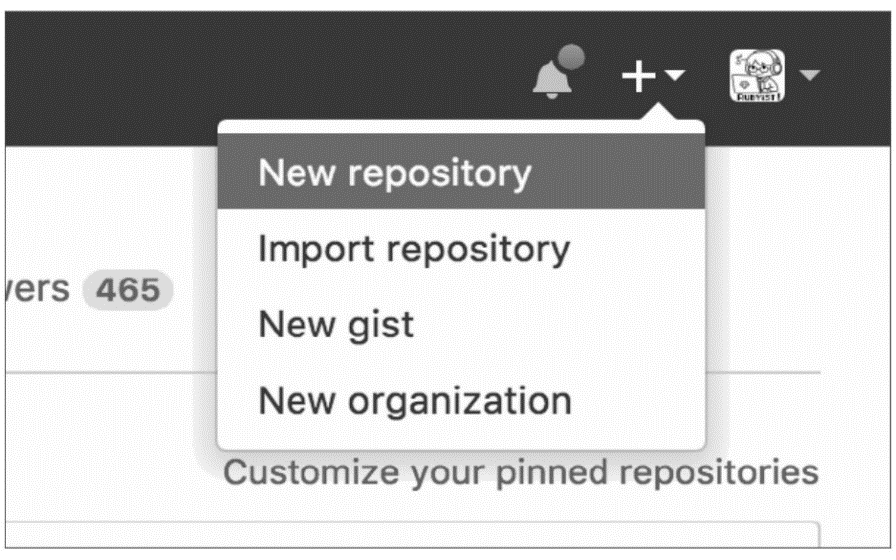


图10-3

接着填写项目名称，如图10-4所示。

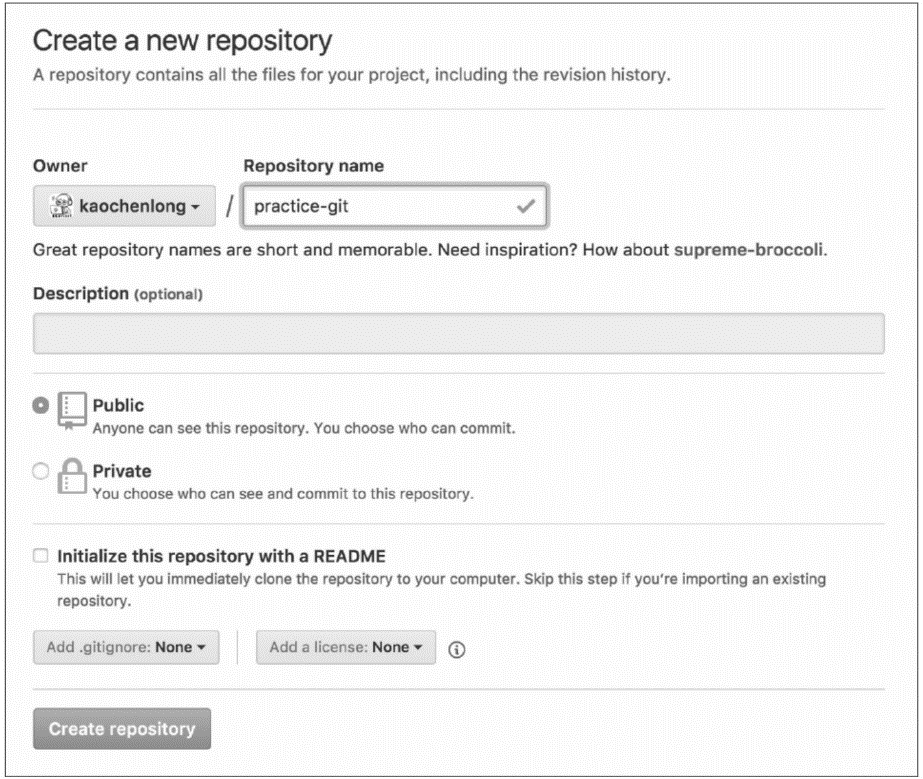


图10-4

这里有以下两点需要说明。

（1）Repository name可任意填写，只要不重复即可。

（2）存取权限选中Public单选按钮，可免费使用，选中Private则需交费$7/月。

单击Create repository按钮，即可新增一个Repository。

接下来会看到引导画面，如图10-5所示。

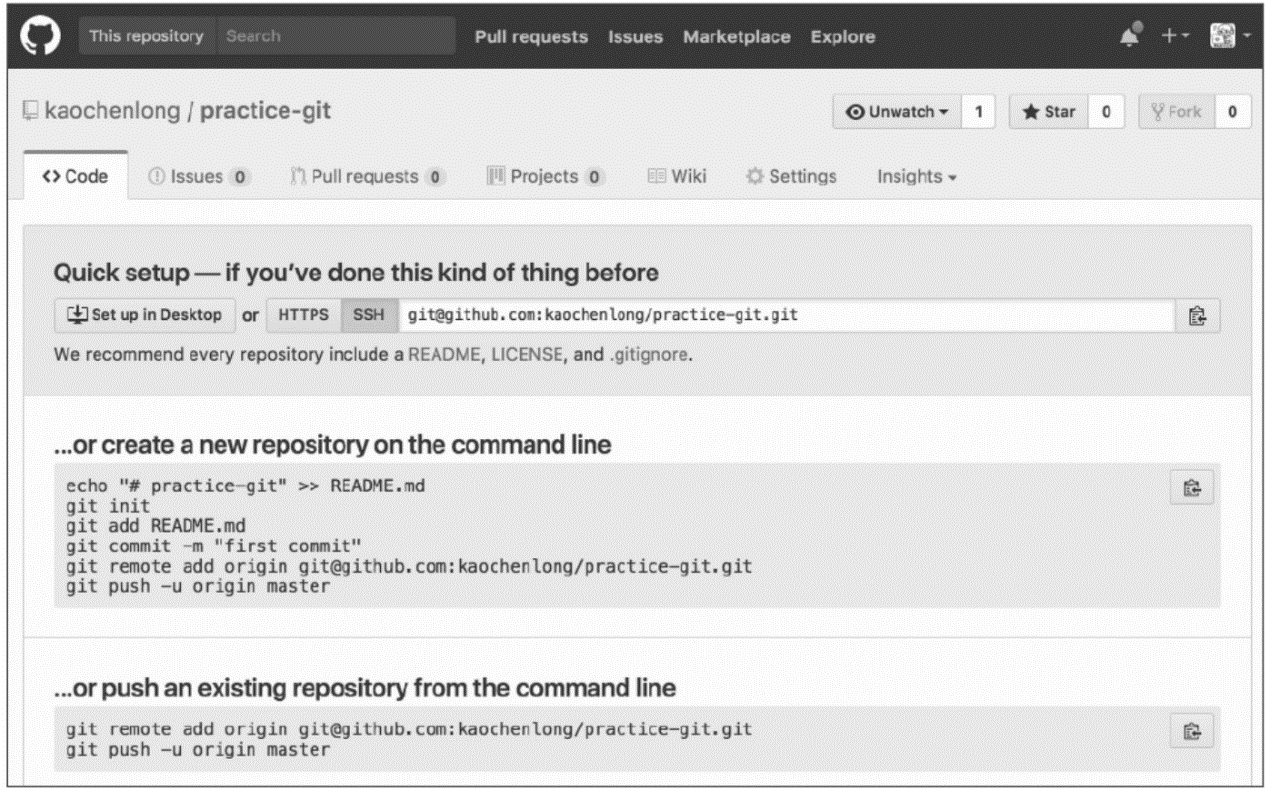


图10-5

这里有以下两点需要说明。

（1）如果是新项目，按照create a new repository on the command line的提示进行操作；如果是要上传现存项目，则按照push an existing repository from the command line的提示进行操作。

（2）在图10-5的中间有两个按钮可供切换，分别是HTTPS按钮和SSH按钮，可根据个人需要进行选择。如果单击SSH按钮，需要设置SSH Key（关于SSH Key的设置，可参阅10.1节的介绍。因为这里已经设置好SSH Key了，所以只需单击“SSH”按钮）。

如果仔细观察，就会发现选择全新开始与上传现有项目的最后两个步骤是一样的。

假设现在什么都没有，要重新开始一个项目，找一个空的目录，然后照着提示进行操作即可。首先创建一个README.md文件：

$ echo "# Practicing Git" > README.md

这个README.md是GitHub项目的默认说明页面，后缀名.md表示采用Markdown格式，Markdown语法可以很轻松地把纯文字格式转换为HTML的网页格式。如果是第一次接触该语法，推荐花点时间学习一下。

