

翻译: 李鸿斌  
内一校: 俞泽恺  
内二校: 苏斌

# 第3章 团队合作与协作开发

---

一个出色的团队不只是其成员的简单累加，打造人们喜爱的产品还需要一个高效的团队。

本章将学习如何建立团队并使用pull requests以达到高度协作开发的目的。了解什么是pull request以及哪些功能可以帮助团队获得优秀的代码评审工作流。

本章中核心主题包括：

- 软件开发是一项团队活动
- 协作的核心:pull request
- 动手实现:创建一个pull request
- 提交更改
- Pull request审查
- 动手实现: 提出建议
- 代码审查的最佳实践

## 软件开发是一项团队活动

设计师兼工程师彼得·斯基尔曼(Peter Skillman)做了一个实验:他让四人一组的成员在棉花糖挑战中相互竞争。挑战的规则很简单——用以下材料构建能支撑最高的棉花糖结构:

- 20块生意大利面
- 1个透明胶带
- 1根绳子
- 1个棉花糖

这个实验并不是针对问题本身，而是关于团队如何共同努力解决问题。在实验中，来自斯坦福大学和东京大学的商科学生团队与幼儿园的孩子们竞争。猜猜谁是赢家？

商科学生检查材料，讨论最佳策略，仔细挑选最有前途的想法。他们以专业理性和智慧的方式行事，但幼儿园的孩子们却是赢家。幼儿园的孩子们并没有决定最好的策略—他们只是尝试并开始试验。他们紧紧地站在一起，简短交流：“这里，不，是这里！”

幼儿园的孩子们并没有因为更聪明或更熟练而获胜。他们获胜是因为作为一个团队他们合作得更好(Coyle D. (2018))。

读者也可以在体育竞技中观察到同样的情况:可以把最好的球员放在队伍中，但如果他们不能组成一个良好的团队，就会输给一个技术不那么好但能完美合作的团队。

在软件工程中，企业需要具有高凝聚力的团队，不能是一起工作但互相独立的专家，而是像棉花糖实验中幼儿园的孩子们那样一起实践的团队。通过寻找E型团队成员取代T型团队成员的演变来做到这一点。I型人员在一个领域有很深的经验，但在其他领域的技能或经验很少。T型人员是在一个领域内拥有深厚经验的通才，并且还拥有跨越多个领域的技能。更高级的是E型人员 —— E表示经验 (Experience)、专长 (Expertise)、探索 (Exploration) 和执行 (Execution)。他们在多个领域拥有丰富的经验，具有成熟的执行技能，总是勇于创新，渴望学习新技能。E型人员是将不同领域的专业知识结合成一个高协作团队的最佳选择( Kim G .、Humble J .、Debois P .和Willis J.)。

通过观察一些pull requests，可以很快观察到团队是如何合作的。谁来审查代码，审查的主题是什么？人们在讨论什么问题？语气如何？如果读者曾经见过杰出团队的pull requests，就会很容易发现进展不顺利的事情。以下是一些读者很容易就能看到的pull request反面案例：

- Pull requests 数量过大，变化较多(批数量过大)。
- Pull requests 仅在某个功能已经完成或在冲刺的最后一天产生(最后一分钟通过)。
- Pull requests 通过但没有任何评论。这通常是因为人们只是为了同意而不干扰其他团队成员 (自动通过)。
- 评论很少包含问题。这通常意味着讨论的是不相关的细节，比如格式和样式，而不是架构设计问题。

之后会向读者阐述代码检查的最佳方式，以及如何避免这些反面事例。接下来先更进一步理解什么是pull request。

## 协作的核心 – pull request

一个 pull request 不仅仅是一项传统的代码审查。还能实现以下几点：

- 代码协作
- 分享知识
- 创建代码的共享所有权
- 跨团队协作

但pull request到底是什么？pull request (也称为merge request)是将来自其他分支的更改集成到Git代码仓库中的目标分支的过程。更改可以来自仓库的分支，也可以来自派生(仓库的副本)。Pull request通常缩写为PR。没有写入权限的人也可以派生仓库并创建PR。这使得开源仓库的所有者无需给每个人对仓库的写入访问权限就能参与贡献。这就是为什么在开源世界中，pull request是将更改集成到代码仓库中的默认方式。

Pull request还可以用于以一种称为内部源代码的开源风格进行跨团队协作(参见第5章，开放和内部源代码对软件交付性能的影响)。

### 关于 Git

Git是一个分布式版本控制系统(revision control system, RCS)。与中央RCS相比，每个开发者都将整个存储库存储在他们的机器上，并与其他存储库同步更改。Git基于一些简单的架构决策。每个版本都存储为整个文件，而不仅仅包括更改部分，并使用哈希算法跟踪更改。修订和文件系统存储为有向无环图(DAG)，该图使用父对象的散列进行链接。这使得分支和合并更改非常容易。这就是为什么Git声明了以下内容: git – the stupid content tracker (参见图3.1中的Git手册页)。

Git是Linus Torvalds在2005年为Linux内核创建的RCS。直到2005年，BitKeeper一直用于版本控制，但由于许可证的变更，BitKeeper不能再免费用于开源项目了。

Git是当今最流行的RCS，有很多关于Git的书(参见echacon S.和Straub B., 2014; Kaufmann M., 2021;等)。Git是GitHub的核心，但在本书中，GitHub作为一个DevOps平台，而不是RCS。

在第11章《基于主干的开发中》将讨论与工程速度相关的分支工作流，但不会深入讨论分支和合并。请参阅补充书目和参考文献部分来了解相关内容。

图3.1为Git的手册页：

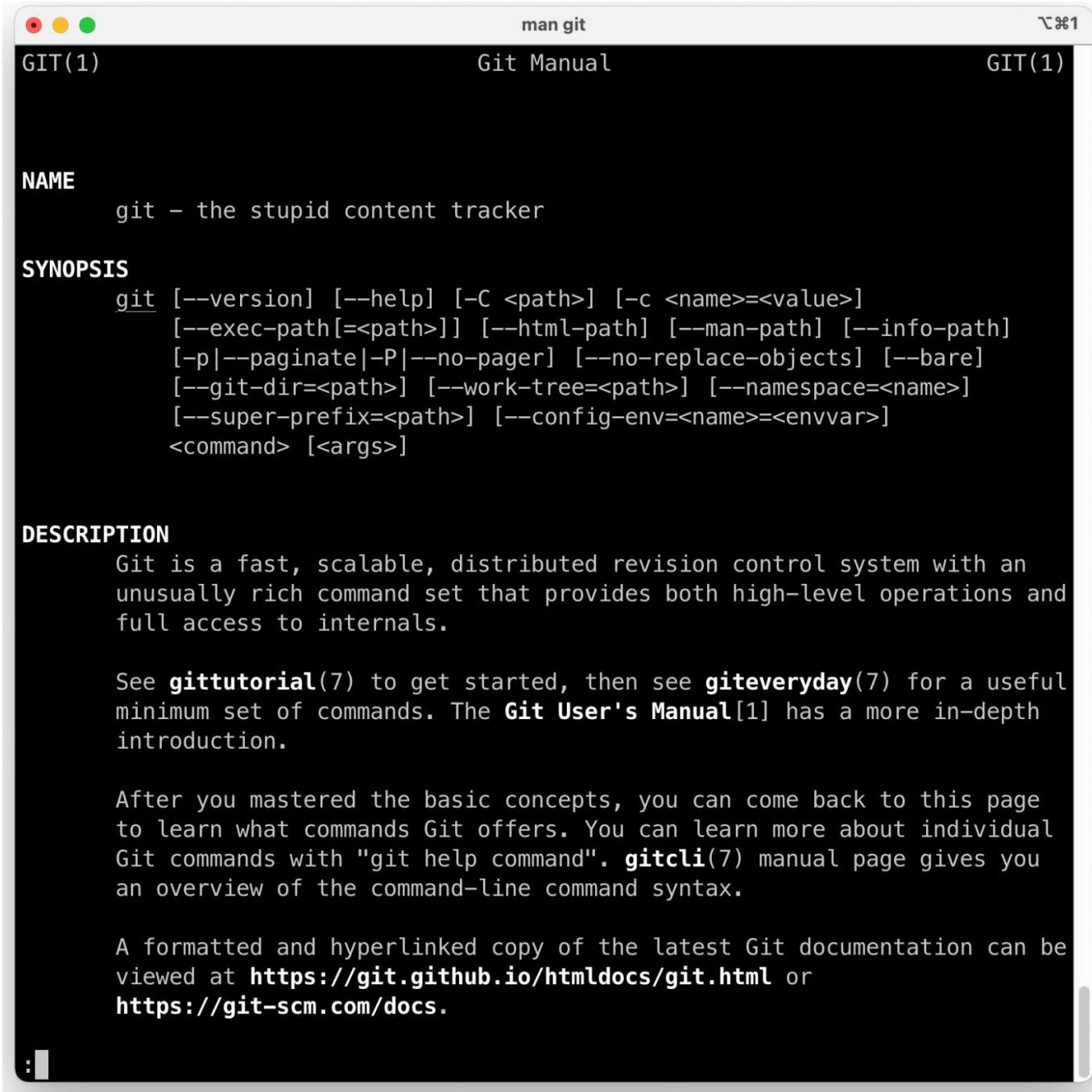


图3.1 – Git手册页 – the stupid content tracker

Git按行对文本文件进行版本管理。这意味着pull request关注已更改的行:可以添加、删除或同时添加和删除一行。在本例中，可以看到更改前后旧行和新行之间的差异。在合并之前，pull request允许用户做以下事情：

- 审查更改并对其进行评论
- 在源存储库中构建并测试更改和新代码，而不需要首先合并它。

只有当更改通过所有审查时，它们才会被pull request自动合并。

在现代软件工程中一切都是代码，它不仅仅是关于源代码。读者还可以在以下方面进行合作：

- 架构、设计和概念文档
- 源代码
- 测试
- 基础设施(以代码形式)

- 配置(以代码形式)
- 文档

任何事情都可以在文本文件中完成。在前一章已经讨论过将markdown作为人类可读文件的标准。它非常适合协作概念文档和文档。如果需要存档或发送给客户一份实际文件，还可以将markdown转变为可移植文档格式(PDF)。用户可以使用图表扩展markdown。例如，使用Mermaid(<https://mermaid-js.github.io/mermaid/>)。markdown是针对人类可读的文件，而YAML Ain't Markup Language (YAML)是针对机器可读的文件。因此，通过源代码、markdown和YAML的组合，可以自动创建开发生命周期的所有工件，并在更改上进行协作，就像在源代码上协作一样！

### 案例

在GitHub，所有事项基本上都是用markdown来处理，甚至法律团队和人力资源(HR)也使用markdown、issues和pull requests来协作。例如在招聘流程中：职位描述存储为markdown文件，并且使用issue跟踪完整的招聘流程。还例如GitHub网站政策(如服务条款或社区指南)，都是markdown格式，而且都是开源的(<https://github.com/github/site-policy>)。

如果读者想了解更多关于GitHub团队协作的信息，请参阅<https://youtu.be/HyvZO5vvOas?t=3189>。

## 动手实现:创建一个pull request

如果读者初次使用pull request，最好自己创建一个来体验它是什么。如果读者已经熟悉pull request，可以跳过这一部分，并继续阅读有关pull request特性的内容。请按以下步骤进行：

1. 打开以下相应的仓库，通过点击右上角的Fork创建一个仓库派生：<https://github.com/wulfland/AccelerateDevOps>。

在Fork中，导航到Chapter 3 | Create a pull request (ch3\_pullrequest / Create-PullRequest.md)。该文件还包含了指令，这样用户不需要在浏览器和本书之间来回切换。点击内建内容上方的Edit铅笔图标编辑该文件。

2. 删除文件中标记的行。
3. 添加几行随机文本。
4. 通过删除超过允许长度的字符来修改一行。
5. 提交更改，但不要直接提交到主分支。将它们提交到一个新的分支，如图3.2所示：

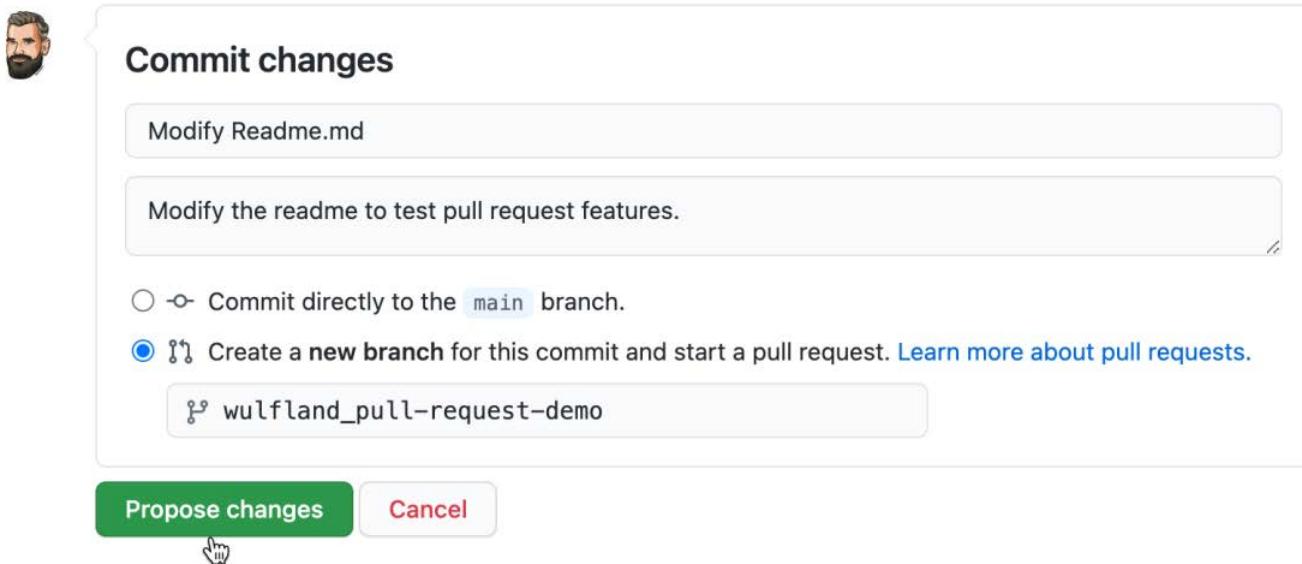


图3.2 – 将更改提交到新分支

6. 浏览器将自动重定向到一个可以创建pull request的页面。输入标题和说明。请注意，在第2章介绍的issue的markdown语法在这里也支持：表情符号（: +1:）、提及（@）、引用（#）、任务列表（- [ ]）和语法突出显示的源代码（```）。用户还可以指配工作负责人、标签、项目和里程碑。

在页面的顶部，可以看到目标分支（base）是main，而要集成的源分支是刚刚创建的分支。“Create pull request”按钮是一个下拉列表。用户还可以选择创建pull request草稿。现在通过单击“Create Pull Request”按钮创建一个Pull Request（参见图3.3）。

The change you just made was written to a new branch named `kaufm_pull-request`. Create a pull request below to propose these changes.

base: main ▾ ← compare: kaufm\_pull-request ✓ Able to merge. These branches can be automatically merged.

**Modify Readme.md**

Write Preview

Test pull request features.

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Reviewers  
No reviews

Assignees  
No one—assign yourself

Labels  
None yet

Projects  
None yet

Milestone  
No milestone

Helpful resources  
[GitHub Community Guidelines](#)

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

图3.3 - 为对文件所做的更改创建pull request

7. 在pull request中，单击“Files changed”并注意对文件所做的更改：删除的行是红色的，添加的行是绿色的，修改的行是删除的行后面跟着添加的行。如果将鼠标悬停在这些线上，左侧会出现一个加号+图标。单击该图标则可以添加单行注释。按住图标并拖动它，则可以为多行添加注释。同样，注释具有与issue相同的所有丰富特性标记支持！添加一条备注，点击Add single comment（见图3.4）：

## Modify Readme.md #1

The screenshot shows a GitHub pull request titled "Modify Readme.md #1". The top navigation bar includes "Edit" and "Open with" buttons. Below the title, a green "Open" button and a message "kaufm wants to merge 1 commit into main from kaufm\_pull-request" are visible. The main interface displays a list of file changes under "Files changed". A specific commit is selected, showing the following content:

```

@@ -6,15 +6,16 @@
 In Chapter 3 – _Teamwork and Collaborative Development_ we learn how to collaboratively work on a project.

 2. Delete the following line:
 3. Add one or two lines here with a random text:
 4. Modify the following line by removing the letters that do not belong:
 5. Commit your changes into a new _branch_:

```

The code editor has a toolbar with "Write" and "Preview" tabs, and a rich text editor toolbar below it. A modal window titled "Add a comment to the change" is open, containing a text area and a "Comment" button. At the bottom of the code view, there is a note: "💡 ProTip! Use ⌘n and ⌘p to navigate between commits in a pull request."

图3.4 - 向更改行添加comment

经典的代码审查和pull request之间的重要区别在于用户可以更新pull request。这使得用户可以处理comment并共同处理issue，直到issue关闭为止。读者可以将编辑文件并提交到新分支，以查看pull request是否会反映更改。

8. 通过打开右上角的菜单并选择Edit file（参见图3.5），直接从pull request编辑文件：

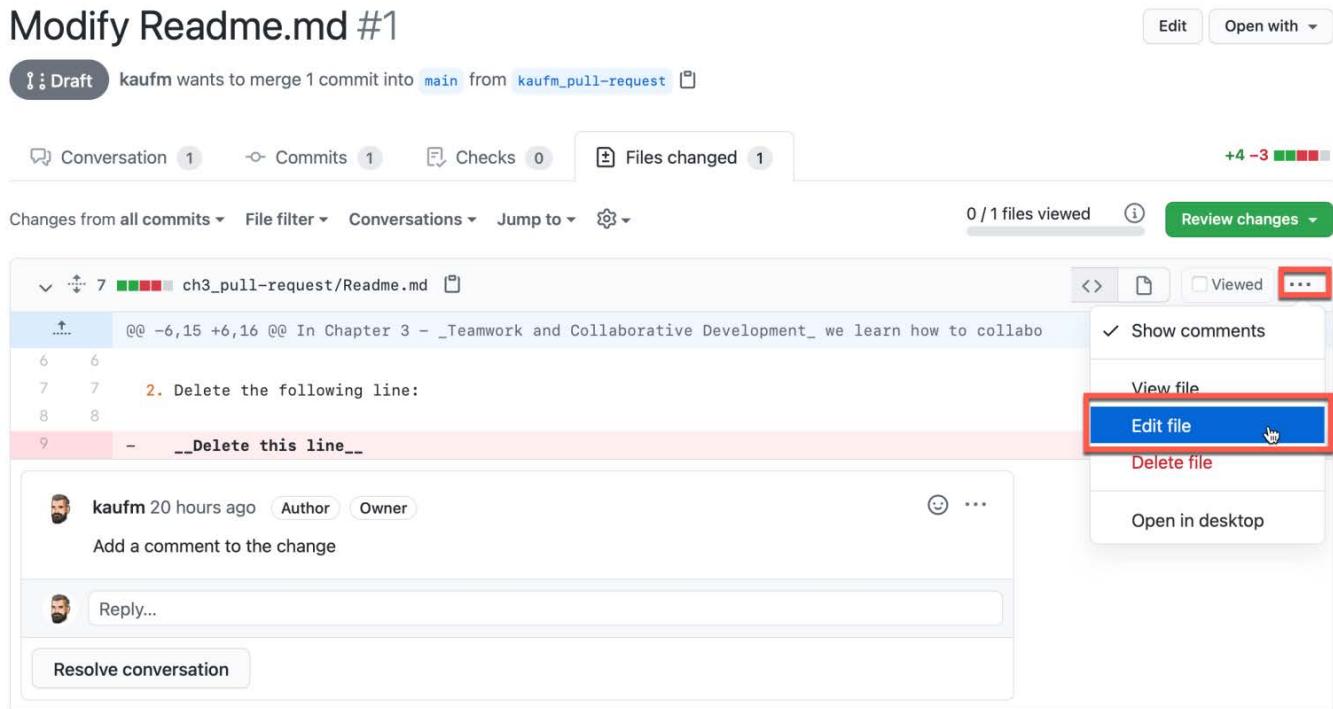


图3.5 - 在pull request中编辑文件

9. 通过添加新的文本行来修改文件。在创建pull request之前，将更改提交到创建的分支（参见图3.6）：

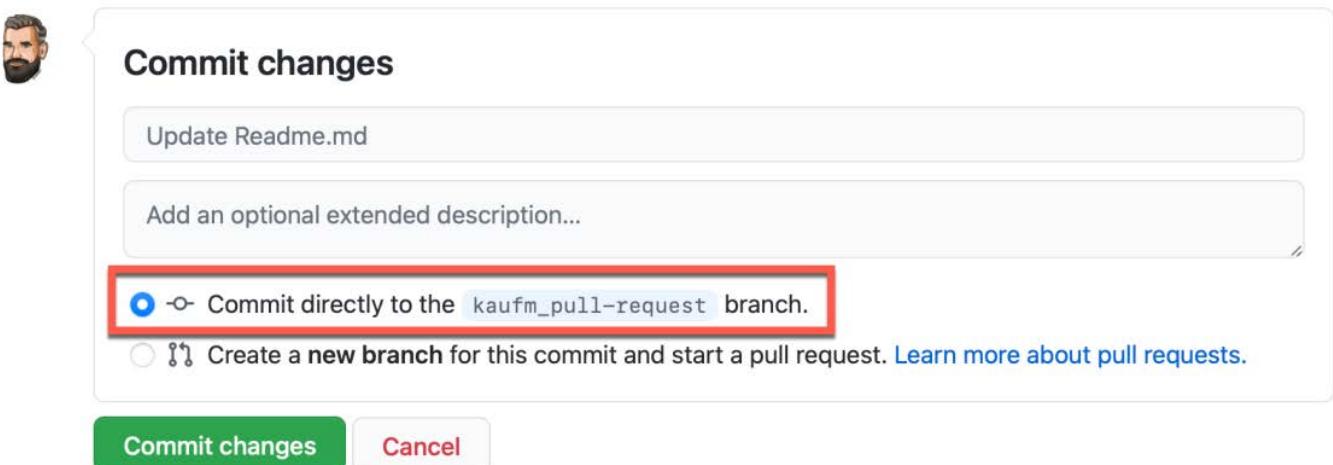


图3.6 - 提交对分支的更改

10. 返回pull request，注意系统会自动显示所做的更改。读者可以在Files changed下查看文件中的所有更改，也可以在Commits下查看单个提交中的更改（参见图3.7）：

