

第3章 GitHub 仓库

本章学习重点：

- (1) 探索 GitHub 仓库
- (2) 创建一个 Hello World 仓库
- (3) 探索仓库 issues, pull requests 和项目看板

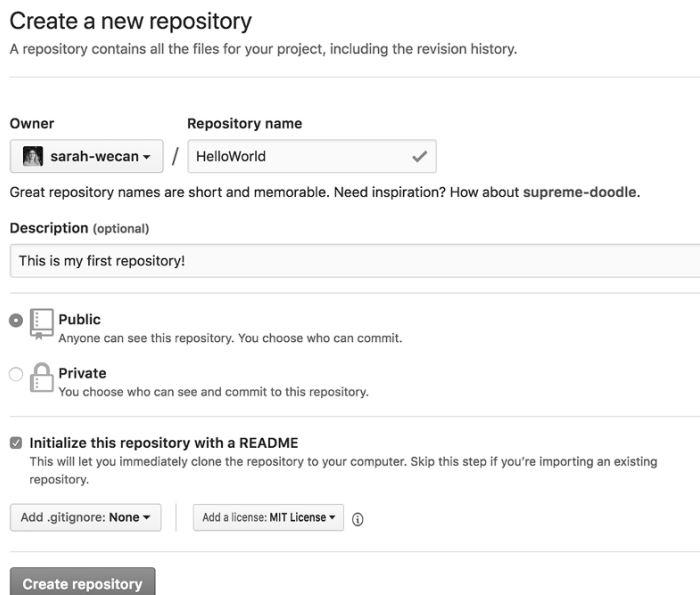
在 GitHub.com 上几乎所有的事情都围绕着仓库进行。本章将介绍如何创建、使用仓库以及如何创建项目看板和 issues。

3.1 创建仓库

一个 GitHub 仓库实质上是一个包含了项目所需的所有文件的文件夹。在该文件夹中记录了项目所有版本的文件，出现错误时可以返回到之前的版本。此外，一个 GitHub 仓库还记录了哪些人通过哪种方式参与到项目的协作中。

为了更好地理解什么是仓库以及仓库是如何组织的，需要先创建一个 GitHub 仓库：

- (1) 单击 GitHub.com 网页左上角的 logo 进入个人主页。
- (2) 单击绿色的“New”按钮，转到如图 3-1 所示的创建仓库页面。



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: sarah-wecan / Repository name: HelloWorld ✓

Great repository names are short and memorable. Need inspiration? How about supreme-doodle.

Description (optional): This is my first repository!

☒ Public
Anyone can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

☒ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: MIT License ⓘ

Create repository

图 3-1 创建仓库页面

- (3) 在“Repository name”输入框中输入仓库名称，这里可以将仓库命名为“HelloWorld”。
- (4) 在“Description”输入框中输入仓库简介。

(5) 选择“Public”。

(6) 勾选“Initialize this repository with a README”，这里不需要添加一个.gitignore 文件。

(7) 从“Add a license”下拉菜单中选择一个许可证。若希望了解更多关于许可证的信息，请参见下方的“软件许可证”。

(8) 单击“Create repository”按钮。

将会显示仓库主页。此时，markdown 文件——README.md——已经在仓库中了。markdown 是一款轻量级标记语言，允许用纯文本格式编写带样式的文档。在 markdown 中，可以将文本设为粗体样式或标题样式，也可以创建一个表格来展示数据。

软件许可证

软件许可证对协作编码至关重要。无论是把自己的代码放在 GitHub.com 与世界分享，还是将代码贡献到他人的仓库中，都应该知道哪些操作是允许的、哪些操作是不允许的。如图 3-1 所示，当创建一个新的仓库时，GitHub 会引导是否需要为该仓库添加一个许可证。单击一旁的问号，页面会跳转到 <https://choosealicense.com/>，该页面会解释三种最常见的软件许可证。

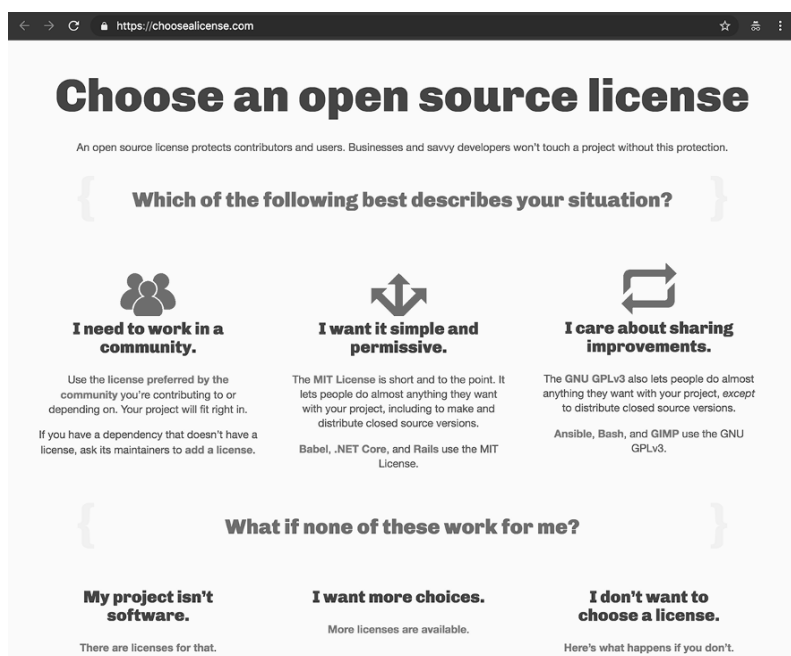


图 3-2 选择许可证

两种适用于公开协作软件的主要许可证分别是 MIT 许可证和 GNU General Public License v3.0 许可证。若项目使用了 MIT 许可证，人们可以对代码进行复制、修改、以闭源的形式再分发。也就是说软件可以被他人以应用的形式分发却不向用户公开软件的源代码。若项目使用了 GPLv3 许可证，人们同样可以复制、修改、贡献自己的代码。但是，假如想分发自己的版本，必须公开源码。本书中的仓库都使用 MIT 许可证，创建仓库时可以自由选择所需的许可证。

图 3-3 为 HelloWorld 仓库的主页。

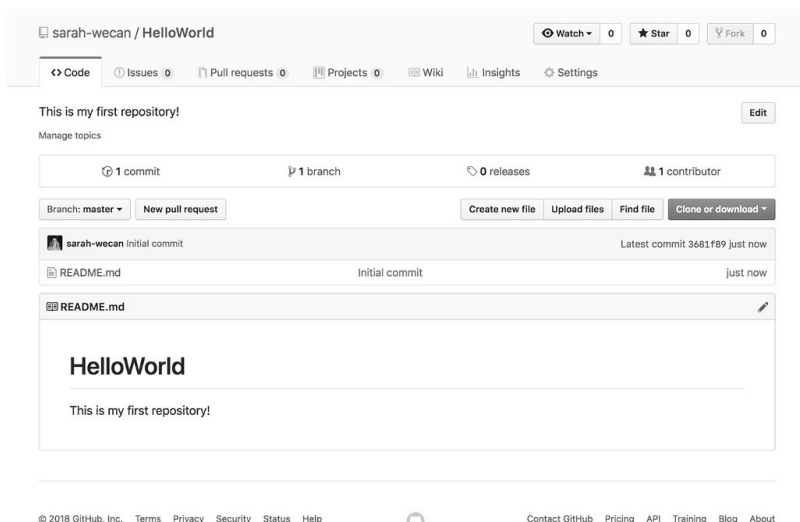


图 3-3 HelloWorld 仓库的主页

在第四章和第五章中，介绍了如何创建一个网站以及从该网站链接到已创建的仓库。

3.2 探索仓库

接下来介绍仓库页面中的各个区域，从整体上认识仓库。

3.2.1 顶部信息

仓库页面顶部显示了仓库创建者的用户名和仓库名称。若该仓库 fork 自其他仓库，还会显示原作者的用户名，单击链接会跳转到原始仓库页面。fork 一个仓库即复制一个仓库，在该副本上做出修改后，还可以将其推送给原作者。本书第 6 章对 fork 仓库进行了深入介绍。

在用户名右方有三个按钮。

(1) **Watch:** 可以根据在仓库中发生的活动类型选择希望接收的通知类型。

(2) **Star:** Star 一个仓库除了可以让使用者快速找到它，还能帮助 GitHub 洞察出使用者的兴趣所在，从而为使用者提供更准确的推荐。单击页面右上角头像，单击“Your Stars”即可浏览所有使用者已 star 的仓库。

(3) **Fork:** 如果使用者不是某个仓库的创建者，就能够 fork 这个仓库。第 6 章会详细介绍 fork。

强烈建议使用者对于大部分仓库在“Not watching”和“Releases only”中二选一。这样的话，使用者只会在被特别提及或主动参与关于某个 issue 或 PR 的讨论时接收到通知。否则的话，使用者的邮箱马上会被记录仓库中任何一举一动的邮件填满。

3.2.2 Tabs

仓库页面上方有七个选项卡，每个选项卡为仓库提供了不同的特性。

(1) **Code:** 在 Code 选项卡中可以查看全部代码以及浏览所有文件。单击文件即可查看

文件内容，单击铅笔图标可以修改文件——相应操作都在浏览器内进行。（在下文“Code 选项卡”有更详细的介绍）

（2）Issues: Issues 是仓库中非常巧妙的功能，能帮助你追踪任务、问题、或建议。本章“使用 Issues 和项目看板”这一节将学习如何创建 issues。

（3）Pull requests: Pull requests 也被称为 PRs，它和 Issues 相似，也有一个标题和一段描述，但还包括了希望被主分支拉取的代码变化部分。贡献代码最保险的方式是新建一个分支并在该分支上改动，然后请求该分支并入到主分支。每个 PR 都提供了一个用于合并两个分支的界面。在界面中，可以看到此分支与主分支中相应文件之间的差异，此外，还能与协作者展开讨论，商洽此分支是否被采纳或在并入前还需再做哪些修改。关于分支的具体内容，请参见第1章。

（4）Projects: GitHub 把项目看板无缝嵌入在仓库中，看板中的每个卡片都能直接与 issues 或 PRs 相关联，因此当特定事件发生时卡片能够自动在看板中移动。想了解更多关于项目看板的信息，请参见本章稍后小节“使用 Issues 和项目看板”。

（5）Wiki: Wiki 的功能主要是用于展示项目文档、项目状态和项目路线图。协作者可以通过 Wiki 了解项目进展和发现可贡献之处。

（6）Insights: 如图 3-4 所示，Insights 选项卡展现了仓库中所有协作者和发生的行动的数据概况。在热门开源项目的仓库内，通过该选项卡读者可以对该仓库的整体状态一目了然。以 TensorFlow 为例，上个月该仓库共有 158 名贡献者！

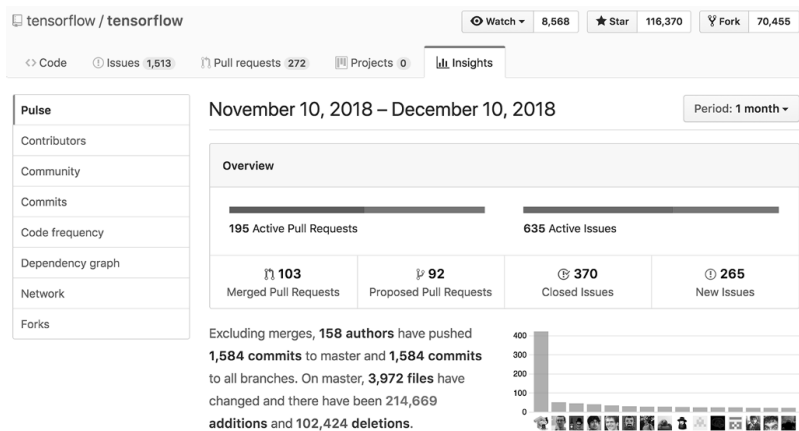


图 3-4 Insights 选项卡

（7）Settings: 只有对仓库拥有相应权限的人，才可以看见 Settings 选项卡。在该选项卡中，可以决定他人对仓库的访问权限和协作者间的合作方式。此外，还能在此集成统计代码测试覆盖率的应用。

3.2.3 Code 选项卡

如图 3-5 所示，Code 选项卡中有许多与仓库相关的在未来的开发中非常有用的重要元数据。

（1）Description and topics: Code 选项卡顶部显示了仓库的描述和主题。为仓库添加主题可以让仓库更易被发现，主题有利于吸引其他开发者参与到你的项目中。

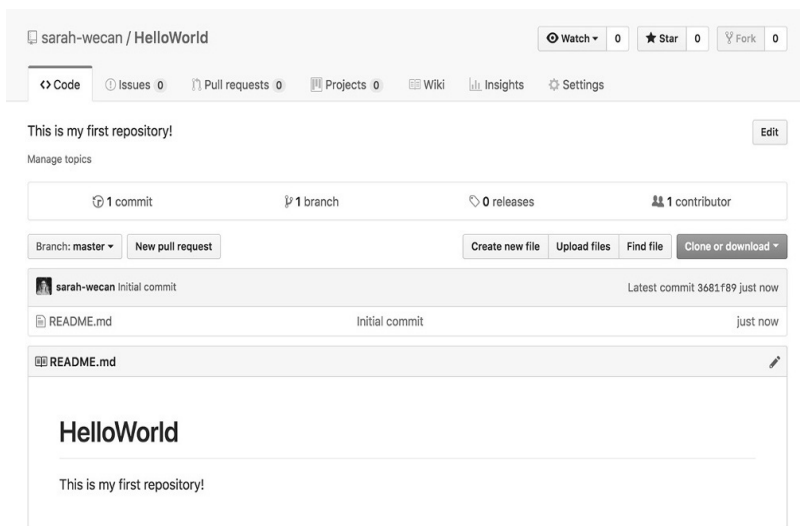


图 3-5 Code 选项卡

(2) **Metadata**: 该条状区域显示了仓库的提交、分支、发布及协作者数目，单击可进入各自的详情页面。

(3) **Action buttons**: 该区域左边有一个用于切换分支的下拉菜单，通过该下拉菜单可以浏览不同分支下的文件。“New Pull request”按钮用于快速创建一个 PR。创建 PR 的最好方式是：切换到另一分支，做一些修改，然后单击“New Pull request”按钮。该区域右边有三个与文件相关的按钮：“Create new file”、“Upload files”、“Find file”。通过最右边的绿色下拉菜单式按钮“Clone or download”可以将代码克隆或下载到本地（见第 4 章）。

