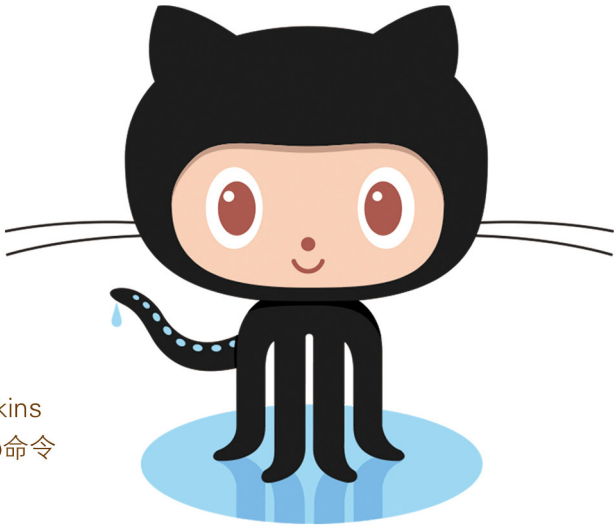
TURING 图灵程序设计丛书

与全世界程序员分享你的代码

GitHub 入门与实践

[日] 大塚弘记 / 著
支鹏浩 刘 斌 / 译



版本管理・分支・合并
Fork・Issue・BUG管理
任务管理・Markdown
代码审查・社会化编程・Jenkins
GitHub Flow・Git Flow・hub命令

中国工信出版集团
人民邮电出版社 POSTS & TELECOM PRESS

● 译者序

“开源”一词在我国 IT 界已经出现了不少年头，但“社会化编程”想必没有多少人接触过。于是在阅读正文之前，容我越俎代庖替作者问一个问题：各位在狭小的空间里呆上一段时间之后，再出门时是否有一种豁然开朗的感觉？相信很多人的答案都是肯定的。对于对日外包出身的我来说，“社会化编程”就给了我这种感觉。或许外包行业在 IT 界只是极端个例，但“让全世界码农看自己的代码”这种事，很多人恐怕想都不敢想吧。

GitHub 正是这样一个平台，我们在这里可以与全世界的开源开发者交流代码或心得。如果您对某款开源软件的源代码感兴趣，如果您想为中意的软件出一份力，如果您自己编写了小程序却苦苦找不到人指点，如果您想跟慕名已久的 IT 界明星（俗称“大神”）聊上几句，那么 GitHub 欢迎您。

GitHub 的纯英文界面或许会令您望而却步，不过不用担心，本书秉承了日系技术书刊一贯的“手把手教学”风格，作者用亲切的语言，简明扼要的介绍，配以生动详实的示例为我们一步步讲解 GitHub 的使用方法，带我们在实践中学习 GitHub。值得一提的是，本书配有一个供各位实践的网站，请感兴趣的读者务必一试。俗话说“读万卷书不如行万里路”，跟着作者一边实践一边阅读本书，相信各位会对这句话有一个更深刻的体会。

有些读者可能要问了，代码是企业的财产，不能随便发到网上给别人看，那 GitHub 对工作又有什么意义呢？这一点作者自然考虑到了。GitHub 面向社会化编程，我们所生活的是一个大社会，我们工作的企业同样是一个小社会，虽然不能强行导入“社会化编程”，但其管理模式仍然值得借鉴。所以如果您是企业的决策者，那么请在本书后半跟随作者一起探讨企业导入社会化编程的利弊，说不定能为您所在的企业带来新的利益。

《GitHub 实战入门》是国内比较少见的对 GitHub 及社会化编程进行系统介绍的书籍。以往我们对于这方面知识，只能通过网络上零零散散

的博客或技术文档进行片面了解，难以把握其全貌。各位读完这本书后相信能得到不少帮助。

最后，对另一位帮忙搭建本书相关网站的译者以及图灵文化的各位编辑致以衷心的感谢，正是有了各位的共同努力，本书才得以出版。同时感谢正在阅读本书的您，有了您的支持，本书才能发挥其价值。