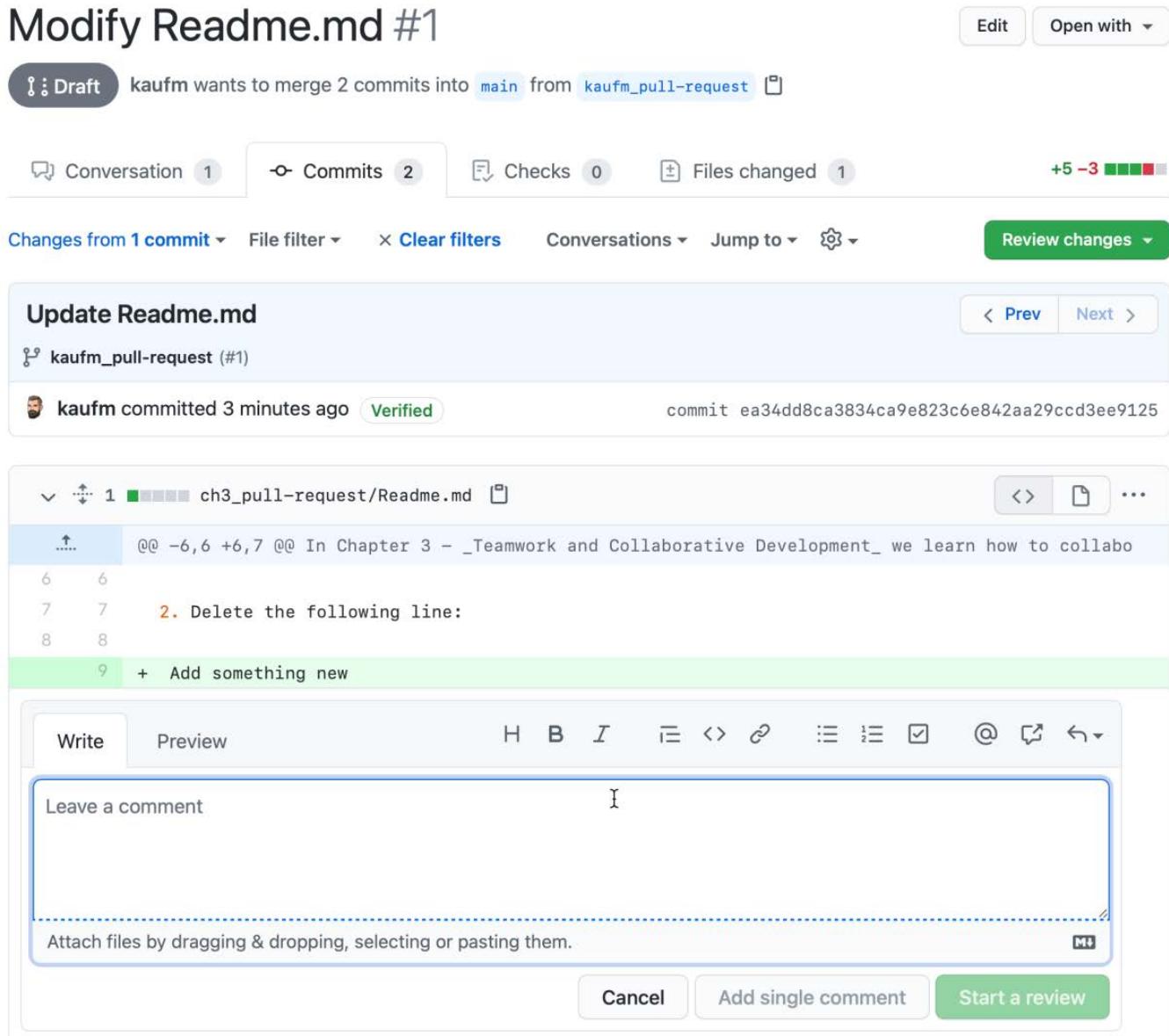


图3.7 -对单个提交中的更改进行comment

11. 如果读者初次接触GitHub上的pull request，有以下几个要点：

- Pull request是关于一个分支到一个基本分支的更改。如果更新分支，则pull request将自动更新。
- 可以使用GitHub中的丰富功能来协作完成所有更改：任务列表、提及、参考、源代码等。
- 可以查看基于每个文件或基于每个提交的更改。这有助于区分重要的更改和不重要的更改（例如重构）。

## 提交更改

GitHub pull request有丰富的功能集，可以帮助用户改进协作流程。

### 创建pull request草稿

创建pull request的最佳时间是什么时候？每个人的观点可能都不同，作者认为：越早越好！理想情况下，用户可以在开始处理某项工作时创建pull request。这样，团队只需查看打开的pull request，就可以知道每个人都在做什么。但是如果过早地开启pull request，审阅者就不知道何时给出反馈。这就是pull request草稿的好处，用户可以尽早创建pull request，每个人都知道工作仍在进行中，审阅者还没有得到通知，但是用户仍然可以在注释中提到相关人员，以便尽早获得代码反馈。

创建pull request时，可以直接在草稿状态下创建（见图3.8）：

### Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

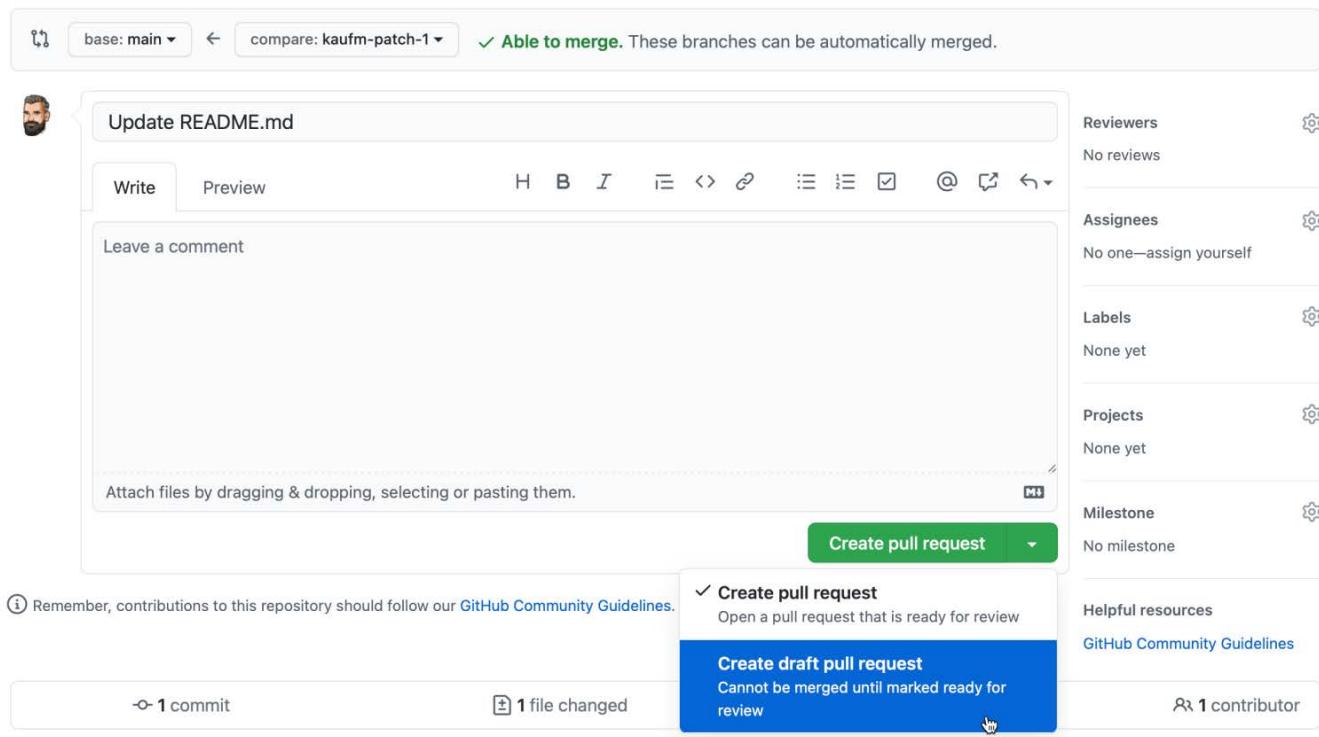


图3.8 - 以草案形式创建pull request

Pull request草稿会被明确标记为Draft，并有自己的图标（参见图3.9）。用户还可以使用draft: true或draft: false作为搜索参数来过滤搜索中的pull request：

## Modify Readme.md #1

The screenshot shows a GitHub pull request titled "Modify Readme.md #1". A blue button labeled "Draft" is visible at the top left. The pull request summary says "kaufm wants to merge 2 commits into `main` from `kaufm_pull-request`". Below the summary, there are tabs for "Conversation" (2 comments), "Commits" (2), "Checks" (0), and "Files changed" (1). A comment from "kaufm" is shown, dated 3 days ago: "Test pull request features." There are "Owner", "Smile", and "More" buttons next to the comment. At the bottom, the text "Symbol for draft pull requests" is displayed.

图3.9 - pull request草案标记示例

如果pull request已处于审阅状态，用户仍然可以通过单击Revieers | Still in progress? | Convert to draft 继续编辑。

如果pull request已准备好接受审阅，只需点击“Ready for review”（见图3.10）：

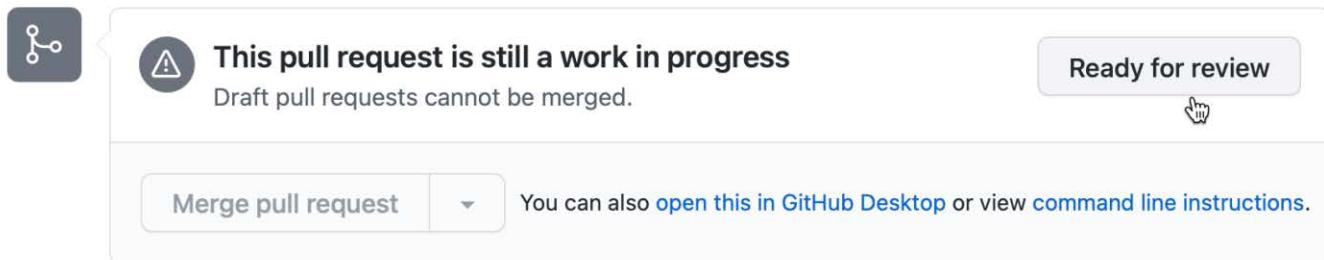


图3.10 - 删除pull request的草稿状态

Pull request草稿是一个很好的功能，可以以透明的方式在仓库内容变更以尽早进行团队协作。

## 代码所有者

当存储库中的某些文件发生更改时，代码所有者（Code owners）可以自动将审阅者添加到pull request中。这个特性还可以用于跨团队协作，并且还可以在早期开发阶段添加许可，而不需要在发布阶段来要求许可。假设开发者在仓库中有设置基础设施的代码，可以使用Code owners的功能要求协作团队中的某个人进行审阅。假设有可以定义应用程序外观的文件，每次修改它们可能需要获得设计团队的批准。Code owners不仅是给以许可，它们还可用于在跨团队的协作社区中传播知识。

Code owners可以是团队或个人。他们需要拥有写入权限才能成为Code owners。如果pull request不是草稿状态，则Code owners将被添加为审阅者。

要定义Code owners，用户需要在存储库的根目录下docs/或者.github/目录中创建一个名为CODEOWNERS的文件。该文件的语法很简单，如下所示：

- 使用@username或@org/team-name定义代码所有者，还可以使用用户的电子邮件地址。
- 使用模式匹配文件以指派代码所有者。顺序很重要：最后匹配的模式优先。
- 使用#表示注释，！表示对模式求反，[]表示定义字符范围。

下面是Code owners文件的示例：

```
# The global owner is the default for the entire repository
* @org/team1
# The design team is owner of all .css files
*.css @org/design-team
# The admin is owner of all files in all subfolders of the
# folder IaC in the root of the repository
/IaC/ @admin
# User1 is the owner of all files in the folder docs or
# Docs – but not of files in subfolders of docs!
/[Dd]ocs/* @user1
```

关于"Code owners"的详细信息，请参阅下面的链接：<https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/creating-a-repository-on-github/about-code-owners>。

Code owners是一个可以跨越团队共享知识很好的方式，并将发布阶段的许可权限转变到变更发生时的早期许可。

## 审阅设置

在合并pull request之前，用户可以要求指定数量的审阅。这在可应用于多个分支的branch protection rule上设置。在Settings | Branches | Add rule下可以创建分支保护规则。在规则中，用户可以设置合并前所需的审阅数量，选择是否要在更改代码时取消审批，以及强制执行代码所有者（code owners）的审批（参见图3.11）：

The screenshot shows the GitHub repository settings for 'kaufm / AccelerateDevOps'. The left sidebar has a 'Branches' tab selected. The main area is titled 'Branch protection rule' for the 'main' branch. It includes sections for 'Branch name pattern' (set to 'main'), 'Protect matching branches' (with three checked options: 'Require pull request reviews before merging' (2 approvals), 'Dismiss stale pull request approvals when new commits are pushed', and 'Require review from Code Owners'), and two unchecked options: 'Require status checks to pass before merging' and 'Require conversation resolution before merging'.

图3.11 - 某一分支的审阅设置

有关分支保护的更多信息，请参见<https://docs.github.com/en/github/administering-a-repository/defining-the-mergeability-of-pull-requests/about-protected-branches#about-branch-protection-rules>。第7章《基于主干的开发》中将更详细地介绍这一内容。

## 设置pull request审阅

如果代码已准备好接受审阅，用户可以手动添加所需数目的检阅者。GitHub会根据修改代码的作者提供审阅者的推荐（见图3.12）。用户只需单击“Request”，也可以手动搜索人员以执行审核：

## Reviewers

### Suggestions

Request review from totosan



[Request](#)



Still in progress? Convert to draft

图3.12 - GitHub审阅人推荐示意

也可以让GitHub自动为团队分配审阅者。在Settings | Code review assignment下可以进行相关配置。用户可以选择自动分配的审阅者数量，并选择以下两种算法之一：

- Round robin: 选择到目前为止收到最少请求的审阅者作为此次的审阅者。
- Load balance: 根据每个成员的审阅请求总数（考虑未完成的审阅）选择审阅者。

可以将某些成员排除在审阅之外，也可以选择在指定审阅者时不通知整个团队。请参见图3.13以了解如何为团队配置代码审阅任务：

The screenshot shows the GitHub team settings interface under the 'Code review assignment' tab. On the left, there's a sidebar with 'Team settings' and three sub-options: 'General', 'Code review assignment' (which is selected), and 'Scheduled reminders'. The main area has a title 'Code review assignment'. Underneath, there's a section with a checked checkbox for 'Enable auto assignment' and a note explaining it routes requests to individual team members. Below that is a dropdown for 'How many team members should be assigned to review?' set to '2'. There's also a 'Routing algorithm' dropdown set to 'Load balance' with a checked checkbox for 'Never assign certain team members' and a dropdown menu containing 'wulfland'. At the bottom, there's a 'Notifications' section with a checked checkbox for 'If assigning team members, don't notify the entire team.' A green 'Save changes' button is at the very bottom.

图3.13 - 管理团队的代码审阅任务

自动合并

Pull request中最令人喜欢的特性之一是自动合并（auto-merge）。这允许用户在处理小的更改时提高速度，特别是在启用了连续部署（continuous deployment, CD）的情况下。如果满足所有策略，自动合并将自动合并更改。如果已经完成了更改，则可以启用自动合并来处理其他更改。如果pull request通过了一定数量的许可并且通过了所有的自动检查，则pull request将自动合并并部署到生产环境中。

## Pull request审阅

如果某个用户已被选中为审阅者，可以对许多更改进行注释、提出建议，并在最后提交带有以下标记之一的审阅：

- Comment
- Approve
- Request changes

在上一节中重点介绍了与pull request作者相关的功能。本节将描述一个帮助审阅者执行审阅并向作者提供适当反馈的功能。

### 审阅pull request中提议的修改

用户可以通过一次查看一个文件的更改来开始审阅。如果将鼠标悬停在行上，则会在左侧看到+图标。它可用于添加单行注释，或者将其拖动到多行上来创建多行注释。如果用户需要添加注释，请选择Start review（开始审阅）以开始审阅而无需提交注释。如果要添加更多评论，则按钮变为Add review comment（添加审阅评论）；可以添加任意数量的注释到审阅。评论只对自己可见，直到提交审阅。用户还可以随时取消审阅。