(4) **Code**: Code 选项卡底部是包含仓库内所有代码的目录，如果在根目录下存在一个 README.md，那么在目录下方会将其显示出来。单击目录中的文件可以跳转到对应页面，可以在页面中浏览或编辑文件。

3.3 修改 README.md

我们建议每个仓库，不管它是公开的还是私有的，都应该在根目录下添加一个 README.md 文件。对任何有意于做出贡献的开发者来说，该文件是他们在此仓库的起点。

README.md 文件一般包含以下内容：

- (1) 项目名称和项目简介；
- (2) 项目成功运行的前置条件；
- (3) 关于安装项目的说明（及依赖）；
- (4) 确保运行测试顺利进行的一切妥当的说明；
- (5) 关于部署项目的说明；
- (6) 项目依赖概览；
- (7) 关于如何为项目做出贡献的指南（包括行为准则）；
- (8) 项目的主要作者和维护者；
- (9) 一个指向软件许可证的链接；

(10) 致谢。

GitHub 倡导共享、开放的软件开发文化。就共享而言，每个人都应该感谢那些给予自己灵感或已被自己所借鉴的来自他人的分享。一款软件的开发离不开前人的工作，很少有软件是从零开始开发的。Grace Hopper 是一位著名的计算机科学家，她创造了世界上第一款编译器和英语编程语言，推动了高级编程语言的发展，使人们能不再只用汇编语言编写程序。虽然人们不必指名道姓地感谢 Grace Hopper 的贡献，但应该意识到有那么一大群人，他们永不停歇地构造、创建、推动着人们的认知边界。

对简易的项目来说，README.md 文件也可作为项目的展示页面。在下面的案例中，项目就是一个人的个人介绍，README.md 文件就是该项目的全部，它包含了所有关于项目的信息！

跟着下面的步骤，在 README 中添加一张大头照，并简单介绍一下深受大家欢迎的 gitHub。

步骤 1：通过仓库的 Code 选项卡，单击“Upload files”按钮。进行步骤 2 至 6 的操作，可以完成如图 3-6 所示页面。

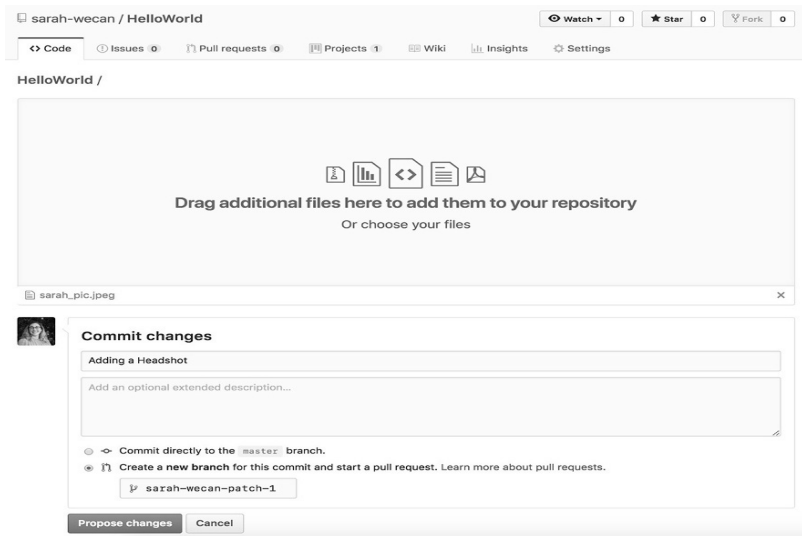


图 3-6 添加文件到 HelloWorld 仓库的页面

步骤 2：找一张大头照（或任何希望被放到 README 中的照片），将它拖进上传区域。也可以点击“choose your files”浏览文件选择照片。

步骤 3：在“Commit changes”一栏中输入标题。在该例中，将标题取为“Adding a Headshot.”

步骤 4：选择“Create a new branch for this commit and start a pull request”。

步骤 5：输入新分支的名称，或直接使用默认的名称。（默认的名称为“sarah-wecan-patch-1”）

步骤 6：单击“Propose changes”按钮。将显示“Open a pull request”页面，添加大头照的 commit 也被包含在该 PR 中，如图 3-7 所示。

步骤 7：加一段用于解释文件变动的描述，然后单击“Create pull request”按钮。如图 3-7 所示，在 Pull request 选项卡中，可以看到 PR 的标题、序号、状态、描述、commit

清单和判断它是否能并入主分支的检查。

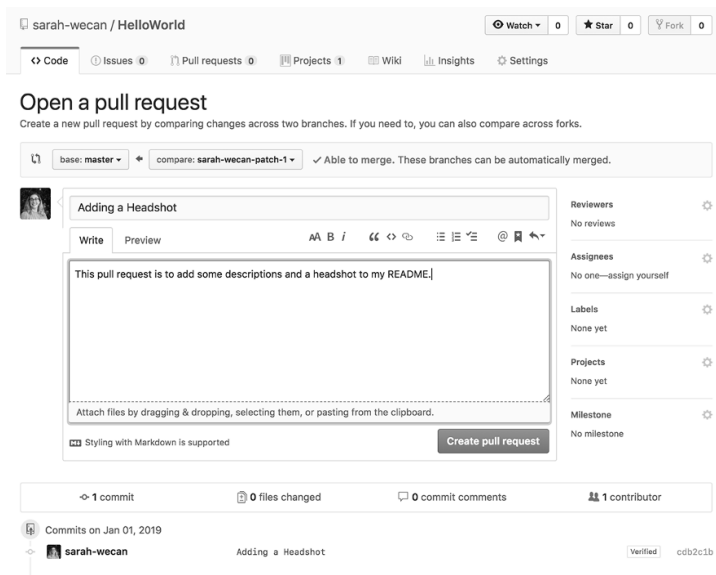


图 3-7 Open a pull request 页面

步骤 8：单击 **Code** 选项卡返回到仓库的代码部分。在下拉分支菜单中选择在步骤 5 中创建的分支。该操作使仓库切换到处于该分支下的状态。正由于刚才对仓库做出了变动，并以步骤 6 中的方式创建了 PR，这些变动已添加到了该分支上。通过步骤 6 的操作一次性完成了：创建新分支、添加大头照、提交变动、发起 PR。

在分支旁边有一个指向刚创建的 PR 的链接。该分支和 PR 相互联系，可以将它们看作是一件事。还能在仓库目录中看到刚才上传的照片文件，如图 3-8 所示。

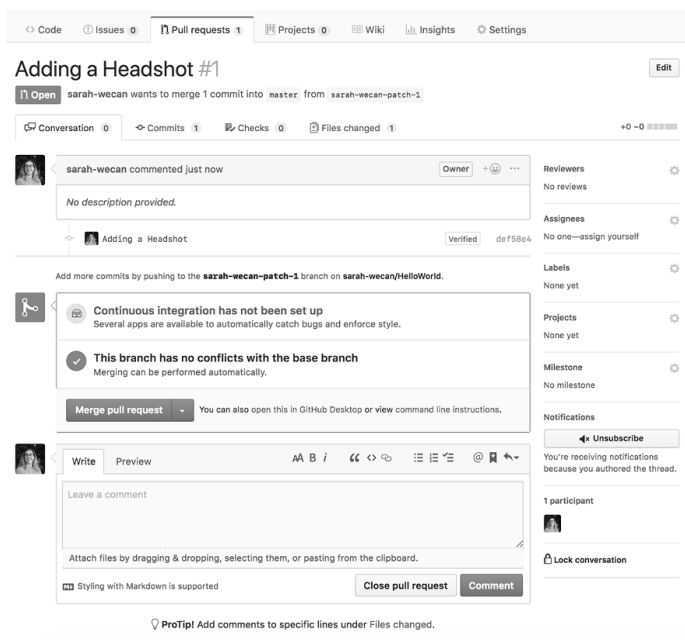


图 3-8 开启着的 pull request 页面

- (1) README 一直显示在仓库目录下方，单击小铅笔图标可以进行修改。
- (2) 用 Markdown 格式写下一些自我介绍，参考图 3-9 中的例子可以包括热衷的事业和兴趣爱好。
- (3) 单击文本输入区域上方的“Preview”，如图 3-10 所示，行左边的红色竖线代表此行被删除，绿色竖线代表该行被添加。
- (4) 确认输入内容后，将页面滚动到文本编辑器底部，为该 commit 取个标题，然后 commit 到刚才创建的分支中。现在可以看到修改后的 README.md 文件的内容。
- (5) 单击“HelloWorld/README.md”中的“HelloWorld”返回到仓库首页。

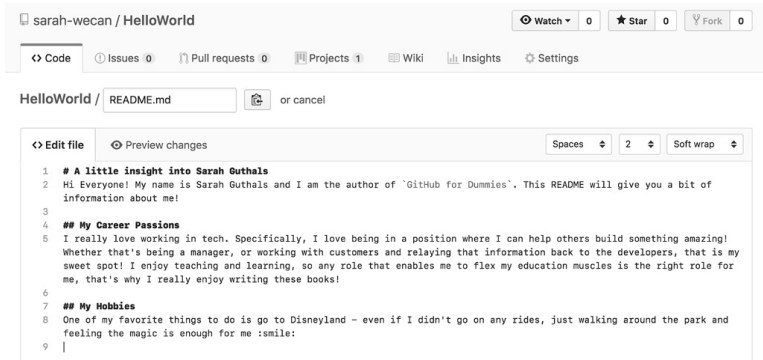


图 3-9 编辑模式下的 README.md

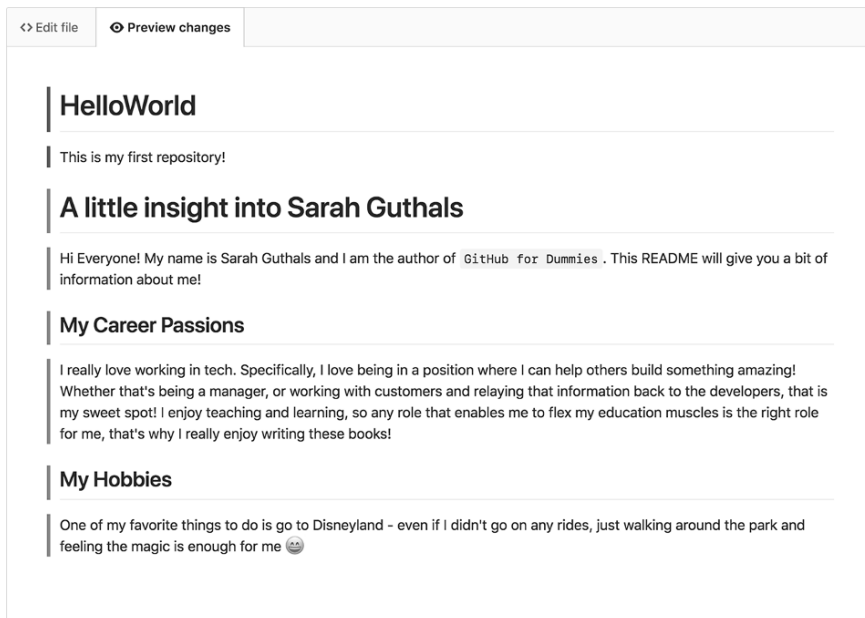


图 3-10 diff 模式下的 README.md

- (6) 再次单击 README.md 上方的小铅笔图标，通过添加下面这行代码引入大头照：
`![headshot](sarah_pic.jpeg)`
- (7) 单击“Preview”进行预览，此时大头照被显示出来了。如图 3-11 所示。
- (8) 滚动到页面底部，commit 所做的修改。

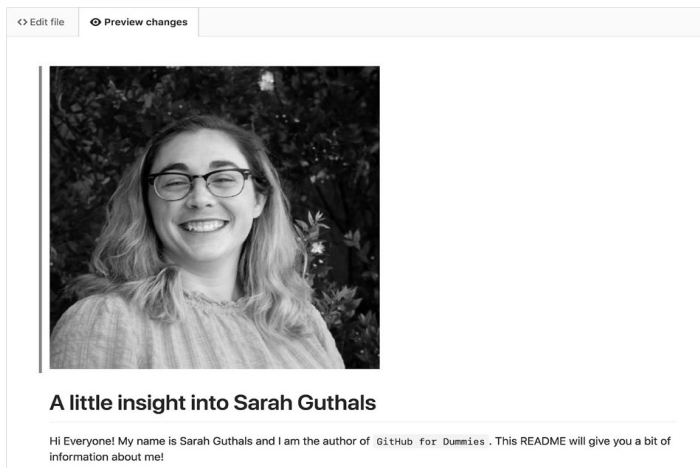


图 3-11 添加大头照后 diff 模式下的 README.md

通过前面的操作，已经对仓库进行了一些修改，但是这些修改都在另一个分支上，而不是在 `master` 分支上。如何将这些修改并入到 `master` 分支中？可以阅读下一小节。

3.4 合并一个 Pull Request

当一个 PR 中的所有更改都准备就绪时（见前一小节），可以通过以下步骤将所做更改合并到 `master` 分支中：

（1）在仓库的 `Code` 选项卡中，单击“`View #1`”按钮访问 PR。如图 3-11 所示，PR 页面中有 3 个不同的 commit。在最早的 commit 中，将大头照文件上传到仓库；第二个 commit 中，在 `README.md` 中添加了些文字；第三个 commit 中，把大头照引入到 `README.md` 中。（具体操作请参照上一节“修改 `README.md`”）

