

第 12 章 在 GitHub 外部进行协作

本章学习重点：

- (1) 在 Slack 中设置 GitHub 集成；
- (2) 在 Trello 中设置 GitHub 集成；
- (3) 安装 Octobox 来管理通知。

虽然 GitHub 可能是一个软件开发项目的中心（因为其托管了关键的交付物，即源代码），但 GitHub 并不是发生协作的唯一地方。软件开发团队使用各种工具来沟通和协调软件开发任务。许多并不是开发者的人员也在软件项目中工作，这些人也希望在他们使用的工具中了解项目的进展。例如，大量的日常协作发生在聊天室，如 Slack。一些人可能会使用 Trello 看板来管理团队的任务。还有一些人可能会使用 Octobox 来关注来自于 GitHub 的最新通知。

在本章中，我们将探讨将 GitHub 信息集成到其他协作工具中的各种方法。本章与第 13 章相反，第 13 章会介绍如何将其他工具的信息集成到 GitHub 以改善软件开发工作流程。

12.1 应用 Slack 聊天

对于许多团队，尤其是分布式团队，聊天是团队成员合作和协调工作的有力方式。这里的聊天不是指在门廊上喝茶，谈论他们一天的情况。聊天指的是基于文本的工具，比如 Slack，来进行同步和异步的沟通。

许多团队发现，使用 GitHub 把重要的通知发布到聊天室里是很有帮助的，这样团队就能随时了解到仓库的情况了。在这一节中，我们将介绍相关设置，将 GitHub 集成到一个流行的聊天软件——Slack 中。在安装这个集成之前，用户需要成为 Slack 工作区的管理员。可以通过链接 <https://slack.com> 创建一个免费的 Slack 工作区。

设置好 Slack 工作区后，安装 GitHub for Slack 集成需要两个关键步骤：

- (1) 在 Slack 工作区安装 GitHub 应用程序。
- (2) 将 Slack App for GitHub 添加到 GitHub 账户。

以下各节将详细介绍这些步骤。

12.1.1 为 Slack 安装 GitHub 应用程序

为 Slack 安装 GitHub 应用程序，包含以下几步。

(1) 进入链接 <https://slack.github.com>，单击浏览器窗口中央的“Add to Slack（添加到 Slack）”按钮。如果在浏览器中没有登录 Slack 工作区，单击“添加到 Slack”按钮会提示用户登录 Slack 工作区。同样地，如果没有登录 GitHub，网站会提示先登录 GitHub。对两者都进行了认证后，会出现一个 Slack 应用的确认界

面，界面上将显示 GitHub 应用对 Slack 工作空间的权限信息。图 12-1 显示了确认界面完全展开的样子（默认情况下是折叠的），可以看到集成后的所有功能。

提示：请确保已将 GitHub 应用添加到正确的 Slack 工作区。如果显示的是错误的工作区，可以通过单击本界面右上方的工作区下拉列表来切换。用户还可以选择将集成安装到哪一个 Slack 工作区。如果没有列出工作区，可以单击“Sign in to another workspace”。

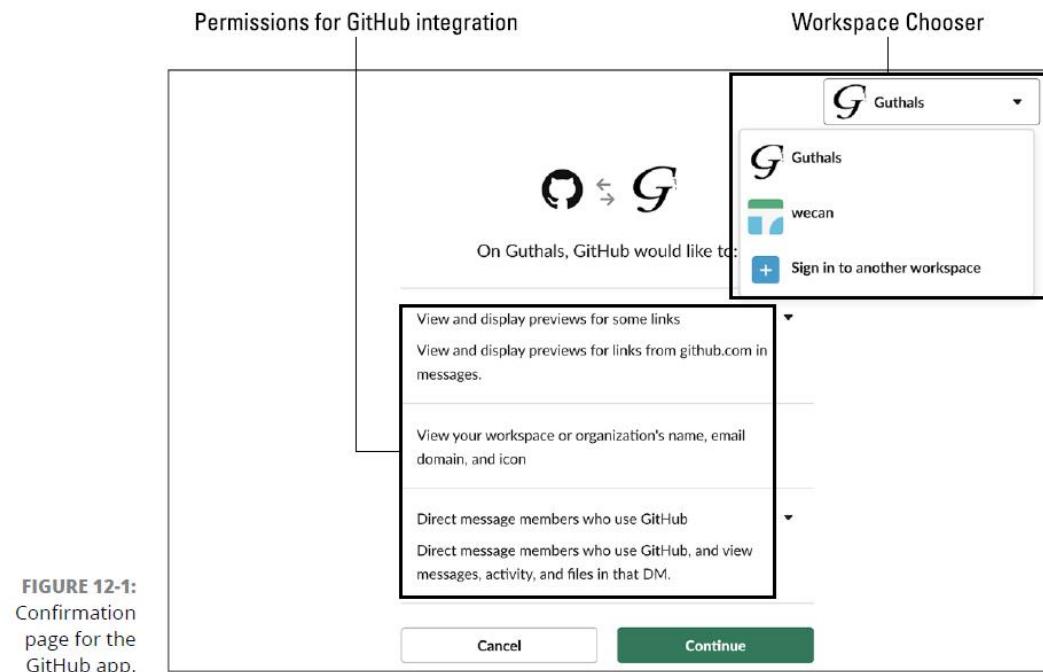


图 12-1 GitHub 应用的确认页面

(2) 单击 Continue 按钮，进入 GitHub 应用配置界面。如图 12-2 所示，该屏幕提示选择该应用适用的频道。

(3) 选择所有公共频道。Slack 工作区的任何公共频道都可以选择利用这个应用，订阅 GitHub 仓库的通知。

(4) 单击 Install 按钮，完成在 Slack 上的集成安装，然后返回到 Slack。

提示：如果安装了 Slack 桌面应用程序，单击 Install 按钮会试图启动该应用程序，并引导用户到安装该应用的工作区。如果用户还没有将工作区添加到桌面应用程序中，将停留在最近使用的工作区中。虽然可能有人会因看起来像没有安装成功而感到困惑，但别担心，事实上已经完成了在 Slack 上的集成安装。只需要把工作区添加到桌面应用程序中，然后继续。

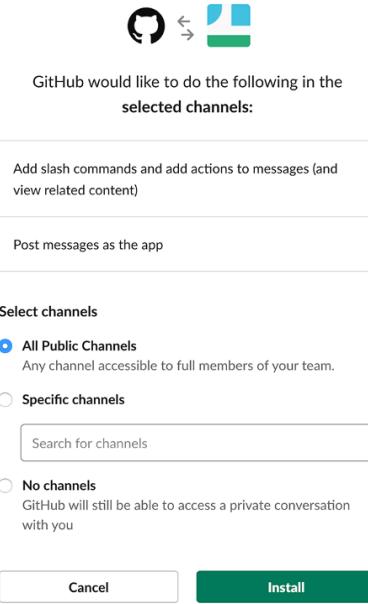


图 12-2 选择 GitHub 应用程序在哪些频道启用

12.1.2 在 Slack 频道中订阅一个仓库

在 Slack 上安装好 GitHub 应用程序后，可以在 Slack 频道中订阅 GitHub 仓库的通知。

(1) 键入以下命令：

```
/github subscribe owner/repository
```

例如，要订阅我们为本书读者创建的仓库（见链接 <https://github.com/thewecanzone/GitHubForDummiesReaders>），可以在 Slack 频道中输入以下命令：

```
/github subscribe TheWeCanZone/GitHubForDummiesReaders
```

第一次运行这个命令时，Slack 会提示将 Slack 账户连接到 GitHub 账户，如图 12-3 所示。这个 Slackbot 消息只有用户自己才能看到。

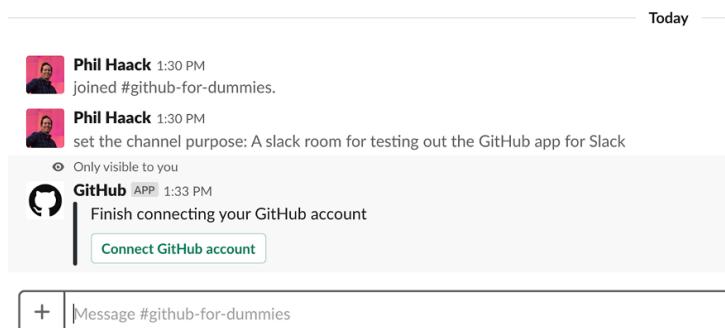


图 12-3 Slack 消息中出现了连接 GitHub 账户的提示

(2) 单击 Connect GitHub Account 按钮，开始连接账户。

如果用户已经在 GitHub.com 上认证并授权了 Slack，无论是之前的步骤还是本章前面“12.2.1 为 Slack 安装 GitHub 应用程序”一节中的步骤，浏览器都会自动认证并重定向到 Slack。

如果用户没有在 GitHub.com 上进行认证，GitHub 会提示先进行认证，并完成本节的其余步骤。认证完成后，GitHub 会提示授权 Slack 应用，如图 12-4 所示。在将 GitHub 安装到其他 Slack 工作区时，用户不需要再做这个操作。

(3) 单击 Authorize Slack by GitHub 继续。授权后，GitHub 会提示选择一个组织或账户来安装该应用。

(4) 单击想订阅的仓库所对应的账户名称。

在这个例子中，我们要安装到 TheWeCanZone.org 中的一个仓库，所以要单击这个选项。单击账户将进入下一步，指定应用程序可以访问哪些软件库，如图 12-5 所示。

(5) 选择“All repositories”，为所有当前和未来的仓库启用应用程序，或者单击“Only select repositories”，选择可能使用该应用程序的特定仓库。

(6) 选择好仓库后，单击 Install 按钮，完成在 GitHub 上的安装。

Slack 现在被列在账户或组织设置中的“应用程序”页面中。用户可以在该页面中更改应用的设置或卸载。

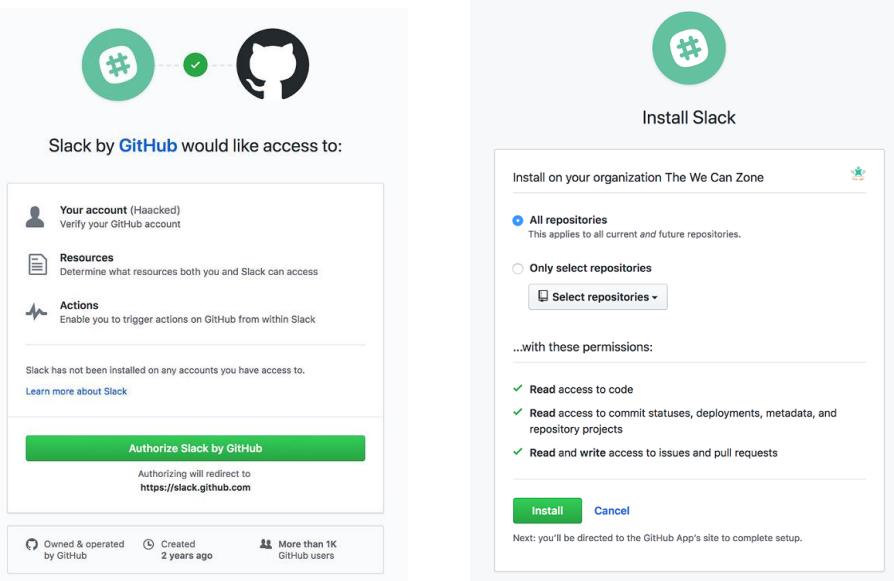


图 12-4 授权给 GitHub 的 Slack 应用

图 12-5 配置 GitHub 上的 Slack 应用

12.1.3 使用 GitHub Slack 集成

安装完成后，就可以在 Slack 频道中订阅 GitHub 仓库了。要查看 Slack 命令的完整列表，请输入以下命令。

```
/github help
```