### 将文件标记为已查看

审阅时，用户会在文件顶部看到进度条。完成一个文件后，用户可以选择“已查看”复选框。文件将被折叠，进度条将显示进度（参见图3.14）：

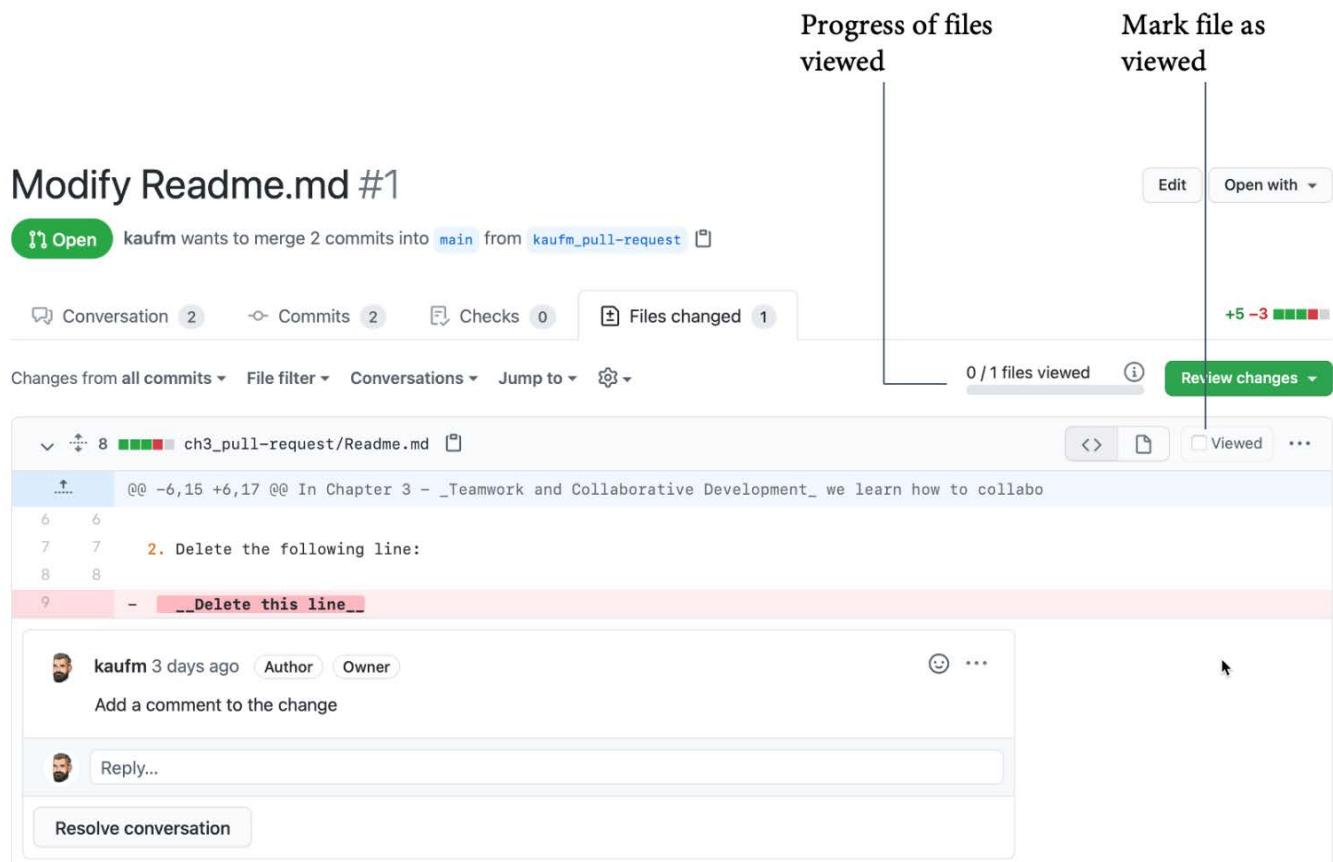


图3.14 - 将文件标记为已查看

### 动手实现-提出suggestions

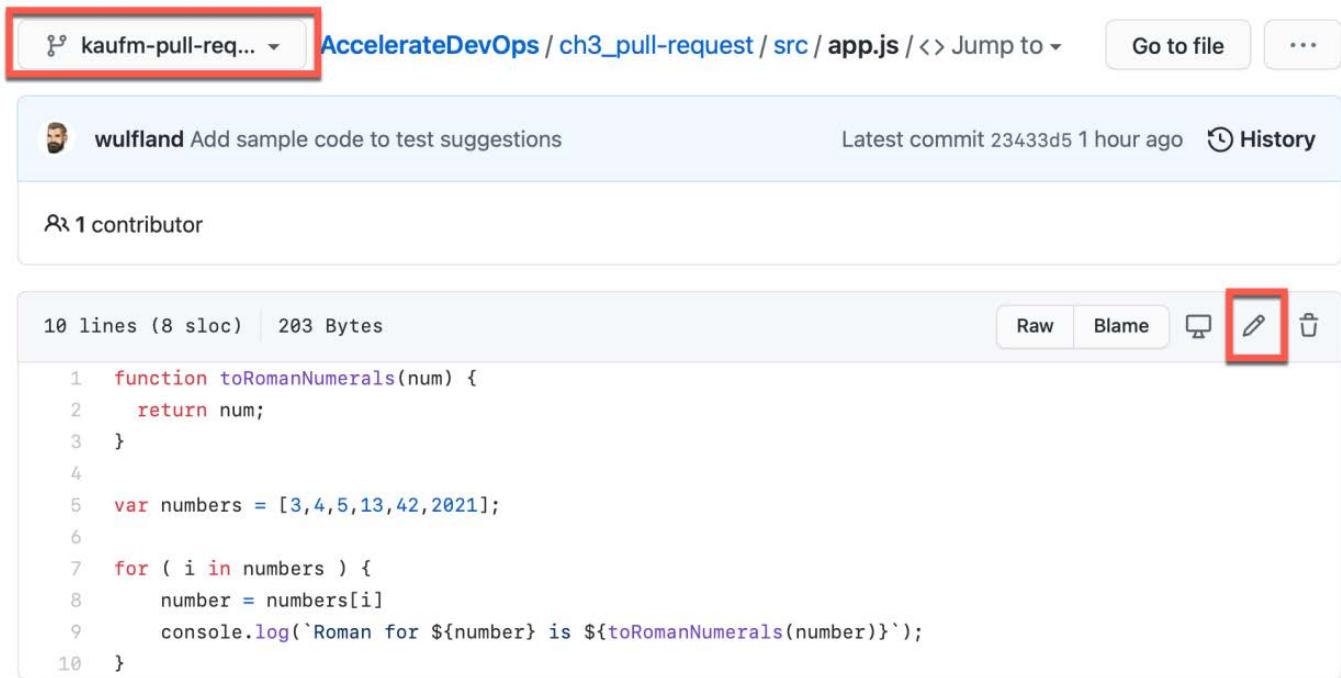
提供反馈的最佳方式是提出使pull request作者可以轻松融入其分支的suggestion。suggestions这个功能非常重要，如果读者从来没有尝试过，那么值得一试：

1. 在上一个实践练习过程里创建的存储库中打开fork：<https://github.com//AccelerateDevOps>。

在fork中，导航至Chapter 3 | Review Changes（ch3\_pull-request/Review-Changes.md），该文件还包含操作指南。

通过单击源代码块右上角的“Copy”图标复制示例源代码。

2. 定位到src/app.js（使用markdown中的链接）。选择在上一个动手练习中创建的分支，并通过单击右上角的Edit图标（铅笔）来编辑该文件（参见图3.15）：



The screenshot shows a GitHub pull request page for a file named 'app.js'. The file has 10 lines (8 sloc) and 203 Bytes. The code contains a function 'toRomanNumerals' that logs Roman numerals for a list of numbers. A red box highlights the file name 'kaufm-pull-req...' in the top left corner of the header. The code editor shows the following:

```
1  function toRomanNumerals(num) {  
2      return num;  
3  }  
4  
5  var numbers = [3,4,5,13,42,2021];  
6  
7  for ( i in numbers ) {  
8      number = numbers[i]  
9      console.log(`Roman for ${number} is ${toRomanNumerals(number)}`);  
10 }
```

图3.15 - 编辑代码文件以添加示例代码

3. 删除第2行，然后按Ctrl + V插入代码。
4. 直接提交到pull request的源分支。
5. 返回pull request并在Files changed下查找src/app.js。请注意，第6到9行中的嵌套循环没有正确缩进。标记第6到9行并创建多行注释。点击Suggestion按钮，会看到代码在comment块中，包括空格（见图3.16）：

# Update Create-PullRequest.md #3

Edit

Open with ▾

 Open kaufm wants to merge 2 commits into main from kaufm-pull-request

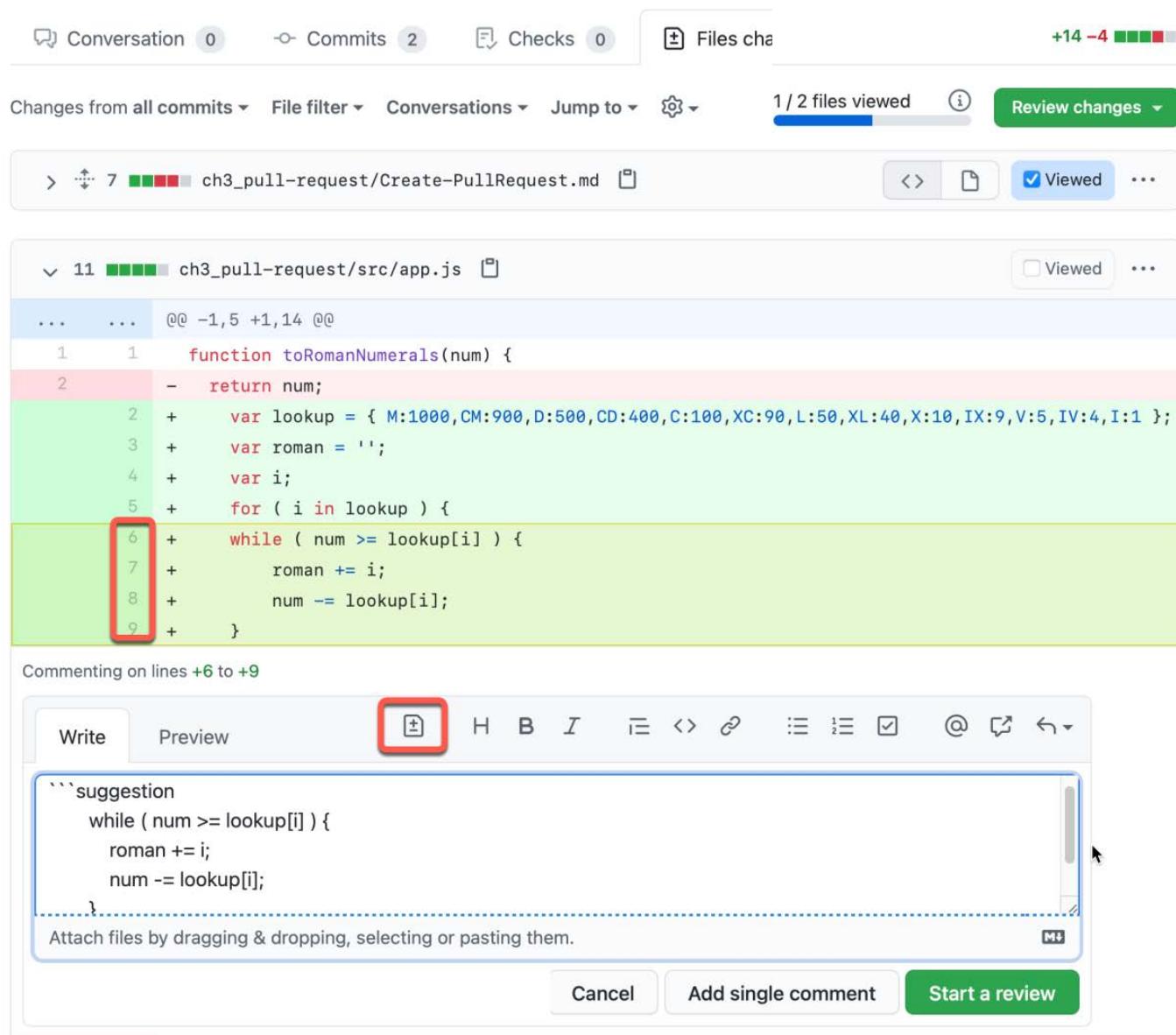


图3.16 - 为多行注释创建建议

6. 注意，comment代码块包含完整的代码，包括空格。在每一行的开头加四个空格以修正缩进。

可以将comment作为审阅的一部分（Start a review）或将comment直接提交给作者（Add single comment）。对于此次实践练习，建议直接提交给作者。

## 将反馈合并到pull request中

假设用户是审阅者和作者，可以直接切换角色。作为作者可以查看pull request的所有建议。

用户可以直接向分支提交建议，或者可以将多个建议批量提交到一个提交中，然后一次性提交所有更改。将更改添加到批处理并在文件顶部应用批处理（参见图3.17）：

## Update Create-PullRequest.md #3

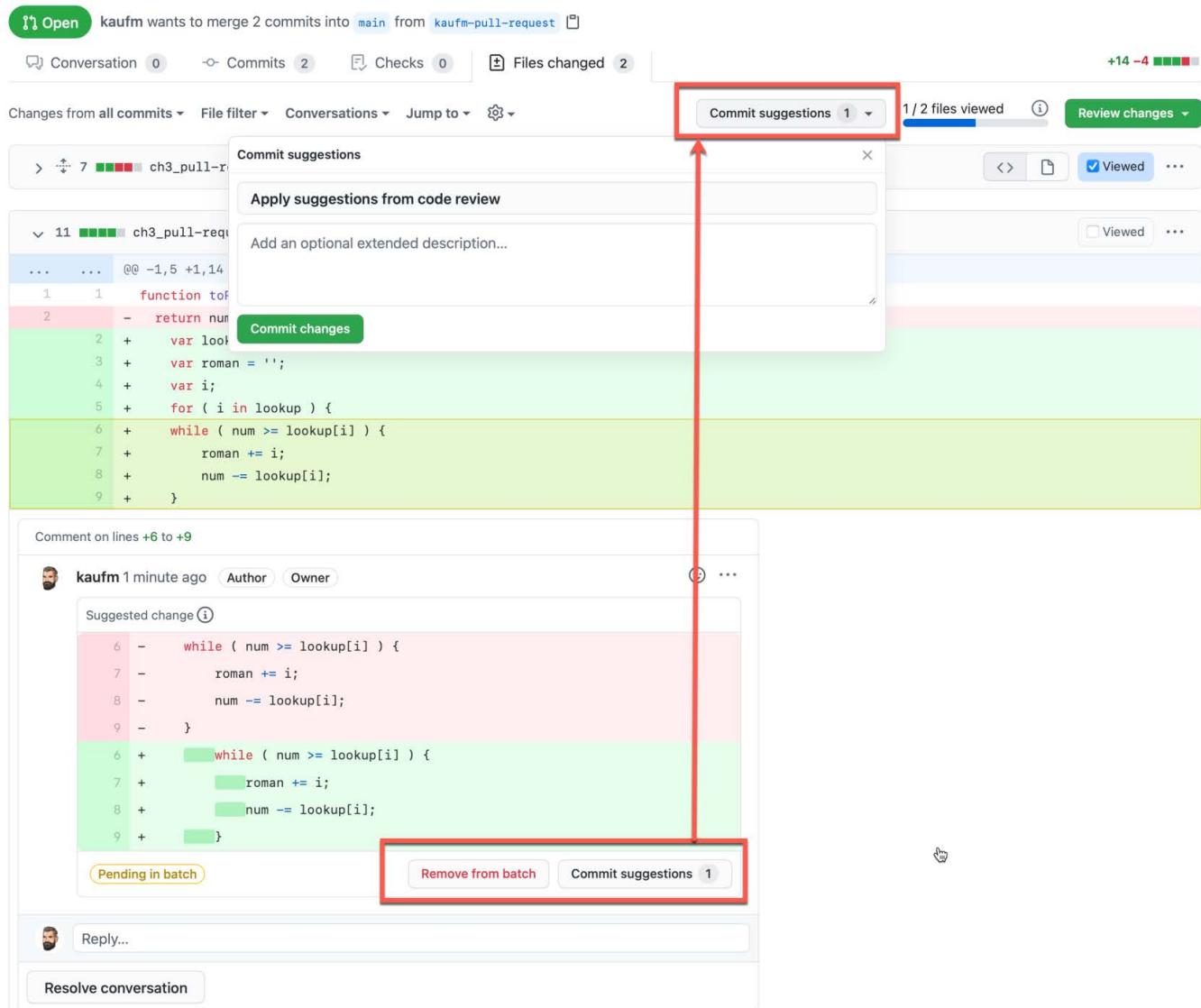


图3.17 - 将建议整合到代码中

建议是提供反馈和提议代码更改的好方法。对于作者来说，将它们合并到代码中是非常容易的。

### 提交审阅

如果已完成审阅并添加了所有注释和建议，则可以提交审阅。作者将被告知审阅结果，并可以回答注释。用户可以留下最后评论并选择以下三个选项之一：

- Approve (批准)：批准更改。这是达到所需审阅者数量的唯一选项！
- Comment (评论)：提交反馈而不批准或拒绝。
- Request changes (请求更改)：指明需要批准更改。

单击“Submit review”完成审核（参见图3.18）：

### Update Create-PullRequest.md #3

The screenshot shows a GitHub pull request interface for a file named 'Create-PullRequest.md'. The top right corner indicates '1 / 2 files viewed' and 'Finish your review 2'. A green button labeled 'Open' is visible. The main area displays the code changes, which include a function 'toRomanNumerals' with several commits. On the right side, there is a 'Finish your review' panel with options to 'Comment', 'Approve', or 'Request changes'. Below this is a 'Submit review' button. A comment from a user named 'wulfand' is shown, with a note that it is pending approval. The bottom right corner has a 'Reply...' button.

**Number of comments**

**Add final comment**

**Approve, comment or Request changes**

**Submit review**

图3.18 - 完成审阅

### 完成pull request

如果要放弃分支中的更改，可以关闭pull request而不进行合并。要将更改合并到基本分支中，有三个合并选项，概述如下：

- Create a merge commit：这是默认选项。它创建一个合并提交，并将分支中的所有提交作为一个单独的分支显示在历史记录中。如果有许多长期运行的分支，这可能会使历史记录变得混乱。用户可以在此处看到此合并选项的表示：

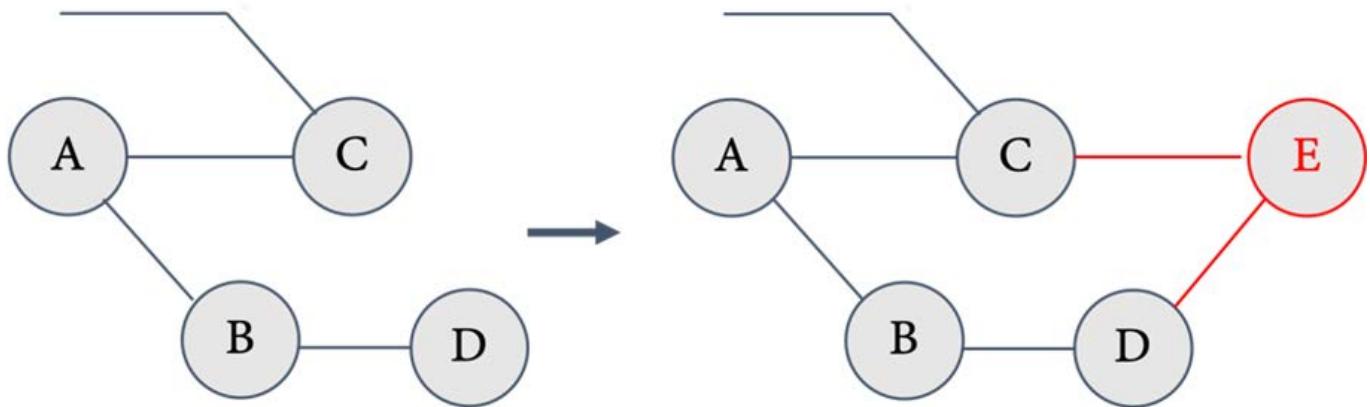


图3.19 - 合并提交时的Git历史记录

- Squash and merge：分支中的所有提交都将合并为一个提交。这将创建一个干净的线性历史记录，如果在合并后删除分支，这是一个很好的合并方法。如果继续处理分支，则不建议使用此方法。可以在此处看到此合并选项的表示：

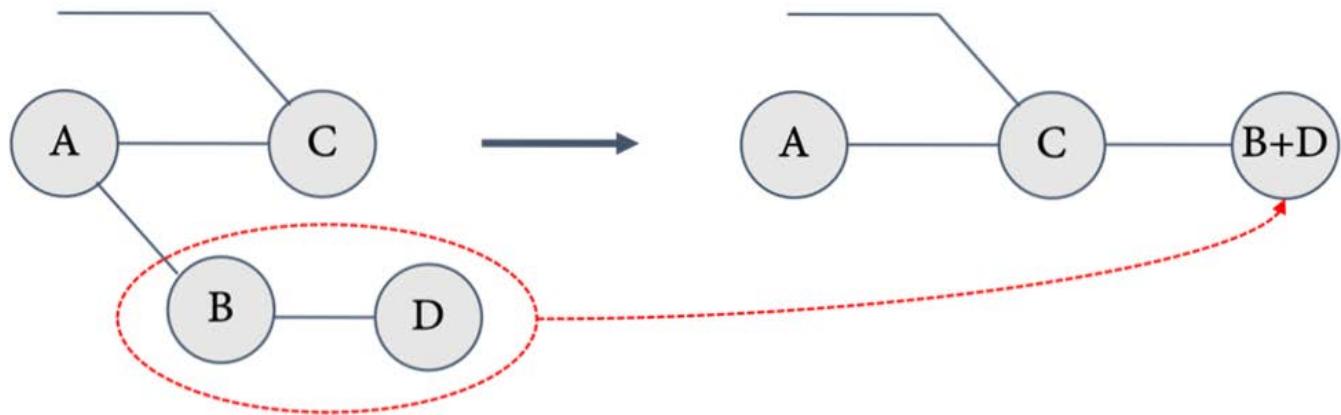


图3.20 -进行了压缩和合并后的Git历史记录

- Rebase and merge: 将分支的所有提交应用到基础分支的头。这也创建了一个线性历史记录，但保留了单个提交。如果继续在分支上工作，不建议使用此方法。可以在此处看到此合并选项的表示：