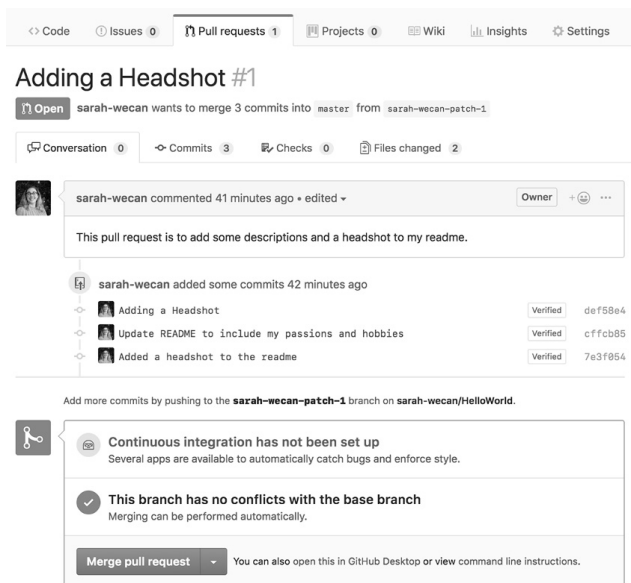


图 3-11 PR 页面

(2) 在 PR 页面中, 单击 “Files changed” 查看仓库中的所有更改。更改分为删除和添加, 红色代表删除, 绿色代表添加。

(3) (可选) ”Files changed “中的 diff 视图是可以改变的, 点击 “Diff Settings” 下拉菜单, 选择 ”Split “, 然后点击 ”Apply and reload “。若选择了垂直分割视图, 屏幕布局会发生改变, 如图 3-12 所示。

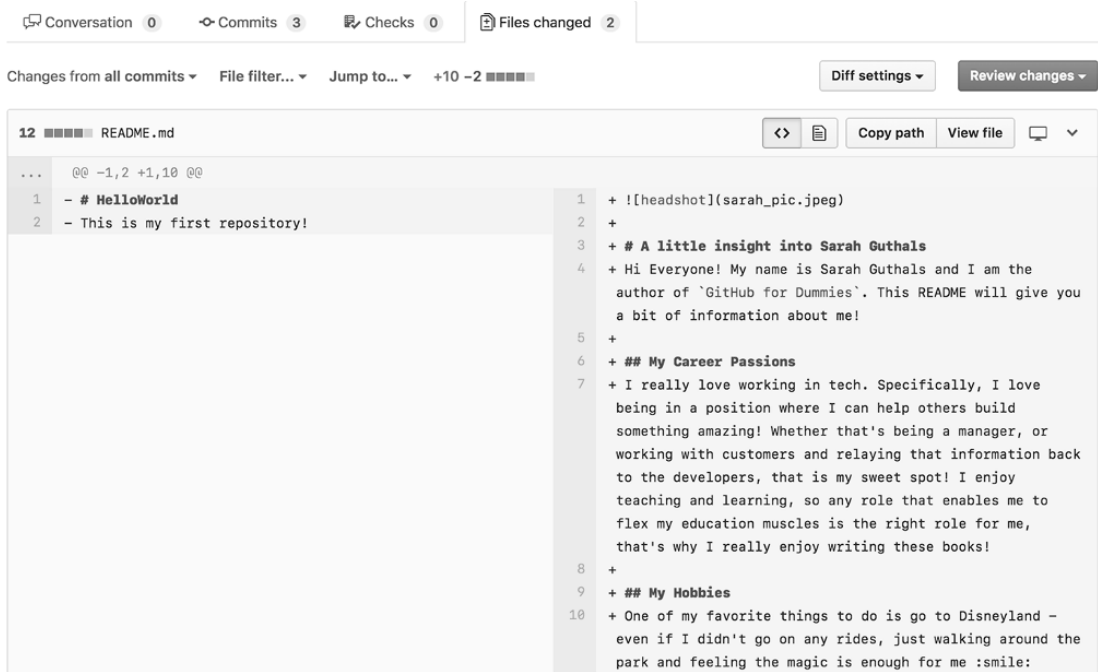


图 3-12 " 垂直布局下的 diff 页面 "

(4) 回到 PR 页面中的 ”Conversation “, 滚动到页面底部, 单击 ”Merge pull request “绿色大按钮, 见图 3-11。按钮所在位置被替换为一个确认合并的会话框, 如图 3-13 所示。

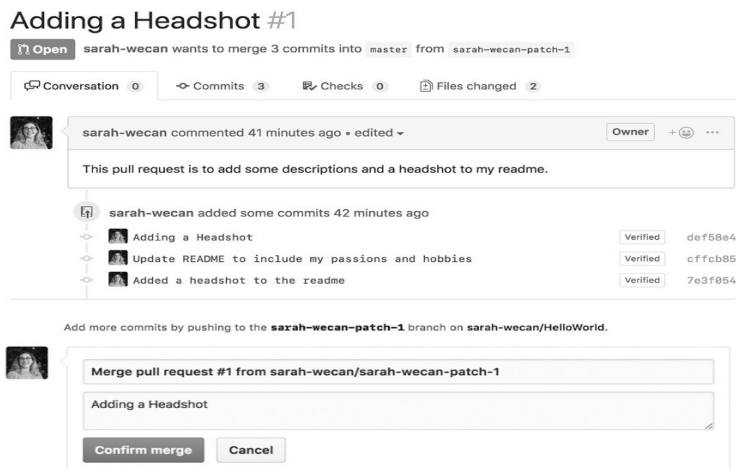


图 3-13 ” 合并 PR 确认会话框 “

(5) 单击” Confirm merge “。出现一条消息提示 PR 已被成功合并，并带有 “Delete branch” 的选项，如图 3-14 所示。

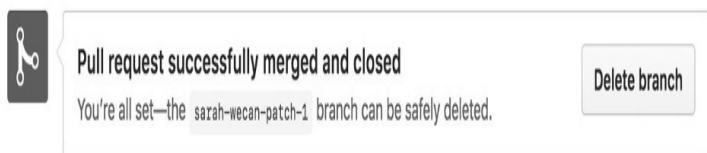


图 3-14 ” PR 被成功合并 “

(6) 单击” Delete branch “。PR 已经被合并，分支也被删除。需要的情况下，该分支可以被恢复。

(7) 单击” Code 选项卡 “查看代码。此时，照片和更改后的 README.md 已经出现在 master 分支上了。

3.5 使用 Issues 和项目看板

GitHub 仓库中的 Issues 是追踪需要修复、添加和更改的内容的好方法。将 Issues 和项目看板结合起来，可以深入了解原本难以追踪的项目。本节将创建 issue 和项目看板，并更改 README.md。

3.5.1 创建一个项目看板和 issue

接下来开始使用 issues 和项目看板。前往仓库首页，然后执行以下步骤。

(1) 进入 Projects 选项卡，然后单击” Create a project “按钮，如图 3-15 所示，显示新建项目看板对话框。

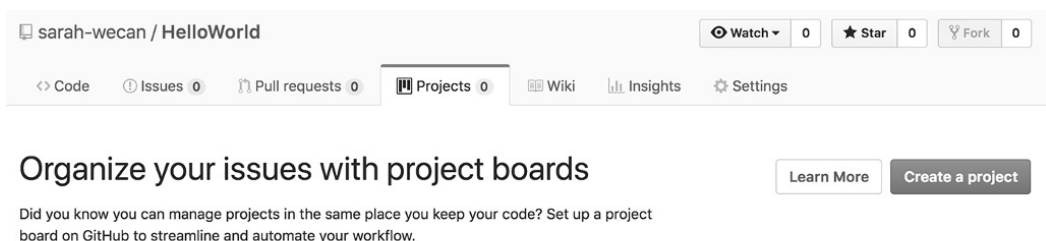


图 3-15 ” 项目看板新建页面 “

(2) 输入看板名称和描述。

(3) 在预置模板选择下拉菜单中，选择一个模板然后单击 “Create project” 按钮。该示例中，选择 “Automated kanban” 模板。在新建的项目看板中有一些 To do 卡片，它们会引导你使用看板中的功能。

(4) 单击 To Do 栏底部的 “Manage” 按钮打开自动化配置会话框。图 3-17 显示了 To do 栏中的自动化配置项。每创建一个 issue 时，它会被自动加入到 To do 栏中。在步骤 8 中，将创建一个 issue。

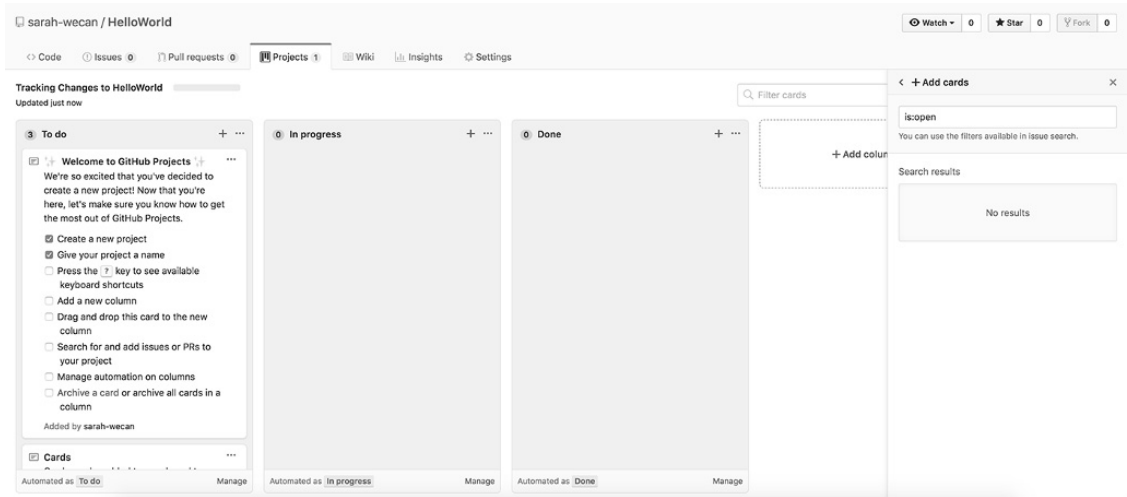


图 3-16 “项目看板初始状态”

(5) 单击 In progress 栏底部的“Manage”按钮。弹出类似的自动化配置会话框（见图 3-17）。每当一个已关闭的 issue 重新开启，或者一个 PR 被创建或重新开启，它们会出现在 In progress 栏中。当执行本节的步骤 11 后，便能在看板的 To do 栏中看到新建的 issue 卡片。

(6) 单击 Done 栏底部的“Manage”按钮。弹出与图 3-17 类似的自动化配置会话框。每当一个 issue 或 PR 被关闭或一个 PR 被合并，与之对应的卡片会从 In progress 栏移动到 Done 栏。可以在“关闭一个 issue”的步骤 14 中看到这一现象。

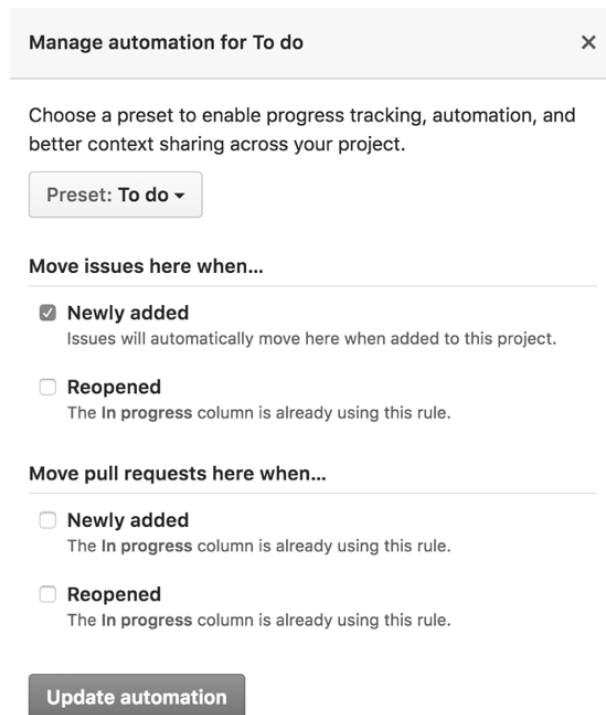


图 3-17 “项目看板 To do 栏中的自动化配置项”

(7) 单击卡片右上方的三个点，选择菜单中的“Delete note”，可删除此卡片。

(8) 在仓库的 Issues 选项卡中，单击“New issue”。

提示：图 3-18 显示了在 Issues 选项卡上方的提示（若之前没有将其忽略的话就能看到）。issue 和 PR 可以设置标签，当希望按标签筛选出需要处理的事项时，这将很有用。若想为一个开源项目做贡献，可以查看 help wanted 或 good first issue 标签。（阅读本书第 5 部分了解更多有关开源软件的内容）

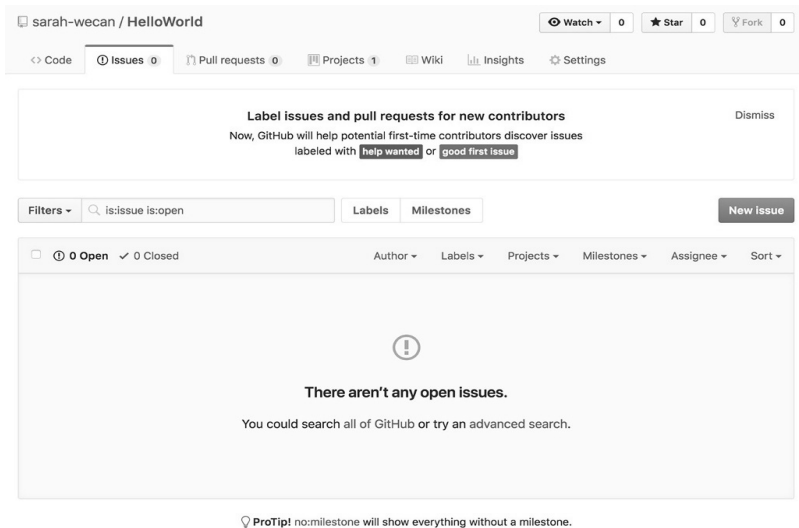


图 3-18 “HelloWorld 仓库的 issues 页面”

(9) 输入 issue 的标题和描述，然后在右方面板中将该 issue 分配给自己。

(10) 在“Projects”面板下，选择“Tracking Changes to HelloWorld”，以此将该 issue 关联到项目看板。之后，便能在“Projects”下看到它，如图 3-19 所示。