该命令的输出如图 12-6 所示。

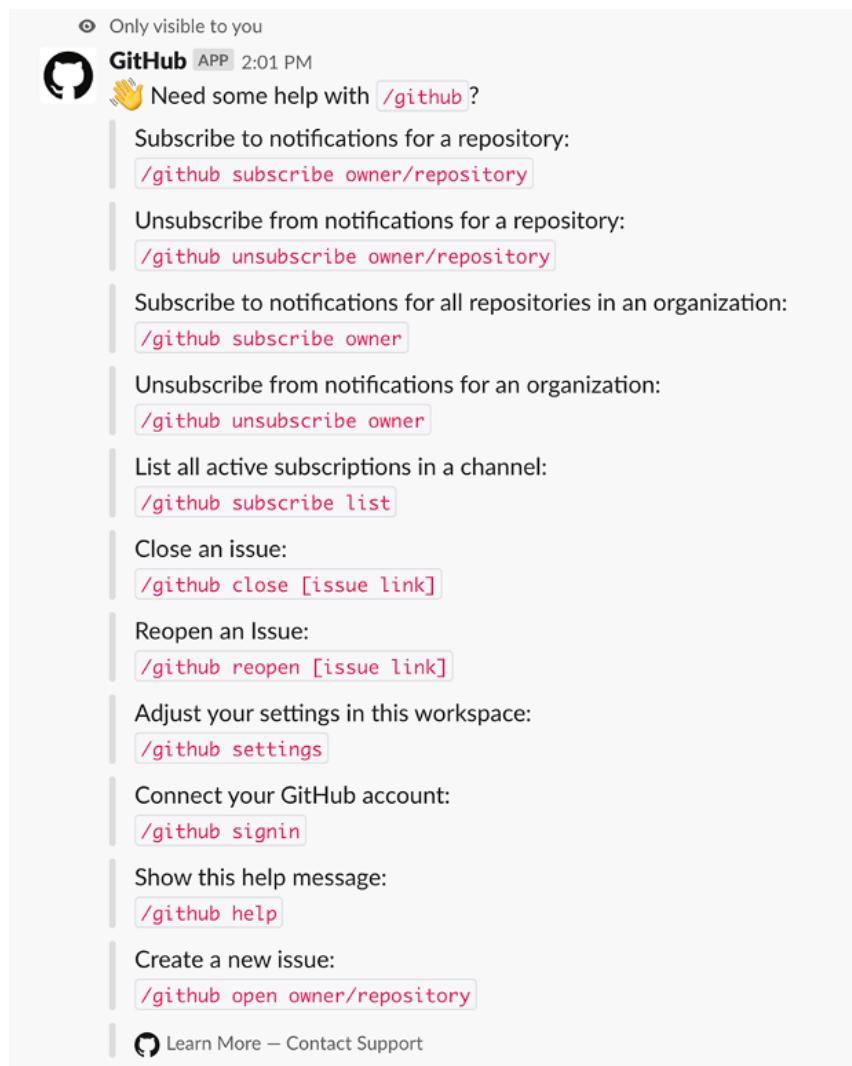


图 12-6 Slack 上可用的 GitHub 命令摘要

下面以打开一个 GitHub Issue 为例进行介绍。

(1) 运行以下命令。

```
/github open TheWeCanZone/GitHubForDummiesReaders
```

弹出一个 Slack 对话框。用户可以使用这个对话框来创建一个新 Issue，如图 12-7 所示。

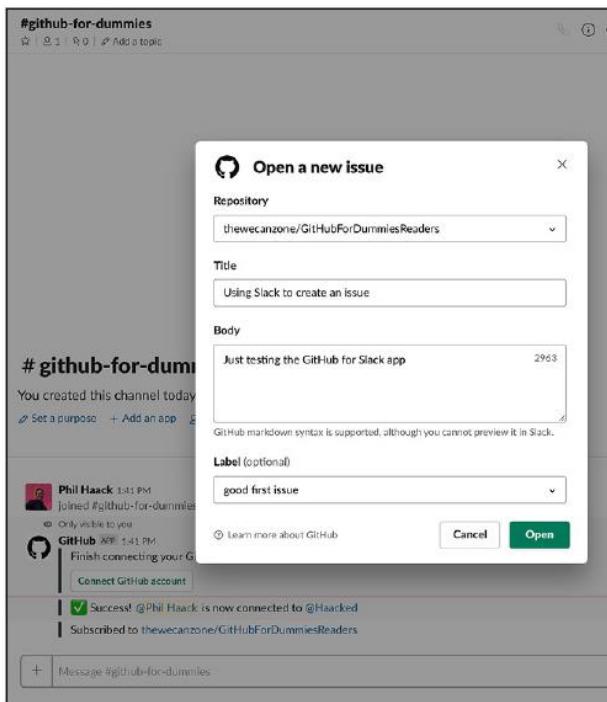


图 12-7 在 GitHub 上创建 Issue 的 Slack 对话框

(2) 在对话框中填入信息，并单击 Open 按钮。单击 Open 按钮，在 GitHub 上创建 Issue。因为之前订阅了该仓库，所以会在 Slack 的频道中收到 Issue 被创建的消息，如图 12-8 所示。

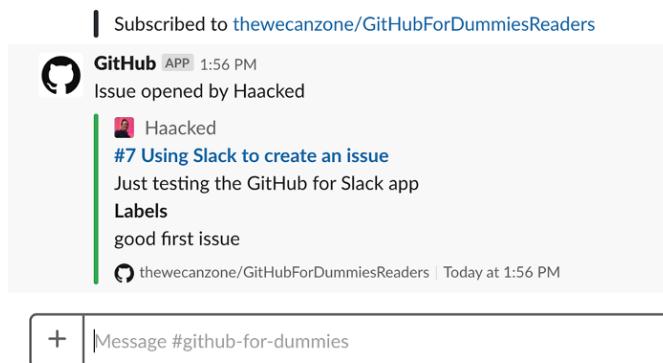


图 12-8 Slack 消息中关于新创建的 GitHub Issue 的信息

提示：鉴于 GitHub 是开源的，如果用户在使用 GitHub 集成时发现了任何 bug，或者在使用过程中有更好的想法，可以单击链接 <https://github.com/integrations/slack>，提出 Issue 或者其他建议和意见。

/github subscribe 命令默认为一个频道订阅了以下关于仓库功能的通知。

(1) Issues：已开放或关闭的 Issue。

- (2) pulls: 新的或合并的 PR (pull requests, 拉请求)。
- (3) statuses: PR 的状态。
- (4) commits: 位于默认分支上的新 commit (通常是 master 分支)。
- (5) deployments: 更新部署状况。
- (6) public: 从私有转为公共的仓库。
- (7) releases: 已发布的版本。

如果只需要订阅一个仓库功能, 请使用 /github subscribe owner/repo [feature] 命令。

```
/github subscribe TheWeCanZone/GitHubForDummiesReaders reviews
```

可以通过使用 /github unsubscribe owner/repo [feature] 命令来取消某个仓库功能的订阅。例如以下命令取消了默认分支的 commit 通知:

```
/github unsubscribe TheWeCanZone/GitHubForDummiesReaders commits
```

一些额外功能默认是禁用的, 如下所示:

- (1) reviews: PR 审查。
- (2) comments: 对 Issue 和 PR 的新评论。
- (3) branches: 创建或删除分支。
- (4) commits:all: 所有被推送到任何分支的 commit 内容。

12.2 Trello 与 GitHub 集成

Trello 是一个协作工具, 使用看板、列表、卡片等形式组织项目。Trello 的灵感来自于丰田公司推广的看板计划系统。看板 (Kanban) 在日语中是告示牌的意思, 其理念是有一个板子, 提供整个项目的状态和进度, 一目了然。

通常, 像 Trello 这样的工具会与 GitHub 结合起来管理项目。项目组可以用 Trello 来管理整个项目进展, 用 GitHub 来托管代码, 并将特定的代码问题交付给开发者。Trello 中的一个卡片可能对应着多个 GitHub Issue。

提示: GitHub 项目看板 (GitHub Project Boards) 本质上是集成了 GitHub 的 Trello。然而, 这个内置的看板并不总是适合每个人的项目团队。有些团队中的非开发人员不想学习 GitHub, 并且他们可能已经在使用 Trello。

通常情况下, 最好把所有的项目放在一个地方管理, 所以如果在 GitHub 之外, 例如在 Trello 中, 可以通过这种集成使其成为开发者工作流程的一部分。

Trello 的 GitHub 集成（在 Trello 中称为 power-up）将卡片与 GitHub 的 Issue、PR 和分支连接起来。在下一节中，我们将介绍如何设置 GitHub power-up。

12.2.1 安装 GitHub power-up

下面的安装说明假定用户已经完成注册（见链接 12-5），并创建了一个项目看板。

提示：如果用户从未使用过 Trello，可以访问 Trello 的指南（见链接 <https://trello.com/en-US/guide>）。Trello 类似于 GitHub 项目看板（见第 3 章）。关于如何创建看板和卡片的具体帮助信息，请访问链接 <https://trello.com/guide/create-a-board.html>。

(1) 打开一个看板，并确保菜单是打开的。如果菜单没有打开，请单击右上方以显示菜单。

(2) 单击菜单中的 Power-Ups 按钮，如图 12-9 所示。当单击 Power-Ups 按钮时，会出现一个搜索对话框。

(3) 搜索 GitHub，找到与 GitHub 相关的 power-ups。

(4) 单击添加按钮以启用 GitHub power-up。

提示：Trello 的 GitHub power-ups（扩展/集成）可能有些是由 GitHub 构建的，还可能有些是由其他人构建的。

由于 GitHub 的 API 是公开的，用户通常可以创建自己的 power-ups/扩展程序。在安装 GitHub power-up 时，请确保已知晓 power-ups/扩展程序的作者。用户很可能想从第三方开发者处安装，而不是从 GitHub 官方安装，因为两者提供的功能可能不同。无论从哪个渠道安装，用户应该明确自己的最终选择。不要认为标题中带有“GitHub”的东西都是由 GitHub 公司制造的。

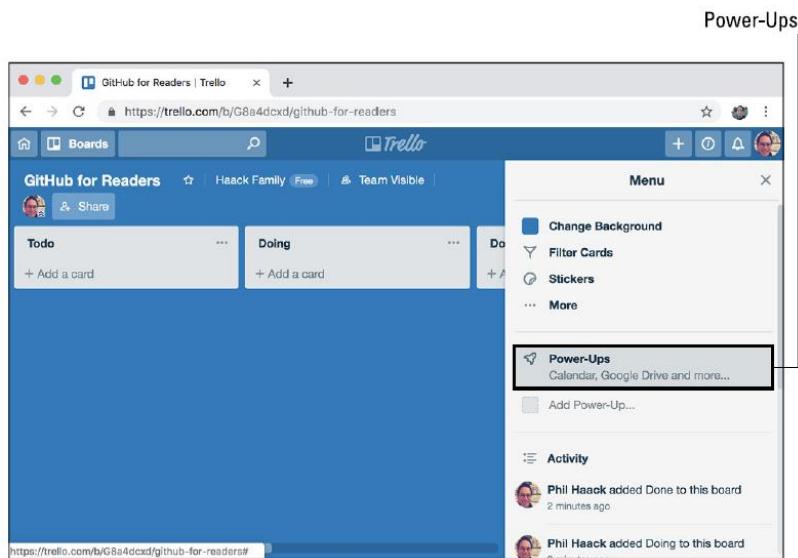


图 12-9 菜单中的 Power-Ups 部分

(5) 启用 GitHub power-up 后，单击齿轮图标进行配置。用户会看到一个菜单，可以选择禁用 power-up，或给其授权。

(6) 单击“Authorize Account”，会出现一个链接到 GitHub 账户的页面。

(7) 单击“Link Your GitHub Account”。GitHub.com 会在浏览器中启动，并提示为 Trello 授权，如图 12-10 所示。在单击“Authorize trello”按钮之前，一定要单击某个组织名称后的 Grant 按钮，确保为该组织授予 Trello 管理权限。

在本例中，将授予 Trello 对 thewecanzone 组织的访问权限，然后单击 Authorize trello 按钮，激活 power-up。

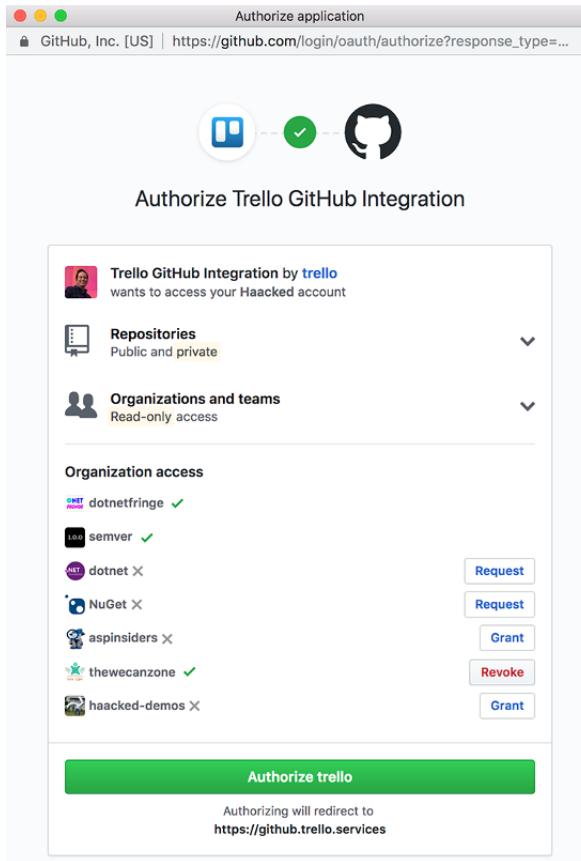


图 12-10 授权给 GitHub 上的 Trello

12.2.2 使用 GitHub power-up

通过 Trello 卡片可以访问 GitHub power-up。当然，需要首先创建好卡片。要在 Trello 看板上使用 GitHub power-up，请遵循以下步骤。

(1) 单击卡片以访问卡片的反面。

图 12-11 显示了用户创建的一个卡片。右下角显示了 GitHub power-up。



图 12-11 带有 GitHub power-up 的 Trello 卡片

(2) 单击 Power-Ups 下 GitHub 按钮。此时会出现四个菜单选项：① Attach Branch；② Attach Commit；③ Attach Issue；④ Attach Pull Request。

(3) 单击 Attach Issue 菜单选项，将弹出一个仓库搜索对话框。

(4) 找到包含要附加到卡片上的 Issue 的仓库。

在选择仓库后，会看到一个 Issue 列表，如图 12-12 所示。也可以搜索 Issue。从下拉列表中单击 Issue，就可以将 Issue 附加到 Trello 卡片上，该 Issue 会显示在 Trello 卡的反面。