支鹏浩

2015 年 4 月 于北京

● 序言

当今世界有众多开发者在使用 GitHub 进行开发。本书旨在指导各位读者在开发现场如何使用 GitHub 进行高效开发。因此，书中除针对 GitHub 进行讲解外，也涉及了开发流程及相关辅助工具的解说。

您在开发现场有没有遇到过以下几件事？

- 代码审查不到位，审查效率低下
- 只有编程者本人能看懂的代码、可靠性不高的代码直接被部署至正式环境中
- 因键入错误、理解错误而造成的低级代码错误导致 BUG 频繁出现
- 没有机会和其他人互相交流代码，共享知识，相互学习、指正、改善
- 没有一个简单高效、能在一天之内添加多个功能的开发流程

GitHub 为我们提供了解决这些问题的机会和功能，而本书则凝练了各种运用 GitHub 的诀窍。

笔者曾为多家企业引入 GitHub，改善其开发流程。本书总结了这些经验，相信能为改善您的开发现场提供一些帮助。

•·····谢辞

本书在编撰过程中得到了多方的大力支持。特此鸣谢 @yamanetoshi、增田贵士（@masutaka）、bakorer、山科佑贵、寺田涉、Tatsuma Murase、杉野康弘、泽义和（排名不分先后）。

另外，长期以来，技术评论社的池田大树为本书的编辑与整理尽心尽力，在此由衷地表示感谢。

2014 年 2 月　大塚弘记

● 本书结构

本书由 10 章及 2 个附录构成。

第 1 章：欢迎来到 GitHub 的世界

讲解 GitHub 是什么，以及有哪些革新之处。在开源软件的世界中，GitHub 为开发者带来了革命性的社会化编程概念。在这里我们将会接触这一概念，并对其带来的优势与功能进行讲解。

第 2 章：Git 的导入

要使用 GitHub，离不开 Git 这一版本管理系统。本章将深入介绍关于 Git 的知识，加深各位对 Git 的理解，同时说明实际操作的相关流程。

第 3 章：使用 GitHub 的前期准备

使用 GitHub 需要开设账户（免费），因此我们将按照顺序为您讲解正式使用前需要进行的一系列设置。

另外，本章还会讲解包括操作示例在内的，实际在 GitHub 上创建仓库并发布代码的相关流程。

第 4 章：通过实际操作学习 Git

在实际操作中学习使用 GitHub 时所必需掌握的 Git 的基本知识和操作方法。

从最基本操作到多人开发时所需的复杂操作，读者都可以随着本章的讲解简单实践一番。

第 5 章：详细解说 GitHub 的功能

本章我们将以图配文，对 GitHub 的功能逐一进行讲解，同时还会详细解说其作为源代码查看器的功能，带您领略方便快捷的 UI。

建议正在使用 GitHub 的开发者也读一读本章，您或许会发现一些将来能用到的小技巧。

第 6 章：尝试 Pull Request

Pull Request 是 GitHub 的代表功能，本章我们将带您亲自动手体会。请务必参考本书内容试着进行一次 Pull Request。

第 7 章：接收 Pull Request

站在仓库维护方的角度，教您在接到 Pull Request 之后应该如何考虑，如何判断，以及该进行哪些操作。

第 8 章：与 GitHub 相互协作的工具及服务

前半部分为您讲解通过 CLI 对 GitHub 进行操作时所需的 hub 命令。另外，在持续集成环境方面，将讲解可与 GitHub 结合使用的 Travis CI 及 Jenkins 的构建及设定方法。

除此之外，本章还会介绍一些能够与 GitHub 共同使用的服务。

第 9 章：使用 GitHub 的开发流程

详细讲解以 GitHub 为中心进行开发的 GitHub Flow、Git Flow 两个开发流程。从两者共通的团队开发心得到各自开发流程的特征，都可以通过本章的讲解实际动手体会。

第 10 章：将 GitHub 应用到企业

总结在企业中采用 GitHub 时需要考虑的问题及一些有用的信息。安全保障、故障信息、事前需要考虑的问题、GitHub Enterprise 的讨论等，这些实际引入 GitHub 时需要考虑或者了解的知识将在本章中进行讲解。

