第 1 章　理解 Git 在 GitHub 中的作用

本章学习重点：

（1）熟悉 GitHub；
（2）探索 Git 的使用方法；
（3）注册 GitHub 账号；
（4）寻找有用的资源。

无论是经验丰富的程序员，还是新手开发者，如果想在计算机或互联网行业获得成功，那么学习如何与他人进行协作开发是非常重要的。GitHub 是世界上支持协同工作流的软件之一，它帮助全世界数百万人共同开发软件。本章将会介绍与他人进行协同开发所需的核心工具。

1.1　GitHub 简介

GitHub 所创建的环境允许将代码存储在远程服务器上，可以与他人共享代码，同时支持多人对同一个文件或者项目进行代码的添加、修改与删除，能够保证该文件的单一可信源。这意味着什么呢？此时可以将 GitHub 与 Google Docs 进行类比，即 GitHub 是一个可以在线与他人合作编写代码，从而不必反复收发邮件交换不同版本代码的地方。Git 可以保证 GitHub 有效运行。

1.2　版本控制

首先，我们要了解版本控制系统（SCM），也称为源代码控制管理。SCM 是能够跟踪编程项目中每个文件的每个版本、该版本创建的时间戳以及对这些文件进行更改的作者的软件。

编写代码是一个不停迭代的过程。例如，当编写一个网站时，我们可以先使用一些基础框架，能够有效运行后再对具体业务进行开发。但是当我们进行业务开发时往往会遇到许多问题导致程序崩溃，最好的办法就是创建版本机制，当我们项目能够有效运行时再进行版本的创建。这样，我们在进行下一阶段开发时，遇到问题导致程序崩溃时就可以回到上一个版本，重新开始。

各种 SCM 能够让程序员更敢于试错，因为不必担心犯下的错误会导致编码全部重来。我们也可以把 SCM 理解成“撤销”按钮，但这个撤销按钮不是逐步撤销的，而是一次撤销一整块。

使用版本控制系统的工作流程如下：

（1）在计算机的文件夹中创建一个项目；

（2）选择版本控制系统跟踪项目/文件夹的变化；

（3）每次项目处于工作状态或者暂时停止编写时，可以让所选版本控制系统将其保存为下一个版本；

（4）如果需要返回到以前的版本，可以要求所选版本控制系统返回到需要的任何以前的版本。

如果是在自己的计算机上单独工作，那么可以使用版本控制系统，而当需要与他人一起工作时，它会变得更加有趣，关于与他人合作的更多信息，请参阅“Git 版本控制”一节。

想要了解更多版本控制的信息，请浏览链接 1-1。

1.3 Git 的版本控制

GitHub 是构建在 Git 上的。Git 是一种版本控制系统，并且是免费开源的，这意味着任何人都可以使用它，可以在其之上构建项目，甚至可以对其进行优化。GitHub 的产品让使用 Git 变得简单。关于单个项目的本地 Git 命令将在后续进行简要介绍。

1.3.1 Git 终端的简单尝试

在 Git for Windows 的帮助下，我们在 Mac、Windows 或者 Linux 系统的计算机上使用的终端是完全相同的。终端是一种应用程序，它能够使用户以一种基于文本的方式与计算机交互，即不是通过双击和拖动来操控，而是通过输入命令来使用计算机。

如果使用的是 Mac 或者 Linux 系统，那么终端已经安装在计算机中了。如果使用的是 Windows 系统，那么需要安装 Git for Windows。请登录链接 1-2 并单击下载，我们使用此应用来访问 Git Bash。Git Bash 是一个模拟器，它允许用户像在 Linux 或 Mac 终端上那样与 Git 交互。本节假设计算机上已经安装了此应用。用户也可以下载 Git GUI，它提供了一个图形用户界面，在 Git Bash 中可能输入的所有命令都可以在 Git GUI 中找到，并且由于其集成了命令行解释器，用户可以在任何文件夹快速地打开 Git Bash 或者 Git GUI。

提示：许多 Windows 系统的开发人员更加喜欢使用 PowerShell 作为终端环境。当然也可以在 PowerShell 中使用 Git，但是如何对其正确配置超出了本书的范围，在此不过多赘述。而本书作者之一 Phil 在网站上提供了一个方便的操作指南，其网站为链接 1-3。

首先，我们要找到终端应用程序，方法如下。

（1）在 Mac 系统的计算机中，可以单击桌面右上方的放大镜，输入

“Terminal”，从应用程序列表中选择终端，按“Enter”键或单击它。

（2）在 Linux 系统的计算机中，同时按 Ctrl-Alt-T 组合键，打开终端窗口。

（3）在 Windows 10 系统的计算机中，单击桌面左下角的 Windows 菜单，

搜索“Git Bash”，从搜索结果列表中选择 Git Bash 应用程序，按“Enter”键或单击它。

当该程序打开后，在终端输入 git --version ，如果安装了 Git，那么应该会看到版本号，如下面的代码所示（因为 $ 已经在行中了，所以我们不需要输入它），否则可以按照链接 1-4 的说明进行操作。

注意：当使用命令行时，必须认真检查所输入的内容。在下面的代码中，第一条指令是输入 git --version。注意，git 和指令的其余部分之间出现了一个空格，但是没有其他空格。同时也应该注意单词版本前的两个“-”。它们很容易被忽略。

在 Mac 或者 Linux 系统的计算机中，出现的代码如下：

```
$ git –version
git version 2.16.3
```

在 Windows 系统的计算机中，出现的代码如下：

```
$ git –version
git version 2.20.1.windows.1
```

接下来，使用终端，输入以下命令，在计算机桌面创建一个名为 git practice 的新文件夹：

```
$ cd ~/Desktop
$ mkdir git-practice
$ cd git-practice
```

如果输入 pwd，则可以看到 git-practice 文件夹，该文件夹在计算机桌面上。显示如下：

```
$ pwd
$ /Users/sguthals/Desktop/git-practice
```

现在，可以使用 init 命令让 Git 跟踪这个文件夹：

```
$ git init
Initialized empty Git repository in /Users/sguthals/Desktop/git-practice
```

接着，在一个空文件中输入检查状态指令，确定其是否为空文件夹：

```
$ git status
On branch master
No commits yet
nothing to commit (create/copy files and use   " git add "   to track)
```

然后，创建一个文件，让 Git 开始跟踪并确认文件是否在文件夹中：

```
$ echo   " practicing git "   > file.txt
$ ls
file.txt
```

在 Mac 系统的计算机中，可以在 Finder 中用 open <path>命令打开这个文件夹：

```
$ open .
```

在 Linux 系统的计算机中，可以通过 nautilus <path>命令打开这个文件夹：

```
$ nautilus .
```

在 Windows 系统的计算机中，可以通过 explorer <path> 命令打开这个文件夹：

```
$ explorer .
```

在上面的例子中，我们在应当输入路径的位置用“.”代替，“.” 告知客户端应当打开当前文件夹。我们也可以使用以上命令打开不同路径的其他文件夹。

当打开文件夹后，双击 file.txt 文件，在不同系统的计算机中会使用不同的应用程序打开该文件。Mac 系统的计算机使用的是 TextEdit，Linux 系统的计算机使用的是 gedit，而 Windows 系统的计算机使用是 Notepad。此时可以看到“practicing git”字符串已经存在于该

文件之中了。

关闭这个文件，现在 Git 将它保存为一个特定的版本。返回终端界面：

```
$ git add file.txt
$ git commit -m " Adding my file to this version "
[master (root-commit) 8d28a21] Adding my file to this version
1 file changed, 1 insertion(+)
Create mode 100644 file.txt
$ git status
On branch master
nothing to commit, working tree clean
```

在刚刚的文本文件中更改文本。再次打开刚刚的文件，将文本改为“Hi! I’m practicing git today!”后保存更改后的文件，并且关闭文本应用程序。

当回到终端再次检查项目状态时，我们看到 Git 已经注意到文件已经改变了:

```
$ git status
On branch master
Changed not staged for commit:
(use " git add <file... " to update what will be committed)
{use " git checkout -- <file>... " to discard changed in working directory)
modified: file.txt
no changed added to commit (use " git add " and/or " git commit -a " )
```

再次提交这个版本的文件，注意 Git 会识别出所有文件的更改，而这些更改都被保存到了新版本中：

```
$ git add file.txt
$ git commit -m " I changed the text "
[master 6d80a2a] I changed the text
1 file changed, 1 insertion(+), 1 deletion(-)
$ git status
On branch master
nothing to commit, working tree clean
```

提示：如果终端满屏都是代码，那么可以输入 clear 来对终端进行清屏，让它看起来更干净整洁。我们可以向上滚动来查看之前输入的所有内容。

假如我们想看到该文件从最初 “practicing git”版本到当前版本的变化过程，那么可以获取提交的日志内容：

```
$ git log
commit 6d80a2ab7382c4d308de74c25669f16d1407372d (HEAD -> master)
Author: sguthals <sguthals@github.com>
Date: Sun Dec 9 08:54:11 2018 -0800
I changed the text
commit 8d28a21f71ec5657a2f5421e03faad307d9eec6f
Author: sguthals <sguthals@github.com>
Date: Sun Dec 9 08:48:01 2018 -0800
Adding my file to this version
```