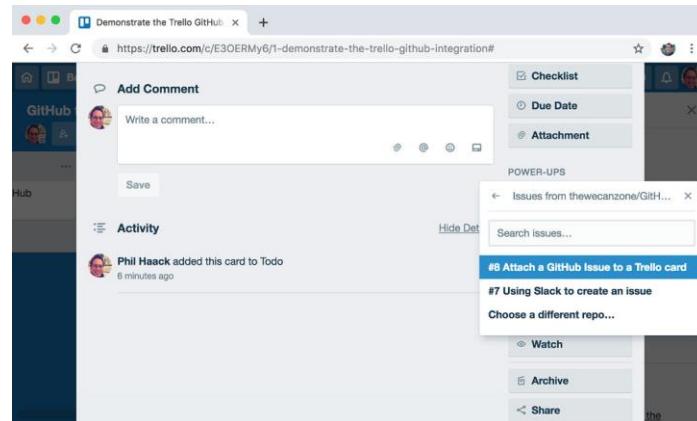


图 12-12 选择要附加的 Issue

一张 Trello 卡可以附加到多个 GitHub item。例如，重复前面的步骤，但选择“Attach Pull Request”而不是“Attach Issue”，将 PR 附加到一个 Issue 上。完成后，会看到一个 Issue 和一个 PR 都附在 Trello 卡上，如图 12-13 所示。

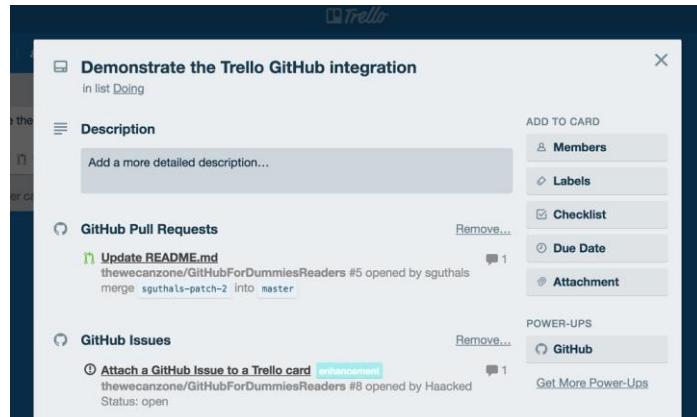


图 12-13 附有 Issue 和 PR 的 Trello 卡片

卡片上显示了几个图标，表明此卡是连接到 GitHub 的。卡片上显示了一个 Octocat 图标，上面有附在该卡上的 GitHub item 的数量。卡片上还显示了 PR 的图标，上面有一个计数，表示连接的 PR 的数量，如图 12-14 所示。

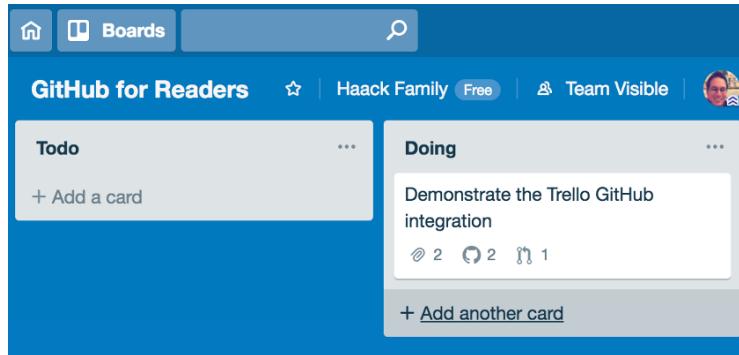


图 12-14 附有 Issue 和 PR 的卡片的正面

当访问 GitHub.com 上的 Issue 或 PR 时，可以看到这个链接是双向的。如图 12-15 所示，GitHub Issue 现在有一个指向 Trello 卡的链接。

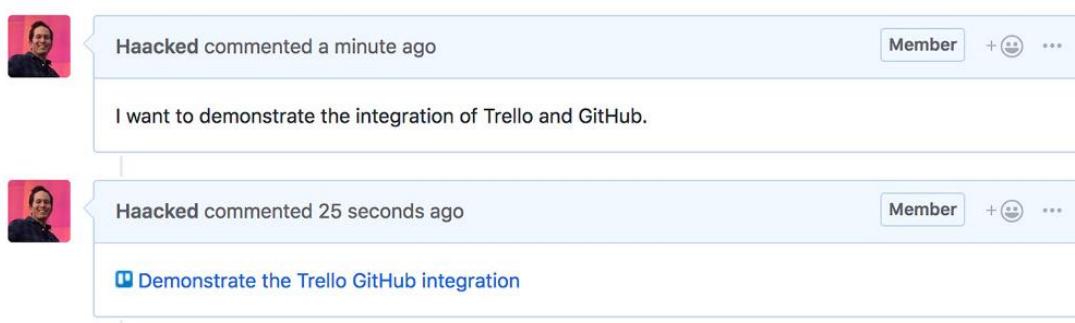


图 12-15 GitHub Issue 与 Trello 板的链接

12.3 使用 Octobox 管理通知

在本章的前面，我们介绍了几个将 GitHub 信息引入其他协作工具的集成。GitHub 的集成可以帮助团队一起工作。在本节中，我们将介绍一个有点不同的 GitHub 应用。

该应用是一个帮助个人管理 GitHub 通知流的工具。当用户参与越来越多的 GitHub 仓库时，通知的数量可能会开始变得令人难以承受。Octobox 为用户提供了一个电子邮件客户端风格的通知视图。安装 Octobox 是非常简单的。

(1) 进入 <链接 <https://octobox.io>>，向下滚动到安装 GitHub 应用程序的按钮。出现一些价格选项，如图 12-16 所示。Octobox 对开源项目是免费的。

(2) 单击“免费安装”以完成安装。

(3) 按照本章前面提到的授权 Slack 和 Trello 的方式，授权该应用程序。

安装和授权步骤完成后，会跳转到 Octobox 收件箱。第一次运行时，需要一些时间来同步用户的通知。当同步完成后，可以看到如图 12-17 所示的页面。

The screenshot shows the Octobox pricing page. It features two main sections: 'Open Collective' and 'GitHub Marketplace'. Both sections are labeled 'FREE FOR OSS' in the top right corner.

Open Collective:
Support the community
Pay by donation on Open Collective.
Private repository access
✓ Unlimited private repositories
✓ Unlimited public repositories
✓ Unlimited collaborators
from \$10 / user / month
Donate on Open Collective
Next: make your donation on Open Collective

GitHub Marketplace:
Support the company
Buy from the GitHub marketplace.
Private repository access
✓ Unlimited private repositories
✓ Unlimited public repositories
✓ Unlimited collaborators
from \$10 / user / month
Subscribe on GitHub
Next: confirm your purchase on GitHub

Wondering why there are two options? [Read more about our pricing strategy.](#)

图 12-16 GitHub 应用程序的下载按钮和 Octobox 的价格选项

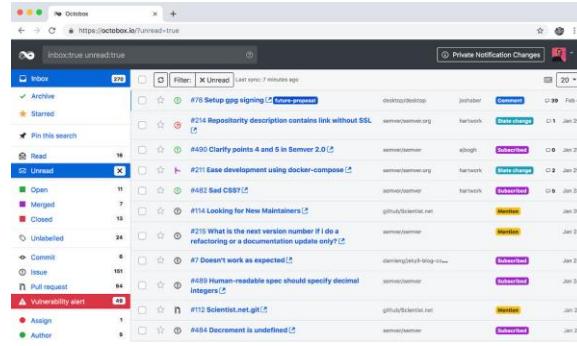


图 12-17 作者的 Octobox 收件箱

在 Octobox 安装和同步完成后，就可以管理用户的通知。Octobox 允许按仓库、组织、类型、行动、状态等搜索和过滤通知。用户可以在 Octobox 的设置页面中设置其自动同步的时间间隔。当 GitHub 上的 Issue 和 PR 的状态发生变化时，Octobox 上显示这些变化。

Octobox 还提供了通知存档和静音功能，这是处理通知的一个很好的方法，尤其是当用户在 GitHub 上面对很多仓库。

在本章中，我们了解到用户可以在 GitHub 以外的许多地方进行项目协作。例如，用户可以用 Slack 聊项目，用 Trello 来管理项目的任务。这两个产品的 GitHub 应用会将信息从 GitHub 同步到这些应用内。这对那些不在 GitHub 上的人来说很有用，因为这些应用可以提供重要的背景信息。这也有助于减少上下文切换，因为用户可能不需要在应用和 GitHub 之间不断来回切换。

最后一个应用，即 Octobox，有不同用途。这个工具填补了 GitHub 产品的空白，使 GitHub 的通知更易于管理。以上只是众多应用中的三个例子，这些应用使得在 GitHub 之外的环境中进行协作成为可能。

第 13 章 GitHub 工作流的集成

本章学习重点：

- (1) 探索 GitHub 的集成；
- (2) 使用 GitHub 集成；
- (3) 随时了解 GitHub 集成的情况。

上一章展示了一些在其他应用程序中获取 GitHub 仓库信息的方法。使用与 GitHub 集成的 Slack 和 Trello 等应用程序，可以帮助开发者进行项目管理。本章会介绍一些将 GitHub 集成到现有开发工作流程中的方法。

本章中展示的集成很多都是开源的，这意味着开发者可以跟踪新功能的开发、通过 GitHub 轻松报告错误、甚至为项目做贡献，这也意味着集成都在快速变化，所以在读者阅读本书时，一些细节可能已经过时了，但是重要的是知道如何获取更新和使用集成。

13.1 在 Atom 中使用 GitHub

Atom 是一个轻量级的开源编辑器，开发者可以直接在它的 GitHub 仓库上跟踪进度和报告 Issue。Atom 同时具有 GitHub 和 Git 集成，在与他人合作时可以帮助改善开发者的开发工作流程。要开始使用 Atom，请确保已经从 <https://atom.io> 下载了 Atom，然后使用下列方式之一打开“GitHub”标签页。

- (1) 选择 Packages ⇄ GitHub，然后选择切换“GitHub”标签页；
- (2) 单击 Atom 窗口右下方的“GitHub”按钮
- (3) 使用 Ctrl+Shift+8 组合键。

13.1.1 查看、签出和创建 PR

当第一次打开“GitHub”标签页时，读者看到的情况取决于在 Atom 中打开的仓库的状态。图 13-1 显示了在 Atom 中没有打开仓库的情况。屏幕空白并不意味着出错，这只是表示当前没有项目，所以“GitHub”标签页没有可操作的元素且没有反馈。

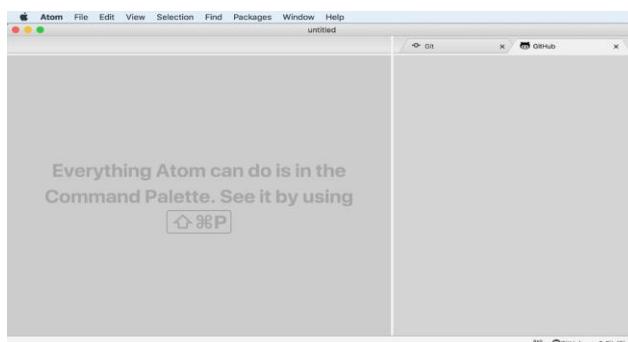


图 13-1 当 Atom 中没有打开项目时的 GitHub 标签页

如果在 Atom 中打开了一个没有与 GitHub 仓库相关联的项目，那么会看到与图 13-1 所示相同的空白标签页。如果该项目是本地项目（例如，用 git init 创建了仓库，或者它只是计算机上的一个文件夹，但未备份到任何类似于 GitHub 的托管平台），那么在 Atom 窗口的底部也有通知，表明该项目不存在远程仓库。

如果项目被托管在 GitHub 以外的地方，比如 GitLab，仍然会看到图 13-1 中的空白标签页，但 Atom 窗口的底部有获取、推送和拉取代码的按钮。这个按钮是 Git 集成的一个功能，并不是 GitHub 特有的。

图 13-2 显示了在 Atom 中打开 GitHub 仓库时看到的登录按钮。当单击登录按钮时，程序将要求转到 <http://github.atom.io/login>，生成认证令牌，并将其粘贴到 Atom 窗口中。

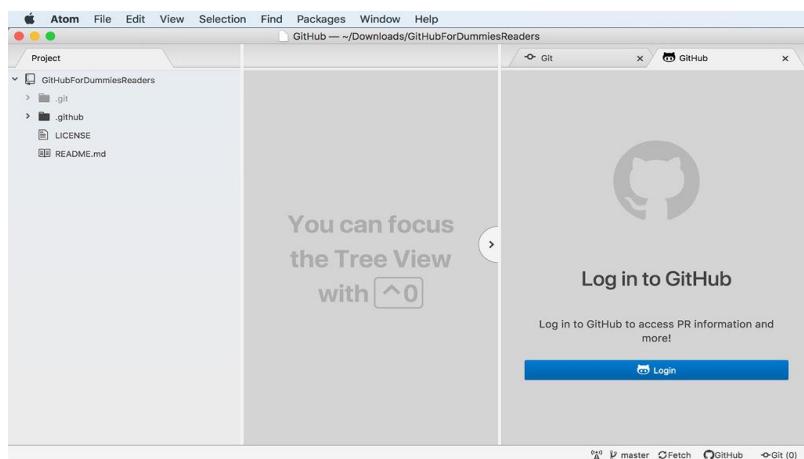


图 13-2 当 Atom 中的 GitHub 项目被打开，但未通过 GitHub 的认证时的 GitHub 标签页

当允许 Atom 的 GitHub 软件包访问个人账户时，会得到一个很长的授权令牌。将该令牌粘贴到 Atom 后，会看到该仓库所有开放的 PR 的列表，以及所在分支的信息（如图 13-3 所示）。可以通过单击“pull request”，然后单击蓝色的 checkout 按钮，来签出与 PR 相关的分支，如图 13-4 所示。