附录 A：辅助 GitHub 的 GUI 客户端

团队中并不是每个人都对 CLI 得心应手。因此，我们为读者总结了辅助 GitHub 的 GUI 客户端的相关知识。

附录 B：通过 Gist 轻松实现代码共享

Gist 能帮助开发者轻松与其他人共享简单的代码示例或日志，我们将在这部分对 Gist 进行讲解。利用 Gist 可以轻松管理日常的小代码片段。

本书内容以敝社《WEB+DB PRESS》Vol.69 的特辑《详解 GitHub——使用 Pull Request 打造高效率的软件开发》[①]为基础，进行大篇幅扩展与修正后作为图书出版。

本书的操作示例是在以下环境中进行的。

- OS X 10.9.1
- git 1.8.5.2

部分 Windows 相关解说中使用了 Windows 8。另外，GitHub 相关解说皆以 2014 年 2 月时的版本为基准。

由于环境和时期不同，操作顺序、页面、运行结果可能会存在差异。

本书中出现的示例仓库，现阶段主要由译者及尝试 Pull Request 的各位读者进行维护。但是在本书出版后，随着时间推移，可能会发生反应变慢甚至没有反应的情况。烦请参照第 7 章的内容以及关于示例仓库的讲解，一同努力维护。

对于您应用本书内容所产生的后果，本书作者、软件开发方及供应方、技术评论社、人民邮电出版社及译者概不负责，特在此声明。

本书中提及的公司名、制品名，皆是各公司实际使用的注册商标或商标。在正文中并未添加™、©、® 标志。

关于本书的补充信息与勘误等，请参考以下网址。
http://www.ituring.com.cn/book/1581

① 詳解 GitHub——Pull Request が織りなす効率的ソフトウェア開発，WEB+DB PRESS vol.69，技术评论社。——译者注

● 目录

第6章

尝试Pull Request

按部就班地创建 GitHub 账号并公开自己的源代码并不是什么难事。不过，刚刚接触 GitHub 的人往往不会或不敢使用 Pull Request 功能。

Pull Request 是社会化编程的象征。GitHub 创造的这一功能，可以说给开源开发世界带来了一场革命。不会用这个功能，就等于不会用 GitHub。

不过，掌握 Pull Request 的难度确实较高，刚刚接触 GitHub 的人在发送 Pull Request 时，往往会遇到找不到对方的项目或者不知道该如何发送等问题。

所以，本书将为各位创造一个亲自动手发送 Pull Request 的机会，请各位不要错过。

6.1 Pull Request 的概要

● 什么是 Pull Request

首先我们来理解什么是 Pull Request[①]。Pull Request 是自己修改源代码后，请求对方仓库采纳该修改时采取的一种行为。

● Pull Request 的流程

下面来看看具体的例子。现在假设我们在使用 GitHub 上的一款开源软件。

在使用这款软件的过程中，我们偶然间发现了 BUG。为了继续使用软件，我们手动修复了这个 BUG。如果我们修改的这段代码能被该软件的开发仓库采纳，今后与我们同样使用这款软件的人就不会再遇到这个 BUG。为此，我们要第一时间发送 Pull Request。

在 GitHub 上发送 Pull Request 后，接收方的仓库会创建一个附带源代码的 Issue，我们在这个 Issue 中记录详细内容。这就是 Pull Request。