然后我们让 Git 展示生成的第一个 commit。当生成一个 commit 时，Git 会分配唯一的哈希值。当我们使用 git log 指令时，该哈希值会以 commit hash 的形式显示在 commit 字符串后，本书的哈希值是以 8d28a2 开始的。当我们在某处需要使用 该哈希值时，请保证所输入的哈希值来自从 Git log 获取的结果。

提示：我们可以使用鼠标拖动标记显示的哈希值，然后右击并选择复制，再输入 git checkout 指令后，再粘贴该值，这样就可以避免输入错误了。注意，在此处复制快捷键 Ctrl+C 与 ⌘-C 无法使用。

```
$ git show 8d28a21f71ec5657a2f5421e03faad307d9eec6f
commit 8d28a21f71ec6567a2f5421e03faad307d9eec6f
Author: sguthals <sarah@guthals.com>
Date: Sun Dec 9 08:48:01 2018 -0800
Adding my file to this version
diff --git a/file.txt b/file.txt
new file mode 100644
index 0000000..849a4c7
--- /dev/null
+++ b/file.txt
@@ -0,0 +1 @@
+practicing git
```

我们可以看到，practicing git 是在初次提交中添加的。

关于如何在命令行中使用 Git 的更多信息，请查看以下资源:

（1）GitHub 中所使用 Git 指令备忘录：参考链接 1-5；

（2）可视化 Git 指令备忘录： 参考链接 1-6；

（3）Git Docs 网页：参考链接 1-7。

另外两个可供学习和理解 Git 的资源可参考链接 1-8 和链接 1-9 ，它们是网站形式，所以其允许使用浏览器进行访问。这样 Windows 系统上的用户就可以体验到 Git 在 Linux 或者 Mac 系统上的工作流程。链接 1-8 是一个非常棒的自学系统。 而链接 1-9 适合那些对于 Git 有一定了解并且想要深入学习的人。

1.3.2　基于协作人员对 Git 进行分支操作

Git 与其他版本控制系统最大的不同就是其具有高效率的分支系统。如图 1-1 所示，分支功能是 Git 的一个组成部分，它本质上包含以下三步：复制代码（每个分支都是一个代码副本），允许我们对特定的副本进行更改，然后将更改提交合并回主（master）分支。

当我们编写代码时，可以添加文件，并将更改提交至主分支。图 1-1 展示了这样一个工作流：两个人在同一文件上进行写作，第一个人创建了一个名为 MyBranch 的新分支，并对该文件进行了一些更改。而第二个人同样创建了一个名为 YourBranch 的新分支，并对同一个文件进行了更改，方框 1 显示了这一过程。

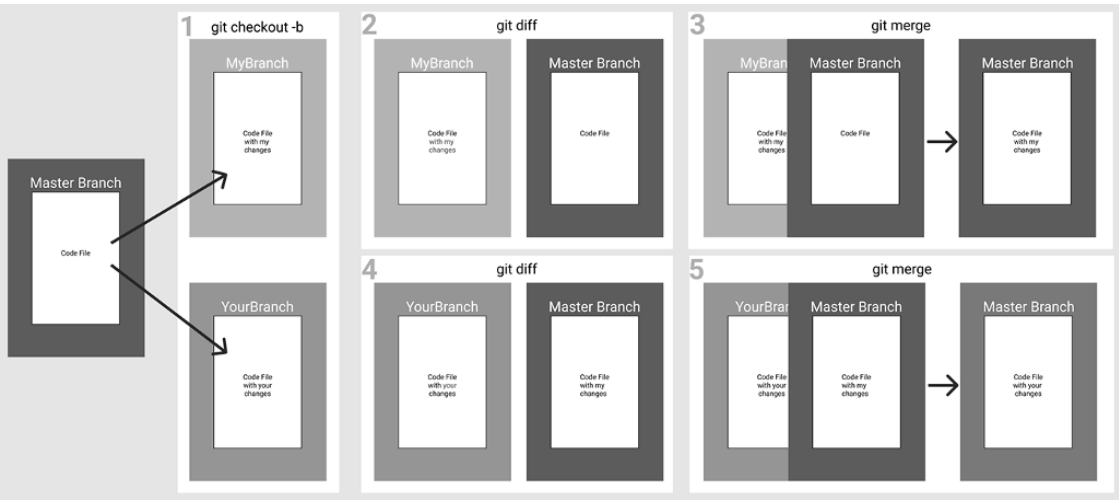

图 1-1 Git 分支的工作流

图 1-1 中的方框 2 显示了 master 分支和 MyBranch 之间文件内容的差异（称为 diff）。从图 1-1 中的方框 3 中可以看到，第一个人将更改与主分支进行了合并。第二个人进行了更改，但在合并之前，他们将确保他们拥有最新版本的主分支，该版本现在包含第一个人的更改。这种差异可以在图 1-1 方框 4 中看到。注意两个文件中的文本不同。最后，第二个人将他的更改提交，其会覆盖第一个人的更改，并且将他的更改与 master 分支进行合并，使得最终版本将与第二个人更改后的文本相同。方框 5 显示了主分支在两个人更改合并后的最终结果。

注意：图 1-1 仅显示了分支的某一种工作流，当多人在处理代码时可能会存在这种工作流，展示此工作流旨在描述解释分支。 更多有关 Git 和分支的更深入概述可以参考链接 1-10。

1.3.3 基于功能对 Git 进行分支操作

另一种使用分支的常见方法是让用户开发的每个功能都位于不同的分支，而忽略构建该功能的开发者。我们可以将“按功能分支”的思想加以扩展，再创建一个分支作为线上分支，该分支是项目用户会看到的分支。我们还可以使用一个开发分支将开发的新功能合并到原有的线上分支中，而此时的线上分支功能并不会受到影响。

这种类型的分支允许我们构建许多不同的功能，然后将它们每个合并到开发分支中。在合并完成后请确保它们都按照我们想要的方式工作，当对该功能分支完成测试确认其可上线后，将其合并到线上分支中。

1.3.4 用于实验的 Git 分支

我们还可以创建分支来测试某些内容是否有效，在测试完成后完全舍弃该分支。

提示：如果我们想要尝试一个全新的网页布局，使用这种类型的分支将会

很有用。我们可以创建三个不同的分支，每个分支都用不同的布局。在决定好喜欢的布局后，我们可以删除其他两个分支，并且将最喜欢的分支合并到主（master）分支。

1.4　Git 与 GitHub 的关系

GitHub 是 Git 仓库的托管平台。在某些情况下，将 Git 仓库放在一个可共享的平台是非常有用的。GitHub 既可以帮助代码进行备份，也可以让多人共同协作编码。作为一个 Git 仓库托管平台，GitHub 在除提供了 Git 的所有特性之外，还增加了一些有用的服务。

注意：在 GitHub.com 中，项目或者仓库将会被 GitHub 的远程服务器储存。

如果将所有的代码都存储在 GitHub.com 上， 如果计算机宕机，代码不会丢失，并且仍然可访问它。

下面是 GitHub 支持的一些 Git 核心特性：

（1）仓库：每个仓库包含与项目相关的所有文件和文件夹，并允许我们控制权限和合作者与代码的交互；

（2）克隆：当我们想要更改代码时，通常会想要在本地计算机上创建项目的副本或克隆。被克隆的项目仍然与 GitHub.com 上的版本紧密相连。这只是本地的复制；

（3）Fork：Fork 一个项目是创建整个项目的副本。当 Fork 一个项目时，GitHub.com 会用我们的所有文件的副本创建一个新的存储库。我们仍然可以更改原始项目，也可以更改新版本，向另一个方向进行开发；

（4）分支：GitHub.com 支持分支功能，甚至提供了一个有用的工具——Pull Request 来比较分支和所合并分支之间的差异；

（5）Commit：GitHub.com 跟踪所有推送到服务器的 Commit，并给一个简单的界面浏览在不同的分支和不同的 Commit 的代码。

1.5　在 GitHub.com 上进行注册

GitHub.com 为每个人提供无限制的免费公开和私有仓库。免费私人账户限三名协作者。我们可以注册一个付费账户，以拥有无限制人数的协作者和一些专业功能。公开意味着任何人都可以看到我们的代码，克隆我们的代码，从而使用我们的代码。GitHub 是开源软件（open source software/OSS）的大力支持者，因此 GitHub 鼓励公开、共享代码。开源软件不仅是公开、共享代码（请参阅第 5 部分）。因为每行代码都可以溯源，所以我们仍然可以因为所写的内容而获得荣誉。但是开源的目标仍然是让所有人都可以使用、扩展和探索这些代码。