注意，在图 13-4 中，详细 PR 视图顶部的多个标签页允许用户检查构建状态，查看该分支上的提交列表，并查看与该 PR 的目标分支相比被修改的文件的差异。

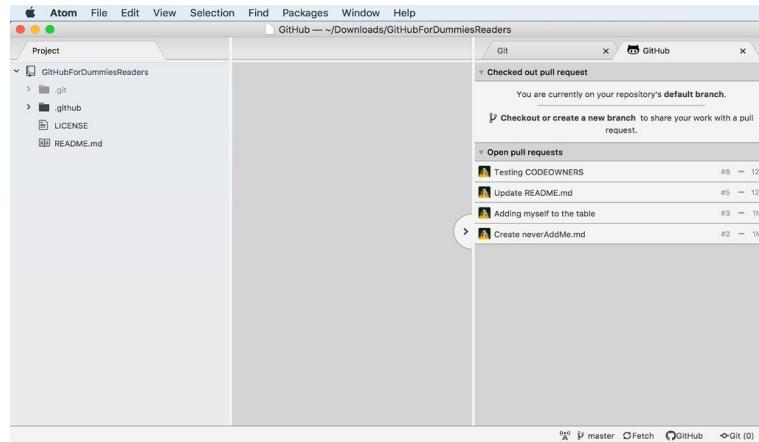


图 13-3 GitHub 标签页显示该仓库的所有打开的 PR

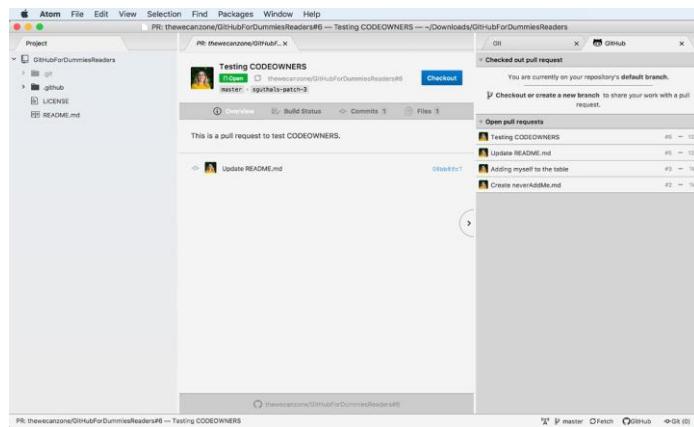


图 13-4 带有“Checkout”按钮的 PR 的详细视图

当在与 PR 关联的分支上时，该 PR 会显示在“GitHub”标签页的顶部，如图 13-5 所示。如果创建或签出了尚未与 PR 相关联的分支，“GitHub”标签页会在顶部的“Open pull request”列表上方增加“Publish”按钮。单击这个按钮可以发布分支，并自动打开 GitHub 网页，在那里可以为该分支创建一个 PR。这个按钮会取代图 13-5 中“Checked out pull requests”列表中的“Testing CODEOWNERS” PR 的位置。

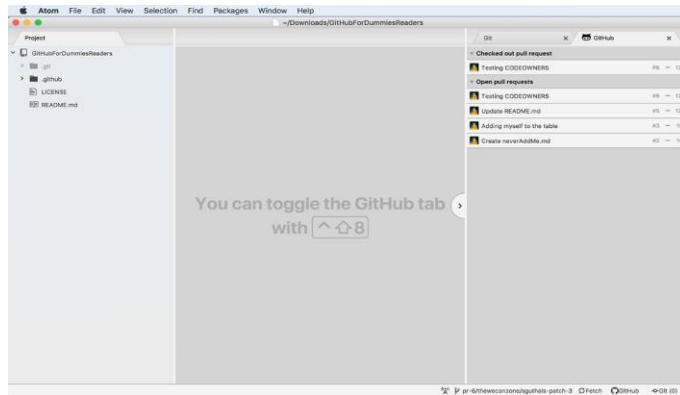


图 13-5 “GitHub” 标签页顶部显示的是当前的 PR

13.1.2 查看 Issues

在 Atom 中可以用便捷的指令面板的命令查看 Issue。首先单击 Packages ⇔ Command Palette 或按组合键 Ctrl+Shift+P 来打开指令面板，随后输入“GitHub: Open”，程序会自动补全“GitHub: Open Issue or Pull Request”选项，如图 13-6 所示，选择该选项。粘贴任何 Issue 的 URL（这也适用于 PR），然后单击打开 Issue 或 PR 按钮，Issue 的详细信息就会在 Atom 中打开，并带有所有评论，就像在 GitHub 网站上一样（如图 13-7 所示）。

除了阅读 Issue，用户不能与之交互，但如果在处理一些代码时需要参考某个 Issue，在 Atom 中查看 Issue 仍然很有用。有了打开 Issue 的功能，用户不必同时打开浏览器窗口，也不必在浏览器和 Atom 之间不断切换。

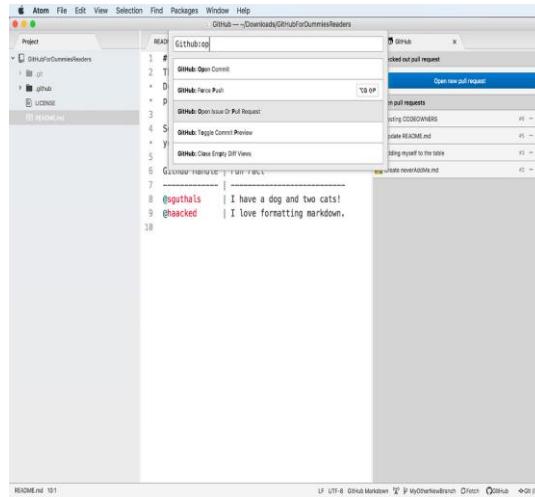


图 13-6 使用指令面板的命令查看 Issue

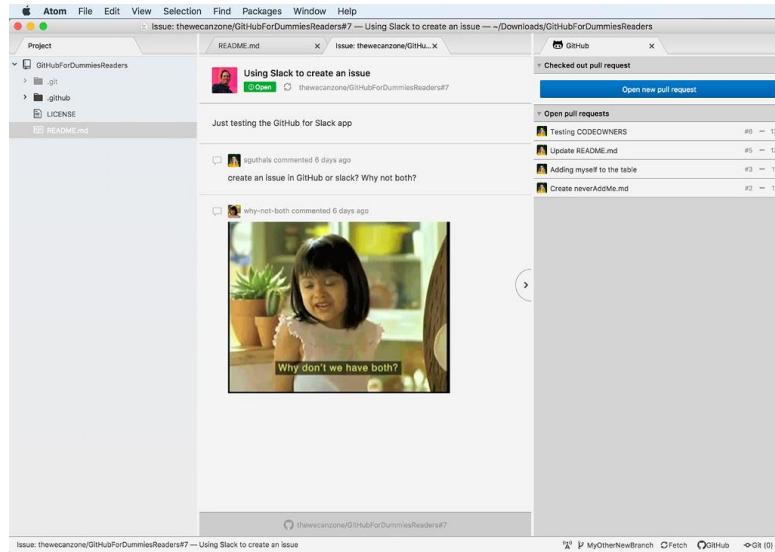


图 13-7 在 Atom 中打开的 GitHub Issue 详情

13.1.3 关注 Atom 的 GitHub 软件包

Atom 的 GitHub 软件包有大量的文档，读者可以在 Atom 手册（Atom Flight Manual）中找到，网址是 [https://flight-manual.atom.io/using-atom/ sections/github-package](https://flight-manual.atom.io/using-atom/sections/github-package)。在这里读者可以找到所有 Git 和 GitHub 功能的列表，并有屏幕截图和深入的描述；也可以在 GitHub 仓库上跟踪新功能或未来功能的进展，网址是 <https://github.com/atom/github>。Atom 的 GitHub 包团队经常使用项目看板、Issue、PR 和 RFC（征求意见稿）来计划、设计和实现新功能。例如，在链接 <https://github.com/atom/github/blob/master/docs/feature-requests/003-pull-request-review.md>。可以找到 PR 审查功能的 RFC。

13.2 在 Visual Studio Code 中使用 GitHub

Visual Studio Code（简称 VS Code）是目前发展最快的编辑器之一。像 Atom 一样，VS Code 是开源的，这意味着读者可以在 <https://github.com/microsoft/vscode> 看到编辑器新功能的发展。2018 年，微软和 GitHub 合作建立了一个开源的 GitHub 编辑器扩展，提供编辑器内的 PR 体验。

安装 VS Code 后，前往扩展市场，搜索“GitHub”，然后单击“GitHub Pull Requests”扩展的绿色 Install 按钮来安装扩展，如图 13-8 所示。

注意：安装扩展后，必须重新加载 VS Code 窗口。重新加载只需要几秒钟，已打开的项目不会受到影响。（现在，需要重新加载窗口的插件，会在安装结束的时候自动在附近显示按钮）

安装完扩展后，就可以登录到 GitHub 了。可以通过三种方式登录 GitHub（见图 13-9）。

- (1) 单击 VS Code 窗口左下方的 Octocat 按钮。

(2) 单击 VS Code 窗口右下方的通知。

(3) 使用指令面板, 按组合键 **Ctrl+Shift+P**, 输入 “GitHub” 并找到登录相关指令。

登录过程会打开网页浏览器, 请授权 VS Code 访问 GitHub 仓库。单击提示后浏览器会重定向到 VS Code。回到 VS Code 后, 系统会弹出提示询问是否信任该 URL, 请单击信任按钮。

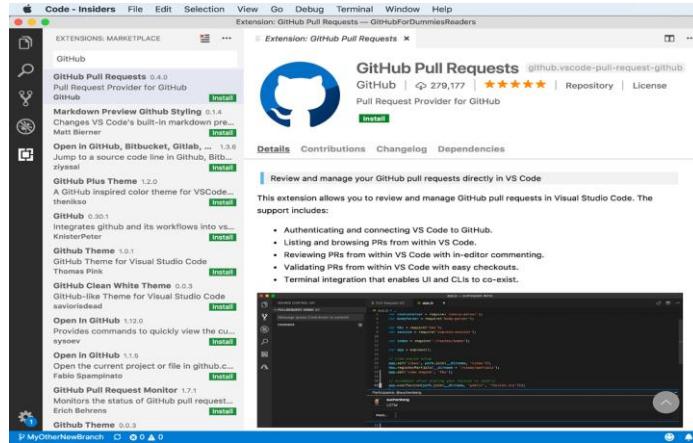


图 13-8 扩展市场中的 VS Code 的 GitHub 扩展

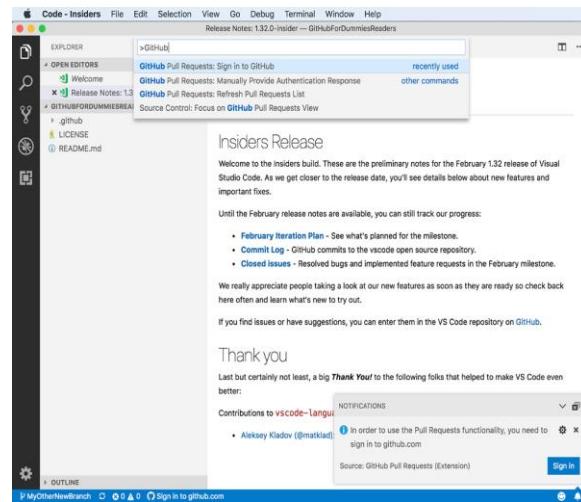


图 13-9 开始 GitHub 登录过程的三种方式

13.2.1 在 VS Code 中与 PR 交互

登录后, 进入 VS Code 左侧的“Source Control” 标签页会看到与该仓库相关的所有 PR。PR 被分为 5 个不同的部分:

- (1) 目前已经签出的 PR;
- (2) 被标记为审查者的 PR;
- (3) 被分配到的 PR;
- (4) 用户创建的 PR;

(5) 所有 PR。

当展开特定的 PR 时，会看到该 PR 的描述，以及所有更改文件。

单击 PR 的描述，会显示用户在 GitHub 上看到的描述，如图 13-10 所示。在这个页面，用户可以查看 PR，留下评论，合并 PR，更新 PR，或者审查 PR。开发者可以在 VS Code 中体验到整个 PR 的过程。

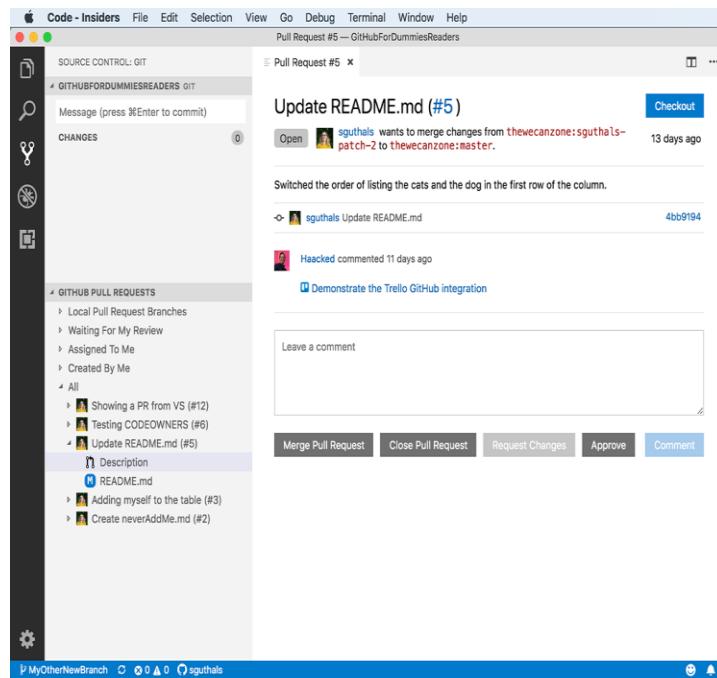


图 13-10 在 VS Code 中与 PR 交互

该扩展的另一个特点是在 diff 中添加行内评论。单击特定的修改过的文件会显示并排布局的 diff，就像在 GitHub 网页上看到的效果。用户可以给任何修改行添加评论，如图 13-11 所示。所有这些操作都同步回 GitHub。