① Pull Request 在网络上也常常被简称为 PR。

发送过去的 Pull Request 是否被采纳，要由接收方仓库的管理者进行判断。一般只要代码没有问题，对方都会采纳。如果有问题，我们会收到评论。

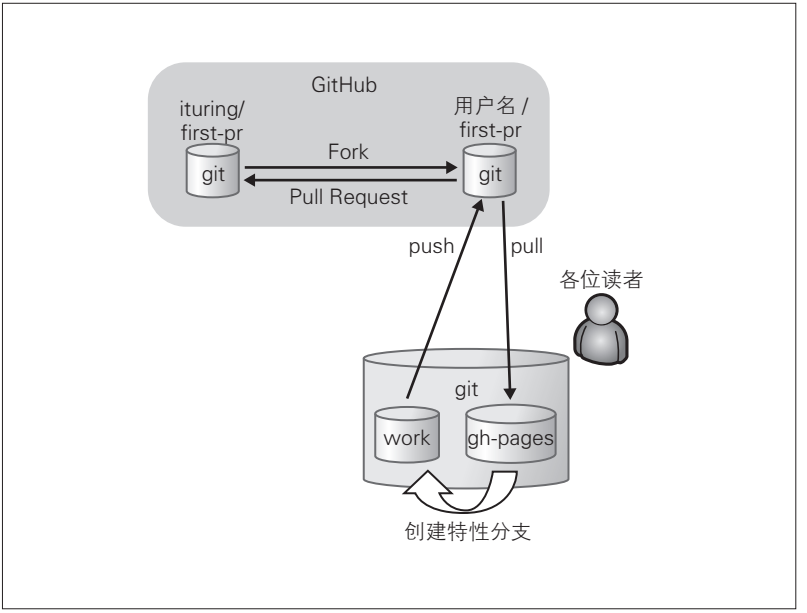
只要 Pull Request 被顺利采纳，我们就会成为这个项目的 Contributor（贡献者），我们编写的这段代码也将被全世界的人使用。这正是社会化编程和开源开发的一大乐趣。

我们为本书专门搭设了一个网站，各位可以对其进行修改，尝试发送 Pull Request。

6.2 发送 Pull Request 前的准备

整体概念如图 6.1 所示。

图 6.1 Pull Request 的概念图



● 查看要修正的源代码

请登录我们为各位准备的网站[①]。该网站的源代码已经在 GitHub 上公开[②]。各位请将自己的感想写入源代码，然后发送 Pull Request。

这个网站通过 GitHub 的 GitHub Pages 功能发布。GitHub Pages 的网站的源代码位于仓库的 gh-pages 分支。访问仓库页面，我们就可以看到源代码。

记述感想时需要修改 index.html 文件。各位不妨先查看它的源代码，对内容有个印象。

● Fork

各位请访问仓库页面，点击 Fork 按钮创建自己的仓库（图 6.2）。

新建的仓库名为“自己的账户名 /first-pr”。在这里我们命名为 hirocastest。

图 6.2 Fork 按钮

● clone

clone 仓库所需的访问信息显示在右侧的中央部分，让我们将它复制下来，把这个仓库 clone 到当前的开发环境中。

```
$ git clone git@github.com:hirocastest/first-pr.git
Cloning into 'first-pr'...
remote: Counting objects: 14, done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 14 (delta 2), reused 0 (delta 0)
Receiving objects: 100% (14/14), 24.05 KiB, done.
Resolving deltas: 100% (2/2), done.
$ cd first-pr
```

① https://ituring.github.io/first-pr/

② https://github.com/ituring/first-pr

first-pr 目录下会生成 Git 仓库。这个仓库与我们 GitHub 账户下的 first-pr 仓库状态相同。现在只要在这个仓库中修改源代码进行 push，GitHub 账户中的仓库就会被修改。

● branch

●…… 为何要在特性分支中进行作业

当前 Git 的主流开发模式都会使用特性分支。关于特性分支的详细知识，我们已经在第 4 章讲解过了。

各位请养成创建特性分支后再修改代码的好习惯。在 GitHub 上发送 Pull Request 时，一般都是发送特性分支。这样一来，Pull Request 就拥有了更明确的特性（主题）。让对方了解自己修改代码的意图，有助于提高代码审查的效率。

●…… 确认分支

我们来查看一下 clone 出的仓库的分支。

```
$ git branch -a
* gh-pages          ←当前分支
  remotes/origin/HEAD -> origin/gh-pages
  remotes/origin/gh-pages
```

开头加了“remotes/origin/”的是 GitHub 端仓库的分支。我们手头的开发环境中只有 gh-pages 分支。

网站中显示的 HTML 位于 /origin/gh-pages 分支。虽然通常情况下最新版代码都位于 master 分支，但由于本次我们使用了 GitHub Pages，所以最新代码位于 gh-pages 分支。

●…… 创建特性分支

我们创建一个名为 work 的分支，用来发送 Pull Request。这个 work 分支就是这次的特性分支。现在创建 work 分支并自动切换。

```
$ git checkout -b work gh-pages
Switched to a new branch 'work'
```