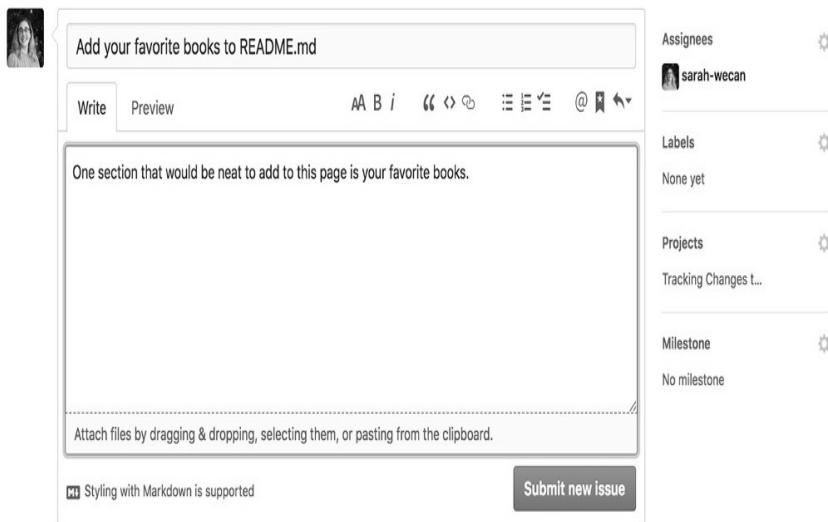


图 3-19 “与项目看板‘Tracking Changes to HelloWorld’关联的新 issue”

(11) 单击“Submit new issue”按钮。与 Pull Request 类似, issue 也有标题、序号、状态、描述以及页面右边的元数据(见图 3-20)。注意, 该 issue 的序号是“#2”, 即使它是在仓库中创建的第一个 issue。这是因为 issue 和 PR 共同编号, 所以之前创建的 PR 的序号是“#1”, 该 issue 的编号是“#2”, 之后创建的 PR 或 issue 的编号将会是“#3”。该 issue 被创建后, 在项目看板中会显示对应的卡片。

单击“Projects”下的看板链接即可跳转到项目看板中, 新卡片出现在 To do 栏中。

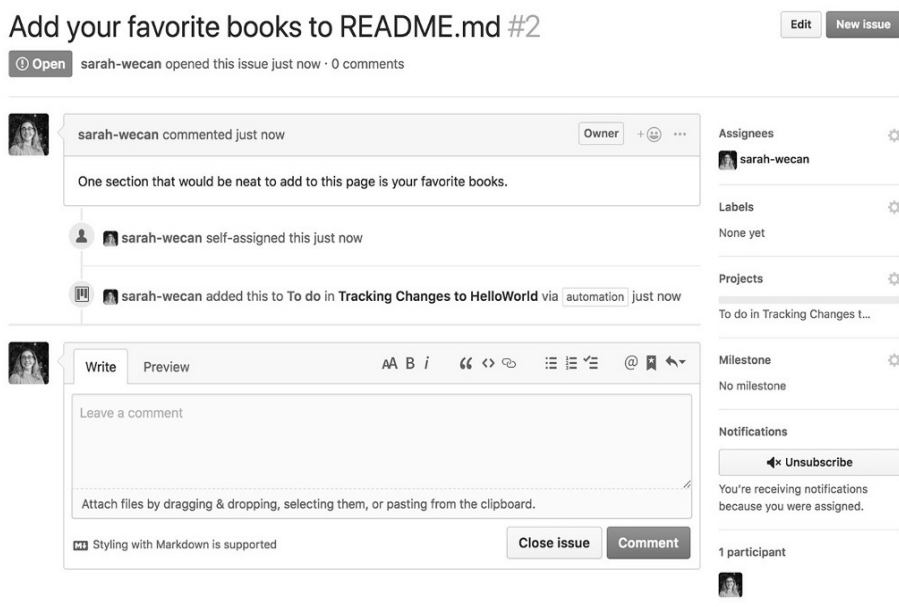


图 3-20 “一个开启着的 issue 页面”

3.5.2 关闭一个 issue

关闭一个 issue 的最好方式是创建一个解决了该 issue 中所描述问题的 PR。理解 issue 和 PR 间的关系对个人项目和开源项目大有裨益。

执行以下步骤来关闭刚才创建的 issue。

- (1) 在仓库 Code 选项卡中点击 README.md 上方的小铅笔图标来编辑文件。
- (2) 在 README.md 底部添加一节内容谈谈你最喜欢的书籍。
- (3) 滚动到页面底部, 填写该 commit 的标题。
- (4) 选择“Create a new branch for this commit and start a pull request”。
- (5) 单击“Propose changes”。
- (6) 填写 PR 的描述。确保在描述中将“closes #2”写为单独一行。当键入#时, GitHub 会提示此仓库中的 issue 或 PR, 可以浏览并选择, 选择后会自动填充到文本框中。
- (7) 将该 PR 与项目看板关联。这与将 issue 和项目看板关联的操作一样。(见前面“创建项目看板和一個 issue”中的步骤 10)
- (8) 单击“Create pull request”
- (9) 返回到项目看板。如图 3-21 所示, 与该 PR 对应的卡片出现在 In progress 栏中。

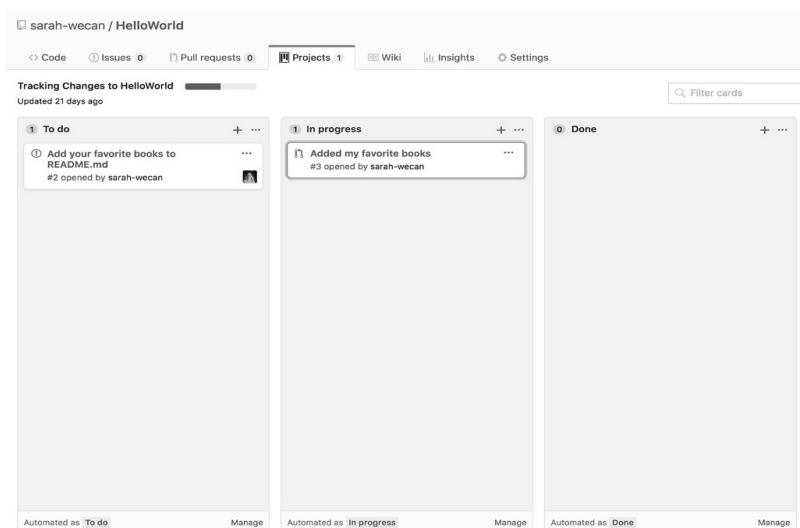


图 3-21 “包含一个开启中的 issue 和一个开启中的 PR 的项目看板”

(10) 单击 PR 卡片标题可以预览 PR。

单击“Go to pull request for full details”返回到 PR 页面。

(11) 单击“Files changed”。

可以看到最近的一节内容已添加到 README.md 中（见图 3-22）。单击“View File”可阅读修改后的 README 文件。

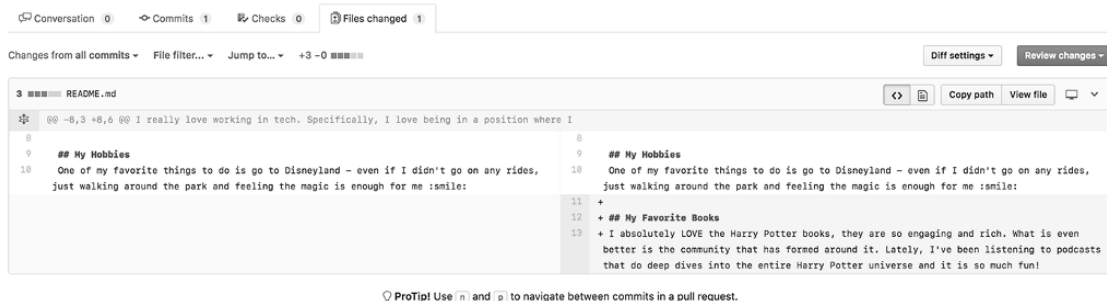


图 3-22 “README.md 的 diff 页面显示了新添文本”

(12) 单击浏览器的后退按钮。

(13) 确认内容无需修改后，单击“Conversation”，单击“Merge pull request”按钮，单击“Confirm merge”按钮，然后单击“Delete branch”按钮。

若在以上步骤的操作中出现問題，请参阅前面“合并一个 Pull Request”小节的内容

(14) 单击 Projects 选项卡并选择之前创建的项目看板。

分别来自 To do 栏和 In progress 栏的两个卡片都移动到了 Done 栏。

第 4 章 建立一个 GitHub 网站仓库

互联网是一项对促进社会发展和推动软件开发十分重要的技术。随着互联网进一步融入人们的日常生活，它带来了崭新的意义、职业生涯机遇以及更多连接你我的方式。互联网的发展促进了一系列社交媒体网站的诞生。社交媒体中的个人主页为人们提供了表达自我的渠道——我是谁？我的兴趣？我的联系方式？但是，社交媒体也创造了新的挑战。

即使人们在 Twitter、Instagram、Snapchat、Facebook 和 LinkedIn 上都有帐号，一般也不会在 LinkedIn 上分享他的私人生活成就（比如生娃），也很少在 Instagram 上分享个人的专业成就（比如创立了一个新项目）。对一般人来说，拥有个人网站意味着人们能借此认识一个多面的他，而不只是为了适应特定社交媒体上社区环境的某个版本的他。这样的话，当人们想了解一个人在做什么新项目时，一般会直接访问他的网站一探究竟。这种互动改善了生活中人们之间的联系以及他们的关系。

即使一些人搭建个人网站是为了聚焦自己的某一方面，但其实他们更喜欢分享内容时的掌控力。本章和第五章会引导读者将任何项目仓库变为一个网站以及在 GitHub.com 上创建一个个人网站。只需几分钟，就能创建并运行一个网站，不需要额外花一分钱。

4.1 GitHub Pages 介绍

GitHub Pages 是一种在 GitHub.com 上部署网站的简单快速的方式。在仓库中的代码即为运行在网站上的代码。更方便的是，还能用 Jekyll 轻松地更换网站的样式。Jekyll 是一款免费开源的网页生成器，它能将 Markdown 文件转换为网页，并支持许多主题。

提示：可以访问 <https://jekyllrb.com> 了解更多关于 Jekyll 的信息，也可以前往 Jekyll 的 GitHub 仓库（<https://github.com/jekyll/jekyll>）了解它的工作原理。有了 GitHub Pages，便能使用 Markdown 或 HTML/JavaScript/CSS 来创建一个网站。

记住：若希望复习一下 Markdown 的使用方法，请访问 <https://guides.github.com/features/mastering-markdown> 阅读 GitHub 的 Markdown 教程。

4.2 将项目仓库变为网站

GitHub Pages 被很好得整合在 GitHub.com 中。GitHub Pages 会在 master 分支中寻找 README.md，并将它作为网站的首页。只需要以下几步操作，就可以很容易地把 GitHub Pages 运行起来！

（1）打开一个 GitHub 仓库；

在本案例中使用第三章中的 HelloWorld 仓库。

（2）在仓库主页中，单击右上方的 Settings 标签页打开仓库设置页面；

设置页面如图 4-1 所示。

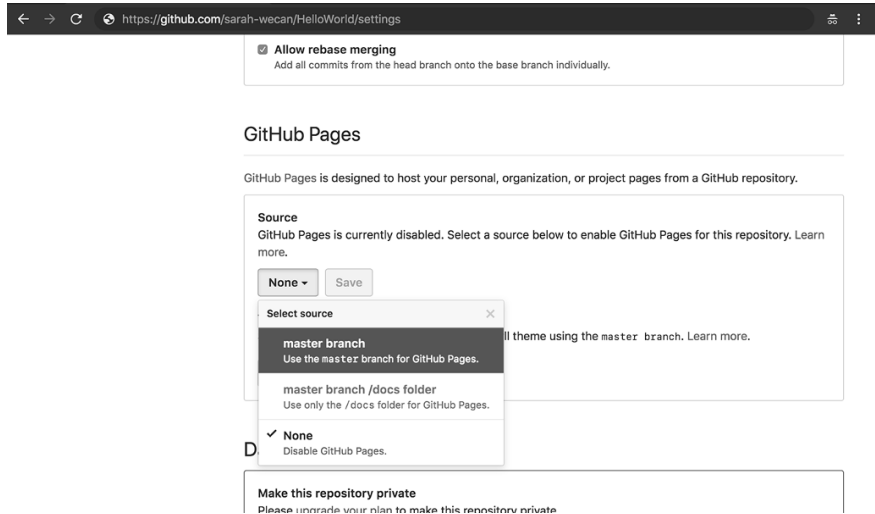


图 4-1 GitHub 仓库设置页面

(3) 在 Source 的下拉菜单中，将 GitHub Pages 的源从原来的“None”替换为“master branch”，然后单击“Save”保存设置；