以下步骤指导我们注册一个免费的 GitHub.com 个人账户:

（1）登录 GitHub.com，填写注册表格；

（2）选择想要的计划；如果只是为了浏览这本书，可以使用免费计划。如果想让私有仓库拥有三个以上的合作者和其他专业 GitHub 特性，可以随时升级到付费计划；

（3）完成简短的调查问卷，这个调查可以帮助 GitHub 了解谁在使用该软件，并帮助他们支持特定于用户的工作流程。进入首页，如图 1-2 所示。

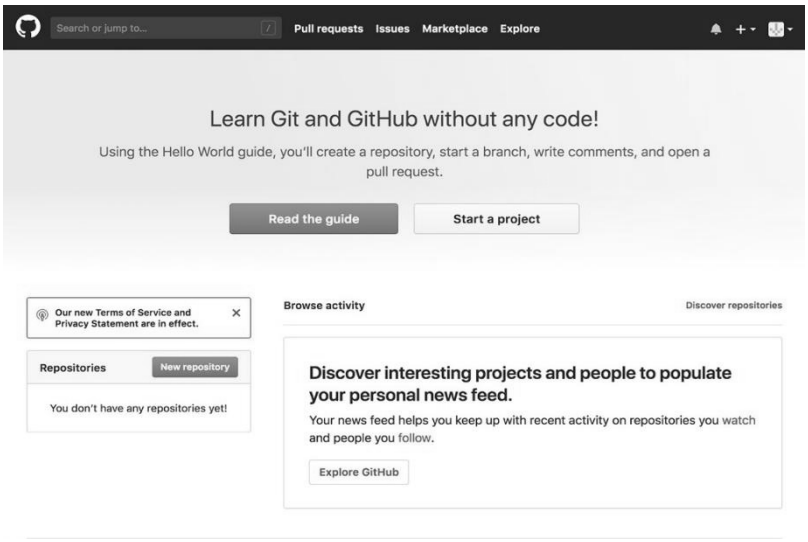

图 1-2 GitHub 登录后的主页

1.6 个性化 GitHub.com 账户

当成为一个更有经验的程序员后，我们可能想要在简历和工作申请中引用自己的 GitHub.com 的内容。越来越多的公司更关注我们的项目，而不是学位或者奖项。例如，在招聘过程中，GitHub 公司不要求申请者提供教育背景信息，而是要求提供 GitHub 网站的个人资料或者作品链接。

要完善自己的 GitHub 个人资料，我们应该：

（1）单击击右上角的头像图标，选择 Your Profile；

（2）在出现的页面上单击 Edit Profile；

（3）在 Personal Settings 页面填写相关内容，如图 1-3 所示；

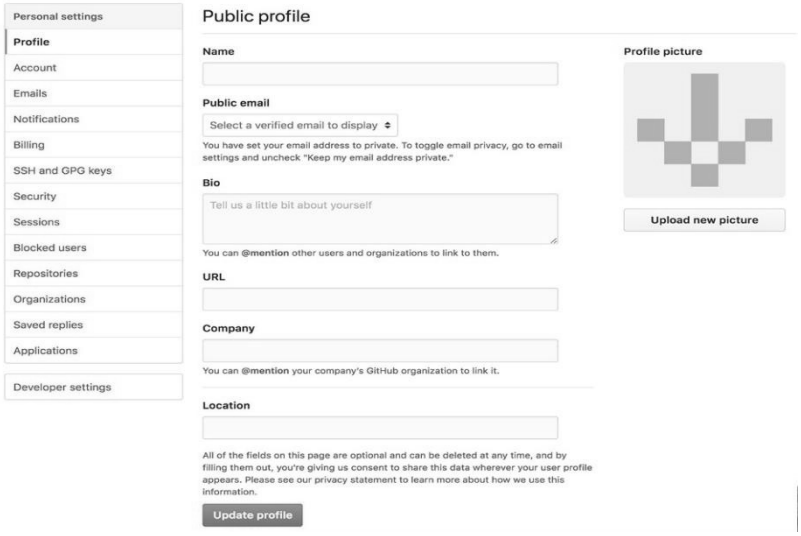

图 1-3 个人设置页面

（4）单击 Update 完成修改。

在个人设置（Personal Settings）页面，还可以调整多个不同的设置，以继续个性化我们的账户。

1.6.1　账户（Account）

在账户设置中，我们可以修改密码、更改用户名或删除账户，如图 1-4 所示。

注意：更改用户名可能会产生意想不到的后果，所以通常不建议这样做。

在更改用户名之后，我们需要确保工作所需的项目正常运行。请检查一下包含我们账户名的链接，测试一下代码，然后测试应用是否有问题。

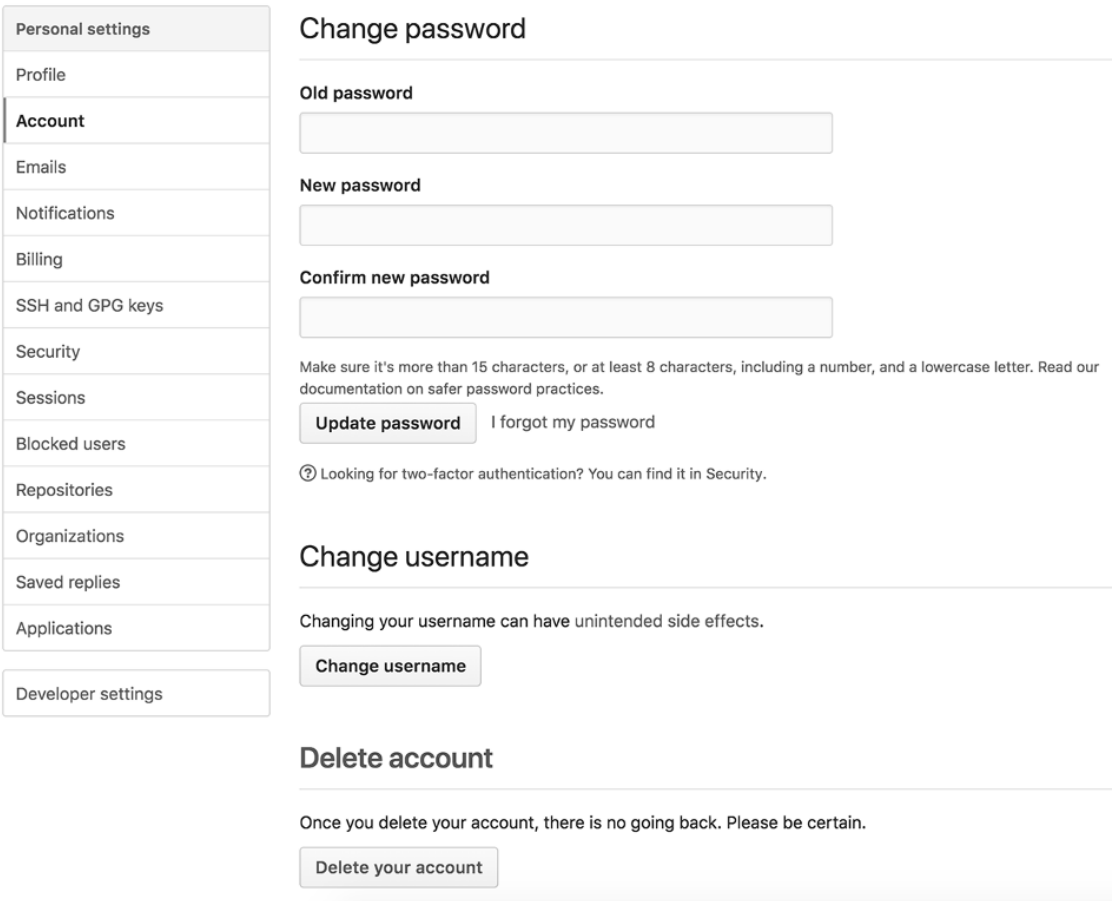

图 1-4　账户设置

1.6.2　电子邮箱（Emails）

GitHub 允许我们链接多个电子邮件地址到自己的账户。注意，我们可以添加电子邮件地址，也可以让我们的电子邮箱不对外公开，甚至 GitHub 会阻止可能暴露电子邮件地址的 Git 命令，如图 1-5 所示。