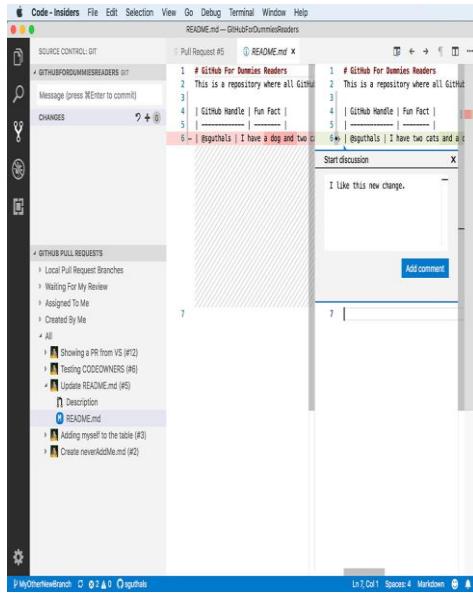


图 13-11 在 VS Code 中添加行内注释

13.2.2 关注 GitHub for VS Code 的 PR 扩展

因为此扩展是开源项目，读者可以在 GitHub.com 上关注它的发展，报告 Issue，甚至为它做出贡献。读者可在 <https://github.com/Microsoft/vscode-pull-request-github> 找到最新的功能和文档，了解如何使用这个扩展来改善开发工作流程。

从事这项扩展工作的 VS Code 和 GitHub 团队经常使用跟踪 Issue 来跟踪他们正在进行的工作。读者可以通过 GitHub 仓库上的“Issues”选项卡，并通过标签“iteration-plan”过滤 Issue，找到这个项目的最新迭代计划。

13.3 在 Unity 中使用 GitHub

Unity 是跨平台的游戏引擎，游戏开发者用它来构建场景并将代码脚本关联到游戏中。开发者可以在任何编辑器中编写代码，或者可以使用 MonoDevelop——与 Unity 伴随提供的编辑器。

提示：集成开发环境（IDE）如 Visual Studio，具有集成性，可为开发者编写 Unity 游戏代码提供更好的体验。在 Unity 中创建新项目后，可以将 GitHub 扩展添加到个人项目中。Unity 与其他编辑器不同，在 Unity 中，每个扩展都被表示为资产，是游戏的一部分而不是开发环境的一部分。所以，如果把 GitHub 扩展添加到 Unity 项目时，该扩展就是游戏的一部分。它们不会出现在成品游戏中，但会在协作构建游戏的过程中起到帮助作用。

首先，在 Unity 中选择 Window ⇄ Asset Store，打开资产商店。搜索 GitHub，会发现 GitHub for Unity 资产，如图 13-12 所示。

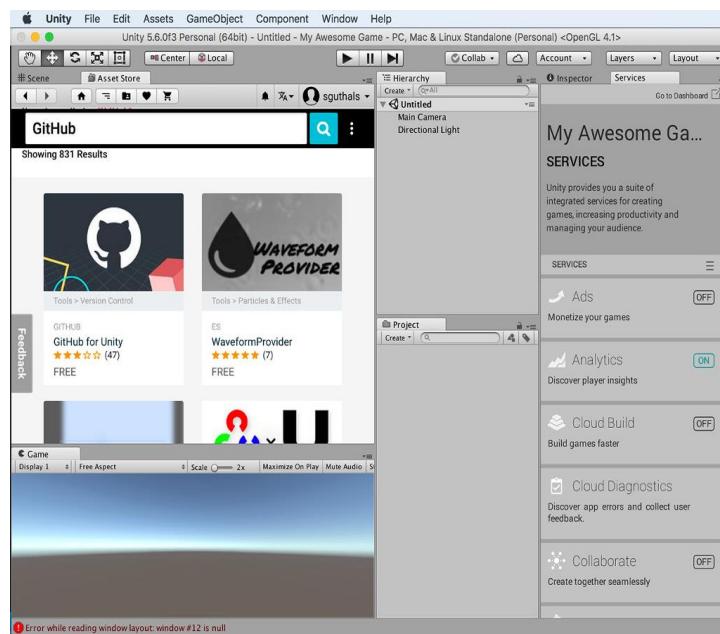


图 13-12 资产商店中的 GitHub for Unity

单击“GitHub for Unity”资产，向下滚动页面单击巨大的粉色 Import 按钮，此时会出现一个弹窗，所有内容将被添加到项目中，如图 13-13 所示。

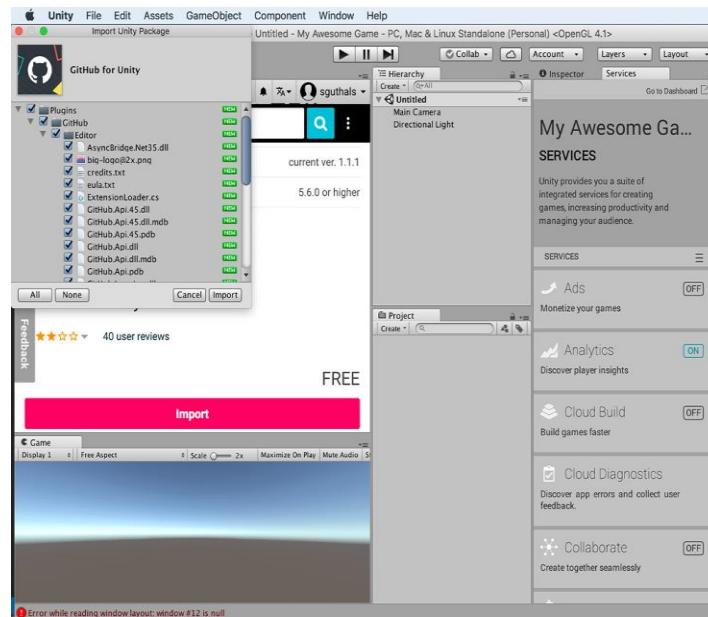


图 13-13 安装 GitHub for Unity 扩展

13.3.1 在 Unity 使用内置的 GitHub for Unity

安装完扩展后，“Window”菜单中可以看到名为“GitHub”的新菜单项。单击“GitHub”后，在 Unity 中会打开新标签页（见图 13-14 的背景）。单击新标签

页中间的为该项目初始化 git 仓库的按钮，GitHub 标签页就会更新，增加一些与 git 操作相关的标签页。

在 GitHub 标签页的右上方，可以单击“Sign in”按钮，登录到个人 GitHub 账户（如图 13-14 所示）。如果用户设置了双因素认证（two-factor authentication），系统会要求提供用户的 2fa 代码。

准备好后，用户可以单击 GitHub 标签页左上方的“Publish”按钮将项目发布到 GitHub。在图 13-15 所示的 Publish 对话框中，选择准备拥有这个项目的 GitHub 组织，输入仓库名称、添加描述，并选择项目公开或私有。

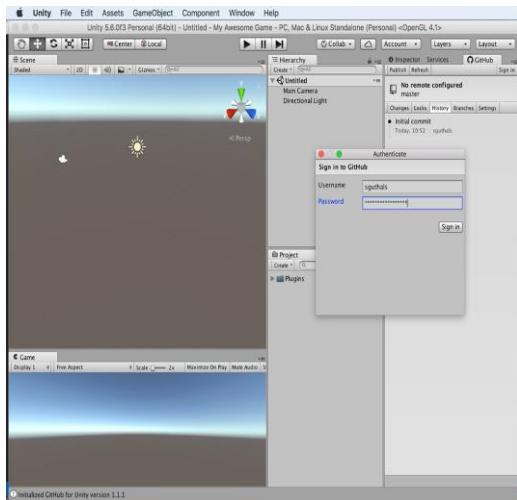


图 13-14 在 Unity 中登录 GitHub

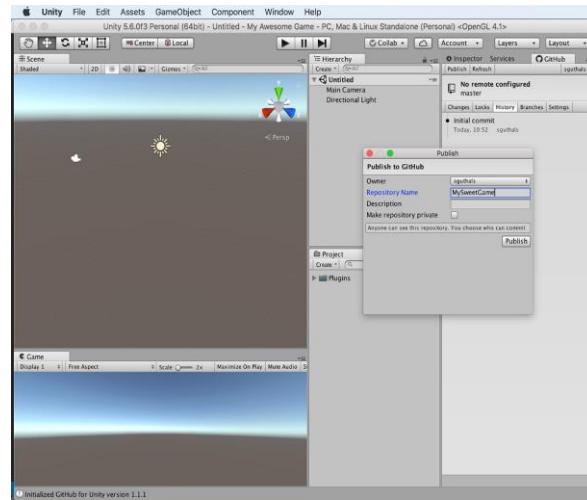


图 13-15 发布项目到 GitHub.com

项目发布后，用户可以像其他代码项目一样，继续进行项目工作。读者可能会发现以下几种有用的功能。

(1) **Changes**，其中列出了所有最近的更改，可以在提交中添加修改，然后将它们提交到用户所在的分支。

(2) **Locks**，用户可以锁定某些文件和资产，以便只有用户可以修改它们。如果正在对一个游戏场景进行修改，而不希望其他人也对该场景进行修改并覆盖当前用户的工作，那么这个锁就变得极为重要。

(3) **History**，显示这个项目所有提交的历史。

(4) **Branches**，在这里可以看到这个仓库的所有分支、创建新的分支、以及切换分支。

(5) **Settings**，在这里可以改变仓库的远程路径，改变扩展使用的 git 的路径，甚至更新 git 用户名或电子邮件，建议用户不要修改这些设置。

(6) **Fetch/Pull/Push/Refresh** 位于 GitHub 标签页的顶部，包含四个按钮，用于将 GitHub 仓库与本地项目同步。

(7) **GitHub Username**，在 GitHub 标签页的右上方，可以看到当前用户名，单击可进入 GitHub.com 个人资料页面，或者在 Unity 里面退出 GitHub。

13.3.2 关注 GitHub for Unity 扩展

GitHub for Unity 扩展是开源的，开发者可以跟踪开发、报告 Issue、甚至通过 GitHub 仓库（<https://github.com/github-for-unity/Unity>）对其进行贡献。始终保持扩展的更新也很重要，每当有新版本发布时，Unity 中就会出现弹窗，用户也可以在 Unity 资产商店中查看，或直接从 <https://unity.github.com> 下载最新版本。

这个扩展的工作团队规模很小，所以如果读者有兴趣为这个项目做贡献，我们强烈鼓励尝试！读者可以随时查看 GitHub 仓库里的 Issue，看看是否有标有“Help Wanted”或“Good First Issue”的 Issue。

13.4 在 Visual Studio 中使用 GitHub

Visual Studio 与 Visual Studio Code 不同。Visual Studio 是一个 IDE，并且是支持开发人员编写代码的全功能应用程序。VS Code 和 Atom 都是编辑器，有广泛的扩展列表，开发者可以将其添加到编辑器中以创造所需要的编辑体验。Visual Studio 有 Mac 版本和 Windows 版本。本节介绍的是 Windows 版本，因为它是原始的、功能更全面的版本，并且集成了 GitHub。

安装 Visual Studio 后，选择 Tools \leftrightarrow Extensions and Updates。弹出的窗口显示了当前安装的所有扩展，以及市场上的其他扩展。如果单击“Online”，出现在最上方的选项是“GitHub Extension for Visual Studio”扩展，如图 13-16 所示。

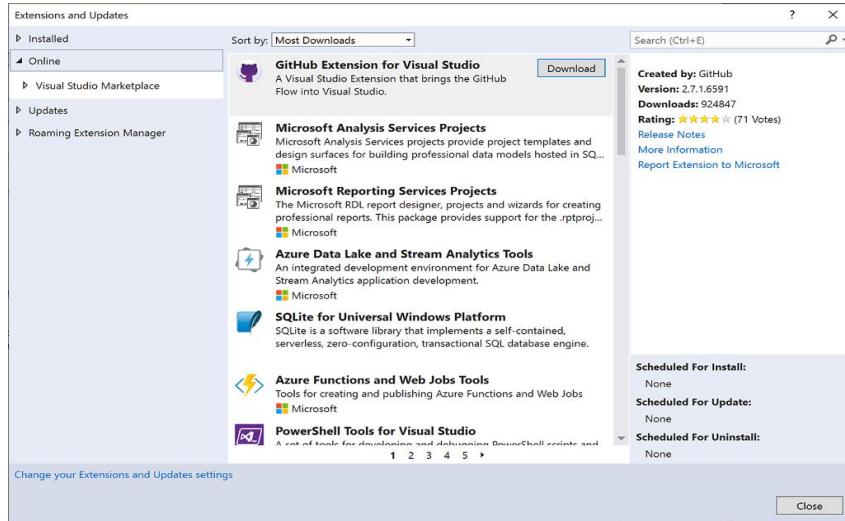


图 13-16 “Visual Studio Marketplace”中的“GitHub for Visual Studio”扩展。

单击下载按钮并关闭 Visual Studio。当 Visual Studio 关闭时，VSIX 安装程序启动并询问是否可以修改 Visual Studio。单击“yes”，然后单击“modify”，扩展会开始安装。安装完毕后，单击“Team Explorer”标签页中的 Connect 链接，连接到个人 GitHub 账户，如图 13-17 所示。当单击 Connect 链接时，程序会弹出窗口要求登录到 GitHub。如果用户设置了双因素认证，还会要求提供 2fa 代码。