网页会自动刷新。再次打开设置页面，可以看到 GitHub Pages 的 Source 已经变为“master branch”了。

(4) 为网站选一个主题；

如图 4-2，展现了一个新页面。浏览并选择一个主题。本案例中选择了“Time Machine”主题。

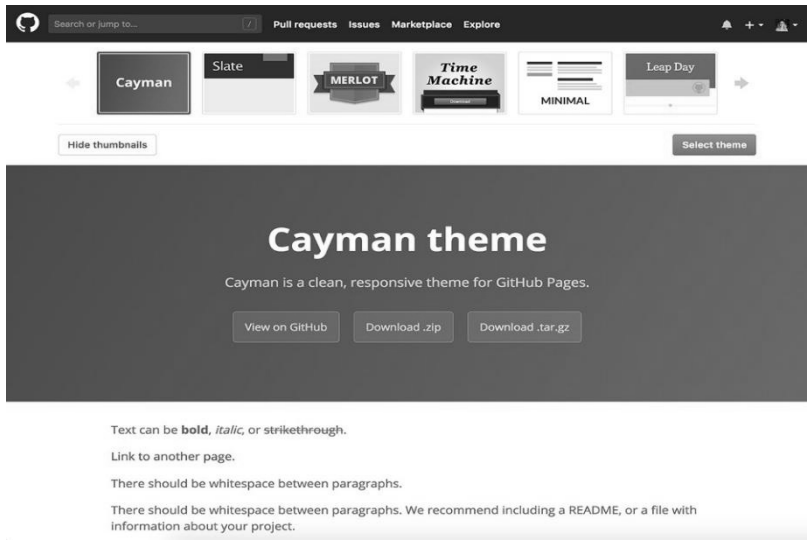


图 4-2 选择 GitHub Pages 网站主题

(5) 在为网站选择一个主题后，单击“Select theme”按钮；
页面会返回到设置页面。

(6) 访问项目网站；

将仓库设置页面滚动到 GitHub Pages 位置，可以看到一则绿色通知，单击上面显示的网站地址，即可访问。

提示：如果没有看到这则通知，尝试刷新页面。通知没被立即显示的原因是因为有时候浏览器会缓存页面，造成页面没被刷新。

(7) 单击链接访问刚创建的网站。

4.3 创建个人网站仓库

创建一个新的仓库来托管创建的个人网站：

(1) 新建一个仓库，并命名为“username.GitHub.io”，其中“username”是我们的 GitHub 用户名；

例如，Sarah 的个人网站仓库名称应取为“sarah-wecan.GitHub.io”。如果不知道如何创建一个仓库，请阅读第三章。

(2) 将仓库设置为公开，初始化 README，根据需要选择一个许可证，然后单击创建仓库；

如何确定是否需要一个许可证，请参见第三章中使用许可证好处的相关介绍。

刷新页面，显示仓库首页。

(3) 用“Automated kanban”模板创建一个项目看板；

有关如何创建一个项目看板，请阅读第三章。

(4) 在仓库主页上方单击 Settings 标签页，滚动页面直到看到 GitHub Pages 部分；

此时网站已经准备好并将发布在一个 URL 上（本案例的是 <https://sarah-wecan.github.io/>）。访问该 URL，可以看到一个简单的页面，页面内容是 README.md 中的内容。若显示 404 了，可以稍等片刻后刷新页面。GitHub Pages 创建网站需要一点时间。

(5) 在设置页面的 GitHub Pages 部分，单击“Choose a theme”然后选择一个主题；

如果已选过主题而想更换，该按钮会显示为“Change theme”。在选择主题后，页面会被转跳至一个或两个页面，这取决于前一次选了哪个主题。

被转跳至设置页面的情况下：可以转跳回到代码页面，在代码页面中会看见一个“_config.yml”文件，该文件描述了主题信息。该情况下，可以跳过步骤 6。

被带到 README.md 的编辑页面的情况下：主题信息已经被自动添加到 README 中。该情况下，需要做步骤 6。

两种情况下，“_config.yml”都会被自动提交到仓库中，该提交会触发 GitHub Pages 根据主题重新生成网站。

(6) 提交 README.md 文件；

README.md 被自动修改，现已包含了 Jekyll 模板信息。将页面滚动到底部，为提交信息添加一个标题，并选择“Create a new branch for this commit and start a Pull Request”。然后单击“Propose file change”。页面刷新后来到了 Pull Request 创建页面。创建 Pull Request 后可以合并它。最后可以转跳回到代码页面，这里便能看到步骤 5 中描述的“_config.yml”文件了。

(7) 在创建 Pull Request 时取一个更具体的标题;

创建 Pull Request 的流程在第三章中已经给出。

修改 Pull Request 的标题, 尽量反映出希望在本次网站迭代中做的所有事情, 并添加一段描述。比如, 可以在描述中添加如下列表:

```
To create a basic website:
```

```
- [ ] Add a theme
```

```
- [ ] Add an index.md file with "Hello World" on it
```

(8) 将该 Pull Request 与项目看板关联并创建该 Pull Request;

为了确保项目看板能够自动跟踪进展, 在 Pull Request 页面右侧的“Project”部分选择在步骤 3 中创建的项目看板。然后单击“Create Pull Request”。

(9) 在合适时更新 Pull Request;

提示: Pull Request 的描述部分不是固定不变的。当刚刚创建一个 Pull Request 时, 我们可能会给出一个列表, 上面是希望在此 Pull Request 并入主分支前要做的事情; 也可能一步到位把所有事情做好了。请每次认真审阅 Pull Request 的描述, 确保它与进展是同步的。举个例子, 如果我们已确切地遵照上述这些步骤, 那么现在就已经选好一个主题了, 所以我们可以勾选掉第一个框“Add a theme”。

(10) 验证项目看板的自动性;

单击仓库上方的 Project 标签页返回到项目看板, 会看到一个新卡片出现在了“progress”栏中。这个卡片上有一个代办列表, 正如在 Pull Request 中描述的那样, 并且列表上的两个事项中的一个已经完成。

(11) 切换到 Pull Request 分支;

返回到代码页面, 将分支切换到与 Pull Request 关联的那个。对本案例来说就是“sarah-wecan-patch-1”分支。

提示: 如果不知道如何在 GitHub.com 上切换分支, 请阅读第三章。

(12) 创建 index.md 文件;

在文件列表右上角, 单击“Create new file”按钮。填写文件名为“index.md”, 在文件中添加一个一级标题“Hello World!”:

```
# Hello World!
```

填写该次提交的标题和描述, 并在该分支中提交这个文件。

(13) 更新并合并 Pull Request;

返回到 Pull Request 页面, 然后勾选第二个框。此时计划所做的所有事情都已经完成, 该 PR 可以被合并。单击“Merge Pull Request”, “Confirm merge”, 然后“Delete branch”。

(14) 验证项目看板的自动性;

对话框中提示 Pull Request 已经从“In progress”栏移动到“Done”栏。可以回到项目看板验证。可以看到卡片被移到了“Done”栏, 并且两个框都被勾选。

(15) 验证网站是否发布。

前往 URL (该案例对应的是 <https://sarah-wecan.github.io>) 可以看到创建的个人网站正用之前所选的主题运行着。

提示: 如果主题未生效, 尝试刷新页面。现在, 我们便拥有了一个可以持续打磨定制的网站了, 可以通过向网站中添加内容来分享给世界。

4.4 为网站创建 Issue

当拥有一个 GitHub.com 网站仓库后（见上一节），可以考虑为网站添加小节。为所有网站中添加或修改的东西创建 issue 可以帮助我们计划和回忆任何修改的细节。

有时候收到一个很好的建议，并不想马上打开计算机开始修改时可以随手打开 GitHub.com，为这个建议创建一个 issue，提醒自己之后处理它；有时编辑网站时，发现一些需要修改的地方也可以创建一个 issue，这样就不需要停下手中的工作去开启一个新任务，而是快速创建一个 issue 等着之后去处理。

为了学习这个在计划阶段使用的 GitHub 特性，前往 Issues 标签页并单击“New issue”，体验 issue 的使用。

在本案例中，一共创建了 4 个 issue。

（1）修改标题和标语。当前网站的标题和标语是自动生成的。若想把标题改为自己的名字，并且选用更能表达自己的标语。创建一个 issue，将 issue 指派给自己，并将它和项目看板关联：

Issue Title: Change the title and tagline

Issue Description: Make the title and tagline something unique to you.

（2）为网站划分小节。不用离开当前的 issue 页面，就可以通过单击“New issue”来创建另一个 issue。该 issue 用于为网站添加小节，将它指派给自己，并关联到项目看板。在本案例中，计划在网站中加入三个小节：

Issue Title: Add a couple of sections to the website

Issue Description: Add three sections to the website:

- [] About Me

- [] Contact Information

- [] Blog

（3）写一篇博客。计划向网站添加的三个小节中的其中一个是博客，创建一个 issue 来提醒自己吧，将 issue 指派给自己，并将其关联到项目看板：

Issue Title: Write a blog post about GitHub

Issue Description: With a blog section ready to go, add a blog post that talks about your interactions with GitHub so far.

（4）链接项目仓库。正如本章先前所讲，可以把项目仓库变成网站。那么为什么不把这个网站链接到个人网站中呢？为此创建一个 issue，将其指派给自己，并关联到项目看板。之后，就可以把自己参与贡献的或感兴趣的项目链接到个人网站中：

Issue Title: Link to the Project Repo

Issue Description: I've turned <http://GitHub.com/sarah-wecan>HelloWorld> into a website and I want to link that website from this one.

如图 4-3 所示，个人网站仓库的 Issues 标签页下有 4 个 issue。它们同时也出现在项目看板的“To do”栏中（见图 4-4）。

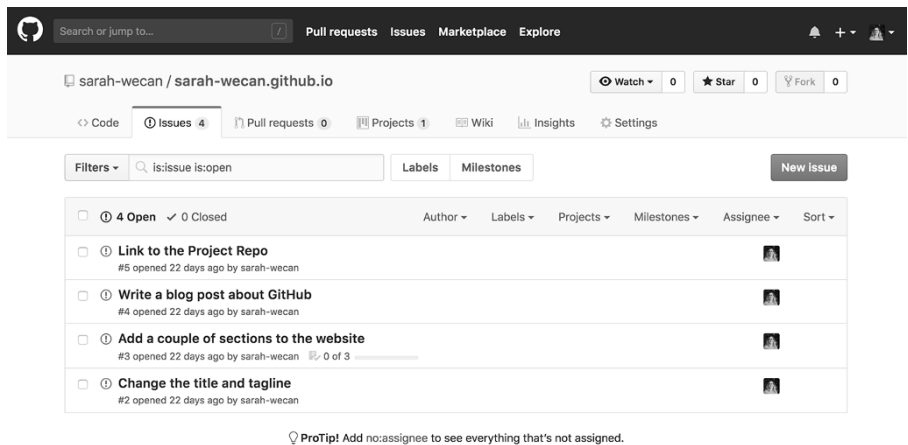


图 4-3 个人网站仓库的 issue 列表

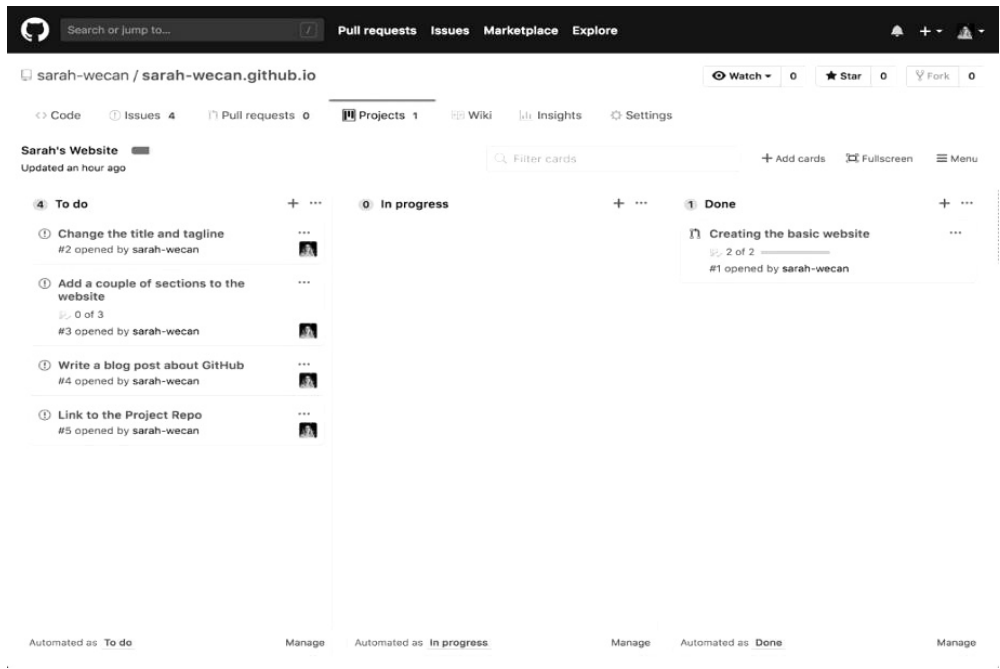


图 4-4 个人网站仓库的项目看板

4.5 设置本地环境

本小节的前提是已经安装了 GitHub Desktop 和 Atom。若还没有安装，请参照第二章内容做好准备。

在本节中，将介绍在本地而不是在 GitHub.com 上编辑个人网站仓库的文件。
提示：当计算机无法联网或者需要浏览许多文件时，在本地编辑文件是很方便的。

4.5.1 在 GitHub Desktop 中克隆仓库

在本地编辑文件的第一步是将网站仓库克隆到计算机中。

(1) 打开 GitHub Desktop, 在菜单栏中选择“File” --> “Clone a Repository”;

可以看到带有三个标签页的会话框: GitHub.com、Enterprise 和 URL。

提示: 如果已安装了 GitHub Desktop, 另一个克隆仓库的好方法是在仓库主页单击“Clone or Download”按钮。当单击按钮后, 展开的小窗中包含了一个“Open in Desktop”按钮。单击此按钮会打开 GitHub Desktop 并将仓库克隆到本机中。

(2) 在克隆仓库会话框的 GitHub.com 标签页中, 显示创建者的所有线上仓库。

警告: 如果在 GitHub.com 上的仓库没有自动出现在这里, 这可能是由于登录失效。可以在菜单栏中找到账户设置选项, 然后登录。如果已登录了仍看不到线上的仓库, 请尝试重新登录。

(3) 选择个人网站仓库, 并选择它在本地的存储位置; 可以使用默认路径来存放该仓库。

(4) 单击“Clone”。刷新 GitHub Desktop 后, 可以看到仓库信息。

4.5.2 在 GitHub Desktop 中游览

GitHub Desktop 提供了许多功能特性, 对我们的日常开发与 GitHub.com 的交互都特别有用。如果需要更多关于 GitHub Desktop 的技术支持, 请访问: <https://help.github.com/desktop>。图 4-5 聚焦了这些技术支持中最核心的六个功能:

(1) 仓库列表: 随着克隆到本地电脑中的仓库增多, 可以单击“Current Repository”下拉列表查看所有已克隆到本地的仓库, 单击任意仓库即可快速切换, 单击“Add”按钮可以快速添加一个仓库。