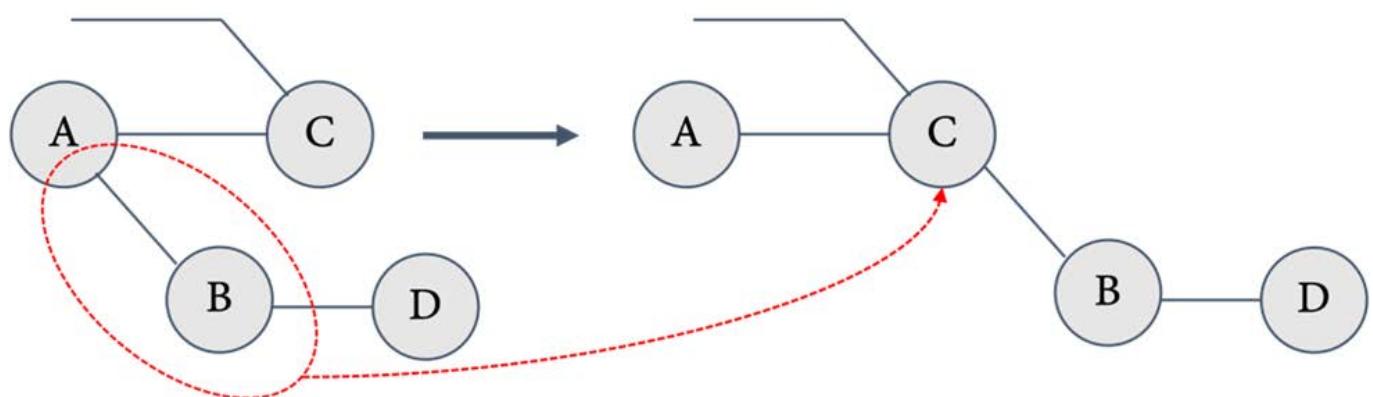
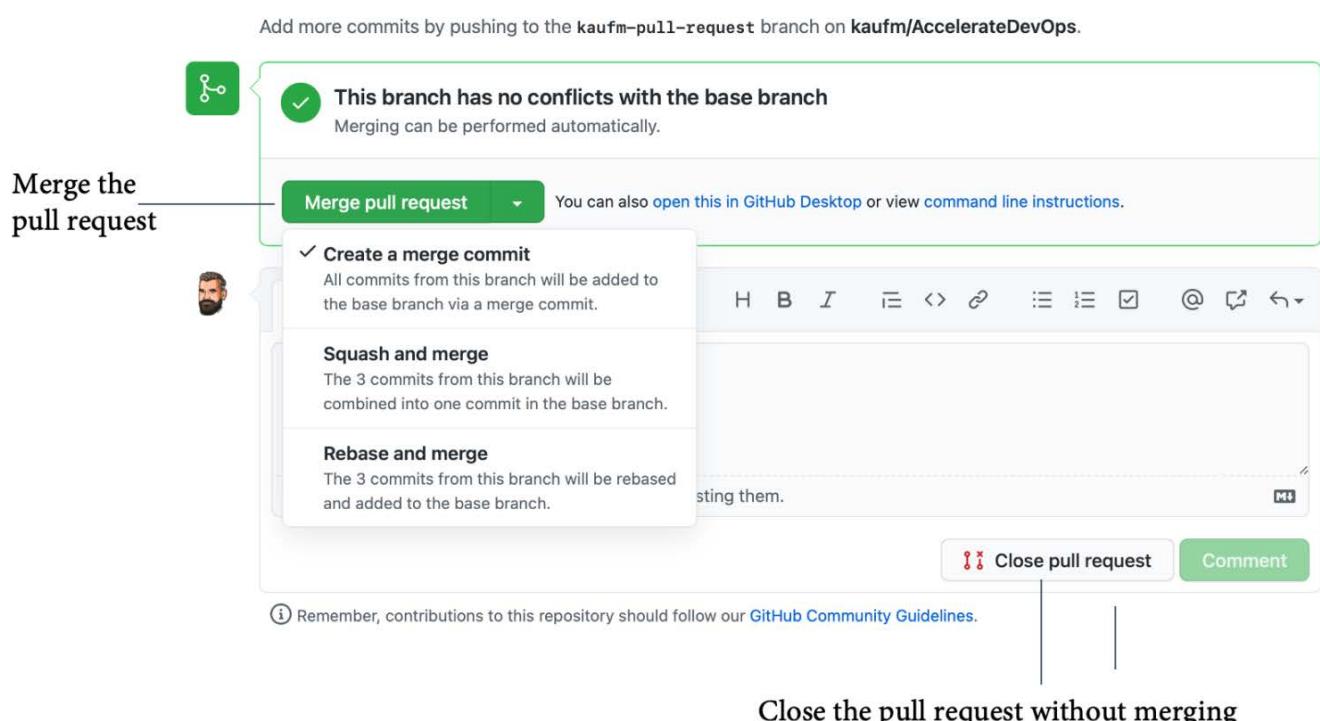


图3.21 - 如果进行了一次变基和合并，Git历史看起来是线性的

选择所需的合并方法，然后单击“Merge pull request”（参见图3.22）：



## 图3.22 -完成pull request

修改合并消息并单击“Confirm merge”以确认合并。合并后，可以根据需要删除分支。

# 代码审阅的最佳实践

Pull request是在所有类型的代码上进行协作的好方法。本章只触及协作工作流的可能性的表面，但是为了让团队有效地协作，读者应该考虑一些有效代码评审的最佳实践。

## Git培训

这一点可能看起来理所应当，但要确保团队在Git方面训练有素。与随机分布在多个提交中的许多更改相比，具有良好提交注释且仅服务于一个目的的精细的提交更容易审查。特别是将重构和业务逻辑混合在一起会使审查成为一场噩梦。如果团队成员知道如何修改提交、修补他们在不同提交中所做的更改，以及如何精心制作好的提交消息，那么最终的pull request将更容易审查。

## 将pull request链接到issue

将pull request链接到启动工作的相应issue。这有助于为pull request给予背景。如果使用第三方集成，请将pull request链接到Jira、Azure Boards工作项或已连接到GitHub的任何其他源。

## 使用pull request草稿

让团队成员在开始处理某项工作时立即创建一个pull request草稿。这样，团队就知道谁在做什么。这也鼓励人们在审阅开始之前使用带有提示的评论来征求人们的反馈。尽早对更改进行反馈有助于在最后进行更快地审查。

## 至少有两名批准人

至少需要两名批准人，当然人数越多越好，具体取决于团队规模，但一个是不够的。拥有多个审阅者会使审阅具有某种动态性。作者注意到一些团队的审阅有很大成效，这仅仅是由于把批准人员从一个变成了两个！

## 进行同级审阅

将审阅视为同级审阅。不要让高级架构师审阅其他人的代码！年轻的同事也应该进行同级审阅来学习。一个好的方法是将整个团队添加为审阅者，并要求一定比例的批准（例如50%），然后人们选择他们想要的pull request。或者可以使用自动审阅在团队中随机分发审阅。

## 自动审阅步骤

许多审阅步骤可以自动化，尤其是格式设置。使用linter检查代码的格式（例如<https://github.com/github/super-linter>），或者编写一些测试来检查文档是否完整。使用静态和动态代码分析自动查找问题。越多地自动化平庸的检查，越多的审查可以集中在重要的事情上。

## 部署和测试更改

合并前自动生成和测试更改。如有必要，使用代码进行测试。人们越是相信改变不会破坏任何东西，他们就越是信任改变这个过程。如果所有审批和验证均通过，则在进行自动合并后发布更改。高度自动化使人们只需关注较小的批量，这使得审阅变得容易得多。

## 审阅准则/行为守则

一些工程师对某些需求的正确做法很有自己的观点，因此很容易发生激烈的争论。如果希望进行激烈的讨论以获得最佳解决方案，但又希望这些讨论以包容的方式进行，以便团队中的每个人都能平等地参与。制定审阅准则和行为守则有助于达到这一目的，如果人们行为不当，可以通过规则指出错误。

## 总结

软件开发是一项团队运动，拥有一个共享代码所有权的团队是很重要的，这个团队在新的变更上紧密合作。如果使用得当，GitHub pull request可以帮助实现这一点。

下一章将了解异步和同步工作，以及异步工作流如何帮助随时随地进行协作。

## 延伸阅读和参考文献

以下是本章中的延伸阅读和参考文献，读者也可以使用这些资料来获取有关内容的更多详细信息：

- Coyle D. (2018). *The Culture Code: The Secrets of Highly Successful Groups* (1st ed.). Cornerstone Digital.
- Kim G., Humble J., Debois P. and Willis J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations* (1st ed.). IT Revolution Press.
- Scott Prugh (2014). *Continuous Delivery*. <https://www.scaledagileframework.com/guidance-continuous-delivery/>
- Chacon S. and Straub B. (2014). *Pro Git* (2nd ed.). Apress. <https://git-scm.com/book/de/v2>
- Kaufmann M. (2021). *Git für Dummies* (1st ed., German). Wiley-VCH.
- Git: <https://en.wikipedia.org/wiki/Git>
- Pull requests: <https://docs.github.com/en/github/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests>
- Code owners: <https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/creating-a-repository-on-github/about-code-owners>
- Branch protection: <https://docs.github.com/en/github/administering-a-repository/defining-the-mergeability-of-pull-requests/about-protected-branches#about-branch-protection-rules>
- Code review assignments: <https://docs.github.com/en/organizations/organizing-members-into-teams/managing-code-review-assignment-for-your-team>
- Auto-merge: <https://docs.github.com/en/github/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/automatically-merging-a-pull-request>
- Pull request reviews: <https://docs.github.com/en/github/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/about-pull-request-reviews>

# 第4章 异步工作：随地协作

---

上一章中介绍了通过pull request的协作开发，以及如何利用它们为代码和构建的产品创建共享所有权。本章将重点关注同步和异步工作，以及如何利用异步工作流程的优势，在分布式、远程和混合团队中更好地协作，以及更好地跨团队协作。

本章将涵盖以下的主题：

- 比较同步和异步工作
- 分布式团队
- 跨团队合作
- 向异步工作流程转变
- 团队和 Slack 集成
- GitHub 讨论
- 页面和维基
- 通过 GitHub Mobile 随时随地工作
- 案例研究

## 同步和异步工作的比较

信息工作者进行的每一项工作本质上都是交流。包括关于编程的一切：开发者必须对未来的开发者（包括自己）沟通他们正在编码的内容，架构，甚至代码本身，以明确如何程序的更改。因此，开发者的沟通方式直接影响开发者完成任务的方式。

### 通信的历史

在人类的历史上，互动和交流方式经常发生变化。在1450年约翰内斯-古腾堡（Johannes Gutenberg）发明印刷术之前，交流大多是纯粹的口头交流，有一些有限的书面交流，这引起了一场印刷革命，让更多的人获得信息，并对宗教和教育产生了很大影响。17世纪，报纸的发明通过极大地缩短从发送者到接收者的时间，再次彻底改变了通信方式。在18世纪，公共邮政系统变得高效，以至于越来越多的通信是通过信件进行的。这使得私人通信迅速发展，就像报纸一样。在19世纪，电报的发明第一次允许人们在很远的距离上进行实时通信。第一部电话是由Philipp Reis于1861年在法兰克福发明的。当时的信息传输仍有波动，因此大多数人没有重视这项发明。直到15年后的1876年，亚历山大-格雷厄姆-贝尔终于为电话申请了专利，这次通讯革命使得实时口头交流成为可能。

以前通信的发展的跨度更多的是与几个世纪有关，而不是与几十年有关。人们有时间去适应对于哪种通信形式是最好的选择，并且人们的选择总是相当清楚和直观的。在过去的30年里，这种情况发生了迅速的变化。在20世纪90年代末，移动电话变得袖珍且价格合理。任何人都可以在任何时间与任何人交谈。有趣的是，这导致了一个新的现象：人们开始给对方发短信，而且往往喜欢异步通信而不是同步通信。随着互联网的兴起，电子邮件迅速取代了信件。但在一开始，互联网并不具有移动性，所以对电子邮件的预期回应仍然是几天的时间。这种情况在2005年左右发生了变化。互联网变得“移动”起来，智能手机允许随时随地访问电子邮件。同时，新的通信形式开始流行。Facebook、Twitter、Instagram和Snapchat。它们允许以文字、语音和视频的形式进行不同种类的沟通，有不同的受众群体（覆盖面和隐私）和不同的信息持久性（生存时间，简称TTL）。

图4.1说明了世界人口的指数增长与人们的通信行为变化之间的关系

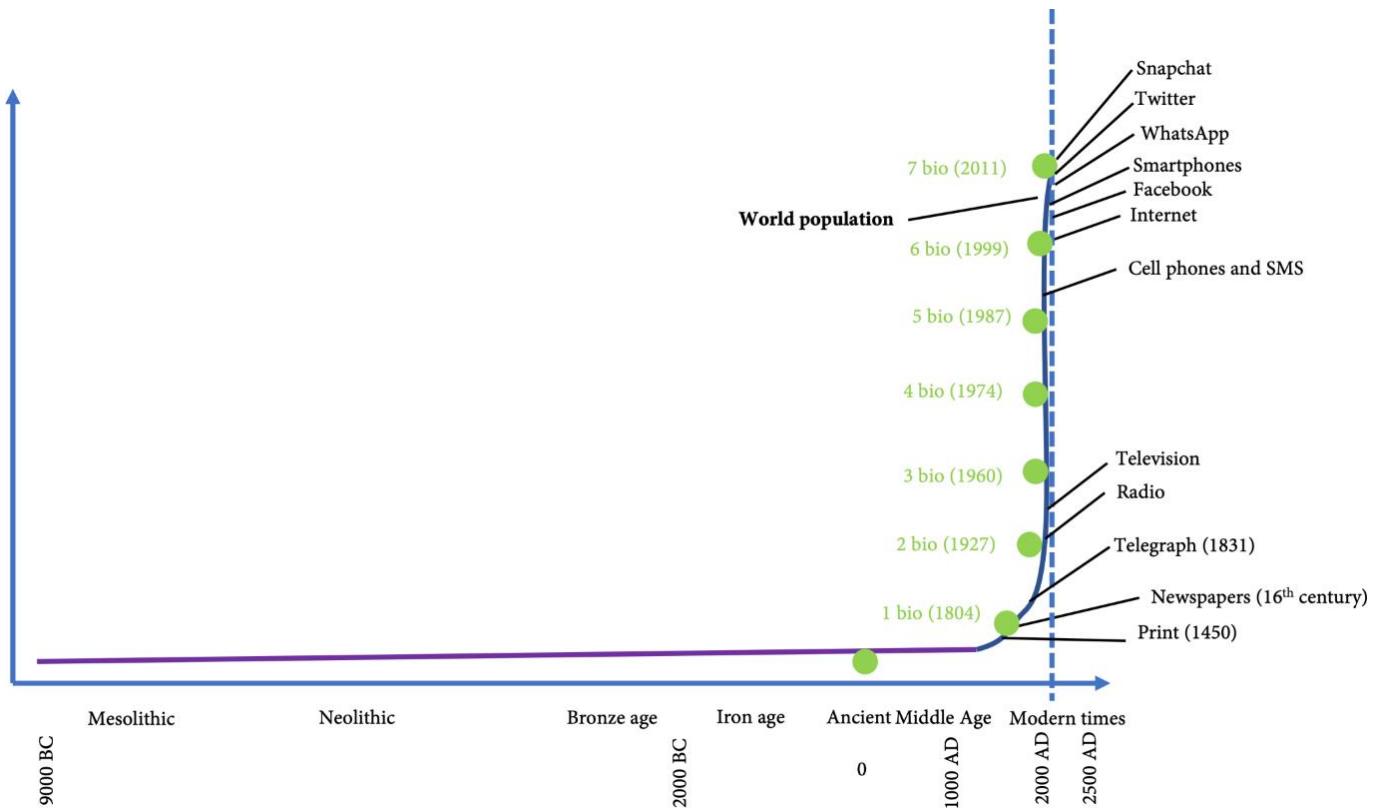


图4.1 - 通信技术的指数式发展与变化

过去30年的快速发展导致了不同类型的通信模式。无论是选择写短信，还是选择打视频电话，或者向一个群组发送一个故事，更多的是取决于个人的喜好，而不是信息的内容。对于特定种类的信息，什么是正确的沟通形式，并没有形成社会的共识。

## 工作和交流

工作不仅仅是沟通。信息工作为对话增加了所需的输出。读者可以把工作分为同步工作和异步工作。同步工作指的是当两个或更多的人实时互动以实现预期的产出。异步工作是指当两个或更多的人通过交换信息以实现预期的产出。

如果读者在一个传统企业工作，异步和同步工作的组合可能仍然像图4.2那样。至少在几年前是这样的。大部分工作是通过电子邮件或会议完成的，而会议通常是在同一个房间里进行的。

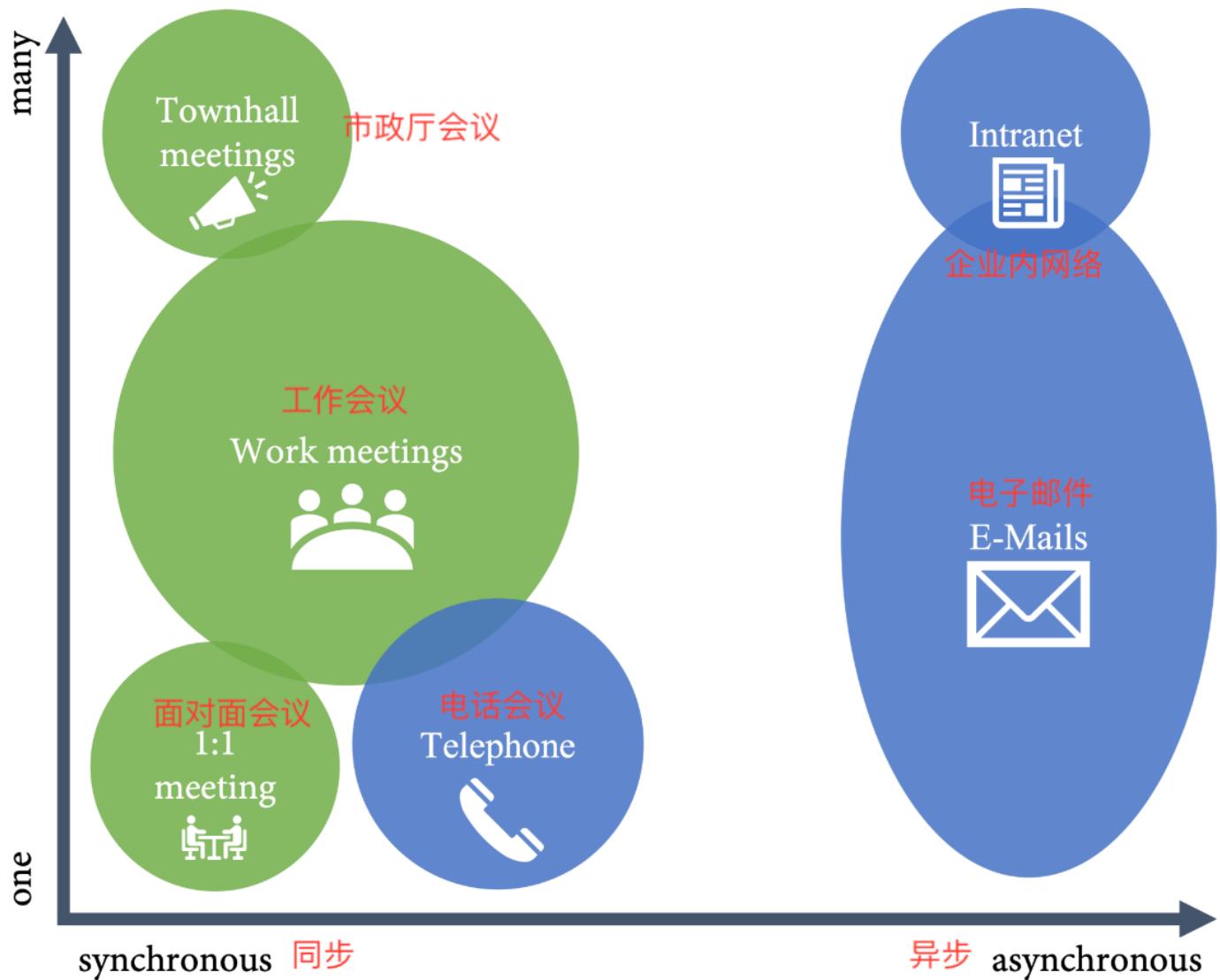


图4.2 - 传统企业中的工作和沟通

大多数异步工作是通过电子邮件和远程方式完成的，而大多数同步工作则是在面对面的会议上完成。主导的工作方式在很大程度上取决于公司的文化。在具有强烈的电子邮件文化的公司里，人们通常会在几分钟内回复收到的电子邮件。在这些公司里，许多人在开会时打开笔记本电脑，人们通常抱怨电子邮件太多。在会议文化浓厚的公司，人们往往不会及时回复电子邮件，因为他们正在参加会议，这导致了更少的电子邮件，更多的会议。

在过去的几年里，这种情况发生了巨大的变化。特别是小公司和初创公司，已经放弃了基于电子邮件的工作模式，而选择了其他异步媒介，如即时通讯。许多公司也发现了远程工作的好处，有些公司只是在大流行病的影响下被迫这样做。

第二章向开发者展示了环境切换是如何扼杀生产力的。因此，对于开发团队来说，异步工作是可取的，因为它允许开发者建立工作项目的拉动，减少环境切换。一个更现代的、为开发者优化的工作组合可以是这样的。