确认是否切换到了 work 分支下。

```
$ git branch -a
  gh-pages
* work  ←当前分支
  remotes/origin/HEAD -> origin/gh-pages
  remotes/origin/gh-pages
```

查看文件列表，我们可以看到网站中显示的 index.html 文件。

```
$ ls
README.md       index.html      params.json
images          javascripts     stylesheets
```

可以用浏览器打开并确认显示。

● 添加代码

用编辑器打开 index.html 文件，以 HTML 形式添加感想。

```
省略
<p>请写明这是对本书内容的实践或描述对本书的感想并发送Pull Request。</p>
↓追加的行
<p class="impression"> 这本书读着很有趣。（@HIROCASTER）</p>
省略
```

请自由添加感想并用 p 标签（Tag）括起，然后关闭编辑器。

● 提交修改

用 `git diff`命令查看修改是否已经正确进行。

```
$ git diff
diff --git a/index.html b/index.html
index f2034b3..91b8ecb 100644
--- a/index.html
+++ b/index.html
@@ -39,6 +39,8 @@

 <p>请写明这是对本书内容的实践或描述对本书的感想并发送Pull Request。</p>

 +<p class="impression"> 这本书读着很有趣。（@HIROCASTER）</p>
 +
 省略
```

然后用浏览器打开，查看显示是否正确。然后确认添加的代码，提交至本地仓库。

```
$ git add index.html
$ git commit -m "Add my impression"
[work 243f28d] Add my impression
 1 file changed, 2 insertions(+)
```

● 创建远程分支

要从 GitHub 发送 Pull Request，GitHub 端的仓库中必须有一个包含了修改后代码的分支。我们现在就来创建本地 work 分支的相应远程分支。

```
$ git push origin work
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 353 bytes, done.
Total 3 (delta 2), reused 0 (delta 0)
To git@github.com:hirocastest/first-pr.git
 * [new branch]      work -> work
```

查看分支，/origin/work 已被创建。

```
$ git branch -a
   master
   * work
   remotes/origin/HEAD -> origin/master
   remotes/origin/gh-pages
   remotes/origin/work   ←已被创建
```

请打开 GitHub 的“用户名 /first-pr”页，确认 work 分支是否被创建，以及是否已包含我们添加的代码。

6.3 发送 Pull Request

参考图 6.3，登录 GitHub 并切换至 work 分支。点击分支名左侧的绿色按钮，会跳转至查看分支间差别的页面（图 6.4）。请在这里通过差

别查看刚刚进行的更改是否正确。这里显示的东西就是我们本次 Pull Request 中包含的提交。

图 6.3 切换分支

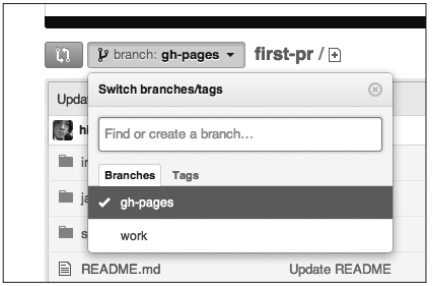
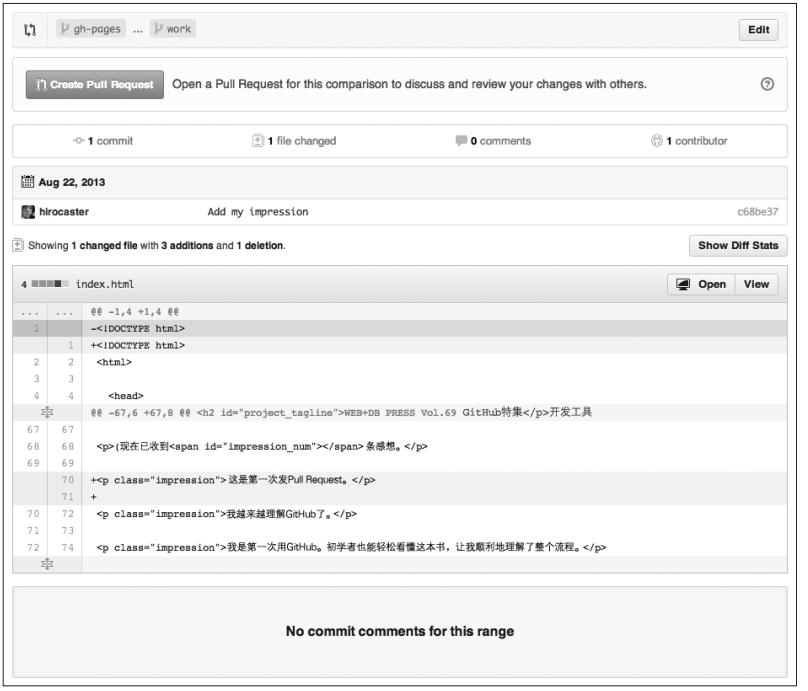