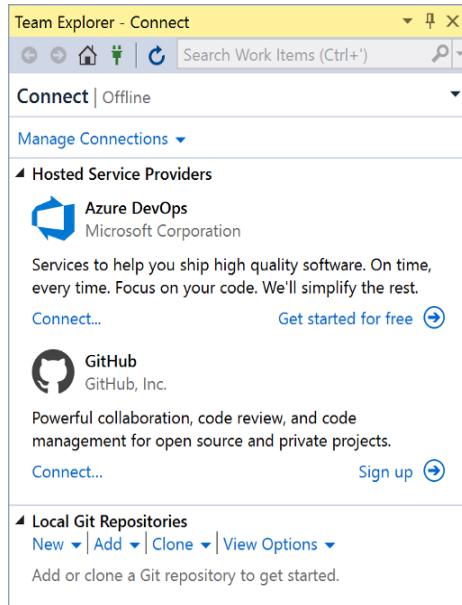


图 13-17 带有 GitHub 连接部分的“Team Explorer”窗格。

连接后，用户可以克隆仓库、创建新的仓库、或者从 GitHub 退出，都可以在团队资源管理器的连接页面上进行。

13.4.1 在 Visual Studio 中查看、创建和审查 PR

当用户在 Visual Studio 中打开连接到 GitHub 仓库的项目时，会在“Team Explorer”窗格的主页上看到额外的项目选项，如图 13-18 所示。

单击“Pull Requests”打开 GitHub 面板，其中列出了该仓库的所有 PR，如图 13-19 所示。在列表的顶部，用户可以选择查看所有打开的 PR、已关闭的 PR，或者只是所有 PR，还可以按作者排序。

双击一个 PR，会打开详细信息。在这里可以看到描述、目标和基础分支、当前状态、评论列表、以及更改的文件列表。还可以单击 GitHub 上的查看链接，打开浏览器窗口查看 github.com 上的该 PR。单击 Checkout <branch-name> 按钮，可以查看与此 PR 相关的分支。

单击添加评论链接，可以添加自己的评论，包括行内和整体评论。当添加评论时，可以把它标记为仅有的评论、批准它、或要求修改。如果双击其中被修改的文件，就会在编辑区打开 diff。如果把鼠标悬停在更改的行上，可以添加行内评论，与 VS Code 中的做法非常相似（参考图 13-11）。用户可以在图 13-20 中看到所有。

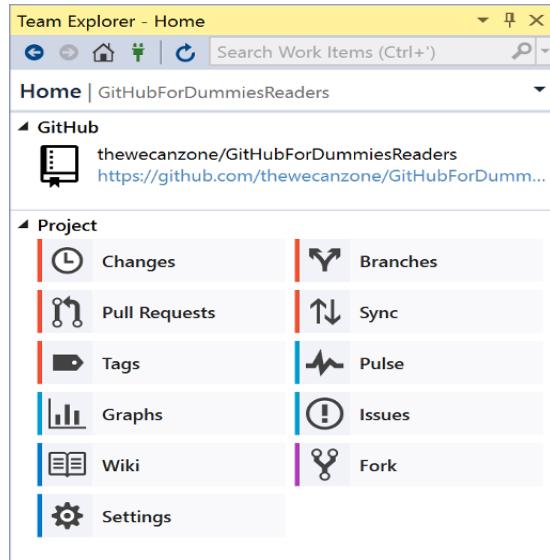


图 13-18 带有额外 GitHub 项目选项的团队资源管理器窗格

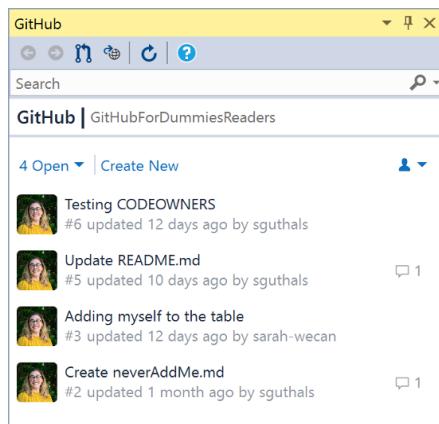


图 13-19 GitHub 窗格中的 PR 列表

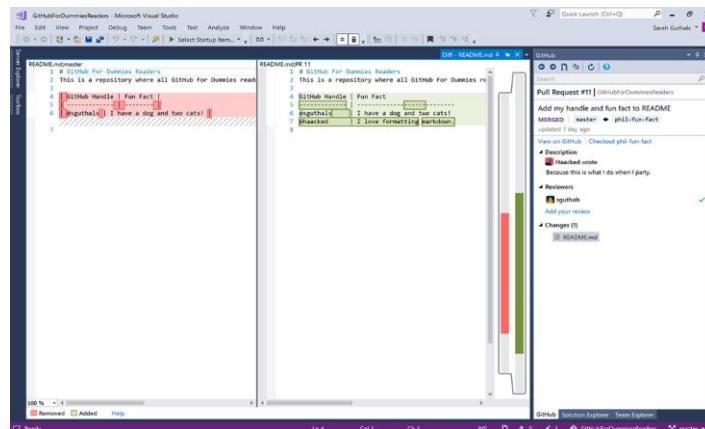


图 13-20 在 Visual Studio 中与 PR 进行交互

13.4.2 关注 GitHub for Visual Studio 扩展

GitHub for Visual Studio 扩展是开源的，读者可以在 GitHub.com 上关注它的进展，并提出 Issue，甚至可以为该项目做出贡献。读者可以登录网址 <https://github.com/github/visualstudio> 查找最新的功能和文档，了解如何使用 GitHub for Visual Studio 扩展来改善开发工作流程。

从事扩展工作的 GitHub 团队经常使用跟踪 Issue 来跟踪它正在进行的工作。读者可以通过 GitHub 仓库上的“Issues”选项卡，用“Next Release”标签过滤 Issue，找到项目的最新跟踪 Issue。

13.5 在 XCode 中使用 GitHub

Apple 已经在 XCode 中集成了 GitHub。安装 XCode 后，可以通过选择 XCode \Rightarrow Preferences 来打开账户标签页，登录 GitHub。单击左下角的加号，选择“GitHub”，然后单击继续，如图 13-21 所示。按照提示登录 GitHub 账户。如果设置了双因素认证，会要求提供 2fa 码。

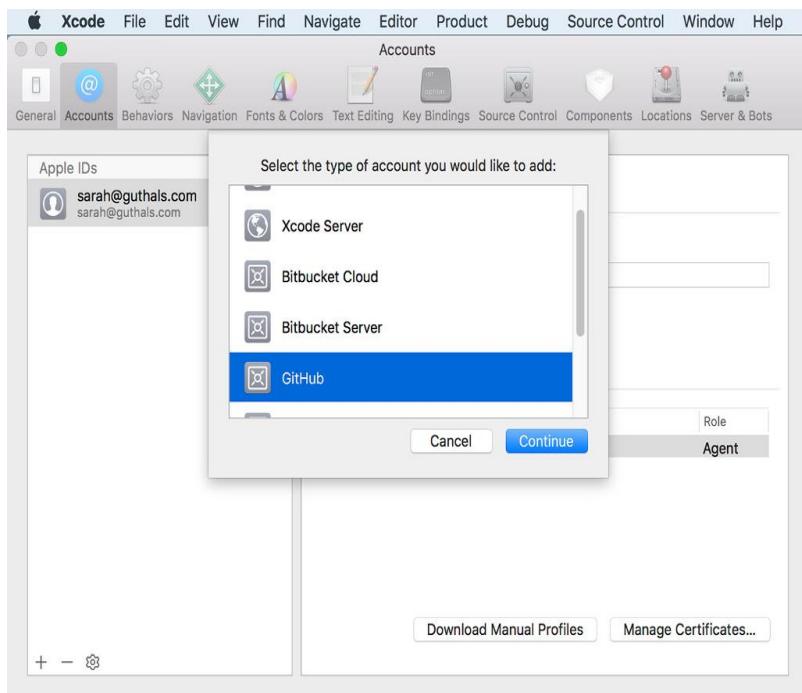


图 13-21 在 XCode 中登录 GitHub。

当用户选择 Source Control \Rightarrow Clone 时，GitHub 仓库会加载到窗口中，也可以输入 URL 进行克隆。当单击某个仓库时，用户会得到一些信息，例如该仓库所使用的主要语言，fork 和 star 的次数，以及 README 的快速链接，如图 13-22 所示。

如果用户在 XCode 中打开了文件，“Source Control”菜单会显示可以执行的其他操作，如“Commit”、“Push”、“Pull”以及“Fetch and Refresh Status”。

这个扩展不是开源的，所以用户最好跟踪最新的 Apple 开发者新闻以了解可能出现的新功能。

用户可以登录以下网址

https://developer.apple.com/library/archive/documentation/ToolsLanguages/Conceptual/Xcode_Overview/UsingSourceCodeControl.html

阅读 Apple 发布的所有源码控制集成。

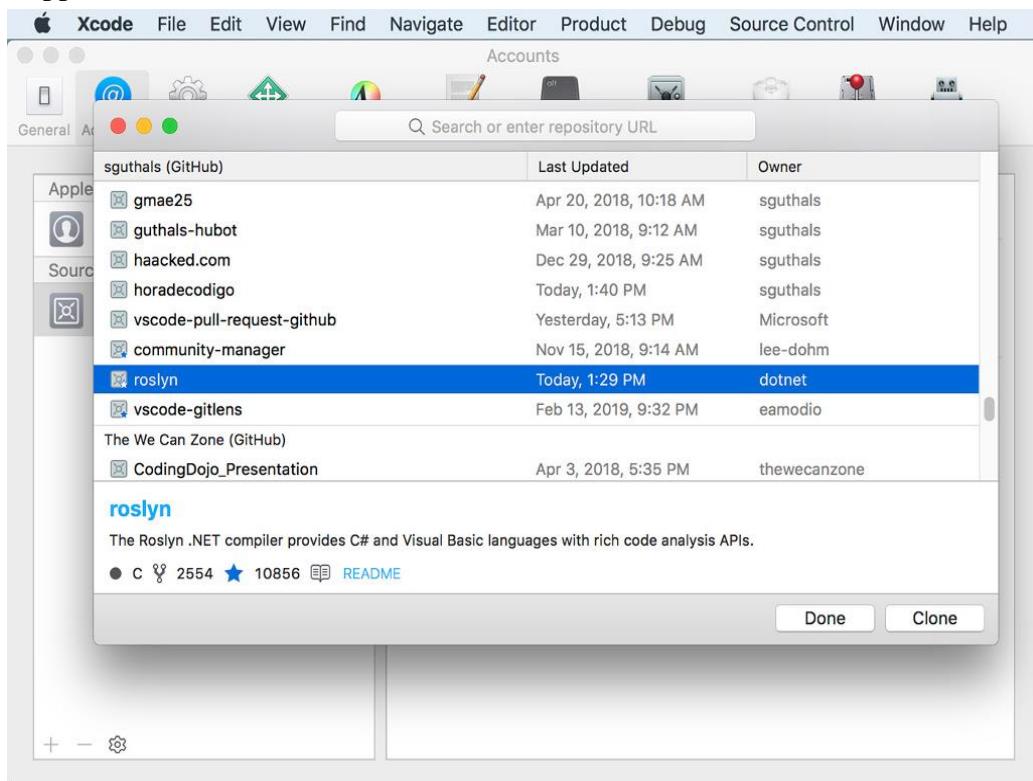


图 13-22 在 XCode 中查看你能克隆的 GitHub 仓库列表。

13.6 为 IntelliJ 使用 GitHub

JetBrains 的 IntelliJ IDE 对打开的任何 GitHub 仓库都有 GitHub pull request 支持。安装 IntelliJ 后，用户可以从开始菜单中选择克隆一个 git 项目，如图 13-23 所示。

在弹出的窗口中输入 URL，然后在窗口的底部，单击“Sign in to GitHub”按钮。系统要求提供 GitHub 凭证，如果设置了双因素认证，还会要求提供 2fa 码。成功登录后，所有能访问的 GitHub 仓库都会出现在下拉列表中，如图 13-24 所示。

在选择仓库并单击克隆后，会有一系列弹出窗口提示你创建 IntelliJ 项目。当项目在 IntelliJ 中打开时，可以通过选择 VCS \Rightarrow Git \Rightarrow View Pull Requests 来打开 GitHub PR 预览。在 IntelliJ 窗口中会打开新的栏目，其中有打开的 PR 列表。如果单击其中某个，就会打开描述和更改文件的列表。如果双击其中某个更改的文件，该文件的 diff 会在新窗口中打开（见图 13-25）。