(2) 分支列表: 通过分支列表可以浏览所有本地分支, 并且提供了一个新建分支按钮。

(3) Pull Request 列表: 和分支列表共用一个下拉列表, 位于该列表第二个标签页, 显示了所有处于开启状态的 Pull Request。

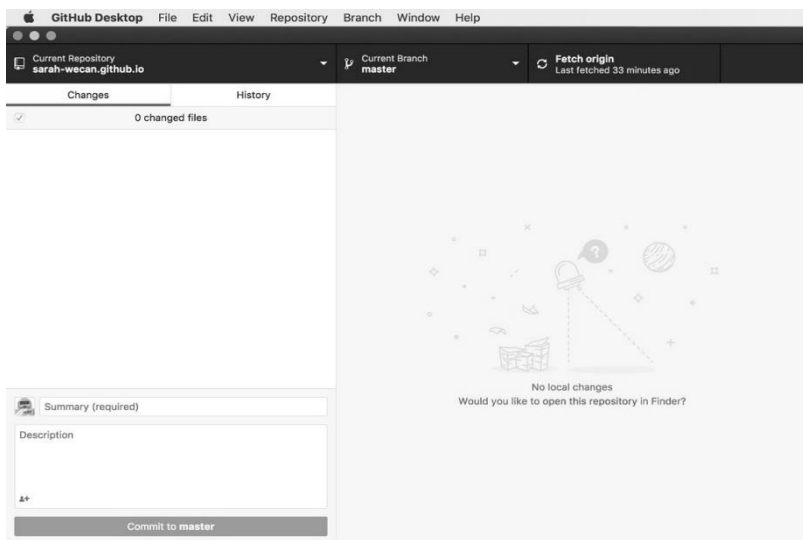


图 4-5 GitHub Desktop 概览

(4) 同步按钮: 当本地作出改动或者远程仓库发生变化时, 需要同步。由于当前(截图

时) GitHub Desktop 并没有在本地或 GitHub.com 上检测到任何改动, 所以可以尝试“fetch”作为一个同步的开始。如果在本地做了改动, 可以选择将本地变动 push 到 GitHub.com; 如果在 GitHub.com 上作出改动, 可以选择将改动 pull 到本地; 如果在本地创建了一个仓库而它不在 GitHub.com 上, 那么可以选择将其上传到 GitHub.com 中。

警告: 如果不把改动 push 到 GitHub.com 上, 别人是无法获取这些改动的。而且这种情况下如果电脑崩溃了, 将丢失所有成果。强烈建议要经常 push 自己的本地代码;

(5) 改动列表: 当开始改动代码时, 任何添加、删除或修改的文件都会出现在改动列表中。可以单击其中的文件在右方查看行级差异。当准备好提交这些改动时, 可以填写“Summary”和“Description”然后单击“Commit to master”按钮。然后, 就能用同步按钮将改动 push 到远程仓库。

警告: 在提交并 push 前, 应该仔细确认当前所处的是哪个分支。虽然可以撤销提交和 push, 尽量还是要避免类似失误。

(6) 历史列表: 在改动列表的右边是仓库的提交历史。提交历史包括了在本地做的改动以及来自 GitHub.com 的那些从未在本地做过的改动。当单击其中一个事件活动, 会在这里看到该活动包含的文件变动情况。

4.5.3 在 Atom 中打开仓库

在本地编辑文件, 可供选择的编辑器有很多, 包括 Atom、Visual Studio Code 或 TextEdit。本书中使用 Atom 编辑器。

在 Atom 中打开仓库步骤如下。

- (1) 打开 Atom; 可以看到一个空白窗口。
- (2) 在顶部菜单栏中选择“File” --> “Add Project Folder”; 出现目录选择窗口。
- (3) 在目录选择窗口中, 找到仓库文件目录, 然后单击“Open”。项目便在 Atom 中打开了。

4.5.4 在 Atom 中游览

Atom 本质上是一个代码编辑器, 但是它的一些特性使它特别适合在 GitHub 仓库上工作。图 4-6 显示了其 Atom 最核心的 6 个特性。

(1) 文件列表: 界面左侧的列表给出了仓库中的所有文件。单击文件名, 可以在编辑区域打开该文件。

(2) 代码编辑器: 位于文件列表的右侧, 可在此处编辑代码。

(3) 分支列表: 在 Atom 窗口右下角有一个分支选择器。目前处于 master 分支。单击分支名称会弹出一个菜单供人们在分支间选择或创建一个新分支。

(4) 同步按钮: Atom 支持与 GitHub.com 上的项目仓库同步。

(5) GitHub 面板开关: 单击 GitHub 按钮, GitHub 面板会出现在窗口中。该面板支持与 Pull Request 相关的操作。

(6) Git 面板开关: 如果单击 Git 按钮, Git 面板将作为挨着 GitHub 面板的第二个标签页出现在窗口中。该面板支持 stage、commit 以及查看仓库提交历史。

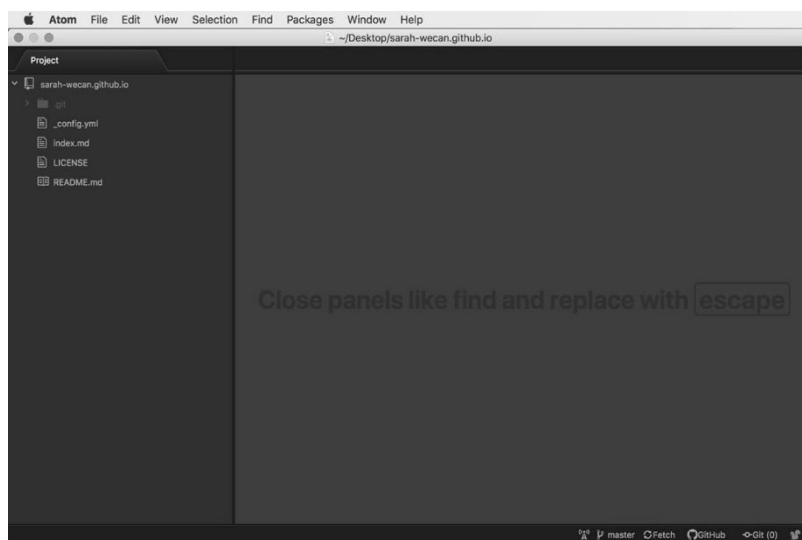


图 4-6 Atom 概览

4.6 寻找 GitHub Pages 资源

在 GitHub 上，有一群人不遗余力地帮助 GitHub 用户探索和学习 GitHub 提供的一切功能特性。除了静态的文档外，GitHub Training Team 还提供了 Learning Labs。

Learning Labs 是一个自我向导的、自动化的教程，它引导读者在 GitHub.com 上作出实际行动，而不仅仅是观看一部视频教程或是阅读一篇文字教程。Learning Labs 上有与 GitHub Pages、GitHub 基础、Markdown、HTML 以及如何运营一个开源社区相关的教程。

第 5 章 使用 GitHub Pages 打造网站

本章学习重点：

- (1) 使用现有的项目定位
- (2) 为已有项目贡献
- (3) 给网站添加标题和小节
- (4) 使用 GitHub Pages 创建博客

在这一章中，将要学习如何高效地跟进一个现有项目，以及使用 GitHub Pages 打造网站的一些细节。本章假设用户已经拥有了一个 GitHub Pages 仓库，如果还没有且希望在该方面得到指导，请阅读第 4 章。

用户可以通读本章，学习如何在 GitHub Pages 上打造一个简易网站，或根据需要跳到特定小节学习某个方法。

5.1 投身到现有的 GitHub 仓库中

用户无论是再次访问一个刚启动的项目还是一个去年参与过的项目，又或是发现了一个从未参与过的项目，都能通过一些简单的方法来快速跟进。此节以第 3 章中创建的 GitHub Pages 网站仓库为例，示范如何跟进一个 GitHub 项目，其技巧适用于任何 GitHub 仓库。

在进入正题前，请确保已经在浏览器中打开并登入了 GitHub.com。如果需要创建一个 GitHub.com 账户的话，请阅读第 1 章。

5.1.1 访问 GitHub.com 仓库

在 GitHub.com 主页左侧有一个仓库列表，列表中是用户最近打开贡献或创建的仓库。仓库列表是会不断增长的，尤其当处于一个大型组织中时。列表上方有一个用于搜索仓库的搜索框，它在仓库数目很多时非常有用。

在主页顶部另有一个搜索框，在这里可以搜索仓库、项目板以及团队（团队是组织中的功能，本书不做介绍）。该搜索框的默认搜索范围视现在所处的位置而定，当前页面为 GitHub 主页，搜索范围是整个 GitHub；若为一个仓库，搜索范围就仅限于该仓库。

不过，无论当前页面在哪，顶部搜索框永远提供了一个全站搜索选项。图 5-1 显示了三种查找特定仓库的方式和两种寻找新仓库的方式。

在找到目标仓库后，单击即可进入仓库首页。

提示：本章给出的例子都与 GitHub Pages 网站仓库有关，进入名为“your-username.github.io”的仓库就能顺畅阅读本章内容。本案例中使用的仓库名为“sarah-wecan.github.io”。

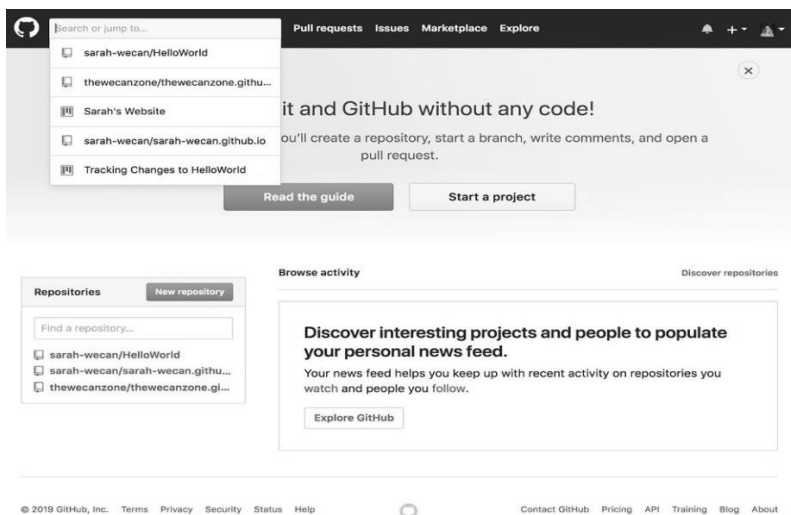


图 5-1 “GitHub.com 首页中查找仓库的地方”

5.1.2 验证在仓库中的权限

用户如果是仓库拥有者或管理员，可以在仓库首页顶部看到一个设置选项卡，对仓库有完全控制权，例如：邀请协作者；改变仓库的可见性（公开/私有）；

删除仓库；归档仓库。

用户如果没有看到设置选项卡，表明不是仓库的拥有者或管理员，但拥有写入仓库的权限。为了验证自己是否具有写入权限，用户可以尝试编辑一个文件，若能够对文件进行修改，且能看到一个如图 5-2 所示的提交框，则说明是项目的协作者，拥有写入权限。

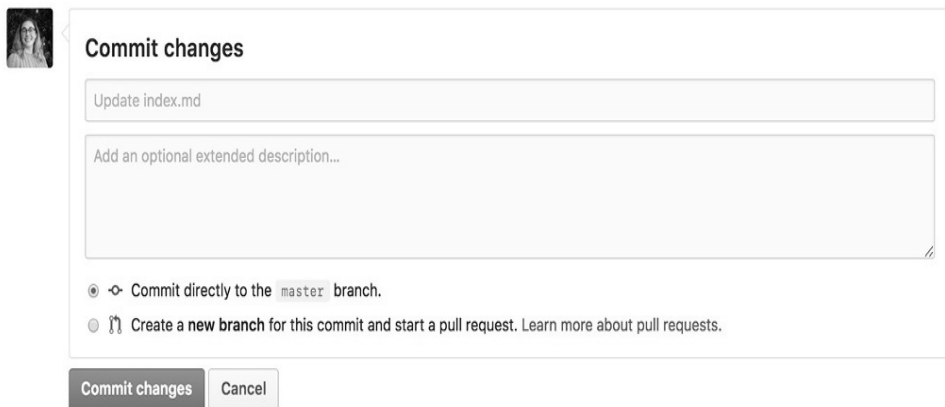


图 5-2 “当用户对仓库拥有写入权限时的提交框”

当用户尝试修改一个文件，看到一则提交框样式如图 5-3 的警告时，说明对该仓库没有写入权限，即该用户不是仓库的协作者。不过仍然可以创建 Issue 或提议修改文件，但需要通过权限更高的用户的同意。如果对如何在没有写入权限的仓库中做贡献感兴趣，请阅读第 6 章。



图 5-3 “当用户对仓库拥有只读权限时的警告和提交框”

5.1.3 跟进项目

无论是作为一个仓库的拥有者、贡献者还是初来乍到者，在投入工作前都需要先跟进项目，以防止上次访问仓库以来有新的 Issue、PR 更新产生。如果用户是一个私有仓库的唯一贡献者，在开始工作前检查一下代办事项，并根据时间合理安排工作。若仅有几个小时，那么开发一个全新的功能不太现实，定位一个 bug 或是做一些小修小改就比较合适。

用户可以通过检查仓库中的四处地方来跟进项目：

(1) 阅读 README.md、CONTRIBUTING.md 和 CODE_OF_CONDUCT.md。除非是在自己的项目里工作，否则需要先阅读上面这三个文件，以确保知道如何在本地配置项目开发环境，如何高效地贡献到项目中，以及如何与其他协作者沟通。并不是所有项目都会提供这三个文件，但只要有就应当先阅读，以确保在项目社区里是一个做出有效贡献的成员。在 GitHub Desktop 的开源项目仓库 (<https://github.com/desktop/desktop>) 中可以找到这些文档的范例。

(2) 调查项目看板。如果项目使用了项目看板应该关注。单击仓库上方的 “Projects tab”，然后选择一个看板。一般情况下，每个看板都有一个用于追踪进展中的或代办的事项的栏目。如果项目看板启用了自动化，那么任何有关 Issue 或 PR（包括新建的）的变化都会反映在项目看板中。本书所举例子，项目看板 “Sarah's Website project board” 下的 “To do” 栏目中有 4 个 Issues。