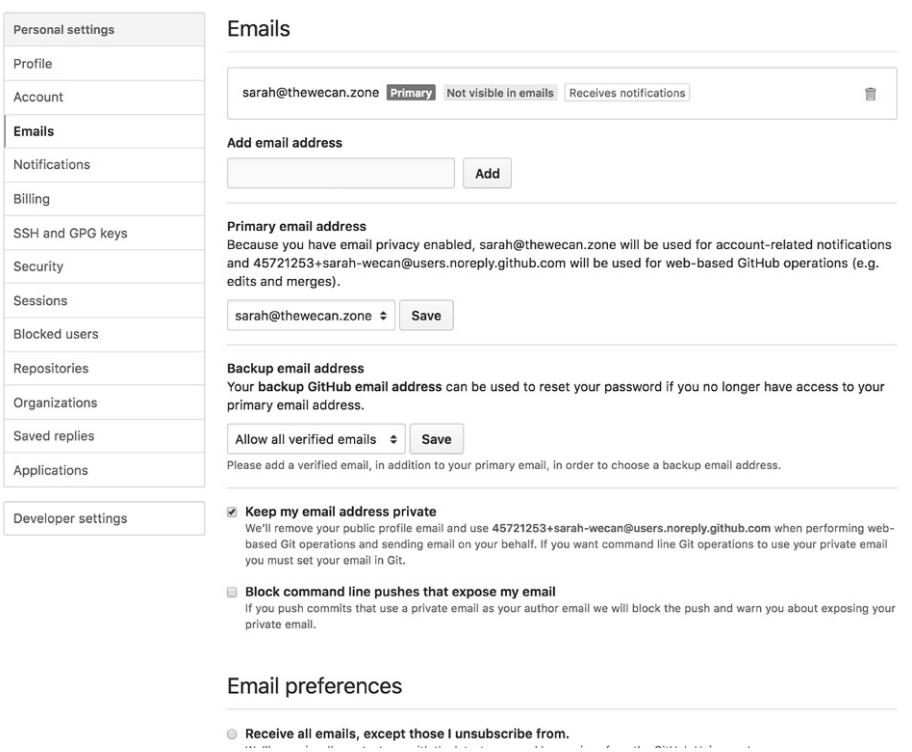

图 1-5 电子邮箱设置

1.6.3 通知（Notifications ）

通知真的会让人不知所措，尽管我们可以为每个仓库选择接收通知的粒度级别，但此页面将为通知创建默认首选项，参见图 1-6。

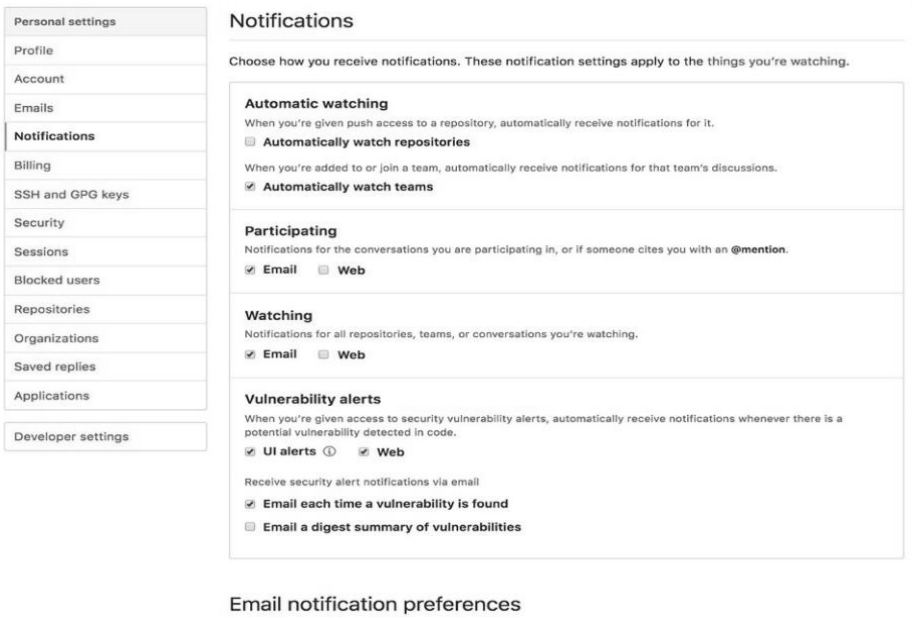

图 1-6 通知设置

注意：我们建议不要选择自动监控仓库，因为任何与用户交互的仓库上发生的活动都将开始显示在该用户的收件箱中，当开始更多的合作时，这会将该用户的收件箱变得十分冗杂。

提示：单击通知设置页面顶部的 things you're watching 链接，查看我们正在监控哪些仓库，从而了解到可能会收到哪些通知。

1.6.4　费用（Billings）

我们可以随时升级计划，以使用专业功能，如无限协作者和高级代码审查工具。我们可以在费用设置页面进行此升级，如图 1-7 所示。除了升级计划，我们还可以购买 Git LFS 数据和其他 Marketplace 应用程序。当然也可以从公司、学校或附属机构获取优惠券。

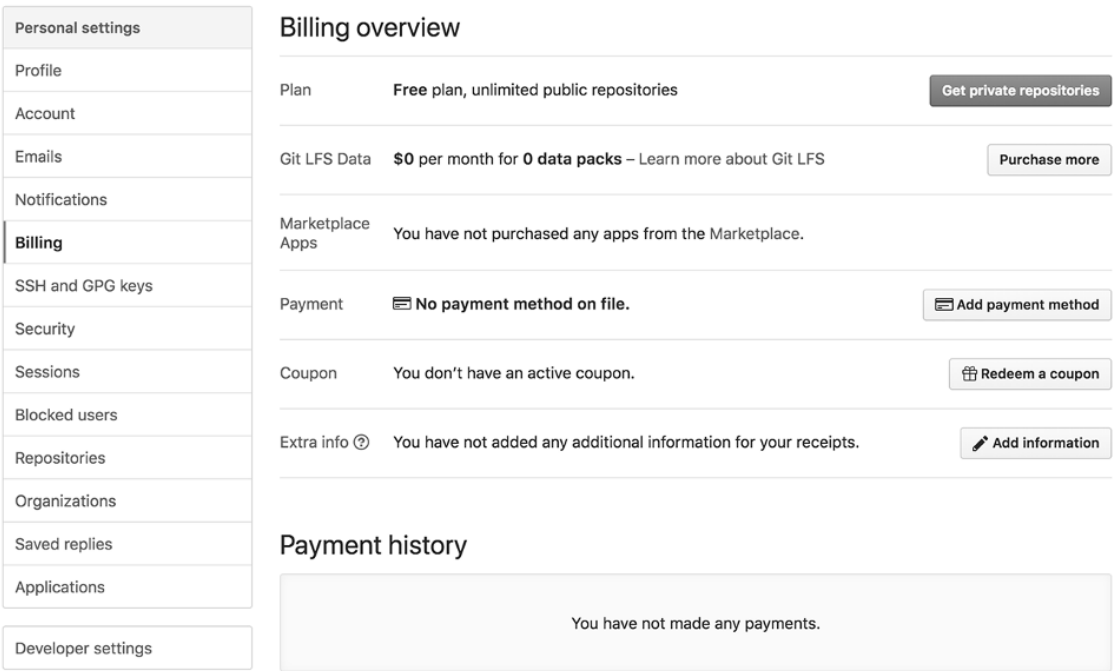

图 1-7　费用设置

提示：Git LFS 代表了 Git 中的大文件存储。一些软件开发需要存储大文件，如电子游戏开发中的游戏场景。如果没有 Git LFS，我们只能上传最大约 100MB 的文件。任何比其更大的文件上传都需要 Git LFS 的支持，其让 Git 支持上传几十 GB 以上的文件。

1.6.5　SSH 和 GPG 公钥

在某些时候，我们可能希望创建一个 SSH 或 GPG 密钥来加密我们与 GitHub 的通信，并确保通信过程在安全的环境下进行。可以在设置中执行此操作，如图 1-8 所示。

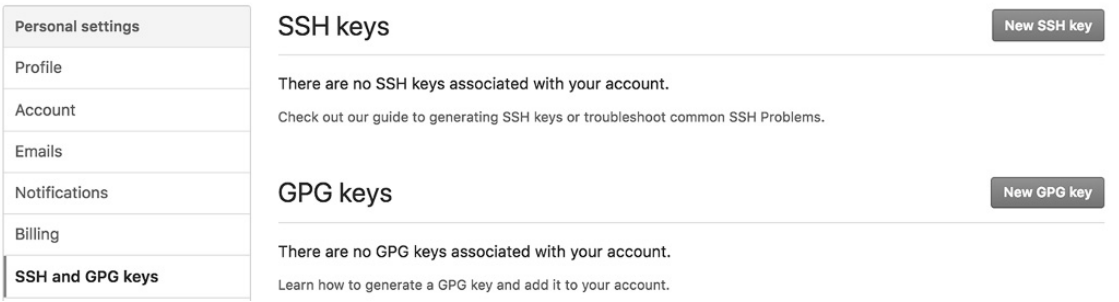

图 1-8 创建 SSH 和 GPG 密钥

注意：SSH 密钥使我们无须每次都输入用户名和密码就可以从本地计算机连接到 GitHub。 GPG 密钥将我们所做的标记和 Commit 标记为已验证，这意味着其他人知道实际上是我们提交了该更改。