图 6.4 查看分支间差别的页面



确认想要发送的 Pull Request 的内容差别无误后，请点击 Create Pull Request。随后显示的表单用于填写请求对方采纳的评论（图 6.5）。现在

让我们在评论栏中简明扼要地描述本次进行 Pull Request 的理由。

图 6.5 填写请求对方采纳的评论

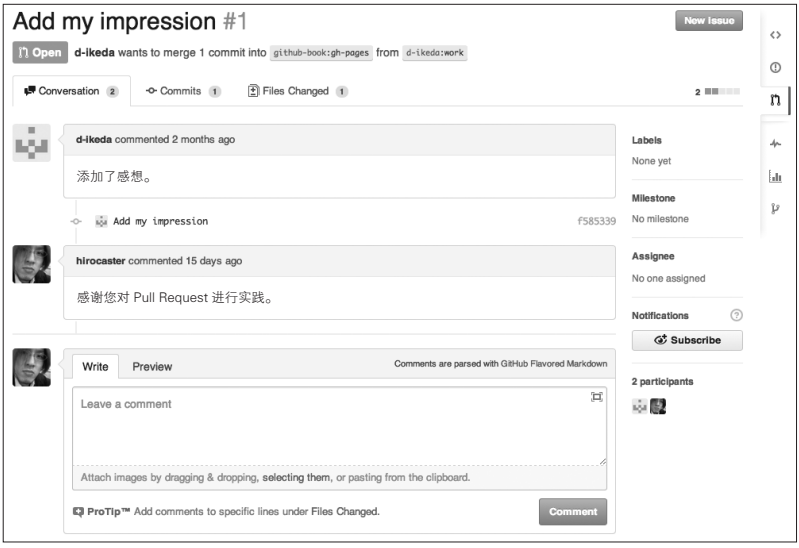
确认没有问题后，点击 Send pull request 按钮。这样一来，Pull Request 的目标仓库中就会新建 Pull Request 和 Issue，同时该仓库的管理者会接到通知。

至此，恭喜各位顺利发送了第一次的 Pull Request。现在我们发送的源代码还没有被采纳，对方仓库不会有任何变化，所以网页也仍然是原样。

如果想查看已发送 Pull Request 的状态，可以登录 GitHub，打开自己的控制面板查看 Pull Request 标签页。点击自己发送的 Pull Request 后会进入如图 6.6 的页面，管理者对 Pull Request 的评论会发到这里。这些在 Conversation 标签页中会按照时间顺序排列显示。只要代码没有问题，我们的 Pull Request 就会被采纳[①]。

① 本书中出现的示例仓库，现阶段将主要由译者及志愿者（包括尝试 Pull Request 的各位读者）进行维护。但是在本书出版后，随着时间推移，可能会发生反应变慢甚至没有反应的情况。烦请参照第 7 章的内容以及关于示例仓库的讲解，一同努力维护。

图 6.6 Pull Request



6.4 让 Pull Request 更加有效的方法

下面为各位介绍在开发现场如何更有效地运用 Pull Request。

● 在开发过程中发送 Pull Request 进行讨论

在软件的设计与实现过程中如果想发起讨论，Pull Request 是个非常好的契机。我们虽然可以像本次示例一样等代码完成后再发送 Pull Request，但在实际开发过程中，这样做很可能导致一个功能在完成后才收到设计或实现方面的指正，从而使代码需要大幅更改或重新实现。