(3) 通读 Issue。活跃的项目里人们更乐意提出新 Issue。单击仓库上方的 “Issues” 选

项卡查看 Issue 列表。可以按照“most recently updated”对列表进行排序，以查看人们在 Issue 中的最新评论。Issue 默认按照“newest”（意味着最新开启）来排序。如果作为仓库的所有者，请为新开启的或更新后的 Issue 分类（阅读后文中“为 Issue 分类”了解为什么以及如何为 Issue 分类）。本书案例仓库中提出了四个 Issue 用于追踪网站的开发进展。

（4）检查 PR。无论是被动还是主动，在着手工作前检查项目的 PR 是明智之举，因为 PR 代表了那些尚未合入主分支的新增、删去或改动的代码。被动检查 PR 即通读最近发起或更新的 PR，从中了解哪些贡献正在等待被合入项目中。检查 PR 能确保不会重复他人的工作，以及避免和他人已有工作发生冲突。如果在 PR 中发现了问题，可以在讨论区留言。

如果用户在项目中拥有相应权限（是否有信心评估 PR 是否应该被合并比较重要），也可以主动检查 PR。如图 5-4 所示，打开一个 PR 后，单击“Files changed”，再单击右上角的“Review changes”按钮，即开始主动检查 PR。在本案例中，仓库中不存在开启中的 PR，只有一个已经关闭了的 PR（初始化网站时的 PR）。

提示：第十一章介绍了检查 PR 的经验。如果想在检查他人项目的 PR 前，先以一种互动的形式学习检查自己项目的 PR，可以通过一个专门为本书设计的仓库来练习如何检查一个 PR，地址是（<https://github.com/thewecanzone/GitHubForDummiesReaders/pull/2>）。

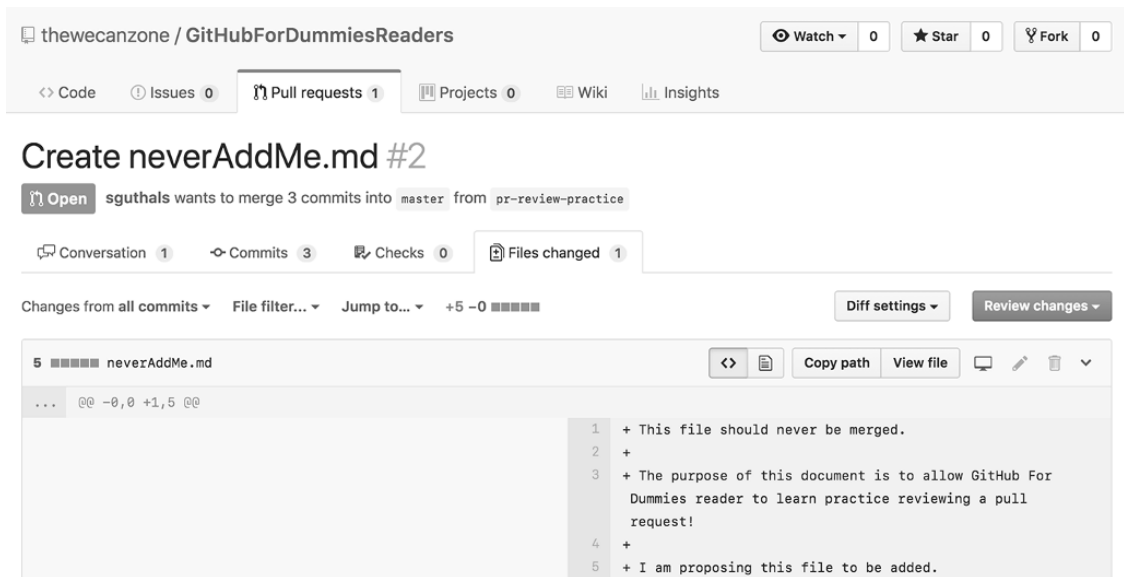


图 5-4 “检查 PR 的地方”

5.1.4 为 issue 分类

如果用户是仓库的所有者，可以每天先花费 30 分钟整理 issue：

（1）添加标签。issue 和 PR 都能被贴上标签。必要的标签如下：

- bug：这个 issue 报告了似乎出问题的地方，需要马上处理。
- good first issue：新贡献者可以从这个 issue 起步。
- help wanted：代码所有者通过该 issue 向社区求助。
- needs investigation：在得出解决方案前需要回答的问题。
- feature/enhancement：该 issue 提出一个新特性或对项目的改进。

一些标签会被默认添加到仓库中，包括：bug、duplicate、enhancement、good first issue、help wanted、invalid、question 和 wontfix。用户可以移除不需要的标签，也可以添加其他标签。

(2) 回复评论。自用户上一次访问后，一些 issues 可能会有新评论。例如，若某人先前提出了一个 issue，然后贴上了“needs investigation”标签来向该用户询问额外的信息，检查 issue 查看是否有后续补充。应该保持 issues 活跃，一旦僵化（人们不再留言或不再有进展），也许应该关闭它们。

(3) 关闭僵化的 issue。任何僵化的 issues 都应该被关闭。距离 issue 提出者最近一次留言的数周后或不打算实现 issue 所描述的功能都是关闭 issue 的合适时机。关闭一个 issue 前用户可以先留言通知将要关闭该 issue，该留言在日后还能帮助用户回忆为何会关闭该 issue。

5.2 准备进行贡献

用户在跟进项目后，需要决定做什么来参与贡献，本书案例中选择项目看板“To do”栏中的 issue#2：更换标题和标语。

提示：其他合适的候选 issues 是那些处于开启状态中，且标签为 bug、help wanted 或 good first issue 的 issues。对提出者来说这些 issues 通常都是十分紧急的或是一个参与贡献的良好起点。

此节假设用户已经将仓库克隆到本地且案例中使用 GitHub Desktop 和 Atom 来解决 issues。如果需要有关如何克隆一个仓库或如何在 GitHub Desktop 和 Atom 中配置项目的帮助，请阅读第四章。

5.2.1 为自身贡献创建一个分支

任何项目，无论是个人私有项目还是大型开源项目，修改或增加代码前强烈建议创建一个分支来容纳所有的改动。分支是 Git 提供的功能，它能拷贝一份代码（每个分支都是一份拷贝），允许用户在该份拷贝上修改，最后将改动合并回主分支。

创建一个分支时，Git 其实不会一模一样地耗时低效地拷贝一份代码。Git 的真正做法充满了智慧，相关的原理细节对用户使用 Git 而言不大重要，在这里就直接称其为“拷贝”。虽然该用语在技术上不太准确，概念上是正确的，能帮助初学者理解分支的作用。

创建分支的好处是让设置一个安全的空间用以试错，以及尝试不同的解决方案，且不必担心威胁到项目的完整性。如果解决方案出现错误，用户只需要删掉这个分支，再创建一个新分支重新开始。创建分支可以保证用户的主分支时刻处于正常运行状态，因为在没有把握新分支切实解决了问题且不会引入新的麻烦之前，是不会将其合并到主分支的。

不同的项目和不同的人采用不同的分支命名法，本书中使用对修改内容的简短描述来命名分支。Atom 是 GitHub 上的开源项目，该项目的贡献者命名分支的习惯是在简短的描述前再加上姓名首字母缩写。可以在 Atom 的 GitHub 仓库中（<https://github.com/atom/atom/branches>）观察到这种命名法。在公开仓库中创建分支前，先看一下别的贡献者是怎么命名分支的，或参考 CONTRIBUTING.md 是否提及命名规范。

记住：本书使用 GitHub Desktop 来管理本地的项目，使用 Atom 作为主要的文本编辑器。如果用户需要安装这两个应用，请阅读第四章。

跟随以下步骤为用户的贡献创建分支：

(1) 确认克隆的仓库。

打开 GitHub Desktop，确认在左上角的下拉列表中被选中的是用户的个人网站仓库。

图 5-5 显示仓库处于正确选中状态。

(2) 创建新分支。

在 GitHub Desktop 的中上方位置单击分支下拉列表，然后单击“New Branch”按钮（见图 5-5）。单击该按钮后，一个会话框出现。

(3) 命名新分支。

输入分支名。在案例中，使用“new-title-and-tagline”作为分支名，因为这描述了计划的修改。

(4) 单击“Create Branch”。

会话框关闭，此时已经切换到新分支上。图 5-5 中显示的分支为 master 分支，但是当完成此步骤后，会自动变更为新建的分支。GitHub Desktop 右上方的同步按钮也会从“Fetch origin”变更为“Publish branch”。

(5) 发布用户的分支。

提示：推荐在对分支做任何修改前将分支发布到 GitHub.com 上，这样在分支中修改或添加的代码将不会丢失。单击界面右上角的“Publish branch”按钮。当分支成功发布后，这个按钮会重新显示为“Fetch origin”，如图 5-5 所示。

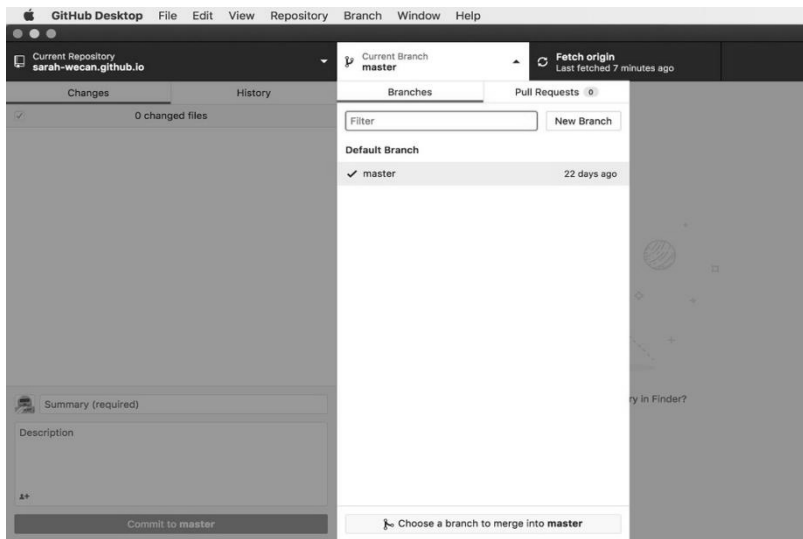


图 5-5 使用 GitHub Desktop 在指定仓库中创建新分支

(6) 用户在 Atom 中打开项目。

打开 Atom 并打开 Project 文件夹。界面左侧是树状文件列表。底部的分支选择器显示当前的分支名。图 5-6 显示了目标分支已在 Atom 中选出。

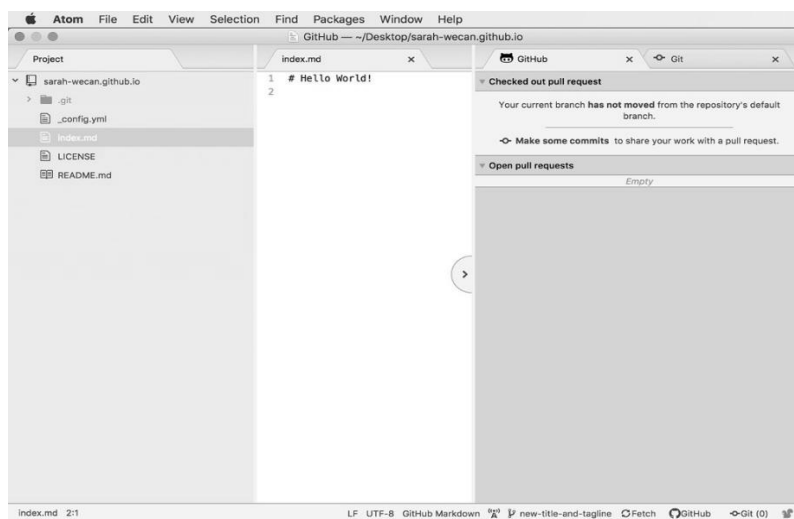


图 5-6 “Atom 的分支选择器显示目标分支已被选出”

提示：如果在 Atom 的分支选择器中切换了分支，那么在 GitHub Desktop 中的分支同样会切换，反之亦然！

5.2.2 确认分支已发布

在编码前，用户应该确认已发布分支并能推送代码到该分支。如果仍在同一台电脑的同一个项目上工作，一切配置都应该照常，但若首次向一个项目提交贡献或刚设置了一台新电脑，就需要确认一下一切配置是否妥当。

接下来的案例将展示如何利用 GitHub Desktop 和 Atom 贡献到 GitHub 网站仓库 (<https://github.com/sarah-wecan/sarah-wecan.github.io>) 中。

总共分为 6 步：

(1) 在 Atom 的树状文件列表中单击一个文件。

文件将在编辑区域打开。图 5-7 显示了打开的 index.md。

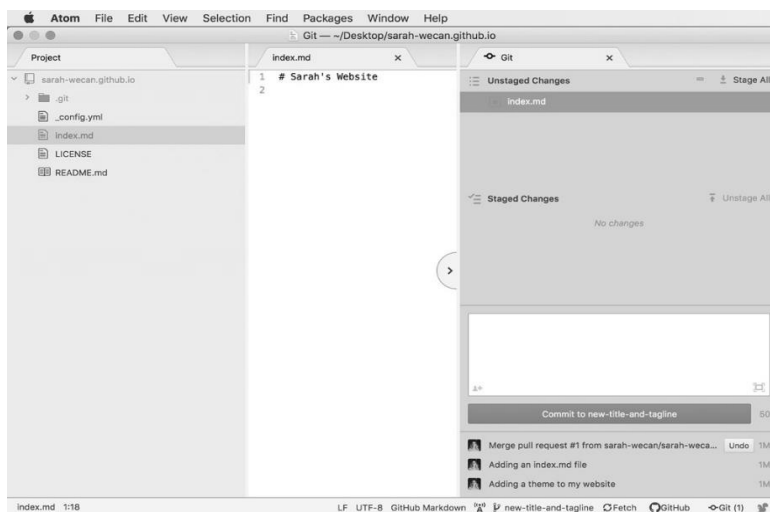


图 5-7 “在 Atom 中编辑文件”

(2) 修改文件。

在这一阶段已经没必要做太多的修改，因为目的是确认能够把所做的更改推送到已经发布的分支中。

(3) 保存文件。

选择“File” --> “Save”以保存修改。单击“Save”后，文件列表中的文件颜色会发生变化，底部的 Git 按钮也会变化以指示有更改发生（见图 5-7）。