提示：另外让 Git 记住我们拥有其证书的方式是使用凭据小助手（credential helper）。但是，GitHub 倾向于推荐使用 SSH，尤其是对于 Windows 用户。有关如何设置此功能的更多信息，请访问链接 1-11。

1.6.6 安全（Security）

要保障我们的用户安全，我们的密码设置得比较复杂，还应当启用两步验证。两步验证表示当我们输入正确的密码后，我们会被要求进一步验证，例如通过应用程序或者短信验证，只有通过两步验证后，才能登录用户。

1.6.7 时域（Sessions）

时域页面允许我们查看自己账户登录或连接到 GitHub.com 的所使用的每个计算机的所在地址、城市和国家。

1.6.8 屏蔽用户（Blocked users）

在屏蔽用户设置中，我们可以屏蔽特定用户访问自己的仓库。

1.6.9 仓库（Repositories）

仓库页面列出了我们已经创建的或被邀请作为合作者的所有仓库，我们还可以从该页离开我们所加入的仓库。

1.6.10 组织（Organizations）

组织允许我们将 GitHub 用户和仓库置于相似的设置下。例如，我们可以将组织中所有仓库的管理权限授予整个组织，而不必分别将每个人添加到每个仓库。虽然组织的相关知识已经超出了本书的范围，但可以在链接 1-12 的 GitHub 帮助页面上阅读它们。

1.6.11　回复模板（Saved replies）

回复模板如图 1-9 所示，对于流行的 OSS 非常有用。例如，如果我们正在构建应用程序的扩展功能，许多人可能会报告原有应用程序的问题，而不是我们的扩展功能问题。我们可以编写一个回复模板，告诉人们在发现错误时可以在哪里提供应用程序的反馈。

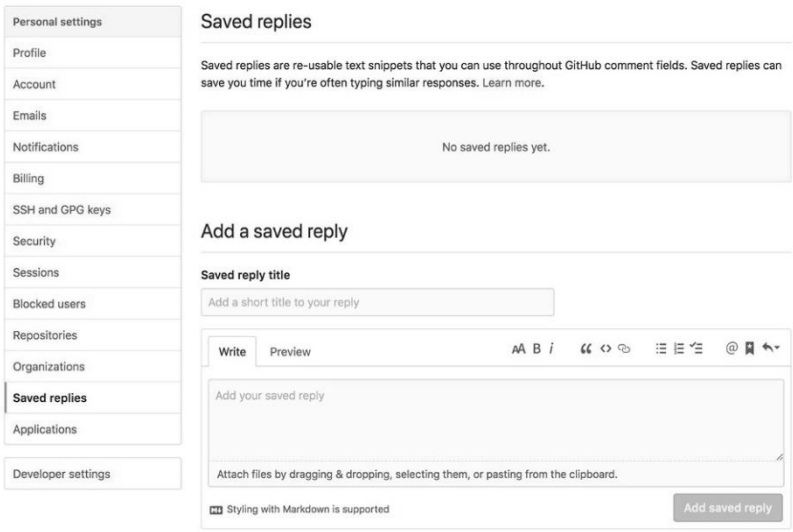

图 1-9　保留回复设置

1.6.12　应用程序（Applications ）

我们可以用自己的 GitHub.com 账户关联三种应用程序:

（1）已安装的 GitHub 官方应用：用我们的 GitHub 账户来登录一些 GitHub 官方应用，例如登录 GitHub 学习实验室（GitHub Learning Labs）；

（2）使用 GitHub 授权登录其他应用程序：我们已授权使用我们的 GitHub 账户登录其他一些应用程序，如登录 Slack；

（3）授权 OAuth 应用程序：授权我们的 GitHub 账号登录 GitHub 证书认证的其他应用程序，如登录 GitHub Desktop。

1.6.13　开发者设置（Developer settings）

设置页面的最后一部分是开发者设置，只有在构建访问 GitHub API 的应用程序时，即应用程序需要以某种方式访问 GitHub 数据时，才使用它。

此部分出现三个设置：

（1）OAuth 应用程序：我们已注册使用 GitHub API 的应用程序；

（2）GitHub 官方应用程序：与 GitHub 集成和扩展的应用程序；

（3）个人访问令牌：类似于 SSH 密钥，允许我们无需身份验证即可访问 GitHub API 的令牌。

1.7 如何发现有用的资源

GitHub.com 帮助（Help）页面（参考链接 1-13）为 GitHub.com 上的每个功能提供了大量文档列表。在右上角的头像跳出的菜单中，我们可以单击帮助链接。在帮助链接中，单击 Contact a Human 会将我们带到 GitHub 联系（Contact）页面（参考链接 1-14），我们可以在其中找到以下资源：

（1）常见问题解答（FAQ）；

（2）帮助文档的链接；

（3）有关 GitHub API 帮助的开发人员文档链接（参考链接 1-15）；

（4）用于指导 GitHub 练习的 GitHub Learning Lab（参考链接 1-16）；

（5）GitHub 社区论坛，我们可以在这里提问并从 GitHub 工作人员或者其他社区成员那里获得问题的答案（参考链接 1-17）；

（6）在 GitHub.com 可以提高我们开发经验的其他资源。

如果这些资源都没有帮助，可以访问链接 1-18 并填写快速表格以提出特定的问题。

第 2 章　设置协作编程环境

GitHub 为新老程序员提供了许多功能。作为一个新手，熟悉 GitHub 功能的好方法就是探索 GitHub.com 官网。本章不仅介绍这些功能，还将指导如何设置好本地环境，以便开始开发。

2.1　探索 GitHub.com

GitHub.com 主页如图 2-1 所示，在这里可以创建新项目，学习一个 topic 或是探索已有的项目。

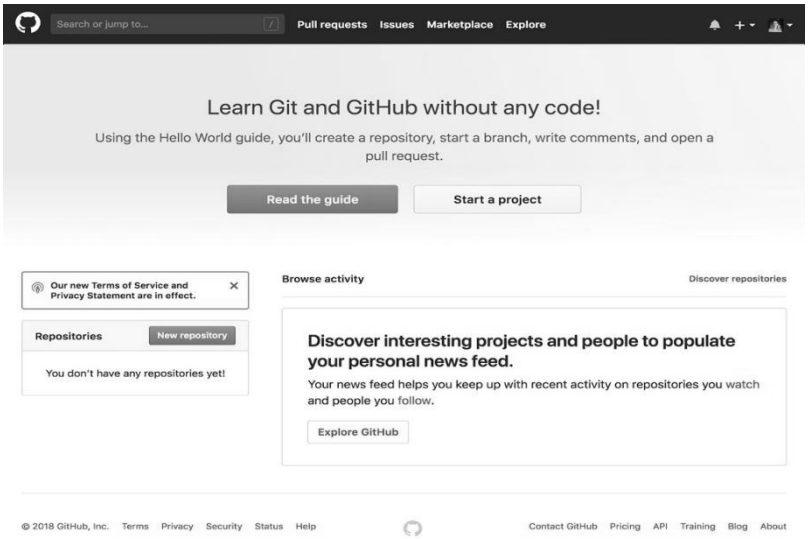

图 2-1　GitHub.com 的主页

顶部菜单栏（如图 2-2 所示）是指向 GitHub 中最重要的几项功能的直接链接，菜单栏将一直停留在页面顶端以供使用。

1）GitHub 主页

如果单击浏览器左上角的 GitHub 标志，将返回主页。阅读“Mona Lisa Octocat”小贴士以获取有关该标志的更多信息。

2）Search Bar

顶部菜单上的搜索框（Search Bar）是十分有用的工具。 在搜索框中不仅可以搜索 GitHub 的所有内容，单击搜索框时，它还会根据最近的活动提供搜索建议。 这些建议有助于快速轻松地找到昨天工作过的仓库。

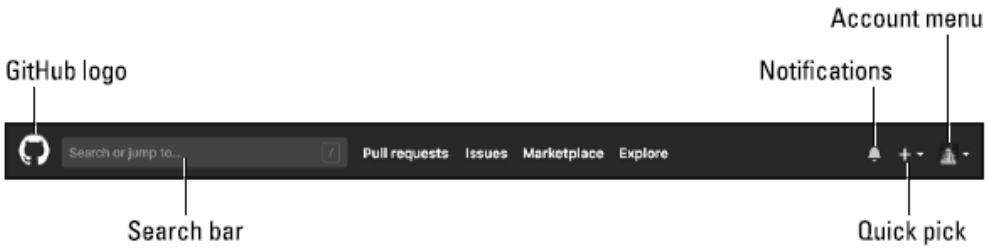

图 2-2 GitHub.com 的顶部菜单

3）MONA LISA OCTOCAT

GitHub 的标志实际是一只拥有五只章鱼般手臂的猫，该标志从 iStock 网站上发现。iStock 是一个可以购买免版税数字图像的网站。该标志的设计师是 Simon Oxley，他同时也是 Twitter 标志的设计师。随后，Cameron McEfee 领导了在 Octodex 网站上展示 Octocats 的工作。Octodex 网站是一个向全世界展示 Octocats 的网站。其中的所有内容都可以在 链接 2-1 上查看。