在 GitHub 上，我们可以尽早创建 Pull Request，从审查中获得反馈，让大家在设计与实现方面思路一致，借此逐渐提高代码质量。这个方法在团队开发大型项目时尤其有效，已将 GitHub 运用到实际开发中的团队请务必试一试。

这个方法执行起来很简单。只要在想发起讨论时发送 Pull Request 即可，不必等代码最终完成。即便某个功能尚在开发之中，只要在 Pull Request 中附带一段简单代码让大家有个大体印象，就能获取不少反馈。如果在 Pull Request 中再加入直观易懂的 Tasklist（请参照第 5 章的“Tasklist 语法”），就能很清楚反映出哪些功能已经实现，将来要做哪些工作。这不但能加快审查者的工作效率，还能作为自己的备忘录使用。

从反馈中，我们不但能获得对自己所提议的新功能的支持和相关改善意见，有时还会被人指出自己没注意到的失误，或者准备编写的代码与其他成员重复等。这样一来，我们最终所完成的代码的质量一定会比原先高出许多。

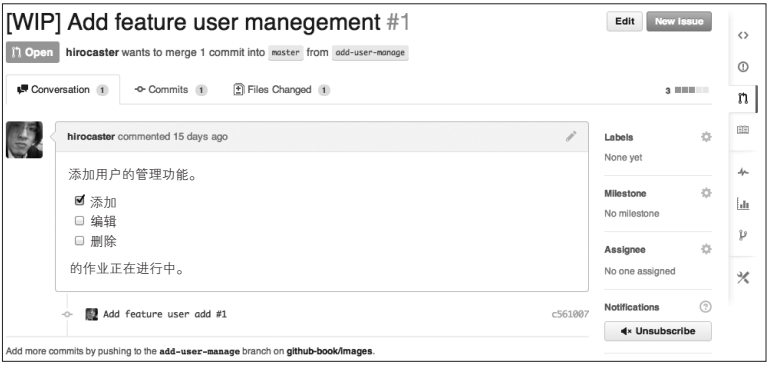
向发送过 Pull Request 的分支添加提交时，该提交会自动添加至已发送的 Pull Request 中。

这一方法要求尽早发送 Pull Request，越早效果越明显。另外还有一件事要记住，就是千万不要在 Pull Request 中添加无关的修改。处理与主题无关的作业请另外创建分支，不然会让原本清晰的讨论变得一团糟。

● 明确标出“正在开发过程中”

为防止开发到一半的 Pull Request 被误合并，一般都会像图 6.7 中所示的那样在标题前加上“[WIP]”字样。WIP 是 Work In Progress 的简写，表示仍在开发过程中。等所有功能都实现之后，再消去这个前缀。

图 6.7 标明仍在开发中的 Pull Request



这种在代码库中边讨论边开发的开发流程，要比以往在完成之后审查再反馈的流程高效得多。这个方法已经被应用到众多的软件开发现场。通过这一方法，开发者可以体验 GitHub 上独有的速度感。各位请务必加以实践。

● 不进行 Fork 直接从分支发送 Pull Request

这个方法也值得在 GitHub 上进行开发的团队借鉴。

一般说来，在 GitHub 上修改对方的代码时，需要先将仓库 Fork 到本地，然后再修改代码，发送 Pull Request。但是，如果用户对该仓库有编辑权限，则可以直接创建分支，从分支发送 Pull Request。利用这一设计，团队开发时不妨为每一名成员赋予编辑权限，免去 Fork 仓库的麻烦。这样，成员在有需要时就可以创建自己的分支，然后直接向 master 分支等发送 Pull Request。