图 13-23 从 git 项目克隆以开始将 GitHub 集成到 IntelliJ

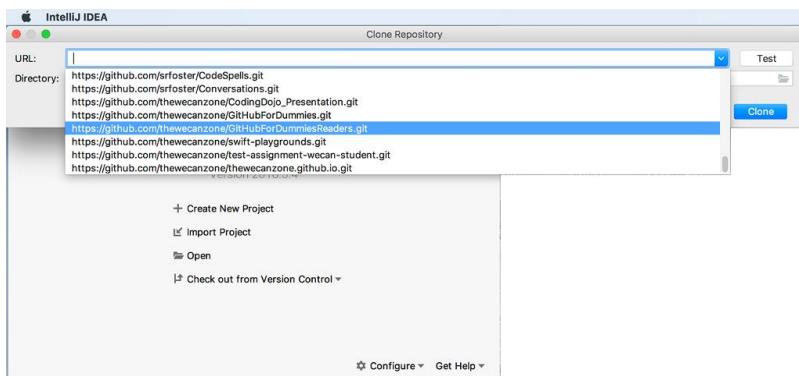


图 13-24 在 IntelliJ 中查看你能克隆的 GitHub 仓库列表

最后，用户也可以在 IntelliJ 中创建 PR。如果在一个新的分支上已经做了修改并提交到分支，可以选择 VSC \Leftrightarrow Git \Leftrightarrow Create Pull Request。在弹出的窗口中，（如图 13-26 所示）指定 PR 的基分支和目标分支、标题和描述。

该 GitHub PR 功能被嵌入到 IntelliJ IDE 中，建议读者关注 IntelliJ 博客和文档，以获得关于该功能发展的最新信息。

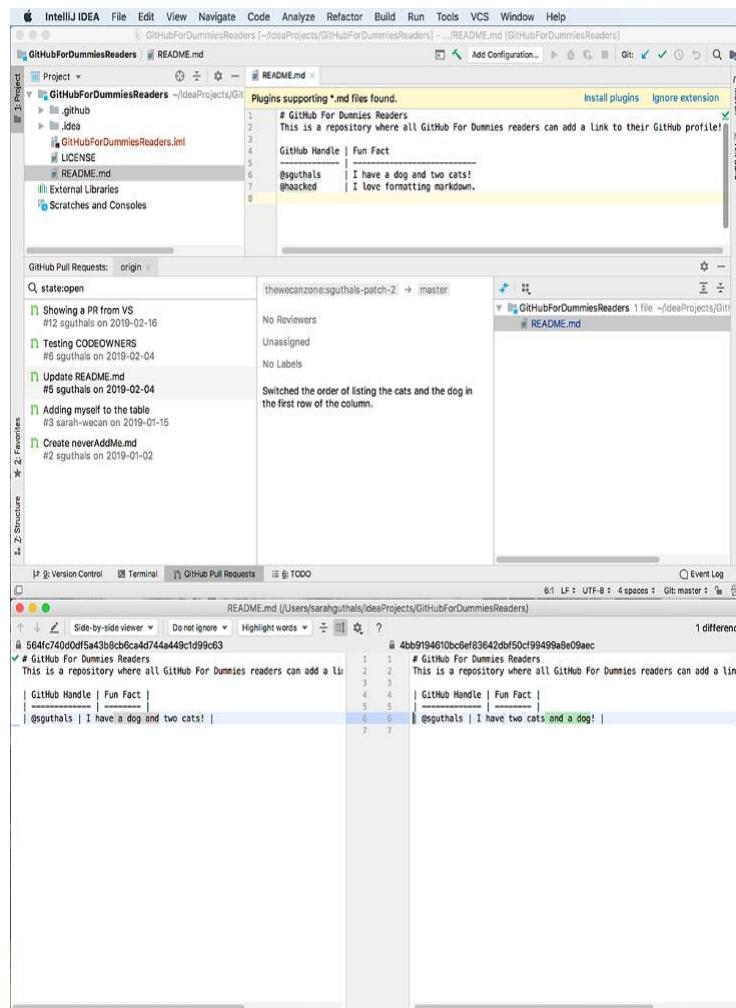


图 13-25 在 IntelliJ 中查看 PR

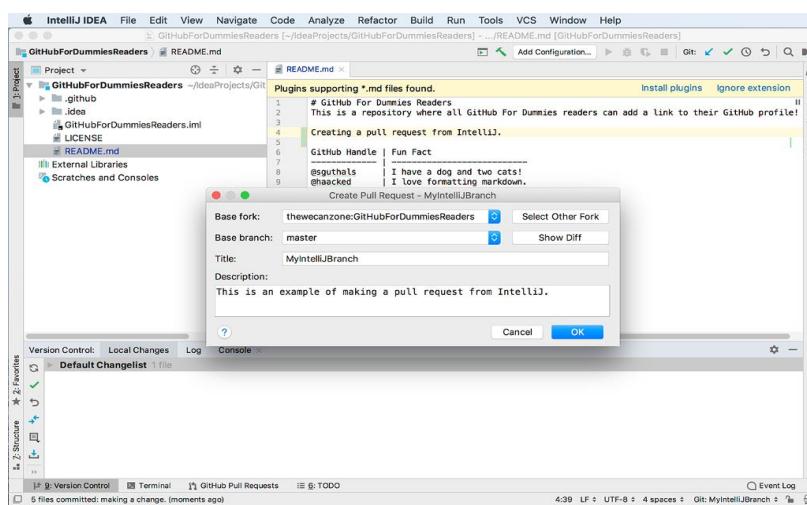


图 13-26 在 IntelliJ 中创建 PR

第 14 章 个性化 GitHub

本章学习重点：

- (1) 使用 Chrome 扩展程序来扩展 GitHub；
- (2) 设置 Probot 以实现 GitHub 的个性化；
- (3) 使用 GitHub Actions 来定制 GitHub。

GitHub 创建 Atom 文本编辑器的目标是建立最可侵入的文本编辑器。Atom 的口号是“一款为 21 世纪而打造的极客化文本编辑器”。Atom 专注于这个口号，因为它认识到开发者对自己的工具可能非常挑剔，开发者对自己的工作方式有很多想法，并投入大量的时间去个性化他们的工具，使工具按照他们想要的方式工作。

GitHub.com 本身也是如此，GitHub 提供了广泛的 API 接口，允许开发者可以编写工具，以各种方式与 GitHub 的数据互动。此外，由于 GitHub.com 在浏览器中运行，任何人都可以建立浏览器扩展来定制使用 GitHub 的体验。本章将介绍一些可用的方法来个性化地使用 GitHub。

14.1 使用浏览器扩展

浏览器扩展可以提升浏览器的使用体验，并且可以为各种可能的场景找到扩展。本节将介绍几个与 GitHub 配合使用的实用扩展，其中一些扩展在多种浏览器中使用，但为了简洁起见，本节主要介绍 Google 浏览器的扩展，如果需更全面的与 GitHub 兼容的浏览器扩展列表，请查看 Awesome browser extensions for GitHub 仓库（见 <https://github.com/stefanbuck/awesome-browser-extensions-for-github>）。

14.1.1 Refined GitHub 扩展

Refined GitHub 扩展是一个开源的扩展，它简化了 GitHub 界面，并为 GitHub.com 增加了一些有用的功能，该扩展可用于 Chrome、Firefox 和 Opera 浏览器。<https://github.com/refined-github/refined-github> 中有相关的源代码，README 文件中有安装扩展的链接。当安装该扩展时，将授予该扩展读取和改变在 api.github.com、gist.github.com 和 github.com 上的数据的权限，它也可以修改已经复制和粘贴的数据。

安装扩展后，地址栏的右边将出现 Octocat 图标，这个图标提供了一些简单的定制选项，其中一个选项可以实现在 GitHub.com 上自定义 CSS。图 14-1 中添加了一些自定义 CSS，实现将仓库名称变大并呈深红色。请注意，当改变 CSS 后，必须刷新页面才能看到更改的效果。

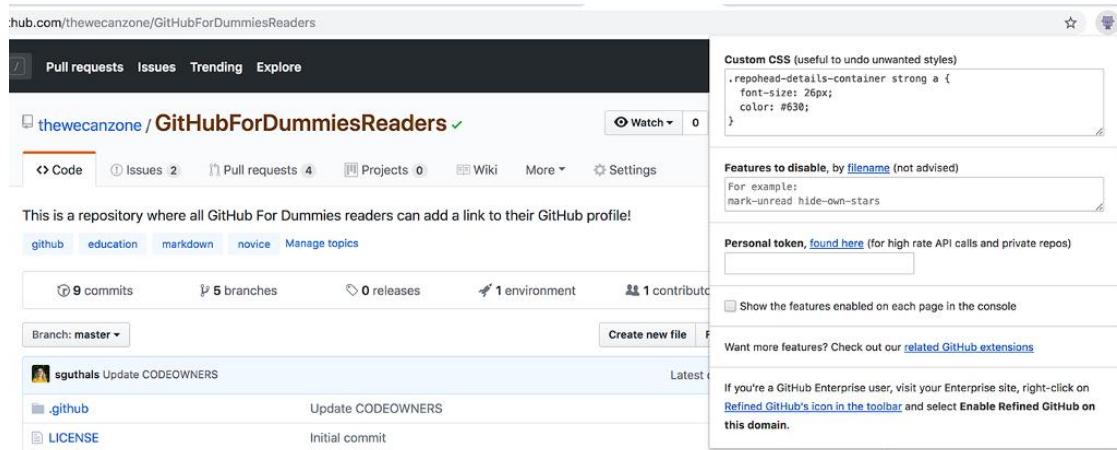


图 14-1 配置 Refined GitHub 扩展

与 Refined GitHub 带来的许多其他增强功能相比，改变 CSS 非常简单。

警示：浏览器扩展通过操纵网站生成的 HTML 来工作，它通常会在页面中寻找已知的标签（如具有已知 ID 的 HTML 元素），然后添加自己的用户界面元素、删除元素或改变元素。然而，它所做的这一切都在实际生成 HTML 的服务器上运行的代码之外。这意味着，如果像 GitHub.com 这样的网站改变了它们的 HTML 标记，某个扩展功能可能会暂时停止工作，直到更新它们的扩展以适应新的变化。因此，如果这些功能中的任何一个停止工作，请在扩展更新后再试。以下是部分增强功能，当安装了这个扩展时，这些功能是默认开启的。

(1) 将 Issue 和 PR 标记为未读。这项功能在 Issue 或 PR 的通知部分增加了标记为未读的按钮，单击该按钮可以将该 Issue 或 PR 标记为未读，从而将其放回通知列表中。

(2) 防止最近推送的分支导致的页面变化。通常情况下，当推送新的分支时，仓库的主页会在代码文件列表上方的黄条中显示最近推送的分支，这会导致页面的其他部分向下移动。Refined GitHub 将最近推送的分支列表显示在右上方，覆盖在 Fork 按钮可能出现的位置，它以叠加的方式显示分支，以确保页面不会变化。

(3) 增加了等待成功检查的选项。如果为仓库设置了持续集成 (CI)，在有人提交 PR 后，可能需要一段时间才能完成所有检查。CI 可能正在运行静态分析或 linter、单元测试、集成测试等。在审查了一些代码并准备合并 PR 后，还要等待所有这些过程的完成，这是比较费时的。Refined GitHub 增加了复选框，提示应该在检查完成并成功后才可以继续合并 PR。

(4) 展示参与互动的用户头像，显示参与了某一条评论的互动的具体用户。通常情况下，评论的互动只显示表情和该表情的数量，如点赞的人数。使用 Refined GitHub，便可以看到提供特定的表情的用户。

Refined GitHub 有许多改进的功能，上述列出的功能只是一小部分。例如，在图 14-1 中，可以禁用扩展的部分功能，也可以设置 GitHub token，以便扩展在私人仓库中的作用。

14.1.2 用 GitHub Selfie 拍摄照片或视频

GitHub Selfie 是一个开源的浏览器扩展（见 <https://github.com/thieman/github-selfies>），它在评论区添加了 Selfie 按钮，让用户使用计算机的摄像头自拍。用户可以选择拍静止的照片，也可以选择拍摄更有趣的 gif 动画。图 14-2 展示了作者使用 GitHub Selfie 拍了一张自拍照，这将有助于更好的协作。



图 14-2 用 GitHub Selfie 拍摄短视频

14.2 GitHub App 和 Probot

第 13 章介绍了一些将 GitHub 与其他应用程序连接的集成，如 Slack 和 Trello，这些集成的共同点是它们都是作为 GitHub 的应用实现的。GitHub 上的应用是扩展 GitHub 的另一种方式。GitHub App 是能够响应 GitHub 上的事件的网络应用，这些事件订阅被称为网络钩子。当 GitHub 上发生应用程序关注的事件时，GitHub 会向应用发出 HTTP 请求，提供该事件的相关信息，然后应用可以采用某种方式响应该事件，通常会通过 GitHub 的 API 回调到 GitHub。

本节将介绍构建一个简单的 GitHub App 的方法。当有人提出二选一的问题时，有人可能会用一张 gif 图来回应，这张 gif 图中有个小女孩问了一个问题：

“Why don’t we have both? ”。在本节中，我们会创建一个 GitHub App，并实现自动发送该图片。

14.2.1 Probot 简介

GitHub App 是需要监听 HTTP 请求的 Web 应用程序。在构建 HTTP 请求时，有很多重要的细节需要处理，如发布到应用中的数据的格式是什么？在刚开始构建

GitHub 应用时，所有这些细节都会让人感到困惑，而且要花时间才能得到正确的答案。

当开始使用 GitHub App 时，使用 GitHub 的 Probot 框架就很方便。Probot 处理了很多构建 GitHub 应用的模板和琐碎的细节。它是一个使用 Node.js 构建 GitHub 应用的框架，并且提供了许多方便的方法来监听 GitHub 事件和调用 GitHub API。Probot 使建立 GitHub App 变得容易，但它并没有解决在哪里托管应用程序的问题。

14.2.2 托管应用程序

GitHub App 可以有多种形式，它可以是在 Heroku 中运行的 Node.js 应用或者在 Azure 中运行的 ASP.NET Core 应用，也可以是在 Google Cloud 中运行的 Django 应用。它只要是持久的且可以通过公共互联网使用的应用即可。为了避免繁琐的配置，我们使用 Glitch 来快速实现一个 GitHub App。

14.2.3 Glitch 简介

Glitch（见 <https://glitch.com/>）是一个网络应用程序的托管平台，它消除了网络应用程序启动和运行的许多阻力。在 Glitch 中创建的任何应用从一开始就在网络上运行，不必考虑如何部署代码的问题，因为任何改变都会自动保存并自动部署。

Glitch 特别关注构建应用程序时的社区协作方式。每个文件都可以由多人实时编辑，就像在 Google Docs 中编辑文件一样。每个项目都可以通过单击 remix 复用。这鼓励了代码共享和相互学习，在建立自己的 GitHub 应用时非常有用。

如果你还没有账户，请确保已在 Glitch 上创建账户。

14.2.4 创建一个 Probot Glitch 应用程序

在对 Probot 有了大致的了解并创建了 Glitch 账户后，就可以在 Glitch 上创建 Probot 应用了。Glitch 允许复用已有的应用程序，并且提供了可复用的 Probot 应用。

要创建应用程序，首先需要在浏览器中输入 <https://glitch.com/edit/#!/remix/probot-hello-world>，这个命令在 Glitch 中基于 probot-helloworld 的例子创建了一个全新的应用程序，其 URL 是随机生成的，如图 14-3 所示，本例中的应用程序被称为 candy-chaffeur。

The screenshot shows the Glitch web interface. At the top, there's a header with a lock icon, a folder icon, and the text "candy-chauffeur". To the right of the folder icon are buttons for "Show" and "Live". Below the header, there's a file tree on the left containing "assets", ".env", ".gitignore", ".travis.yml", "LICENSE", "README.md" (which is currently selected), "app.json", "index.js", and "package.json". On the right, under the "Markdown" tab, is the content of the README.md file:

Welcome to Probot on Glitch

This is the Glitch equivalent of running `create-probot-app` to generate a new probot app locally. Updates to your code will instantly deploy and update live.