Mona Lisa Octocat 也有一个经过验证的 Twitter 账户，见图 2-3。我们可以在链接 2-2 上关注该账户以获取所有最新动态。

图 2-3 Mona Lisa Octocat 的 Twitter 账户

Mona 发布的最受欢迎的应用程序之一是“Build Your Own Octocat”，这个应用可以用于创造自己喜好的 Octocat，应用可以在链接 2-3 上找到。

要了解 Octocat 的完整历史，请访问链接 2-4。

4）Pull request

Pull request 按钮会跳转至由你创建的、被分配、被提及或者被要求审查的 Pull request 列表。一个 Pull request 是对仓库代码的一个提议。开始使用 GitHub 时，这个页面通常没有

任何内容。在开始与他人协作工作后，会有许多的 Pull request 需要处理。关于 Pull request 的更多内容请参阅第 3 章。

如果单击 Pull request 按钮，在页面下端会出现一个 Pro Tip，如图 2-4 所示。Pull request 的搜索栏提供了多种方法来进行搜索，以确保能够准确地获取要查找的内容。事实上，整个页面（参考链接 2-5）都致力于高效的搜索。在 GitHub.com 各个地方都可能出现 Pro Tips，在探索 GitHub 的过程中一定要留意它们。

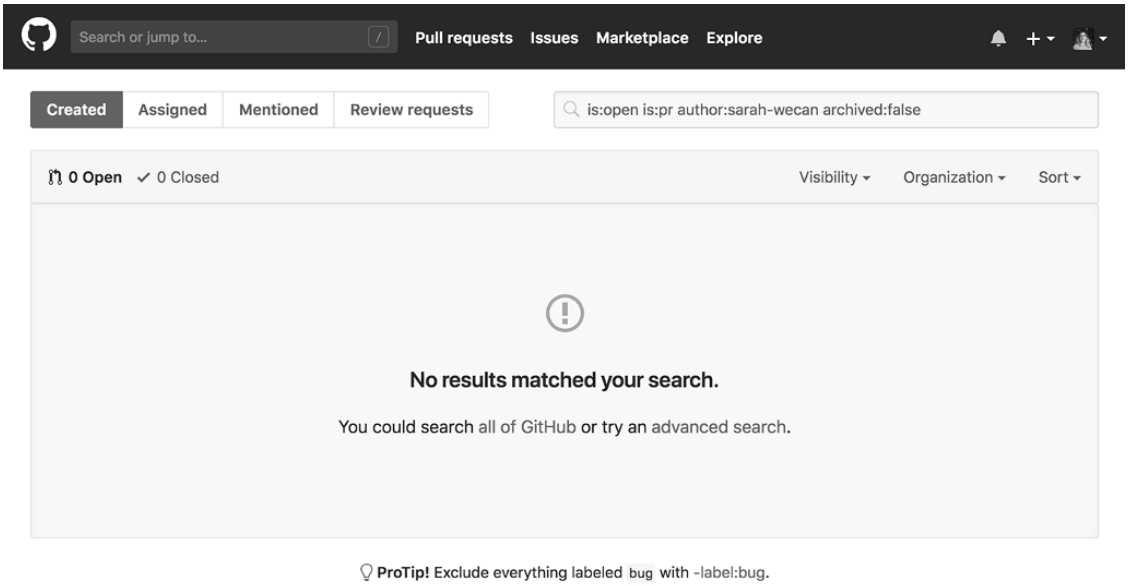

图 2-4　如何在 Pull request 找到 ProTip

5）Issues

Issues 列表与 Pull request 列表几乎相同。 Issue 和 Pull request 之间的主要区别在于：Issues 或是用于报告一个错误或是用于提出一个新功能。Issues 还与 Pull request 有一个不同点在于，Issue 不包含代码更改，因此不需要审阅者。

6）Marketplace

GitHub 上的 Marketplace 是查找有助于协作编码工作流程的应用程序和工具的好地方。例如能够在项目中使用的持续集成应用程序 AppVeyor，将 AppVeyor 连接到一个仓库时，它会持续运行测试并部署应用程序，以确保添加的每行代码都不会破坏已经构建的内容。

7）Explore

Explore 链接会跳转到可能感兴趣的领域（参见图 2-5）。在这里可能会找到 GitHub 主办或支持的活动和挑战。例如，在编写这本书的时候，GitHub 刚刚发布了“The State of the Octoverse”，它提供了很多关于 GitHub 的有趣分析——例如，GitHub 用户在 2018 年做出了 11 亿的代码贡献。

8）Notifications

铃铛图标将引导跳转至通知列表。请参阅第 1 章，了解如何更改通知设置。

9）Quick pick

加号图标提供了一个可以在任何时候执行的快速操作列表，例如创建一个新仓库；从另一个 SCM 导入仓库；创建一个 gist （一种共享代码、注释和代码片段的快速方法）或者创建一个新的组织。

10）Account Menu

账户菜单：单击个人头像时，将会出现账户菜单。在这里可以跳转至个人首页、个人仓库、star 过的项目、创建的 gist、帮助文档和设置，并可以注销用户。

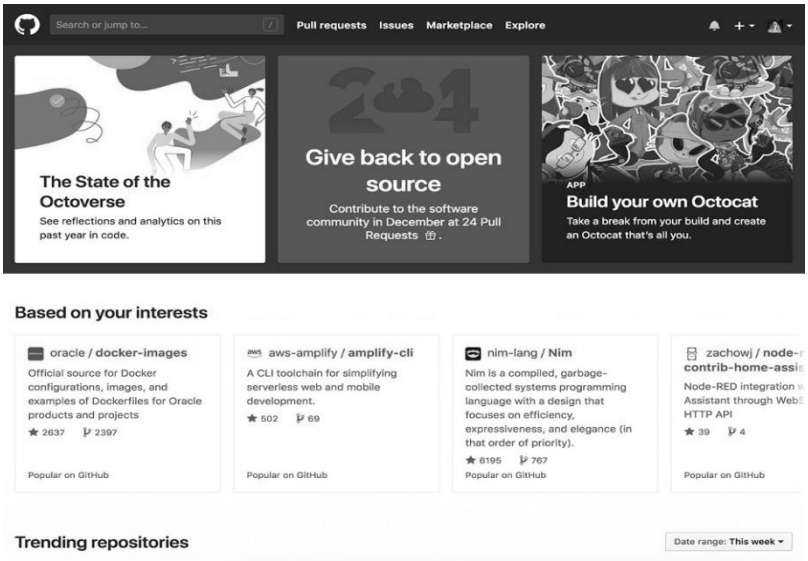

图 2-5　GitHub.com 上的库列表

2.2　个人首页

从本质上说，个人首页是个人项目经历的一个公开展示页面。要查看你的个人首页，请单击头像，然后从出现的菜单中选择 Your Profile，如图 2-6 所示。

个人首页页面的顶部菜单栏提供了一些快速链接，这些链接将会链接到个人仓库、star 的仓库、关注者以及正在关注的用户。顶部菜单下面是经常访问的仓库和贡献图。贡献图会追踪每天编写了多少代码。还可以将对私有仓库的贡献和活动概览也包括进来，这是一个新的特性。

注意：编写代码的数量和频率并不是衡量软件开发的标准。的确，练习得越多，越能获得成长，但练习必须是经过思考的。如果不挑战自己，花费时间思考，设计想要编写的代码，只是随机修改代码让方块变绿，还不如让格子保持空白。