其实，这一方法已经被 GitHub 实际运用到开发之中[①]。关于这一开发流程的具体内容将在第 9 章详细说明。

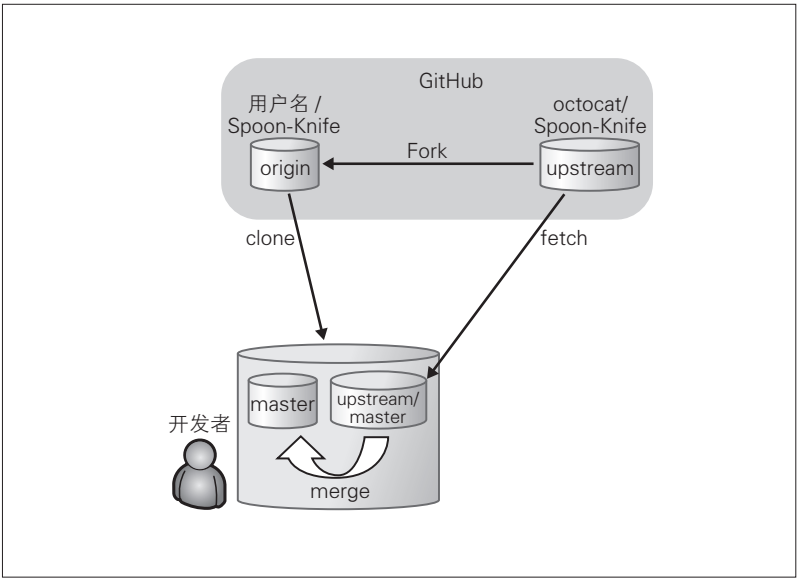
6.5 仓库的维护

Fork 或 clone 来的仓库，一旦放置不管就会离最新的源代码越来越远。如果不以最新的源代码为基础进行开发，劳神费力地编写代码也很可能是白费力气。下面就让我们学习如何让仓库保持最新状态。

通常来说 clone 来的仓库实际上与原仓库并没有任何关系。所以我们需要将原仓库设置为远程仓库，从该仓库获取（fetch）数据与本地仓库进行合并（merge），让本地仓库的源代码保持最新状态（图 6.8）。

① https://github.com/blog/1124-how-we-ues-pull-requests-to-build-github

图 6.8 将仓库更新至最新状态



● 仓库的 Fork 与 clone

将 octocat/Spoon-Knife 作为原仓库，在 GitHub 上进行 Fork，然后 clone。

```
$ git clone git@github.com:hirocastest/Spoon-Knife.git
Cloning into 'Spoon-Knife'...
remote: Counting objects: 24, done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 24 (delta 7), reused 17 (delta 1)
Receiving objects: 100% (24/24), 74.36 KiB | 68 KiB/s, done.
Resolving deltas: 100% (7/7), done.
$ cd Spoon-Knife
```

● 给原仓库设置名称

我们给原仓库设置 upstream 的名称，将其作为远程仓库。

```
$ git remote add upstream git://github.com/octocat/Spoon-Knife.git
```

今后，我们的这个仓库将以 upstream 作为原仓库的标识符。这个环境下只需要设定一次。

● 获取最新数据

下面我们从远程仓库实际获取（fetch）最新源代码，与自己仓库的分支进行合并。要让仓库维持最新状态，只需要重复这一工作即可。

```
$ git fetch upstream
From git://github.com/octocat/Spoon-Knife
 * [new branch]      master     -> upstream/master
$ git merge upstream/master
Already up-to-date.
```

我们通过 `git fetch` 命令获取最新的数据，将 upstream/master 分支与当前分支（master）合并。虽然本次示例没有可以合并的内容，但这一操作确实可以将最新的源代码合并至当前分支。

这样一来，当前分支（master）就获得了最新的源代码。各位在创建特性分支，编辑源代码之前，建议先将仓库更新到这一状态。一般情况下 master 分支都会获取最新代码，很少需要 Fork 的开发者亲自进行修正。

6.6 小结

本章中我们简单学习了 Pull Request 的发送方法。想必各位已经发现，发送 Pull Request 时不单要敲一敲代码，还需要进行很多其他工作。

在实际开发现场，Pull Request 多少都会与传统的习惯或规范有些冲突。但是，诸多团队的实践表明 Pull Request 确实有其显著的效果。作为一名投身于开源开发的程序员，应当尽早适应这一设计。

笔者认为，对这种标准的设计或规范采取“总之先试试看”的态度，往往可以给现场带来活力，促进成员成长，给开发带来速度感。建议各位积极地去尝试。