(4) 单击 Git 按钮打开 Git 面板。

在“unstaged”区域将会显示更改。

(5) 暂存并提交更改。

单击“Stage all”按钮暂存所有更改，并填写 commit 信息，然后单击“Commit to xxx”。图 5-8 显示了这一步骤。单击“commit”按钮后，刚刚的 commit 将出现在按钮下的 commit 列表中，旁边还有一个撤销按钮。

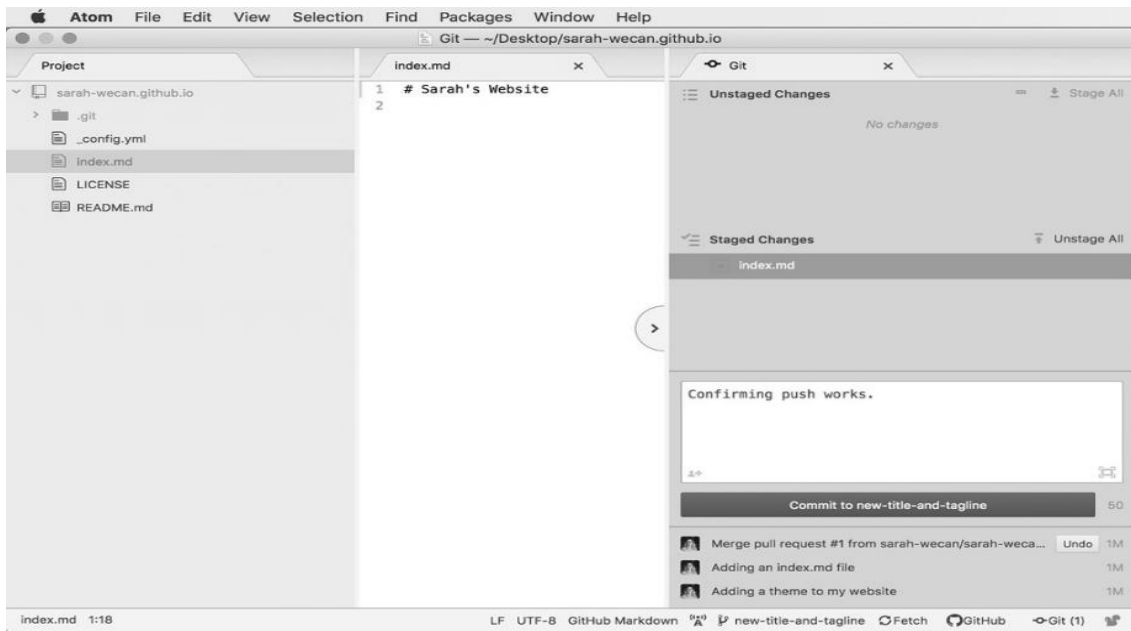


图 5-8 “在 Atom 中暂存和提交更改”

(6) 推送合并并更改。

Atom 底部的 Sync 按钮会根据仓库状态改变触发动作。在提交被创建前，该按钮显示为“Fetch”（见图 5-8）。在创建提交后，该按钮会显示为“Push 1”，这意味着用户有 1 个 commit 可以推送。单击“Push 1”把更改推送至 GitHub。然后打开 GitHub.com 并前往仓库。案例中的仓库地址为(<https://github.com/sarah-wecan/sarah-wecan.github.io>)。仓库中将显示一则 Pull request，提示可以发起一个 PR（见图 5-9）。

- 接下来请创建并合并 PR。如果需要额外帮助，如何合并一个 PR 可参看第 3 章，可以用 PR 做什么可参看第 8 章。
- 有时，PR 会产生冲突。阅读“解决合并时的冲突”以了解如何解决它们。

- 检测到新分支

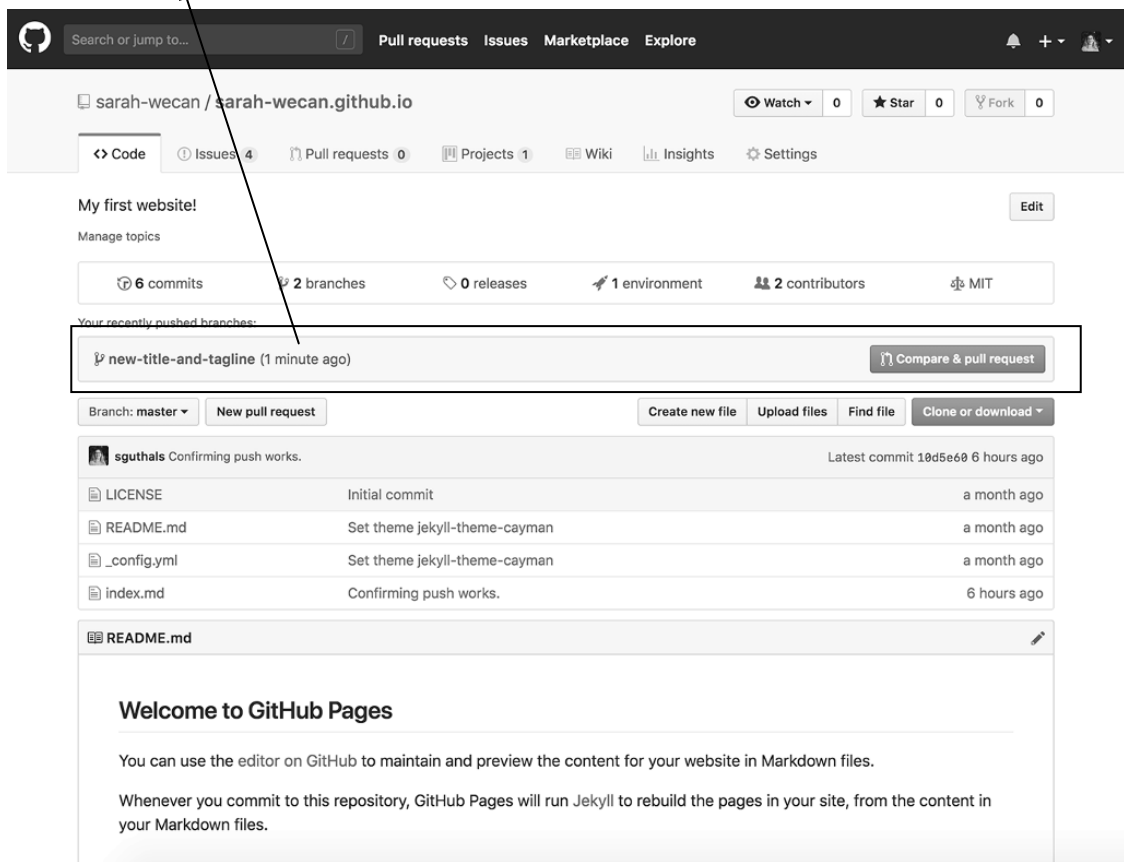


图 5-9 “GitHub.com 提示一个分支已准备好被合并”

5.2.3 解决合并时的冲突

有时分支会处于一个 GitHub.com 无法决定怎样合适地将其合并的状态。这种情况有时会在一系列复杂的事件后发生。用户基于主分支创建了一个新分支并修改时，如果还有其他用户也在该主分支上修改，则需将修改暂存、提交并推送到自己的新分支中。如果没有先把主分支上的更改拉取到自己的分支中，那么所做的更改就可能会和他用户的更改产生冲突。例如，两人同时修改了同一行内容，但修改的起点位置却不一样；

假设用户创建分支时，某行是

```
Hello World!!!
```

然后将它改成

```
Sarah's Website
```

但是其他用户在主分支上却将该行改为

```
Hello Friend!
```

这时若用户发起 PR 时，GitHub 会感到困惑：它不清楚到底要把“Hello World!!!”改成什么。GitHub 并不会做出选择，相反它会询问用户希望的做法。在创建一个会制造冲突的 PR 后，不会出现合并 PR 的按钮，取而代之的是一个提示解决冲突的按钮。如果单击该按钮

会打开一个新页面，该页面会引导用户进行选择（见下图）：

- 个人分支中的代码（上半部分）
- 主分支中的代码（下半部分）

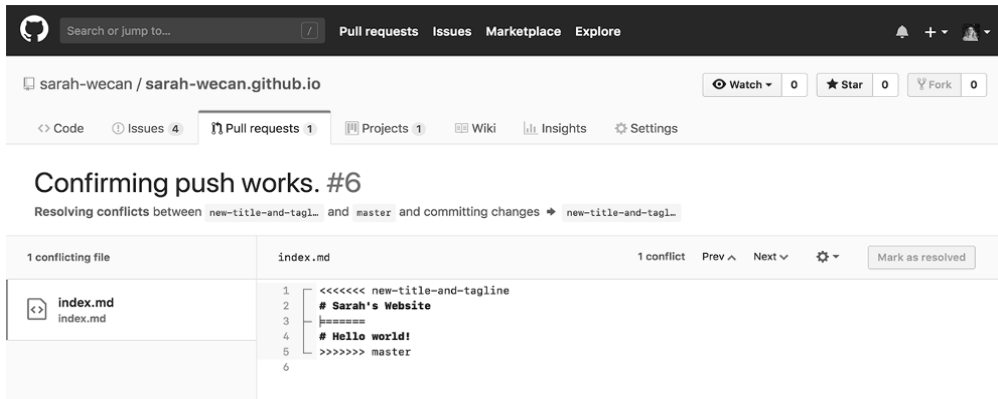


图 5-10 冲突后的选择

可以通过删除分隔符，保留希望留下的代码。在本案例中，最后留下的代码是
Sarah's Website

之后，通过单击“Mark as resolved”将变化提交到个人分支。

5.3 打造个人网站

本节内容承接第 4 章中创建的个人网站仓库，展示一些打造个人网站的基本实践。本节中提到的技巧对任何由 GitHub Pages 生成的网站都是适用的。

5.3.1 修改标题和标语

若想修改网站的标题和标语的话，请在 Atom 中打开 `_config.yml` 文件。在该文件唯一的一行代码（即配置主题的代码行）之上添加两行内容：

```
title: <Your Name>
description: <Your Description>
theme: jekyll-theme-cayman
```

5.3.2 为网站添加小节

利用 Jekyll 和 GitHub Pages 向网站中添加小节十分容易，因为可以使用 Markdown 和 HTML。首先，添加一些社交媒体用户名。用户打开 `config.yml` 文件，然后按照个人意愿添加一些社交媒体用户名。在案例中，添加了 Twitter 和 GitHub 的用户名：

```
title: Your Name
description: Your Description
twitter_username: Your Twitter username
github_username: Your GitHub username
theme: jekyll-theme-cayman
```

然后，打开 `index.md` 文件，将小节包含进来。案例中文件内容如下：

```
# My Projects
Here are a list of projects that I am have been working on:
# My Interests
I'm interested in teaching novice coders about computer science!
# My Blog
I'm really excited to blog my journey on GitHub.com.
# Get in Touch
<ul>
  <li>
    <a href="https://twitter.com/{{ site.twitter_username }}">Twitter</a>
  </li>
  <li>
    <a href="https://github.com/{{ site.github_username }}">GitHub</a>
  </li>
</ul>
```

保存、暂存、提交并推送这些变化到个人分支中，然后发起 PR 并将其合入主分支，数分钟后刷新个人网站页面就能看到变化。

5.3.3 创建一篇博客

在网站上写博客是一种很棒的分享 GitHub 之旅的方式。一旦开始探索与创造，用户就可以与社区中志同道合的伙伴分享自己的所学所创。这部分内容会指导如何创建一篇博客并将其链接到 `index.md` 文件中。

首先，创建一个名称为“`_layouts`”的新目录，然后在该目录下创建一个叫“`post.html`”的文件。在 `_layout/post.html` 文件中加入以下代码来创建一个博客风格的布局：

```
layout: default

<h1>{{ page.title }}</h1>
<p>{{ page.date | date_to_string }} - {{ page.author }}</p>

{{content}}
```

注意：请确保 `_layouts` 目录的名称无误，因为 Jekyll 通过搜索 `_layouts` 目录来获取用户自定义的布局，否则它会使用主题提供的默认布局。特定布局的名称（比如更改 `post.html` 的名称）可以更改，但是布局名称必须与使用该布局的文件中的布局元数据相匹配，正如接下来的代码片段所示。

使用布局和特定的命名规则，Jekyll 可以推断出标题、日期和作者，并将它们显示在博文或主页中。然后，创建一个名称为“`_posts`”的目录，并在其中创建一个名称为“`YEAR-MONTH-DAY-TITLE.md`”的文件。本案例创建了“`_posts/2019-01-01-new-year.md`”，文件内容为：

```
---
layout: post
```

```
author: sguthals
```

```
---
```

Write your blog post here.

最后，在 `index.md` 文件中，将以下内容添加到“`My Blog`”小节中：

```
<ul>
  {% for post in site.posts %}
    <li>
      <a href="{{ post.url }}">{{ post.title }}</a>
    </li>
  {% endfor %}
</ul>
```

保存、暂存、提交并推送这些变化到分支中，然后发起 PR 并将其合入主分支。数分钟后，刷新个人网站页面就能看到变化。

5.3.4 链接项目仓库

可以使用与“为用户网站添加小节”中描述的链接社交媒体相同的方法将 GitHub 项目仓库链接到自己的网站中。虽然直接添加仓库链接很高效，但还可以为项目仓库创建 GitHub Pages，就像第 4 章介绍的那样。

打开 `index.md` 文件，然后添加下方代码，将其中的链接替换为自己的项目链接。案例中的代码演示了将项目仓库网站和项目仓库本身链接到 `index.md` 中。

```
<ul>
  <li>
    <a href="https://sarah-wecan.github.io>HelloWorld/">Hello World Project </a>
  </li>
  <li>
    <a href="https://github.com/thewecanzone/GitHubForDummiesReaders">GitHub For Dummies Repo
</a>
  </li>
</ul>
```

提示：本书并未提及其他许多自定义 GitHub Pages 网站的方法。Jekyll 和 GitHub 一起为用户提供了一种独特的体验——只需少量代码就能实现复杂的配置。