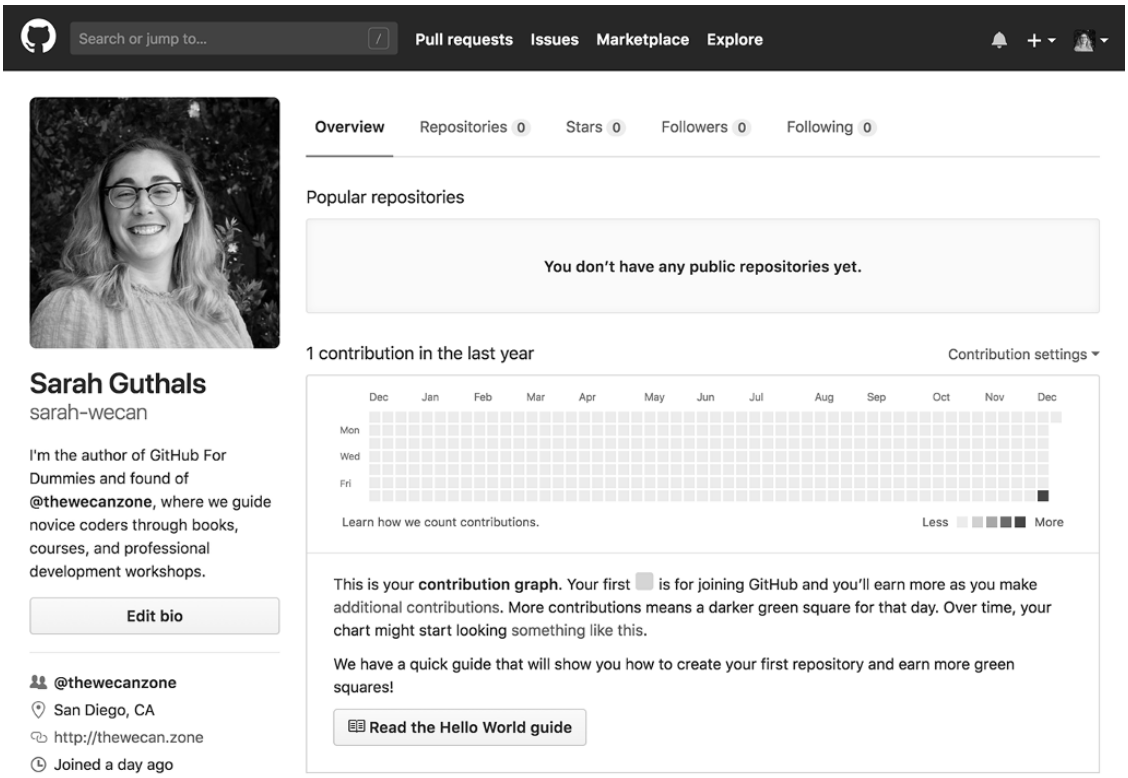

图 2-6　My Profile 页面

2.3　了解 GitHub Desktop

GitHub Desktop 是一个免费的开源应用程序，它使 Mac 和 Windows 用户更容易在他们的本地计算机上管理仓库以及与 GitHub 建立连接。

Desktop 是开源的，这意味着可以跟踪其新特性的开发、与正在构建应用程序的实际仓库中的开发人员进行联系，甚至可以选择添加感兴趣的特性。在链接 2-6 上可以找到这个仓库。

在个人仓库上安装此应用程序需要如下步骤。

（1）登录链接 2-7，单击下载正在使用的平台的对应版本。

这本书的编写是以谷歌 Chrome 和 Mac 为基础的。GitHub Desktop 也可以在 Windows 系统的计算上工作，因为它是用 Electron 构建的，这使得它可以在这两种操作系统上工作。

（2）下载完成后，单击已下载的文件

文件将会自动解压缩。在 Mac 上，GitHub 桌面应用程序将出现在下载文件夹，并且出现在所下载 zip 文件的旁边。在 Windows 上，解压缩文件后，应用程序立即打开。

（3）在 Mac 上，拖动紫色 GitHub 桌面应用程序到 Applications 文件夹。

（4）在 Mac 上，进入应用程序文件夹，双击 GitHub 桌面图标。

打开应用，如图 2-7 所示。注意：可能会出现一个警告，显示“正在试图打开一个从互联网下载的应用程序”。如果出现此警告，请单击“打开”。

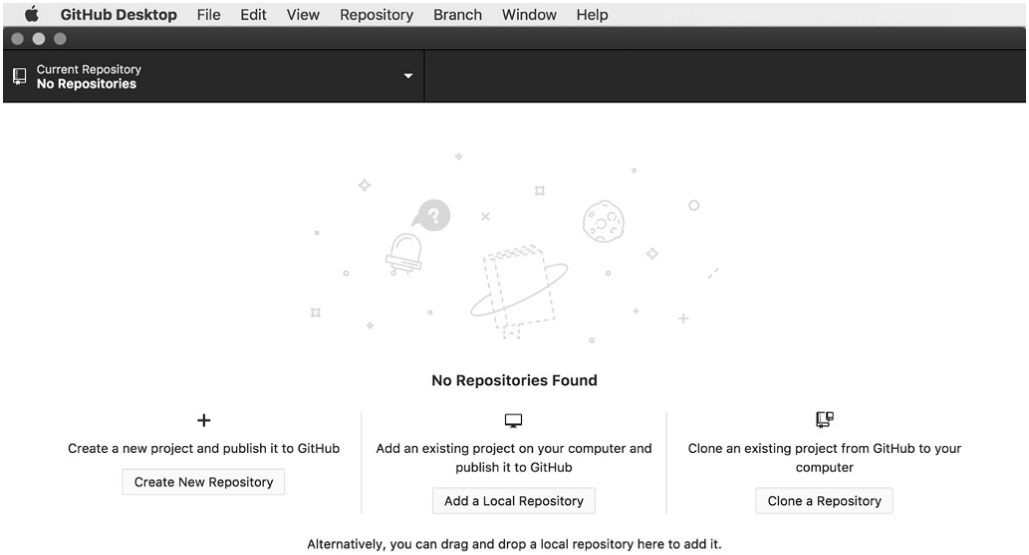

图 2-7 GitHub 桌面应用程序默认视图

2.4 设置 Github Desktop

必须先登录个人 GitHub 账号才可以正常使用 GitHub Desktop。如果还未拥有 GitHub 账号，请参考第 1 章。如果已经拥有一个 GitHub 账号，并且已经下载了 Github Desktop，则可以按照以下步骤设置 GitHub Desktop。

（1）打开 Github Desktop 应用程序。

（2）选择 File 下的 Preferences。

（3）在账户选项卡上，单击 GitHub.com 上的 Sign In，系统弹出“登录”对话框，如图 2-8 所示。

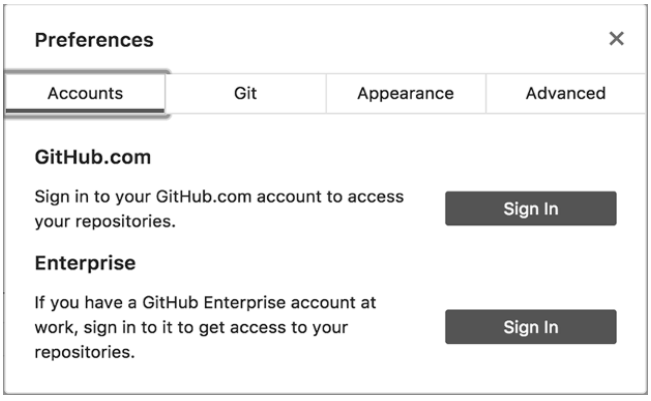

图 2-8 GitHub Desktop 的 Sign in 对话框

（4）输入用户名和密码，然后单击“登录”按钮或单击“使用浏览器登录”。单击 Sign in 时，所有对话框将关闭。

（5）重复步骤 1 重新打开首选项。

确认你的账户与头像，姓名和 GitHub 用户名出现在 GitHub.com 行下，

此时已成功登录。

（6）单击 Git 选项卡，Github Desktop 会自动填写你的信息。

（7）在 Appearance 选项卡中，可以选择 Light 或者 Dark 主题，本书中的截图是在 Light 主题下。

（8）在 Advanced 选项卡上设置其他首选项，如编辑器和使用数据。

本书使用 Atom 和 Visual Studio Code 作为示例编辑器，但是你可以选择

喜欢的任何编辑器。本书建议同意发送使用数据，这是默认勾选的。该选项可以帮助 GitHub Desktop 开发团队了解用户的反馈并加以改善，从而使其改进。如果你的计算机上还没有 GitHub 仓库，可以先停止设置。如果已有一个仓库，请参阅第 4 章。

注意：虽然 GitHub 的团队主要负责 GitHub Desktop 的开发，但他们的一

部分职责是支持希望为项目做出贡献的社区成员。不要犹豫，可以在链接 2-8 上与该团队联系。

2.5　Atom 的介绍

Atom 是一个免费的、开源的编辑器。和 GitHub Desktop 一样，Atom 也建立在 Electron 上，可以在 Mac 或 Windows 系统的计算机上运行。Atom 是可扩展性的，这意味着可以向它添加自己想要的功能。通过访问 Atom 团队的仓库可以了解团队成员正在做什么，可以参考链接 2-9

Atom 是一种轻量级编辑器，安装并不需要耗费很长时间。若需要安装 Atom，请访问链接 2-10 并单击下载。

如同前一节中安装 GitHub Desktop 的过程一样，当 Atom 完成下载时，单击以解压文件。在 Mac 上，Atom 应用程序出现在下载文件夹中。将 Atom 应用程序拖放到应用程序文件夹中。在 Windows 或 Mac 上，双击应用程序打开它。此时，将出现如图 2-9 所示的内容。