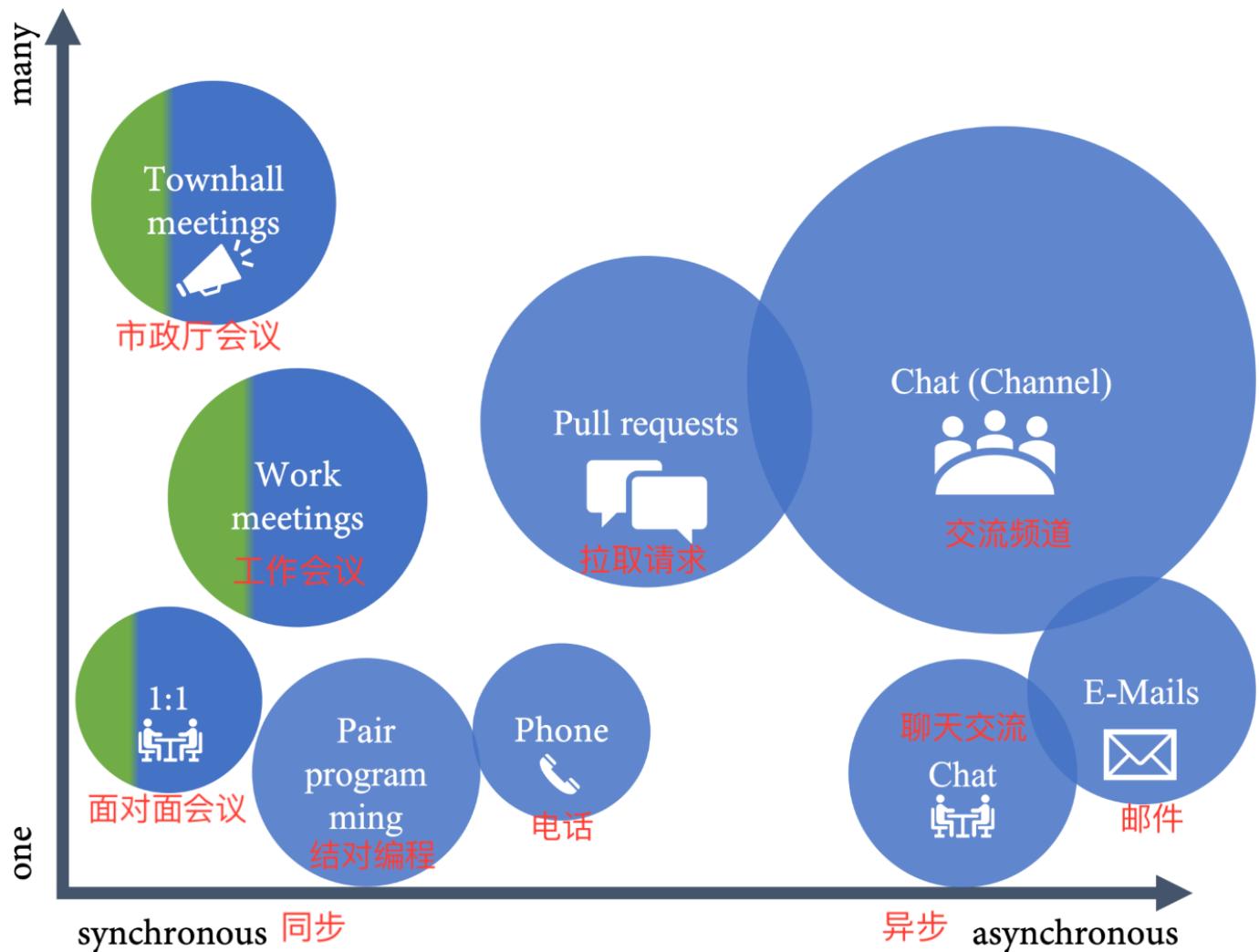


图4.3 - 对开发者工作和交流的优化模式

开发者同步工作时间越少，他们就能更专注地投入到工作中，而不需要进行环境切换和额外计划。重要的是要有这个意识：人们以同步的方式执行什么样的工作，人们可以异步地做什么？人们当面进行什么样的工作，人们可以远程进行什么样的工作？

### 面对面和远程工作

同步工作可以当面进行，也可以远程进行。两者都有其优点和缺点。

如果要必须说服某人，亲自会面是可取的。与电话或远程会议相比，销售人员总是更喜欢当面会谈，因为他们更适合社交和关系/团队建设。对于关键的反馈和敏感的问题，当面讨论也比远程讨论好。复杂的讨论或需要创意的问题也可以从当面会谈中受益。

远程会议的优势在于，由于减少了通勤时间，所以效率更高。人们可以在实际所在地点独立参与，这使得公司可以拥有一个跨越多个时区的团队。远程会议可以被记录下来，这使得人们即使不能参与，也可以观看会议。

远程会议的计划应与面对面的会议不同。一个8小时的研讨会（2x4）在面对面的情况下效果很好，但在远程的情况下就不行。远程会议应该更短，更集中。如果他们面对的是他们的计算机，人们往往会迅速分心。

在未来几年里，我们将看到越来越多的混合工作模式。混合工作模式使员工能够在不同的地点自主工作：在家，在路上，或在办公室。66%的公司正在考虑为混合工作模式重新设计办公空间，73%的员工希望有更灵活的远程工作选择（参见<https://www.microsoft.com/en-us/worklab/work-trend-index/hybrid-work>）。在组

织会议时，混合工作将是一个很大的挑战。远程会议适合个人，而现场会议适合团队。将两者结合起来将是一个挑战，不仅是对会议室的技术设备，而且对负责组织会议的人来说也是如此。

## 分布式团队

那些几乎100%远程办公、团队分布在全球各地的科技公司已经存在了相当长的一段时间了。作者认识一家公司，它有一个完全的远程招聘流程。每位员工都有预算来投资他们的家庭办公室或者在联合办公空间租一些物品。公司分布在全球各地，每年只在一起开一次会。

随着大流行病以及远程和混合工作的兴起，越来越多的公司开始看到拥有分布式团队的好处，其中包括：

- 不限制在某个大都市地区招聘，有更多的人才和更多的专家可供招聘（人才战）；
- 在其他地区招聘往往伴随着降低成本；
- 如果产品针对多个市场，有来自这些不同背景的团队成员帮助了解客户是有益的。（多样性）；
- 通过提供支持，可以自动拥有更多的覆盖时间，这意味着工程师在正常工作时间之外的无用任务减少。

分布式团队也有其挑战，最大的挑战是语言问题。非母语人士在沟通上有更多问题，如果想跨越许多国家，需要一个好的基础语言--最可能是英语。此外，文化方面的问题可能会使沟通更加困难。在远程招聘过程中，团队建设和文化适应必须发挥更大的作用。

如果想用更多的远程工程师来增加团队规模，一定要对时区进行相应的规划。这取决于会议数量，至少在时区正常工作时间的工作时间有1-2小时的重叠。这意味着通常最多可以在一个方向上开展大约8个小时，才能有1小时的重叠。如果在一个方向已经有4小时，只能在另一个方向增加一个最多4小时(图4.4)。

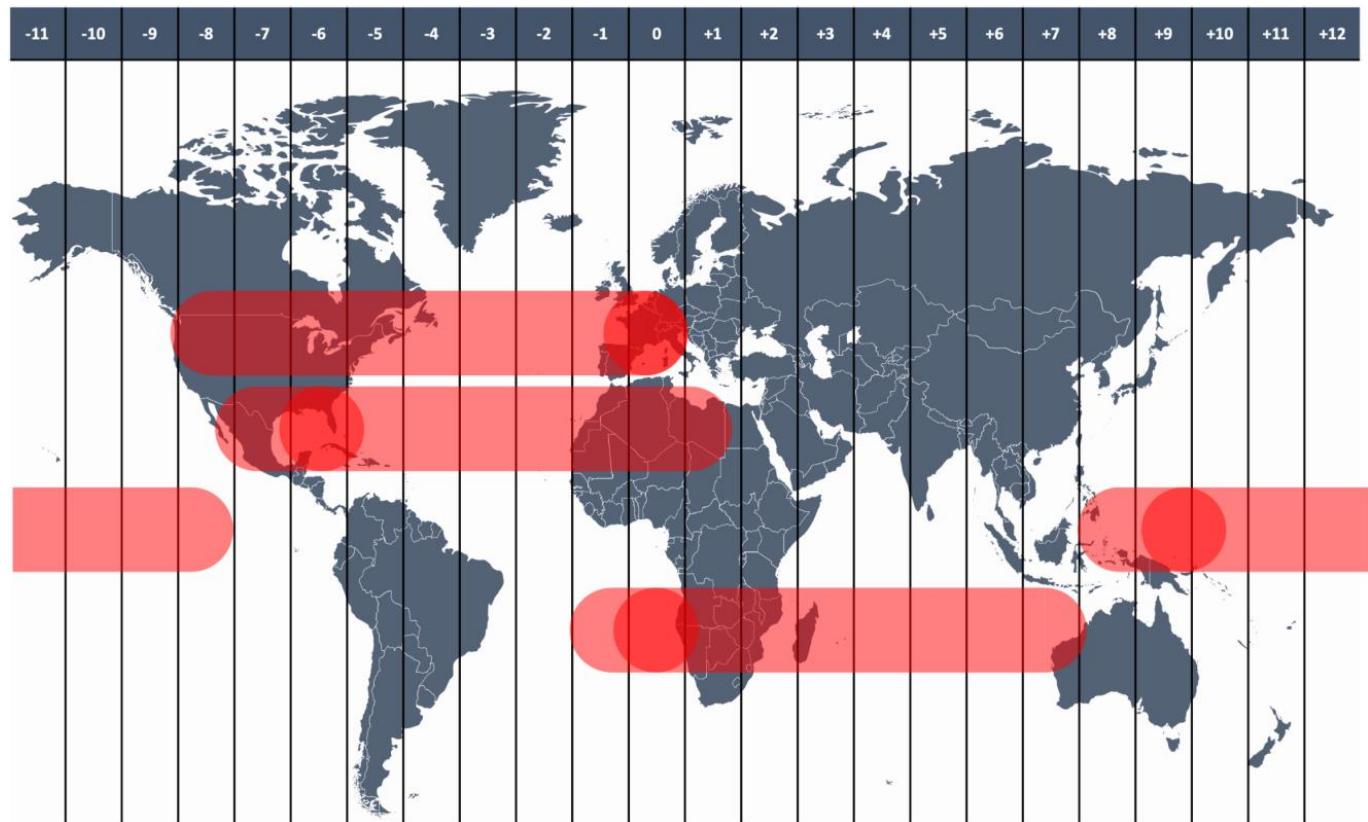


图4.4 - 根据跨时区覆盖安排会议

考虑到夏令时和不同的工作时间，使跨时区的重叠规划成为一项相当复杂的任务！

分布式团队有其优势，在未来几年将会看到更多这样的情况。如果已经开始方式实践，这是很好的，它允许从其他国家、其他时区聘请专家。这意味着让所有的沟通用英语或公司所在地区的另一种通用语言，并有尽可能多的异步工作流程。

## 跨团队合作

为了加速软件交付，公司希望团队可以尽可能地自主。能否在任何时候向最终用户提供价值而不依赖其他团队是对速度的最大影响因素之一，然而这需要在各团队之间进行一些协调：设计、安全和架构是一些必须跨越团队界限的共同关注点。良好的跨团队合作是整个团队健康一致性的标志。

好的跨团队合作通常不需要管理层的参与，而是直接将正确的人员聚集在一起解决问题。日常工作所需的会议越少，效果越好。

## 向异步工作流程的转变

为了更多地向异步工作方式转变，并允许远程和混合工作，有一些最佳做法，可以很容易地采用，例如：

- **更喜欢聊天而不是电子邮件。** 依靠电子邮件的工作流程有很多缺点：没有共同的历史记录；如果一个团队成员生病或离开，另外的成员的工作可能会受阻等等。尝试将所有与工作有关的对话转移到聊天平台，如微软团队或Slack。
- **让（大多数）会议变得可有可无。** 让所有与工作有关的会议成为可有可无的。如果他们认为会议没有价值就离开。这有助于使会议更加集中，准备更加充分，因为没有人愿意成为自己会议的唯一参与者。当然，有一些团队建设或行政会议不应该是可选的。  
·**记录所有的会议。** 录制所有的会议，让人们有机会了解，即使他们不能参与。录制的会议可以用更快的速度观看，这有助于在更短的时间内消化会议内容。  
·**要有目的性。** 要有意了解什么是会议，什么是异步工作流程（聊天、issues、pull requests和维基）。  
·**审查设置。** 一定要了解自己的指标，并定期检查自己的设置。会议是否成功，或者它们是否可以转移到issues或pull requests的讨论中？在issues和pull requests中的讨论是否要花很长时间，有些事情是否可以在会议上更快解决？不要太频繁地改变设置，因为人们需要一些时间来采用，但要确保至少每2或3个月审查和调整设置。  
·**使用提及和代码所有者。** 使用提及和代码所有者（第3章）来动态地召集合适的人完成任务。这两个功能对跨团队协作也很有帮助。  
·**把一切都当做代码。** 试着把一切都当作代码，像对待代码一样进行协作：基础设施、配置、软件架构、设计文件和概念。

## 团队和Slack整合

如果开发者喜欢即时通讯而不是电子邮件，可以使用GitHub for Microsoft Teams (<https://teams.github.com>) 或Slack (<https://slack.github.com>) 的集成功能。这些功能允许开发者在聊天频道中直接接收通知，并与issues、pull requests或部署进行互动。Slack和Teams的功能非常相似：

- **通知：**订阅版本库中的事件。开发者可以用分支或标签过滤器来过滤通知。
- **GitHub链接的细节：** GitHub链接会自动展开，并显示链接指向的项目的细节。
- **打开新的issue：** 直接从开发者的对话中创建新issue。
- **互动：** 从开发者的渠道直接处理issues、pull requests或部署批准。
- **安排提醒：** 在开发者的频道中接收代码审查的提醒。

安装很简单，开发者必须在Microsoft Teams或Slack中安装GitHub应用，并在GitHub中安装组织中相应的Teams或Slack应用。

安装完毕后，开发者就可以与GitHub机器人互动并发送消息了。在Teams中，可以用 @GitHub 提到机器人，而在Slack中，可以用/GitHub 来做。如果开发者提到机器人，会收到一个可以使用的命令列表（见图4.5）。

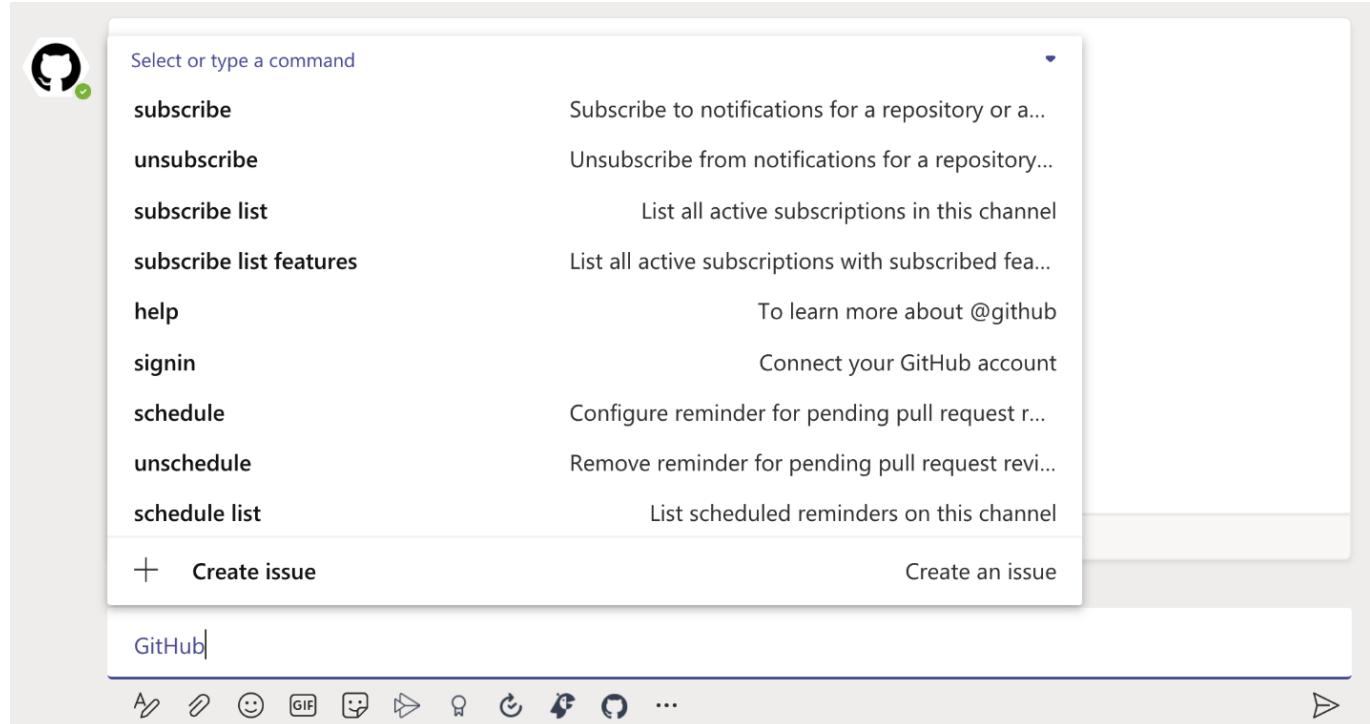
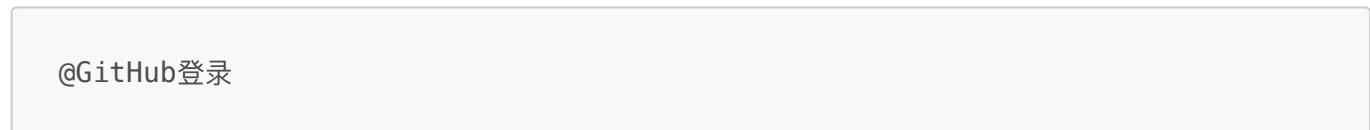


图4.5 - 向GitHub机器人发送消息

开发者必须使用的一个命令是signin。这将把GitHub账户和Teams/Slack账户连接起来。



之后，开发者可以订阅通知或安排提醒。链接的展开和与问题的互动无需配置任何东西就可以进行。图4.6显示了Team中的一个issue，它是由对话创建的。开发者可以直接对该issue进行评论或关闭它。

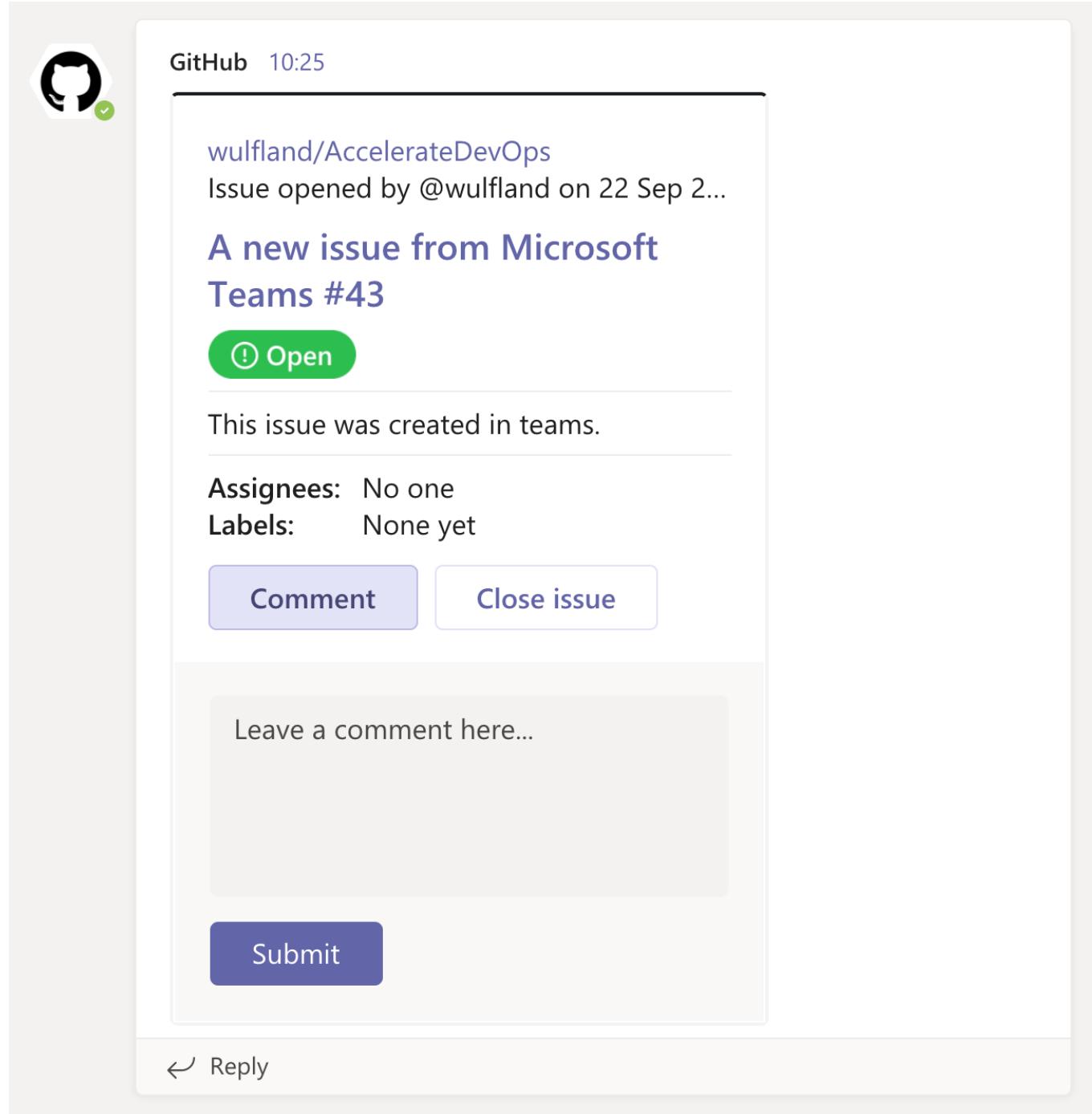


图4.6 - 与Microsoft Teams进行issue互动

聊天集成是一个强大的功能，当工作流程越来越多地通过聊天而不是通过会议或电子邮件启动和管理时，它就会派上用场。

## GitHub讨论

在第二章中介绍了如何使用GitHub问题和GitHub项目来管理工作。GitHub讨论是一个社区论坛，允许成员提出问题，分享更新，并进行开放式的对话。讨论会是一个很好的方式，通过提供一个不同的地方进行长时间的讨论和问答(Q&A)来减少问题和拉取请求的负荷。