接下来就是我们熟悉的Git了（用git init指令针对目录进行Git初始化）：

$ git init

Initialized empty Git repository in /private/tmp/practice-git/.git/

$ git add README.md

到这里都还是Git的基本招式。如果忘记了怎样操作，可参阅5.2节。在这之前，所有的操作都是在自己的计算机上进行的，接下来就要准备把内容推上远端的Git服务器上了。首先，需要设置一个远端节点。例如：

$ git remote add origin

git@github.com:kaochenlong/practice-git.git

这里有以下三点需要说明。

（1）git remote指令主要进行与远端有关的操作。

（2）add指令是指要加入一个远端的节点。

（3）这里的origin是一个代名词，指的是后面那串GitHub服务器的位置。

按照惯例，远端的节点默认使用origin这个名称。如果是从服务器上Clone下来的，其默认名称就是origin。关于Clone的使用，可参阅10.5节。

但这只是惯例，不用该名称或之后想要更改都可以。例如，更改为七龙珠dragonball：

$ git remote add dragonball git@github.com:kaochenlong/practice-git.git

总之，它只是指向某个位置的代名词。设置好远端节点后，接下来就是把内容推上去：

$ git push -u origin master

Counting objects: 3, done.

这个简单的Push指令其实做了以下几件事。

（1）把master分支的内容推向origin位置。

（2）在origin远端服务器上，如果master不存在，就创建一个名为master的分支。