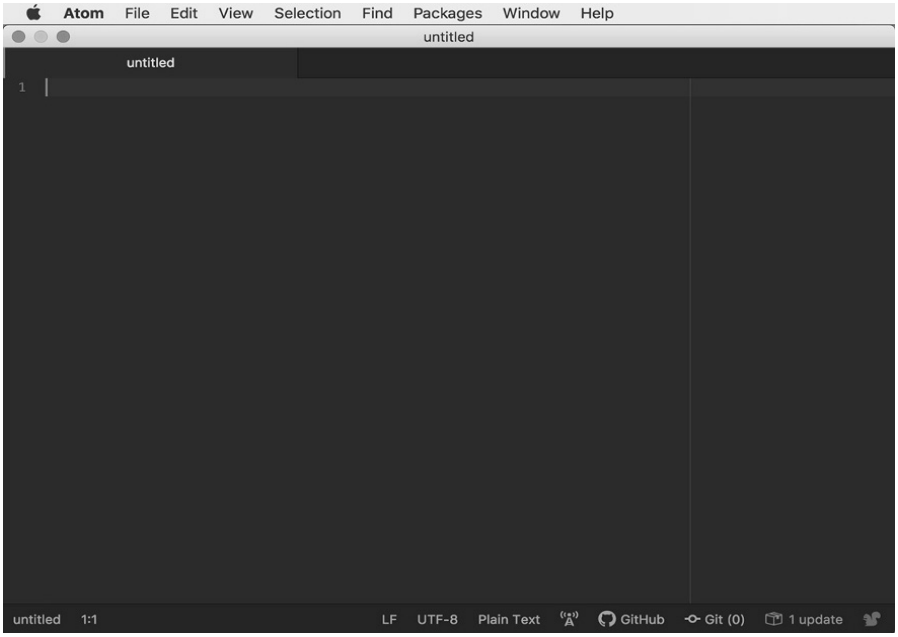

图 2-9　Atom 的默认页面

此时可能会出现一个警告，显示“正在试图打开一个从互联网下载的应用程序”。如果出现此警告，请单击“open”。

以下是关于 Atom 的几个注意点。

更新：每次启动 Atom 时，请确保检查右下角是否需要进行任何更新，以使软件保持最新版本。单击 Update 链接时，Settings 选项卡将打开，包将出现在屏幕上。在图 2-10 中，可以看见之前已经安装过的包 teletype，它需要更新。在单击 Update to 0.13.3 或 Update All（如果有多个需要更新的包）之后，重新启动 Atom。我们可以单击 Squirrel 图标转到 Atom About 页面，以确保 Atom 编辑器也是最新的。

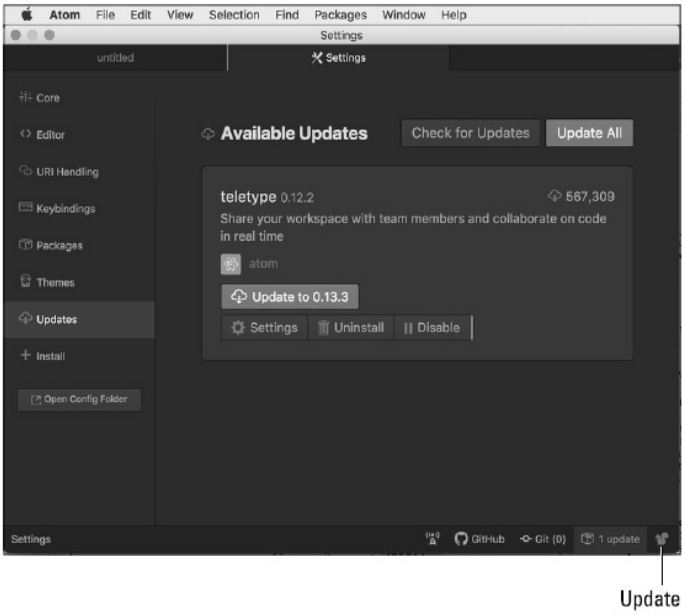

图 2-10　Squirrel 图标跳转到 About 页面，在这里可以更新 Atom

Git 和 GitHub 标签：这两个标签在应用程序底部的右边，更新按钮旁边，你可以通过选择 Packages➪GitHub➪Toggle Git/GitHub 标签轻松打开 Git 和 GitHub 标签。

请注意 Atom 中的一些菜单项有热键绑定，例如图 2-11 中的 Git 和 GitHub 选项卡。热键是可以在键盘上按下的特殊的按键组合，以便在特定的应用程序中实现某些功能。之前可能已经介绍了一部分。 例如在浏览 Internet 时，可以通过在 Mac 上输入⌘-T 或在 Windows 上输入 Ctrl+T 来打开一个新选项卡。在试图提高自己编码的专业度并且寻找提高编码效率的方法时，使用热键是一个非常有效的策略。

首选项：使用 Atom 时可以依照个人习惯指定许多首选项。 本书中不会对此逐一解释，但鼓励读者浏览它们，并使用其来设置 Atom，这可以在极大程度上提高编码效率。请通过选择 Atom➪preferences（如图 2-12 所示）来找到首选项。

软件包：Atom 拥有超过 8000 个软件包，通过安装软件包，可以使其最符合个人使用习惯。在 preferences 或链接 2-11 上可以找到这些软件包。 介绍每一个包超出了本书的范围，但若在使用 Atom 的过程中觉得功能有限，本书鼓励读者探索并搜索其中一些包。

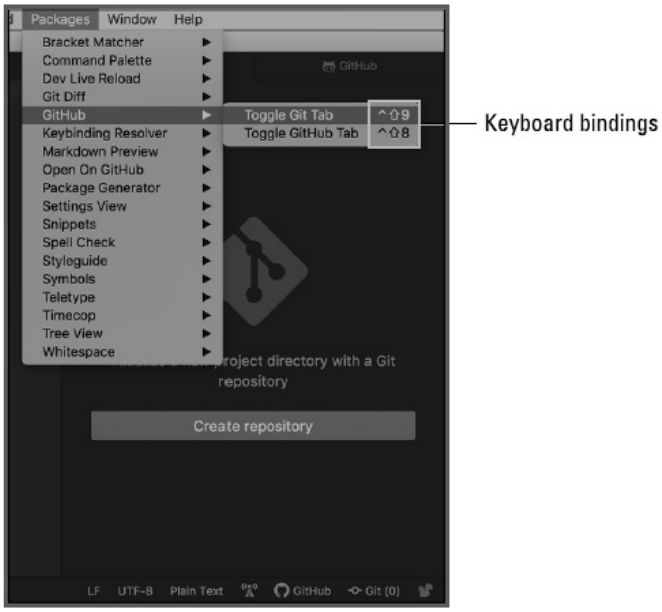

图 2-11　用于切换到 Git 和 GitHub 页面的热键

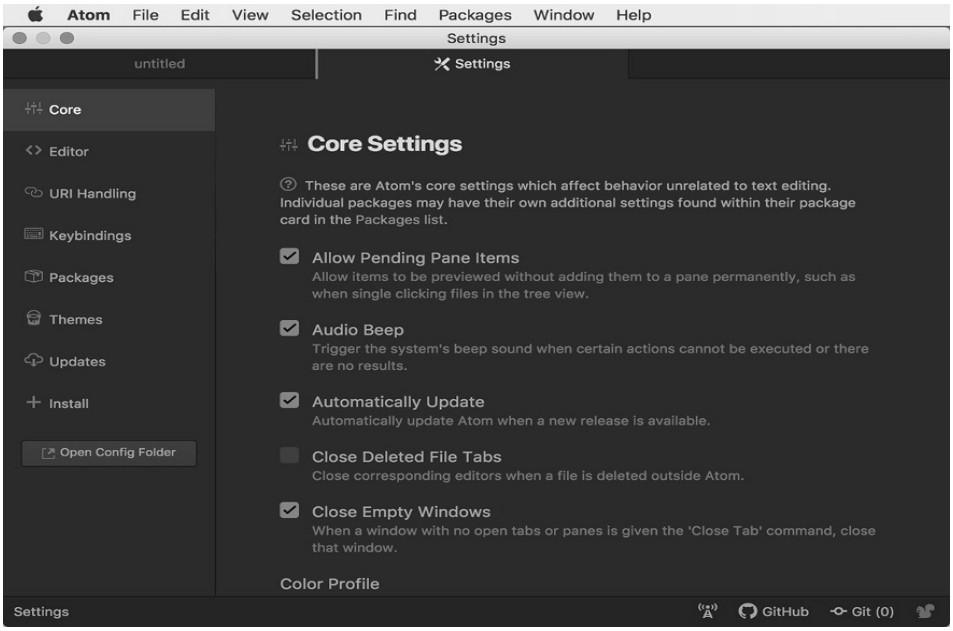

图 2-12　Stom 的 Settings 页面

注意：若需要寻找本书之外的资源，请在链接 2-12 上查看 Atom 文档，或者在链接 2-13 上查看 Atom 手册。这两种资源都可以帮助指导解决可能遇到的任何问题。 如果这两种资源不包含该问题的解决方法，可以通过访问链接 2-14 上的开放源代码仓库，与使用 Atom 的开发人员联系。