### 开始使用讨论区

要开始使用 GitHub讨论，则必须在仓库的 "Settings|Options|Features" 中勾选Discussions将其启用。一旦勾选了这个选项，开发者的仓库里就会有一个新的主菜单项，即Discussions。

## 注意

GitHub讨论是在本书编写时仍处于Beta版的功能。有些功能后来可能已经改变。可以在<https://github.com/github/feedback/discussions/> 参与讨论，当然，这本身就是一个GitHub讨论。

讨论是按类别组织的。开发者可以在讨论中搜索和过滤，就像可以搜索和过滤问题一样。讨论本身可以被标注，并标明评论的数量，并标明是否被认为是回答。开发者可以将最多四个讨论固定在页面的顶部，以发布一些重要的公告。排行榜显示了在过去30天内回答问题最多的用户，对他们最有帮助。图4.7显示了讨论的界面。

**Pinned discussions** ————— 固定的讨论

**Search and filter** ————— 搜索和筛选

**Categories** ————— 分类

**Leader board** ————— (most helpful members) 行榜  
(贡献最多的成员们)

# comments and resolved state

图4.7 - GitHub 讨论区界面

## 讨论类别

开发者可以通过点击类别旁边的编辑按钮来管理类别。可以编辑、删除或添加新的类别。一个类别包括以下内容：

- 图标
- 标题
- 描述 (可选)

有三种类别：

### 1. 问题/答案:

讨论类别可以提出问题，建议答案，并对最佳建议答案进行投票。该类别是唯一允许将评论标记为已回答的类型。

### 2. 不限成员名额的讨论:

一个可以进行对话的类别，不需要一个明确的问题答案。很适合分享技巧和窍门或只是交流。

### 3. 公告:

与开发者的社区分享更新和新闻。只有维护者和管理员可以在这些类别中发布新的讨论，但任何人都可以评论和回复。

## 开始讨论

开发者可以通过点击 **Discussion|New discussion-** 来启动一个讨论。要开始一个新的讨论，必须选择一个类别，并输入一个标题和一个描述。开发者也可以选择给讨论添加标签。描述有完整的Markdown支持。这包括对issue、pull request和其他讨论的引用(#)，以及对其他人的提及(@)，带有语法高亮的代码，和附件(见图4.8)。

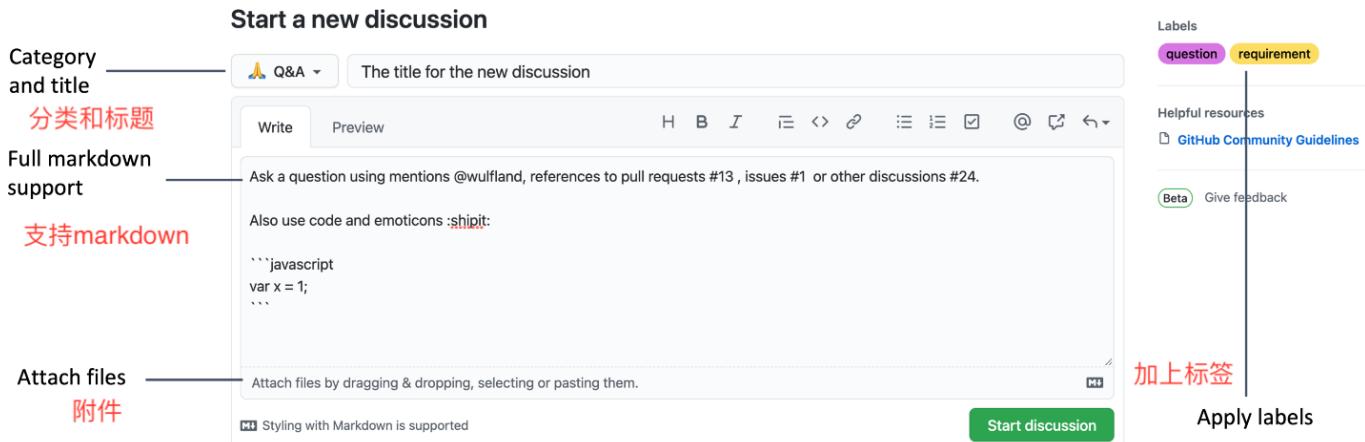


图4.8 - 新建讨论

## 参与讨论

开发者可以评论或直接回答原始的讨论描述，或者可以回答现有的评论。在每一种情况下，都有完整的Markdown支持。开发者可以在所有评论和原始描述中添加表情符号形式的反应，还可以给讨论或评论/回答加分。在右边的菜单中，可以将一个讨论转换成一个issue。作为管理员或维护者，还可以锁定谈话，把它转移到另一个资源库，把讨论固定在论坛的顶部，或删除它。图4.9给出了一个正在进行的讨论的界面。

## 图4.9 - 参与讨论

讨论是一个与同行和跨团队边界进行异步协作的好地方。有关讨论的更多信息，请参阅 <https://docs.github.com/en/discussions>。

## Pages和维基(wikis)

开发者有很多以协作的方式分享内容的选择。除了issue和讨论之外，还可以使用GitHub Pages和维基。

### GitHub Pages

GitHub Pages 是一种静态网站托管服务，可以直接从 GitHub 的仓库中提供文件。开发者可以托管普通的超文本标记语言（HTML）、层叠样式表（CSS）和JavaScript文件，自己建立一个网站。也可以利用内置的预处理器Jekyll(<https://jekyllrb.com/>)，它可以用Markdown建立好看的网站。

GitHub Pages网站默认托管在github.io域名下（如<https://wulfland.github.io/AccelerateDevOps/>），但也可以使用一个自定义域名。

GitHub Pages 是为公共存储库提供的免费服务。对于内部使用（私人仓库），需要GitHub Enterprise版。

注意 GitHub Pages 是一项免费服务，但它不能用来运行商业网站。官方禁止运行网店或任何其他商业网站。仓库配额为1GB，每月的带宽限制为100GB。详情请参考 <https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages> 获取更多信息。

了解GitHub页面的最好方法是看它的实际操作：

1. 如果读者已经在前几章的实战练习中fork了<https://github.com/wulfland/AccelerateDevOps>仓库，可以直接进入fork的仓库。如果没有，请点击版本库右上角的Fork按钮，创建一个分支。这将在<https://github.com//AccelerateDevOps>下创建一个分支。
2. 在复制的仓库分支中，导航到 **Settings|Pages**。选择想运行网站的分支main并选择 /docs 目录作为网站的根目录。读者只能选择版本库的根目录或 /docs，不能使用其他文件夹。点击保存来初始化网站（见图4.10）。

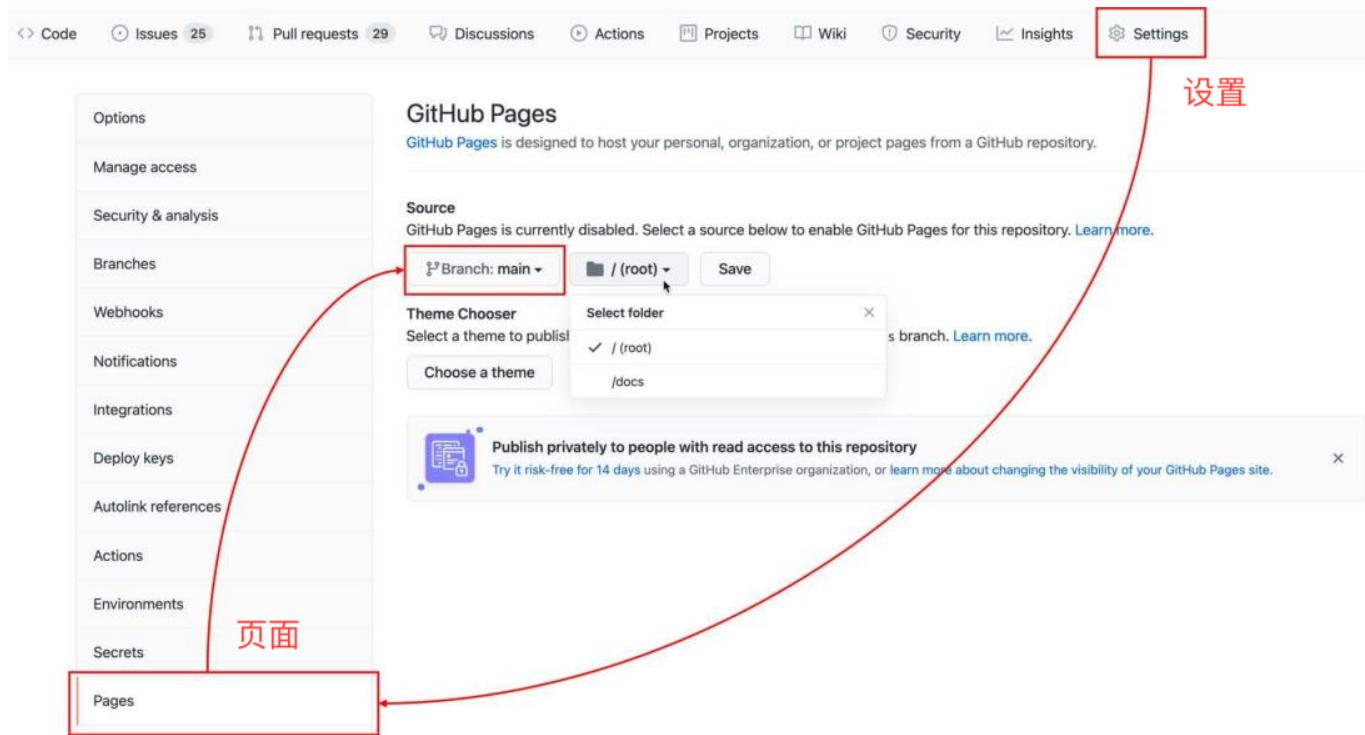


图4.10 - 在仓库中建立GitHub Pages

3. 几分钟后网站创建完成。点击链接，如图4.11所示，如果还不能使用，请刷新页面。

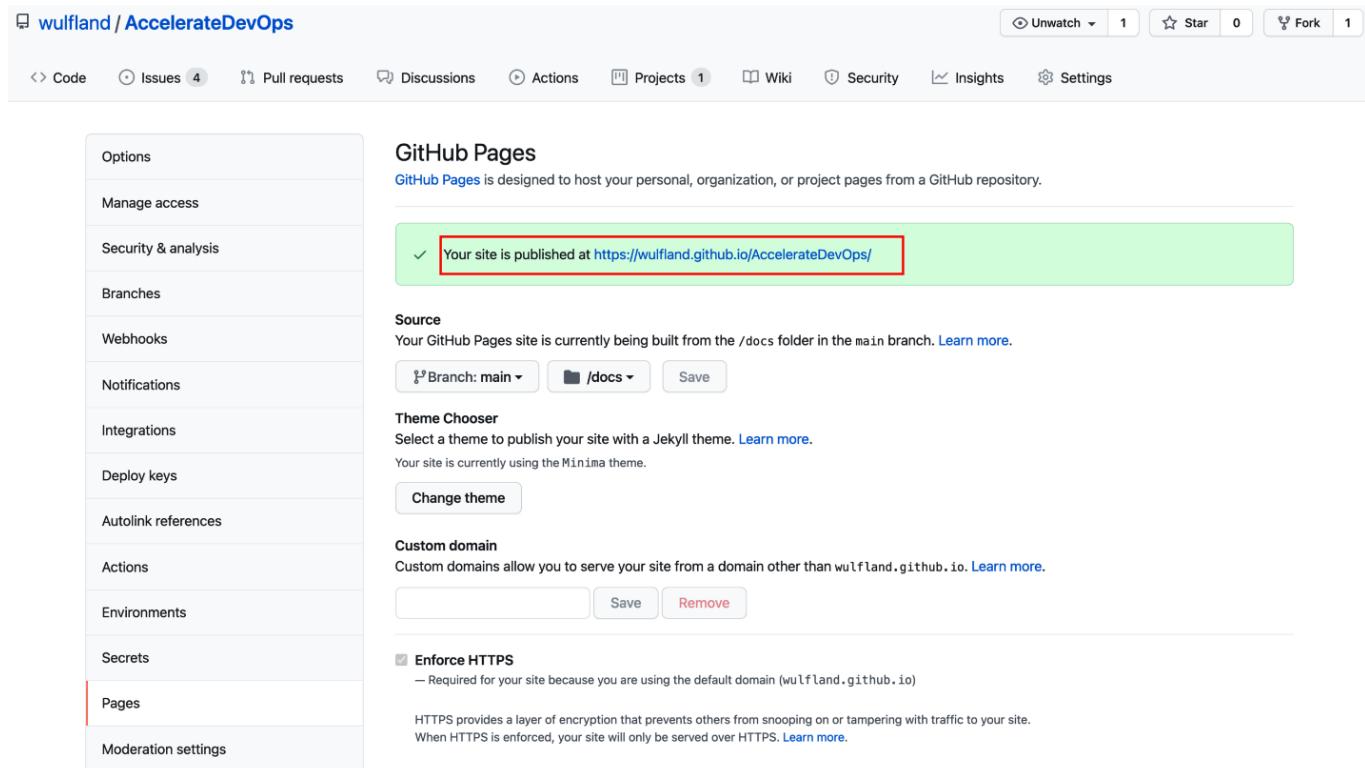


图4.11 - 导航至Web页面

4. 检查网站，页面有一个带有静态页面的菜单和一个显示带有摘录的帖子的菜单（见图4.12）。

## Accelerate DevOps with GitHub

[Get started](#) [About markdown](#) [About](#)

## Static Pages

### 静态页面

This is the homepage `index.md` of the type `home`. `Pages` are displayed in the top menu and `Posts` after this section.

Use the following page header for the homepage:

```
---
```

```
layout: home
```

```
---
```

To see excerpts of your posts and not only the heading add `show_excerpts: true` to your `_config.yml`.

## Posts

Aug 13, 2021

### [Posting in Jekyll](#)

Creating individual blog posts in GitHub pages is easy. In this post I will show you the basics.

Aug 12, 2021

### [Writing Posts with Markdown](#)

You can use markdown to write and format your blog posts. In this post I will show you how to use markdown syntax to create nice looking posts.

Aug 10, 2021

### [Posting Source Code](#)

Source code can be `inline` or as blocks. Blocks can have syntaxhighlighting for many different languages.

[subscribe via RSS](#)

## Accelerate DevOps with GitHub

Accelerate DevOps with GitHub

 [wulfland](#)

 [mike\\_kaufmann](#)

This is a sample Jekyll website that is hosted in GitHub Pages. It demonstrates how to create nice content using markdown.

图4.12 - Jekyll网站

5. 在代码中检查 `/docs/_config.yaml` 配置文件。在这里可以为网站全局配置，如标题和描述：

```
title: Accelerate DevOps with GitHub
description: >-
  This is a sample Jekyll website that is hosted in GitHub Pages.
  ...
```

开发者可以选择主题用来渲染网站，每个主题都有自己的特点，建议查看文档。本书有默认的Jekyll主题minima。要渲染Markdown，可以使用Kramdown或GitHub Flavored Markdown（GFM）。Jekyll也支持不同的插件。minima主题支持jekyll-feed，通过`show_excerpts`选项，可以设置是否在主页上显示文章的摘录：

```
theme: minima
Markdown: kramdown
```

```
plugins:  
  - jekyll-feed  
show_excerpts: true
```

许多主题支持附加选项。例如可以设置在网站上显示的社交媒体账户：

```
twitter_username: mike_kaufmann  
github_username: wulfland
```

通常情况下，静态页面按字母顺序显示在顶部导航栏。为了对页面进行过滤和排序，可以在配置中添加一个部分。由于要添加一个新的页面，在About.md之前添加一个my-page.md条目：

```
header_pages:  
  - get-started.md  
  - about-Markdown.md  
  - my-page.md  
  - About.md
```

直接将修改提交到main分支。

6. 在/docs文件夹中，在右上角选择 **Add file|Create new file**。输入my-page.md作为文件名，并在该文件中添加以下内容：

```
---  
layout: page  
title: "My Page"  
permalink: /my-page/  
---
```

用户可以再添加一些Markdown内容并直接提交到main分支。

7. 前往/docs/\_posts/文件夹。在右上角再次选择 **Add file|Create new file**。输入YYYY-MM-DD-my-post.md作为文件名，其中YYYY是当前年份，MM是两位数字的月份，DD是该月的两位数字的日期。添加以下内容，并将日期替换为当前日期：

```
---  
layout: post  
title: "My Post"  
permalink: /2021-08-14_writing-with-Markdown/  
---
```

在页面上添加一些更多的Markdown内容，并直接提交到main分支。

8. 后台进行编译处理，然后刷新页面，就可以在起始页上看到页面和帖子，用户可以点击它们（见图4.13）。

## Accelerate DevOps with GitHub

[Get started](#) [About markdown](#) [My page](#) [About](#)

This is the homepage `index.md` of the type `home`. `Pages` are displayed in the top menu and `Posts` after this section.

Use the following page header for the homepage:

```
---
```

`layout: home`

```
--
```

To see exc excerpts of your posts and not only the heading add `show_excerpts: true` to your `_config-yml`.

## Posts

Aug 14, 2021

### My Post

This my first blog post on Jekyll !

Aug 13, 2021

### Posting in Jekyll

Creating individual blog posts in GitHub pages is easy. In this post I will show you the basics.

图4.13 - 在Jekyll上创建新页面并推送

现在可以看到在GitHub Pages中发布内容是多么容易。Jekyll是一个非常强大的工具，几乎可以定制一切，包括主题。安装Ruby和Jekyll时，还可以离线运行网站进行测试（见<https://docs.github.com/en/pages/setting-up-a-github-pages-site-with-jekyll/testing-your-github-pages-site-locally-with-jekyll>以了解更多细节）。然而，这是一个非常复杂的话题，超出了本书的范围。

使用Jekyll的GitHub页面是一种很好的方式来展示内容，并像在代码上一样通过pull request对内容进行协作。读者可以把它作为一个技术博客或用户文档使用。在一个分布式的团队中，可以用它来发布每个冲刺阶段的结果，还可以发布短视频。这有助于与他人交流成果，即使他们不能参加冲刺审查会议。

## 维基(Wikis)

GitHub在每个仓库中都有一个简单的维基，也可以选择在代码旁边创建开发者自己的基于Markdown的维基。

### GitHub的维基(wiki)

每个资源库中都有一个非常简单的维基。可以选择以不同的格式来编辑页面。Markdown, AsciiDoc, Creole, MediaWiki, Org-mode, Prod, RDoc, Textile, 或reStructuredText。由于GitHub中的其他内容都是Markdown, 本书认为这是最好的选择, 但如果已经有其他格式的维基内容, 它可以帮助把内容转移过来。

### 注意

其他的编辑格式, 如AsciiDoc或MediaWiki, 有更高级的功能, 如自动生成的目录 (ToC)。如果开发者的团队已经熟悉了语法, 可能更有意义, 但同时学习Markdown本身和另一种Markdown语言可能弊大于利。

维基是非常简单的。有一个可以编辑的主页, 开发者可以添加一个自定义的侧边栏和页脚。与其他页面的链接在双括号中被指定为[[页面名称]]。如果创建一个单独的链接文本, 可以使用[[链接文本|页面名称]]格式。如果创建了一个还不存在的页面的链接, 会被显示为红色, 开发者可以通过点击该链接来创建一个页面。

wiki是一个Git仓库, 其名称与仓库相同, 扩展名为.wiki (<name\_of\_repository>.wiki)。开发者可以克隆一个仓库, 在本地用wiki的分支工作。但到现在为止, 还没有办法使用pull request来协作修改, 这是Wiki最大的缺点, 也是GitHub wikis的最大缺点!