Getting Started

To get your own Glitch-hosted Probot up-and-running, follow these steps. If you need more detail, the [Probot Docs](#) are a great place to go to learn more.

1. [Configure a new app on GitHub](#).
 - Hit the "Show" button on the top left of this page to find the URL. It will look something like `https://random-word.glitch.me/probot`, except the domain will be specific to your app.
 - For the Webhook URL, use this URL (again, updating the domain to match yours): `https://random-word.glitch.me/`. Notice that we left off the `/probot`.
 - For the Webhook Secret, just use "development". until Step 4.
 - Save your changes.
 - On the **Permissions & webhooks** tab, add read and write permissions

图 14-3 Glitch Probot 应用程序的 README 页面

图 14-3 左边的窗格显示了应用程序中的文件列表。README.md 文件包含了设置 hello-world Probot 应用的说明，仔细按照这些说明来设置 GitHub App 示例。其中一段说明提到了运行以下命令：

```
cat my-app-name.2018-06-20.private-key.pem | pbcopy
```

警示：上面命令的目的是把私钥文件的内容复制到剪贴板上，这样就可以把它粘贴到 Glitch 文件中了。然而，这个命令只在 Mac 系统上有效，在 Windows 系统上，可以运行以下命令（需要更改文件名以匹配相应的文件名）。

```
type my-app-name.2018-06-20.private-key.pem | clip
```

当完成以上操作后，就可以在仓库中安装该应用程序，然后创建一个新的 Issue。过几秒钟后，可以看到创建的评论上面写着“Hello World”。

14.2.5 定制应用程序

在 Glitch 中创建了一个 Probot 应用并安装在 GitHub 上后，可以自定义该应用对 Issue 评论的响应方式。按照 README 中的步骤操作，我们只订阅了 Issue 事件，并没有包括 Issue 评论，现在我们要同时订阅 Issue 评论事件。

<https://github.com/settings/apps> 中有相关的应用列表，单击 Edit 按钮，导航到应用程序，然后在左边的导航窗口中，单击 Permissions & events，向下滚动到订阅事件部分，勾选 Issue comment 复选框，如图 14-4 所示。

Subscribe to events

Based on the permissions you've selected, what events would you like to subscribe to?

- Security advisory** ⓘ
Security advisory published, updated, or withdrawn.
- Issue comment** ⓘ
Issue comment created, edited, or deleted.
- Issues** ⓘ
Issue opened, edited, deleted, transferred, pinned, unpinned, closed, reopened, assigned, unassigned, labeled, unlabeled, milestone, or demilestone.
- Label** ⓘ
Label created, edited or deleted.
- Milestone** ⓘ
Milestone created, closed, opened, edited, or deleted.
- Public** ⓘ
Repository changes from private to public.
- Pull request** ⓘ
Pull request opened, closed, reopened, edited, assigned, unassigned, review requested, review request removed, labeled, unlabeled, or synchronized.
- Pull request review** ⓘ
Pull request review submitted, edited, or dismissed.
- Pull request review comment** ⓘ
Pull request diff comment created, edited, or deleted.
- Repository** ⓘ
Repository created, deleted, archived, unarchived, publicized, or privatized.
- Watch** ⓘ
User stars a repository.

图 14-4 更新 GitHub App 的事件订阅

单击底部的“Save changes”按钮，保存这些更改。现在需要更改 Glitch App，编辑 index.js 文件，用以下代码替换该文件的内容。

```
module.exports = (app) => {
  // Listens to new Issue comments
  app.on("Issue_comment.created", async (context) => {
    // Retrieves the comment text
    const message = context.payload.comment.body;
    if (message.indexOf(" or ") > -1) {
      const params = context.Issue({
        body: "(https://media3.giphy.com/media/3o85xI03317RlmLR4I/giphy.gif)",
      });
      // Creates a comment with a markdown image
      return context.github.Issues.createComment(params);
    }
  });
};
```

这段代码的功能是监听新的 Issue 评论事件，寻找单词“or”，如果找到该单词，就创建 markdown 格式的评论。

这种方法并不是最优的一些自然语言处理（NLP）和人工智能（AI）的相关算法可以更好地实现相关功能。

14.2.6 安装应用程序

在创建并运行应用后，其他人可以安装使用该应用程序。例如，我们已经建立了一个版本的应用程序，那么可以进入 <https://github.com/apps/why-not-both>，单击

右上方的绿色安装按钮来安装该应用程序。安装完成后，在仓库的 Issue 中发送一段带有单词“or”的评论，其交互界面如图 14-5 所示。



图 14-5 应用程序使用界面

14.3 使用 GitHub Actions

上节通过创建 GitHub App 来实现 GitHub 的个性化，这就要求我们在 GitHub 之外托管我们的应用。事实证明，GitHub 有一种功能可以让我们的应用不需要托管在 GitHub 之外，这可以减少扩展 GitHub 时的成本，这个功能被称为 GitHub Actions。

GitHub Actions 是 GitHub 较新的、最令人兴奋的功能之一。在撰写本书时，它仍是测试版功能。GitHub Actions 使得在 GitHub 上创建自定义工作流成为可能。它可以实现自定义的逻辑来响应 GitHub 上的事件，上一节写了一个 GitHub 应用程序来做实现这个功能。有了 GitHub Actions，我们不需要构建自定义的应用程序，可以使用别人编写的现有 Action 来建立工作流，也可以编写自己的 Action，在 Docker 容器中运行。

提示：希望在读者看到本书时，GitHub Actions 已经普遍可用。但如果 GitHub Actions 还在测试阶段，请发邮件到 support@github.com，要求加入 GitHub Actions 测试项目。

为了演示 GitHub Actions，请考虑以下场景。当我们在自己的仓库中合并 PR 时，这个 PR 的分支会一直存在。GitHub 提供了删除该分支的按钮，但很多人都忘记了这样做，那么这些 PR 的分支会一直存在。Jesse Frazelle 写了一个 GitHub Action，可以在自己的工作流中使用。她的博文 “The Life of a GitHub Action”（见 <https://blog.jessfraz.com/post/the-life-of-a-github-action/>），是了解更多关于

GitHub Actions 生命周期的细节的好文章。当我们想要删除这些 PR 分支时，就可以使用她写的 GitHub Action。

14.3.1 创建 GitHub Actions 工作流

下面是为一个仓库创建 GitHub Actions 工作流的过程。

(1) 单击 Actions 选项卡，查看现有的工作流。如果没有任何工作流，就会看到“Create a new workflow”的按钮，用来创建一个新的工作流，如图 14-6 所示。

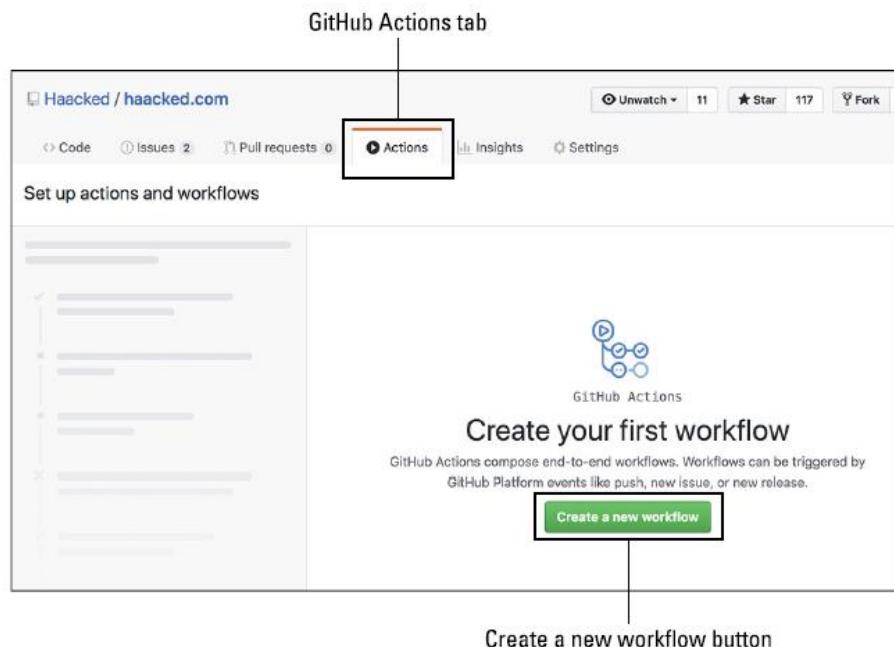


图 14-6 创建 GitHub Actions 工作流的页面

(2) 单击“Create a new workflow”按钮，调出工作流设计视图，它可以帮助创建工作流，并连接到 Action。工作流被存储在仓库的 .github 目录下一个名为 main.workflow 的文件中。如果喜欢用文本建立工作流，也可以使用编辑器视图。

(3) 切换到文本编辑器，输入以下代码。

```
workflow "on pull request merge, delete the branch" {
    on = "pull_request"
    resolves = ["branch cleanup"]
}
action "branch cleanup" {
    uses = "jessfraz/branch-cleanup-action@master"
    secrets = ["GITHUB_TOKEN"]
}
```

再切换回可视化设计器，提交一个简单的工作流，如图 14-7 所示。

(4) 单击“Start commit”按钮，将这个新的 main.workflow 文件提交到仓库中。

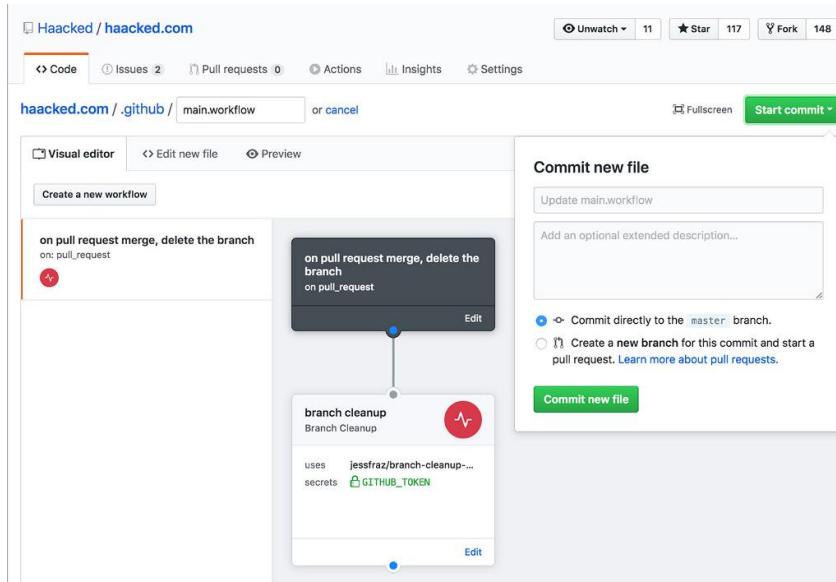


图 14-7 提交一个简单的工作流

14.3.2 测试 GitHub Actions

在提交这个新的文件到仓库后，工作流就被激活了，可以通过创建一个新的 PR，然后将其合并来测试该工作流。几秒钟或几分钟后，就可以看到该 PR 分支被删除了。PR 上的更新内容如下：

```
github-actions bot deleted the branch-name branch 1 minute ago
```

提示：我们可以使用现有的 Actions 创建更复杂且更有用的工作流，也可以在 GitHub 市场上安装 Actions，或者引用别的仓库的 Actions。读者还可以编写自定义的 Actions，但是这超出了本书的范围，读者可以参考 Jesse Frazelle 在 <https://github.com/jessfraz/branch-cleanup-action> 写的 Actions，了解一下它的作用。

GitHub Actions 可以以几乎无限可能的方式实现 GitHub 个性化，满足不同用户的需求。