另外, 维基不支持嵌套页面。所有页面都在资源库的根目录。开发者可以使用侧边栏来创建一个使用Markdown嵌套列表的层次结构的菜单:

```
[[Home]]
* [[Page 1]]
  * [[Page 1.1]]
  * [[Page 1.2]]
```

如果开发者想让菜单的部分内容可折叠, 可以使用

#### ► 详细信息

GitHub Markdown功能。这将在Markdown中创建一个可折叠的部分, 通过可以定制标题:

```
* [[Page 2]]
* <details>
<summary>[[Page 2.1]] (Click to open)</summary>
  * [[Page 2.1.1]]
  * [[Page 2.1.2]]
</details>
```

请注意, 空行不能删除。结果如图4.14所示。

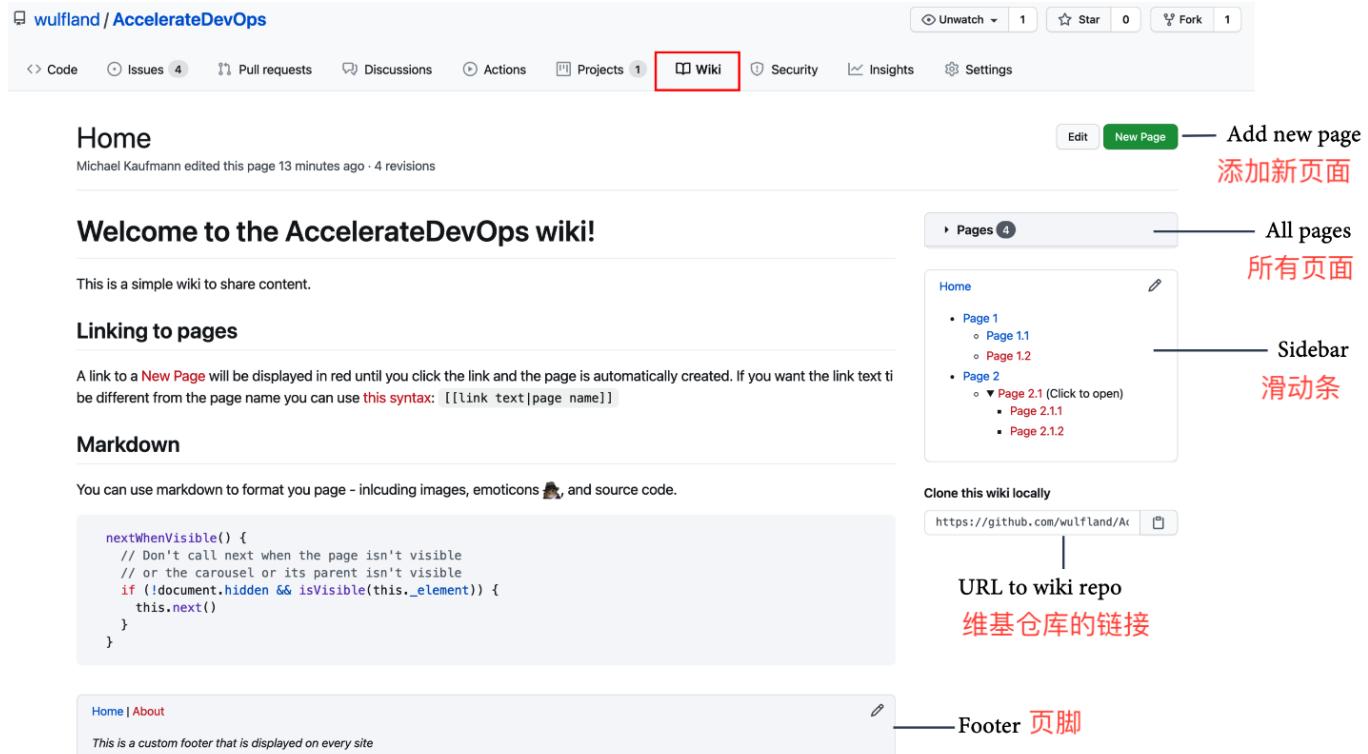


图4.14 - GitHub wiki结构

GitHub维基是一个非常简单的维基解决方案，但缺乏其他维基解决方案所具有的许多功能，尤其是不能使用pull request，这限制了它在异步工作流程中的优势。但幸运的是，可以在自己的仓库中托管Markdown，自己建立一个自定义的wiki。

### 一个自定义的维基(wiki)

如果开发者不喜欢GitHub Pages的复杂性，但又想在维基上处理pull request，则可以直接把Markdown文件放到仓库里。GitHub会为开发者所有的Markdown文件自动渲染一个自动生成的目录ToC（见图 4.15）。可能读者已经在GitHub仓库的README文件中注意到了这一点。

Auto-generated ToC for all markdown files in GitHub

对于所有markdown 自动生成目录

Header 1

- Head 1.1
- Head 1.2
- HEAD 1.2.1
- HEAD 1.2.2
- Head 1.3
- HEAD 1.3.1
- HEAD 1.3.2

an build a custom wiki usi  
n changes and you have the  
Code extensions that can

图4.15 - GitHub Markdown文件自动目录

自定义维基的问题在于导航。使用Markdown嵌套列表和相对链接来建立一个导航系统是很容易的。开发者也可以用细节使其可折叠：

```
<details>
  <summary>Menu</summary>
  * [Home] (#Header-1)
  * [Page1] (Page1.md)
    * [Page 1.1] (Page1-1.md)
    * [Page 1.2] (Page1-2.md)
  * [Page2] (Page2.md)
</details>
```

但如果开发者需要它出现在每个页面上，必须在改变它时将其复制粘贴到所有页面上。开发者可以将其自动化，但它仍然会使历史记录变得混乱。最好是在每个页面上都有类似面包屑的导航，人们可以用它来导航回到主页，并从那里使用菜单。读者可以在这里看到一个Markdown的自定义导航的例子：

[https://github.com/wulfland/AccelerateDevOps/blob/main/ch4\\_customWiki/Home.md](https://github.com/wulfland/AccelerateDevOps/blob/main/ch4_customWiki/Home.md)。

从社区论坛，到简单的Markdown维基，再到用Jekyll制作的完全可定制的网页，在GitHub上为开发者的工作托管额外内容有很多选择。为手头的工作选择合适的方案不容易。开发者需要尝试找出适合自己的团队的方法。

## 通过 GitHub Mobile 随时随地的工作

大多数时候，开发者会在浏览器上协作处理 GitHub 的issues、pull requests和讨论。但也有其他选择，可以帮助开发者随时随地浏览GitHub。

GitHub Mobile 是一款移动应用程序，可通过其市场平台 (<https://github.com/mobile>) 在安卓和苹果上使用。这款应用可以让开发者访问所有的issues、pull requests和所有仓库的讨论。它有夜晚模式和日间模式，并可以把开发者喜欢的仓库固定在开始屏幕上（见图4.16）：

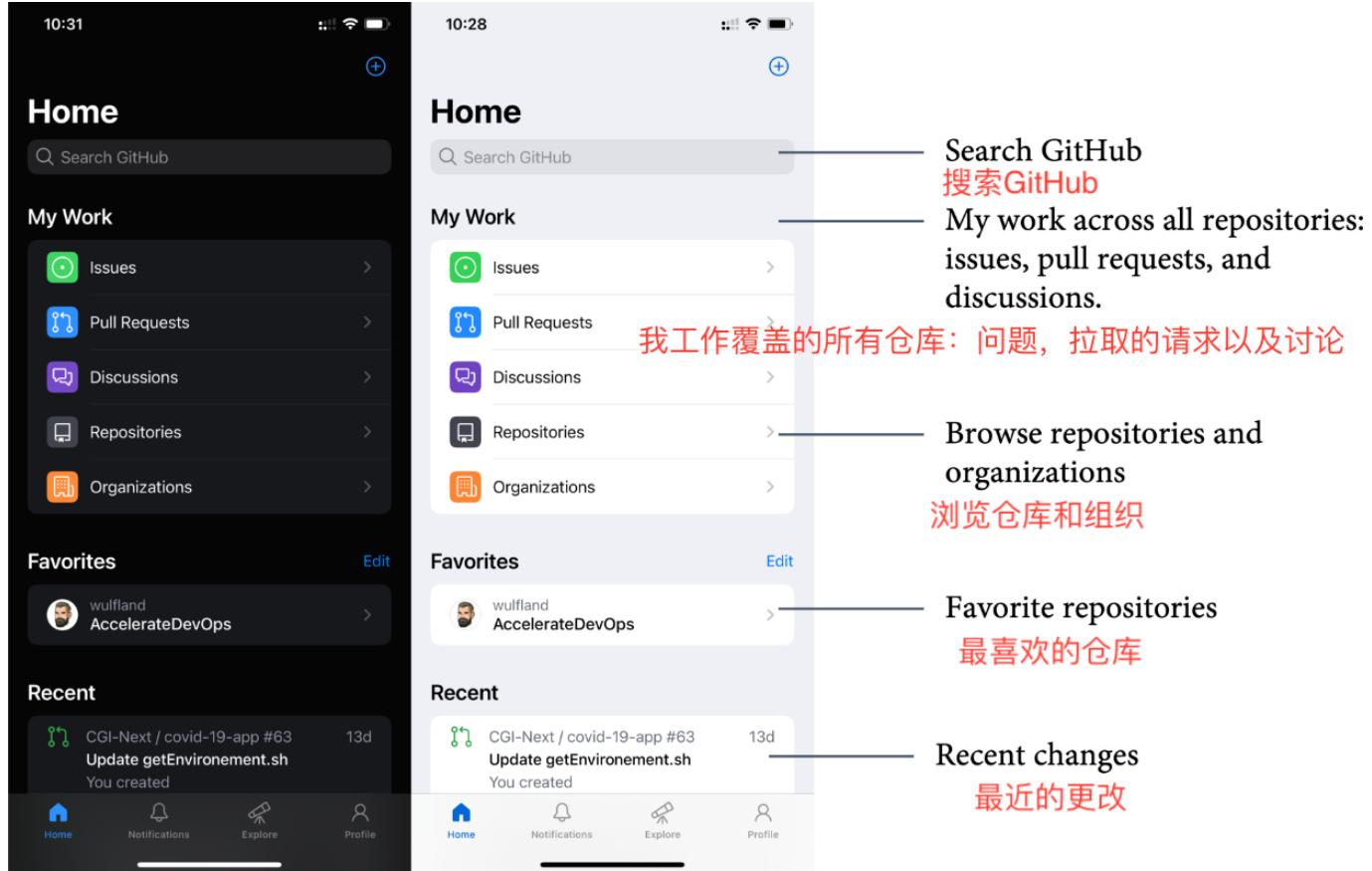


图4.16 - GitHub Mobile首页的夜间模式和日间模式

作者非常喜欢GitHub的移动应用--它做得非常好，可以帮助开发者在日常工作中在工作站或笔记本电脑以外进行issues和讨论的协作。开发者可以配置通知，这样当被提及和分配时，或者当有人要求审查时，就会得到通知。通知会出现在开发者的收件箱中，开发者可以使用可配置的轻扫动作，将通知标记为完成、阅读或未读；保存通知；或取消订阅通知源。默认的标记选项是完成和保存。收件箱界面如图4.17所示。



图4.17 - GitHub Mobile通知界面

第一次使用该应用程序时留下最深刻印象的是代码审查体验在移动设备上的效果。开发者可以打开换行功能，这样就可以很容易地阅读代码，看到改动，并对其进行评论（见图4.18）。

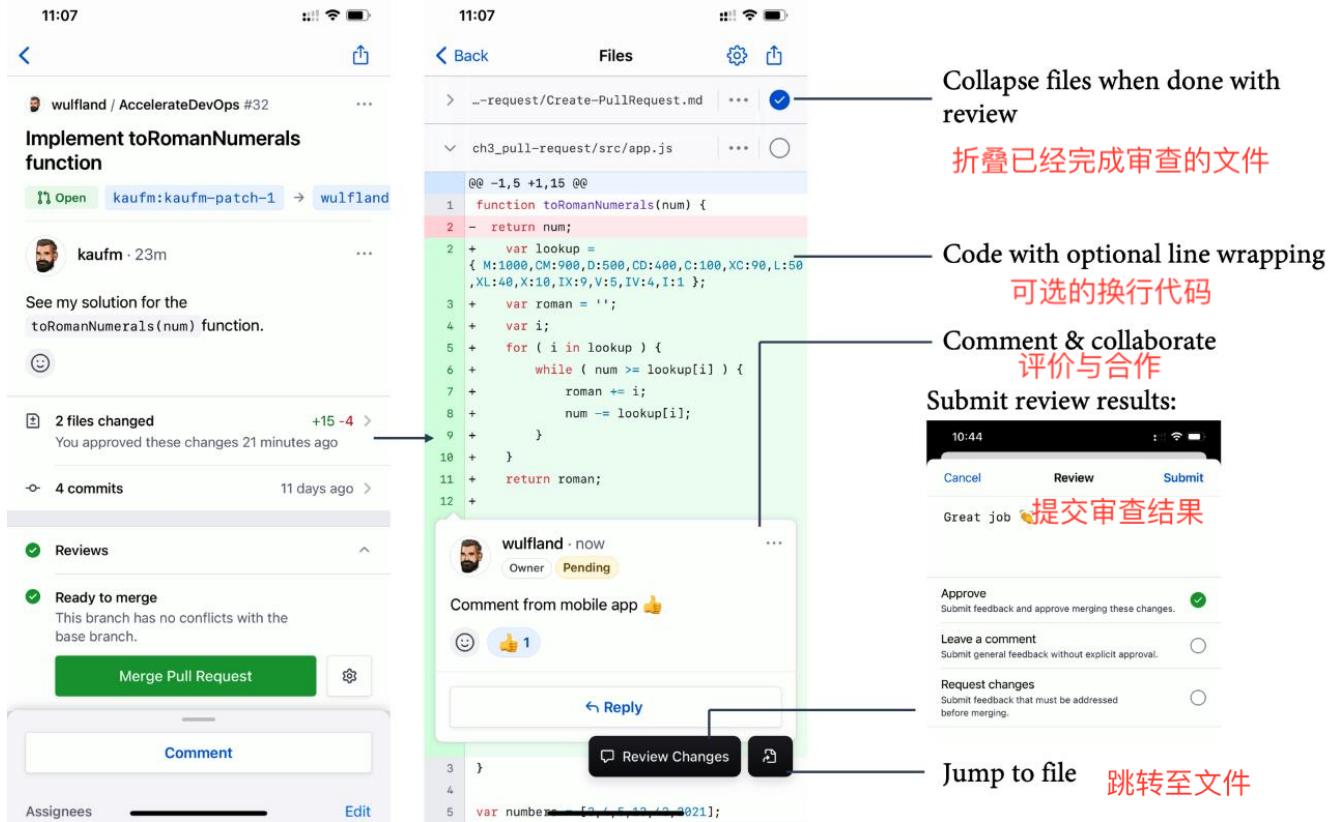


图4.18 - GitHub Mobile 上的Pull request审查

GitHub移动版是一个很好的工具，即使开发者不在办公室，也可以为团队成员解围。它允许开发者参与讨论，并对代码修改和问题进行评论。在旅途中审查小改动可能性可以帮助开发者的团队转向小批量的工作，开发者的审批等待时间更短。

## 案例研究

在Tailwind Gears的两个试点团队做的第一件事就是把他们的代码转移到GitHub仓库。一个团队已经在Bitbucket服务器上使用Git。对该团队来说，迁移就像把仓库推送到一个新的远程仓库一样容易。另一个团队使用的是团队基础服务器（TFS）的版本控制，在推送到GitHub之前，必须先在服务器上将代码迁移到Git上。两个团队都决定参加为期两天的Git培训，以便能够充分利用Git的力量，制作易于审查的良好提交。他们使用pull request草稿，以便团队中的每个人总是知道其他人在做什么，而且他们暂时设定了至少两个必要的审查员。

许多工作仍在仓库之外，例如在Word、Excel和Visio文档中，这些文档存储在公司的SharePoint服务器上。一些文件被转换为便携式文件格式（PDF），并由管理层签字确认。在发布符合某些规定的产品之前。有太多的文件需要一次性转换Markdown。这些团队在他们的代码库中创建了一个基于Markdown的自定义维基，以使所有东西都接近代码。他们添加链接到SharePoint中的当前文档。每当需要对文件进行修改时，内容就会被移到Markdown文件中，链接也会被删除。而不是签署PDF文件，管理层被添加为相应文件的代码所有者，并直接在拉取请求中批准更改。与审计日志一起，这对所有必要的合规性审计都是有效的。

当迁移到新平台时，许多方面都与两个团队有关，以后也会与其他团队有关，因为他们也会迁移到新平台上。这就是为什么他们要创建一个共享的平台仓库。该仓库包含GitHub讨论，以便与所有工程师合作，即使是那些还不在这两个团队中的人。一个技术博客是使用GitHub页面建立的，以分享技巧和窍门。Jekyll网站也被用来合作制定共同的审查准则和行为守则。

## 总结

本章介绍了同步和异步工作的优势和劣势。读者可以用它来创建有效的异步工作流程，从而实现更好的跨团队协作，使远程和混合团队能够跨越多个地区和时区。通过本章，可以了解到GitHub讨论、Pages和维基如何帮助开发者为代码和需求以外的主题制定异步工作流程。

下一章将解释开放和内部资源对软件交付性能的影响。

## 阅读和参考

以下是本章的参考资料，可以通过它们获取更多信息：

- History of communication: [https://en.wikipedia.org/wiki/History\\_of\\_communication](https://en.wikipedia.org/wiki/History_of_communication),  
<https://www.g2.com/articles/history-of-communication>, and <https://www.elon.edu/u/imagining/time-capsule/150-years/>
- History in general: <https://www.dhm.de/lemo/kapitel> (German)
- World population growth: <https://ourworldindata.org/world-population-growth>
- Hybrid work: <https://www.microsoft.com/en-us/worklab/work-trend-index/hybrid-work>
- Work trend index: <https://www.microsoft.com/en-us/worklab/work-trend-index>
- GitHub Discussions: <https://docs.github.com/en/discussions>
- GitHub Pages: [<<<<< HEAD](https://docs.github.com/en/pages)
- GitHub Mobile: [=====](https://github.com/mobile)
- GitHub Mobile: <https://github.com/mobile>



# 第5章 开源和内源对软件交付性能的影响

2001年6月1日，微软前首席执行官史蒂夫·鲍尔默在接受《芝加哥星期日泰晤士报》的采访中提出：

“Linux可以说是一种癌症，从知识产权的意义上来讲，它会把自己附着在它所接触的一切事物上。”（Greene T.C. (2001)）

他关注的不仅仅是Linux，而是所有的开源许可证。今天，微软超过了Facebook、Google、Red Hat和SUSE成为了世界上最大的开源贡献者。微软不仅有许多开源产品，例如PowerShell、Visual Studio Code和.NET，而且还为Windows 10提供了一个完整的Linux内核，以便用户可以在其上运行任何发行版本。微软总裁布拉德·斯米特承认，“当开源在本世纪初呈爆发式出现时，微软站在了历史的错误一边”（沃伦·T (2020)）。

如果读者去了解对开源贡献最大的10家企业，会发现里面有所有制造商业软件的大型科技企业：

	Company	Active Contributors	Total Community
1	Microsoft	5,368	10,924
2	Google	4,907	9,635
3	Red Hat	3,211	4,738
4	IBM	2,125	5,062
5	Intel	1,901	3,982
6	Amazon	1,742	4,415
7	Facebook	1,350	4,017
8	GitHub	1,122	2,871
9	SAP	811	1,606
10	VMware	786	1,604

表5.1 -开源贡献指数，2021年8月2日(<https://opensourceindex.io/>)

到底在过去二十年发生了什么变化，以至于有影响力的科技企业现在都开始拥抱开源？

本章将介绍自由和开源软件的历史，以及为什么在过去几年中它变得如此重要。本章还将介绍开源对读者的项目速度影响，以及在企业如何使用开源的原则来实现更好的跨团队协作（内部开源）。

本章将涵盖以下主题：

- 自由和开源软件的历史
- 开源与开放开发的区别
- 企业采用开源的好处
- 实施开源战略
- 开源和内部开源
- 内包的重要性
- GitHub赞助商

# 自由和开源软件的历史

如果想要理解开源，必须回到计算机科学的早期。

## 公共领域的软件

在20世纪50年代和60年代，与必要的硬件相比，软件的价格很低，主要由学者和企业研究团队开发。源代码与软件一起发布是很正常的一—通常都会成为公共领域软件。这意味着软件是免费的，开发者没有所有权、版权、商标或专利。这些开放与合作的原则对当时的黑客文化产生了巨大影响。

在20世纪60年代末，操作系统和编译器的兴起增加了软件成本。这是由不断增长的软件行业与硬件供应商竞争所推动的，硬件供应商会将其软件与硬件捆绑在一起销售。

在20世纪70年代和80年代，销售软件使用许可证开始变得普遍，1983年，IBM停止将其源代码与购买的软件一起发布。

## 自由软件

理查德·史泰尔曼认为这在道德上是错误的，他于1983年创立了GNU项目，不久后他创立了自由软件运动。

自由软件运动认为，如果软件的使用者被允许执行以下操作，则认为软件是自由的：

- 可以出于任何目的运行程序
- 研究软件并以任何方式更改它
- 重新发布程序并制作副本
- 改进软件并发布改进

Richard在1985年创建了自由软件基金会（FSF）。FSF因为以下说法闻名：

“自由是指像言论自由一样，而非是畅饮啤酒那样。”

这意味着“自由”一词意味着分配的自由，而不是成本的自由（开源是自由的，而不是免费的）。由于大部分自由软件已经是免费的。软件（自由软件）与免费软件和零成本有关联。

自由软件运动创造了一个概念叫做copyleft。这授予用户使用和修改软件的权利，但保留了软件的自由状态。这些许可证的示例包括GNU通用公共许可证（GPL）、Apache许可证和Mozilla公共许可证。

今天仍在数百万设备上运行的大多数优秀软件都是使用这些版权许可证发布的；例如，Linux内核（由Linus Torvalds于1992年发布）、BSD、MySQL和Apache。

## 开源软件

1997年5月，在德国维尔茨堡举行的Linux大会上，埃里克·雷蒙介绍了他的论文《The Cathedral and the Bazaar》（Raymond, E.S.1999）。他在论文里介绍了自由软件原则、黑客文化以及软件开发的好处。这篇论文引起了大量关注，并促使Netscape企业将其浏览器Netscape作为免费软件发布。

Raymond和其他人希望将自由软件原则介绍给更多的商业软件供应商，但“自由软件”一词对商业软件企业也有负面影响。

1998年2月3日，自由软件运动的许多重要人士在帕洛阿尔托举行了一次战略会议，在会议上讨论了自由软件的未来。与会者包括埃里克·雷蒙、迈克尔·铁曼和克里斯汀·彼得森，他们提出了开源一词，支持自由软件。

开放源代码倡议（OSI）是由埃里克·雷蒙德和布鲁斯·佩雷斯于1998年2月创立的，雷蒙德担任第一任主席（OSI 2018）。

1998年，在出版商蒂姆·奥莱利（Tim O'Reilly）的历史性自由软件峰会（后来被命名为开源峰会）上，该术语迅速被早期支持者采纳，如莱纳斯·托瓦尔兹（Linus Torvalds）、拉里·沃尔（Larry Wall）（Perl的创建者）、布莱恩·贝伦多夫（Brian Behlendorf）（Apache）、埃里克·奥尔曼（Eric Allman）（Sendmail）、吉多·范·罗森（Guido van Rossum）（Python）和菲尔·齐默曼（Phil Zimmerman）（PGP）（O'Reilly 1998）。

但是Richard Stallman和FSF拒绝了新的开源术语（Richard S.2021）。这就是为什么自由开源软件（FOSS）运动存在分歧，今天仍然使用不同的术语。

20世纪90年代末和21世纪初，在网络泡沫中开源和开源软件（OSS）这两个术语被公共媒体广泛采用，并最终成为更流行的术语。

## 开源软件的兴起

在过去的二十年中，开源越来越流行。Linux和Apache等软件很大程度上驱动着互联网的发展。一开始OSS很难进行商业化，第一个想法是围绕开源产品提供企业级支持服务。在这个方面做的比较成功的企业是红帽和MySQL，但是具体实施很困难，而且没有商业许可所提供的规模。因此，大量投资于构建OSS的开源企业开始创建开放核心产品：一种免费的开源核心产品，以及商业附加组件，客户可以购买这些产品。

软件商业模式从传统许可证向软件即服务（SaaS）订阅的转变帮助开源企业将其OSS商业化。这促使传统软件供应商发布他们的软件——至少是核心软件——开源来和社区互动。

不仅微软、谷歌、IBM和亚马逊等大型软件企业成为了大型开源企业，而且像红帽和MuleSoft这样的纯开源企业也获得了很多价值和市场认可。例如，红帽于2018年被IBM以320亿美元收购。同年，Salesforce以65亿美元收购了MuleSoft。

因此，今天的开源并不是来自于创造替代自由软件的革命思想。为云提供商提供软件和平台服务的大多数顶级软件都是开源软件（Volpi M.2019）。

## 开源与开放开发的区别

OSS指的是根据许可证发布的计算机程序，该许可证授予用户使用、研究、修改和共享软件及其源代码的权利。

但将源代码公开在版权许可下只是第一步。如果一家企业想要拥有开源的所有收益，它必须采用开源的价值观，这就引出了一种叫做开放开发的东西，意味着开发者不仅可以访问源代码，而且开发者必须使整个开发和产品管理透明。这包括：

- 需求
- 架构和研究
- 会议
- 标准

.NET团队是一个很好的例子，他们在Twitch和YouTube上主持社区站（<https://dotnet.microsoft.com/live/community-standup>）。

开放式发展还意味着创造一个开放和包容的环境，让每个人都能安全地提出变革。这包括一个强大的道德规范和一个干净的代码库，该代码库高度自动化，允许每个人快速轻松地做出贡献。

# 为企业提供开源的好处

开源如何与更好的开发绩效相联系，以及企业如何从良好的开源战略中获益？

## 使用开源软件更快地交付

根据来源的不同，新产品已经包含70%到90%的开源代码。这意味着开发者自己编写的代码将减少70%到90%，可以显著减少开发者的上市时间。

除了在产品中重用开源代码之外，还有很多平台工具可以作为开源工具使用。可重复使用的GitHub操作、测试工具或容器编排等等。在大多数情况下，开发者可以使用开源软件来更快地交付软件。

## 通过吸引社区参与，打造更好的产品

如果开发者在公开环境中开发产品的某些部分，读者可以利用社区的蜂巢思维来构建更好、更安全的软件。它还可以帮助读者从世界各地的优秀工程师那里获得有关读者正在做什么的早期反馈。

特别是对于复杂、关键和安全相关的软件，与社区合作通常会带来更好的解决方案：

“问题越大，开源开发人员就越被吸引，就像磁铁一样，一起致力于解决这个问题。”

(Ahlawat P.、Boyne J.、Herz D.、Schmieg F.和Stephan M. (2021) )

## 使用弃用风险较低的工具

使用开源可以降低工具过时的风险。如果读者可以自己构建工具，那么必须自己维护它们——这不是读者的首要任务。使用小供应商提供的工具或由合作伙伴构建工具会带来无法维护或合作伙伴退出市场的风险。相反，选用开源工具可以显著降低这些风险。

## 吸引人才

让工程师能够在工作中利用开源，并在工作时间为开源项目做出贡献，这会对企业的招聘能力产生重大影响，参与社区并参与开源将有助于吸引人才。

## 影响新兴技术和标准

许多新兴技术和标准都是公开开发的。对这些计划的贡献可使企业能够影响这些技术，并成为前沿开发的一部分。

## 通过学习开源项目来改进流程

如果接受开源，企业可以学习协作开发，并应用这些原则来改善企业内部的跨团队协作（称为内部开源）。

## 实施开源战略

尽管拥抱开源的好处很多，但也有一些风险必须解决。在产品和工具链中使用开源软件时，必须严谨并且遵守许可证。如果开源组件造成损害，开发者还必须自己承担责任，因为开发者没有可以起诉的供应商。此外，如果承担了太多的依赖关系（直接或间接），其中一个依赖关系中断，也会带来风险。

注意

在第14章中读者将了解软件包中的11行代码和一个名称的冲突是如何造成严重破坏并摧毁互联网的大部分内容的。

这就是为什么企业应该制定开源战略。该策略应该定义什么类型的开源软件开发人员可用于何种目的。对于不同的目的，可能有不同的规则。如果想在产品中包含开源，企业将需要某种治理策略来管理相关风险。

该策略还应定义是否允许开发人员在工作时间为开源做出贡献，以及这方面的条件。

本章节将不深入探讨该战略的细节。这在很大程度上取决于企业计划如何使用开源以及如何开发和发布产品。只要确保企业有一份关于开源战略的文档（即使它一开始很简短）。它将随着开源的成熟度和经验的增长而发展。

本章节的一个建议是实施一个精英中心或社区，它帮助读者制定一个策略，如果开发人员有疑问或不确定开源组件是否符合要求，他们可以求助于该策略（Ahlawat P.、Boyne J.、Herz D.、Schmieg F.、Stephan M.2021）。

## 开放和内部开源

开源的成功在于其开放和协作的文化。让合适的人自愿进行远程异步协作，有助于以最佳方式解决问题。原则如下：

- 开放式协作
- 开放式沟通
- 代码评审

将这些原则应用于组织内的专有软件称为内部开源。这一术语从2000年开始由蒂姆·奥莱利（Tim O'Reilly）提出。内部开源可以是打破与企业其他部分隔绝的部分并促进团队和产品之间强大协作的好方法。

但是像开源和开放开发一样，仅仅使代码可用不足以创建内部开源文化。许多因素影响内部开源方法能否成功：

- 模块化产品架构：如果企业有一个大型的整体架构，这将使人们无法做出贡献。此外，代码的质量、文档以及开发者理解代码和贡献的速度对内部源代码的采用有很大影响。
- 标准化工具和流程：如果每个团队都有一个工具链和工作流，需要避免其他工程师参与其中。拥有一个通用的工程系统和类似的分支和CI/CD方法将有助于其他人专注于问题，而不必首先学习其他工具和工作流。
- 自主性和自组织性：只要组织向团队提出需求，并且工程师忙于在他们的截止日期完成工作，那么对其他团队的贡献就不会发生。只有当团队能够自主地确定优先级并以自组织的方式工作时，他们才能自由地参与其他社区——无论是开源还是内部开源。

内部开源可以帮助打破企业部门之间的壁垒，提高工程速度。但这也与高水平的DevOps成熟度有关。内部开源是随着企业的DevOps能力和开源成熟度的提高而发展的。因此，将其视为对于企业的加速是输出而不是输入。

### 注意

从技术上讲，内部开源通常是通过在企业中激活派生来完成的。这与分支工作流紧密相连，本书第11章将详细介绍。

## 内包的重要性

许多企业认为软件开发不是他们的核心业务，因此他们倾向于将其外包。外包意味着一家企业雇佣另一家企业或自由职业者来执行特定的功能。外包通常不是一个坏方法：如果企业有另一家专门从事一项技术的企业帮助做这些工作，这样就可以把自己的员工和投资专门放在核心产品上。专业企业通常会做得更便宜、更好——而自己培养这些技能的人才可能需要花费大量时间和金钱。

但现在软件基本上是所有产品的关键区别。不仅是数字客户体验，智能制造和供应链管理也能带来竞争优势。定制软件正在成为核心业务的一部分。由于这一点，许多企业已经制定了软件开发的内包战略——即内部招聘和雇佣软件开发人员和DevOps工程师。

问题是软件开发人员和DevOps工程师的市场竞争激烈（所谓的人才争夺战）。这通常会导致合作伙伴在核心产品上工作，开发人员维护工具的分散局面。

一个好的内包策略是问问自己，软件是否是业务的核心，也就是说，它是否能带来竞争优势：

- 核心软件应由内部开发人员开发。如果不能雇佣足够熟练的开发人员，可以与你信任的合作伙伴之一的工程师共同寻找并扩充你的员工。但目标应该始终是以后用自己的工程师替换这些开发人员。
- 辅助软件可以外包。在最好的情况下，可以使用现有的产品。如果没有，可以让合作伙伴构建它。这里是开源发挥作用的地方：企业可以利用现有的开源解决方案，或者让合作伙伴在开放环境中构建解决方案。这降低了企业成为唯一客户和解决方案过时的风险。由于软件只是对业务的补充，不在乎其他企业是否使用它。相反，核心软件使用的越多，软件过时的风险就越小。此外，如果软件是在操作环境中开发的，质量是可靠的。

支付给其他企业或个人为自己开发特殊的开源软件，或为现有的开源解决方案添加功能并不常见。但随着越来越多的企业采用内包战略，以及人才争夺战的持续，这将在未来几年显著呈现。

## GitHub赞助商

一开始，开源策略似乎与内包策略相冲突，但问题更为复杂。对于核心软件来说，为开源项目提供一个小功能可能比自己实现一个解决方案更有用。但在许多企业，团队层面在自己做还是向别人买的决策中总是倾向于做，因为用金钱购买或资助某些东西的过程太复杂。一个好的内包策略应该始终包括一个轻量级和快速的过程，并在工具和软件供应链上投入一定的预算。如果企业内部开发人员不足，购买软件或赞助开源贡献者应该没有问题。

让团队能够投资于开源项目的一个好方法是利用一个名为GitHub赞助商的功能。它允许您投资于产品依赖的项目（软件供应链），并保持这些项目的活力。它还可以让维护人员自由地编写新请求的特性，而不必自己实现它们。

一个积极的副作用是赞助对开源社区来说是显而易见的。这是一个很好的营销，增长了企业信誉，可以帮助吸引新的人才。

如果个人开发者或组织是GitHub赞助商计划的一部分，可以为他们提供赞助。也可以代表组织赞助他们。这种赞助可以是一次性或每月支付，并且可以在个人资料或组织的个人资料中看到（见图5.1）：

The screenshot shows the GitHub profile page for the curl project. At the top right, there is a 'Sponsor' button with the text 'Sponsor a project or person'. Below it, there are sections for 'Pinned' repositories: 'curl' (a command-line tool and library for transforming data with URL syntax), 'curl-www' (the curl and libcurl website contents), and 'curl-up' (all things related to the curlup conference series). To the right, there is a 'People' section showing profiles of contributors and a 'Sponsors' section showing logos of organizations that have sponsored the project. At the bottom left, there is a 'Repositories' section with a search bar and filters for 'Type', 'Language', and 'Sort'.

Sponsor a project or person 赞助一个项目或者个人

Sponsors of a project or person 一个项目或者个人的赞助者

### 图5.1 – GitHub赞助商中启用的组织配置文件

GitHub赞助商不会从用户帐户收取任何赞助费用，因此这些赞助中的100%归受赞助的开发者或组织所有。

### 赞助商级别

赞助商可以设置不同的赞助级别。这可以通过一次性赞助或者是每月定期付款来实现（见图5.2）：

**Dennis Doomen**  
denniscoomen  
The Hague, Netherlands

I'm the author of [Fluent Assertions](#), a very popular .NET assertion framework, [Chill](#), a BDD-style testing framework that helps you write better unit tests, [Liquid Projections](#), a set of libraries for building Event Sourcing architectures and I've been maintaining [Coding Guidelines for C#](#) on since 2001. I also keep a blog on my [everlasting quest for better solutions](#) and you can reach me on Twitter through [@ddoomen](#).

I use my sponsor funds to:

- explain my wife that those thousands of hours I spent in the evenings were not wasted
- domain names for all the projects I maintain
- development licenses and tools
- help me save for a better development rig

I also share a part of the funds with my partner-in-crime [Jonas Nyrup](#), who has been helping maintain Fluent Assertions for the last three years.

You are one of 5 sponsoring denniscoomen!

Featured work:

- [FluentAssertions/fluentassertions](#): Fluent API for asserting the results of unit tests that targets .NET Framework 4.6, 4.7, .NET Standard 1.3, 1.6 and 2.0. Supports the latest frameworks: MSTest, MSTest2, Gallio, NUnit, xUnit, NHUnit.
- [denniscoomen/CSharpGuidelines](#): A set of coding guidelines for C# 5.0, C# 6.0 and C# 7.0, design principles and layout rules for improving the overall quality of your code development.

Sponsoring as [wulfand](#)

Hover to review the badge you'll get that shows @denniscoomen you're a sponsor.

Select a tier: [Monthly](#) [One-time](#)

**\$5 a month** [Select](#)

**Durasteel Monthly Sponsor**

For a sponsorship at this level you'll receive:

- A badge on your profile
- A shoutout on Twitter

**\$10 a month** [Select](#)

**Carbonite Monthly Sponsor**

For a sponsorship at this level you'll receive:

- A badge on your profile
- A shoutout on Twitter
- Your name on a repository of your choice

**\$25 a month** [Select](#)

**Beakar Monthly Sponsor**

For a sponsorship at this level you'll receive:

- A badge on your profile
- A shoutout on Twitter
- Your name on a repository of your choice
- Your name on the landing page of the project of your choice

Sponsor as a yourself for on behalf of your organization 代表组织作为自己的赞助商

Sponsoring can be monthly or one-time 赞助可以是每月或一次性的

Select one of the tiers the owner has created 选择所有者创建的层之一

图5.2 – 月度或一次性赞助的选项

所有者每月最多可以设置10层，一次性付款最多可设置10层。这使他们能够将制定的回报链接到不同的层。例如：

- 知名度：可以在网站或社交媒体上提及赞助商。也可能有徽章（如银牌、金牌和白金赞助商）用于区分不同级别的赞助。
- 访问：赞助商可以访问私有存储库或早期版本。
- 优先排序：赞助商的bug或功能请求可以优先排序。
- 支持：一些赞助商也为解决方案在一定程度上提供支持。

## 赞助目标

赞助账户可以设定融资目标。目标可以基于赞助商数量或每月的赞助金额（美元），并显示在赞助页面上（见图5.3）：

## Become a sponsor to Jan De Dobbeleer

The screenshot shows the GitHub Sponsors page for Jan De Dobbeleer. At the top, it says "Become a sponsor to Jan De Dobbeleer". Below that is a profile picture of Jan De Dobbeleer, his name, and his location: Diest, Belgium. It also mentions he is a CTO, Microsoft MVP, GitKraken Ambassador, and creator/maintainer of `oh-my-posh`. A progress bar indicates "90% towards 10 sponsors goal" with "10ch and 8 others sponsor this goal". To the right, there is a section titled "Goal for amount of money or number of sponsors". Below this, a dropdown menu shows "Sponsor as wulfand". A tooltip says "Hover to review the badge you'll get that shows @JanDeDobbeleer you're a sponsor." Under "Select a tier", there are four options: "\$10 a month" (selected), "\$1 a month", "\$3 a month", and "\$10 a month" again. Each tier has a "Select" button and a small description: "\$10 a month" says "a friend wouldn't hesitate 😊", "\$1 a month" says "Buy me a coffee ❤️", and "\$3 a month" says "Some more coffee? 😊".

Goal for amount of money or number of sponsors 对于金额或者是赞助商数量的目标

图5.3 – Python的赞助目标是每月获得12000美元

赞助目标可以与某些里程碑联系起来。例如，维护人员可以在他们辞去日常工作并开始全职工作时设置一定的金额。组织还可以设置雇佣新开发人员以帮助维护项目所需的金额。

## 总结

在本章中，读者了解了自由和开源软件的历史、价值观和原则，以及它对软件交付性能的影响。一个好的开源战略，再加上一个良好的内包战略，以及团队赞助和资助开源项目的能力，可以帮助企业显著缩短上市时间，并让工程师开发对企业至关重要的功能。将这些原则应用于企业内部资源，可以帮助建立协作文化，实现更好的跨团队协作。

在下一章将学习GitHub操作的自动化。

## 进一步阅读和参考

有关本章内容的更多信息，请参阅以下材料：

- Greene T. C. (2001). Ballmer: Linux is a cancer:  
[https://www.theregister.com/2001/06/02/ballmer\\_linux\\_is\\_a\\_cancer/](https://www.theregister.com/2001/06/02/ballmer_linux_is_a_cancer/).
- Warren T. (2020). Microsoft: we were wrong about open sourc:  
<https://www.theverge.com/2020/5/18/21262103/microsoft-open-source-linux-history-wrong-statement>.
- Raymond, E. S. (1999). The Cathedral and the Bazaar: Musings on Linux and OpenSource by an Accidental Revolutionary. O'Reilly Media.
- O'Reilly (1998). FREEWARE LEADERS MEET IN FIRST-EVER SUMMIT O'Reilly

- Brings Together Creators of Perl, Apache, Linux, and Netscape's Mozilla (Press Release): <https://www.oreilly.com/pub/pr/636>.
- OSI (2018). Open Source Initiative - History of the OSI: <https://opensource.org/history>.
- Richard S. (2021). Why Open Source Misses the Point of Free Software: <https://www.gnu.org/philosophy/open-source-misses-the-point.en.html>.
- Volpi M. (2019). How open-source software took over the world: <https://techcrunch.com/2019/01/12/how-open-source-software-took-over-the-world/>.
- Ahlawat P., Boyne J., Herz D., Schmieg F., & Stephan M. (2021). Why You Need an Open Source Software Strategy: <https://www.bcg.com/publications/2021/open-source-software-strategy-benefits>.
- Inner Source: [https://en.wikipedia.org/wiki/Inner\\_source](https://en.wikipedia.org/wiki/Inner_source).
- GitHub Sponsors: <https://github.com/sponsors>.