

与全世界程序员分享你的代码

GitHub 人门与实践

[日] 大塚弘记/著 支鵬浩 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







● 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 $^{\text{TM}}$ 、 $^{\text{C}}$ 、 $^{\text{R}}$ 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

0	第1章 欢迎来到GitHub的世界	· 1
1.1	什么是 GitHub	2
	GitHub 公司与 octocat ····································	2
	并不只是 Git 仓库的托管服务	3
	GitHub 的使用情况····································	
	Column 专栏:GitHub 与 Git 的区别·······	
1.2	使用 GitHub 会带来哪些变化····································	Z
	协作形式变化	
	在开发者之间引发化学反应的 Pull Request ·······	Ę
	对特定用户进行评论	
	GitHub Flavored Markdown	
	Column 专栏:还可以这样写!!······	
	能看到更多其他团队的软件	7
	与开源软件相同的开发模式	8
1.3	社会化编程	(
1.4	为什么需要社会化编程	10
	不要闭目塞听,要接触不同的文化	10
	会写代码的程序员更受青睐	11
	GitHub 最大的特征是"面向人"	11
1.5	GitHub 提供的主要功能	12
	Git 仓库······	12
	Organization ·····	12
	Issue····	13
	Wiki	13
	Pull Request ·····	
	Column 专栏:GitHub 上受到瞩目的软件 ····································	
1.6	小结	14

	参考资料	14
0	第2章 Git的导入	17
2.1	诞生背景	18
2.2	什么是版本管理 ······	18
	集中型与分散型······	19
	集中型 ·····	
	分散型	
	集中型与分散型哪个更好	
2.3	安装	
	Mac 与 Linux······	21
	Windows	
	组件的选择 ·····	
	设置环境变量	
	换行符的处理····································	
	本书所用的环境	
2.4	初始设置	
2.7	设置姓名和邮箱地址	
	皮直妊石和助相地址 提高命令输出的可读性······	
٥.		
2.5	小结	25
0	第3章 使用GitHub的前期准备····································	27
3.1	使用前的准备	28
	创建账户	28
	设置头像	
	设置 SSH Key······	
	添加公开密钥	
	使用社区功能	

3.2	实际动手使用	31
	创建仓库	31
	Repository name	32
	Description	32
	Public、Private·····	32
	Initialize this repository with a README	
	Add .gitignore	
	Add a license····	
	连接仓库	
	README.md·····	
	GitHub Flavored Markdown ·····	
	公开代码	34
	clone 已有仓库······	
	编写代码	
	提交	
	Column 专栏:公开时的许可协议	
	进行 push ····································	
0	第4章 通过实际操作学习Git	39
4.1	基本操作	40
	git init——初始化仓库 ····································	40
	git status——查看仓库的状态·······	40
	git add——向暂存区中添加文件·······	41
	git commit——保存仓库的历史记录········	
	记述一行提交信息	
	记述详细提交信息	42
	中止提交	43
	查看提交后的状态	43
	git log——查看提交日志·······	43
		10
	只显示提交信息的第一行	
	只显示提交信息的第一行 ······· 只显示指定目录、文件的日志 ······	44

	git diff——查看更改前后的差别······	45
	查看工作树和暂存区的差别	
	查看工作树和最新提交的差别	46
4.2	分支的操作	47
	git branch——显示分支一览表·······	48
	git checkout -b——创建、切换分支·······	48
	切换到 feature-A 分支并进行提交······	48
	切换到 master 分支······	
	切换回上一个分支 ······	
	特性分支	
	主干分支	51
	git merge——合并分支······	51
	git loggraph——以图表形式查看分支·······	52
4.3	更改提交的操作	53
	git reset——回溯历史版本·······	53
	- 回溯到创建 feature-A 分支前 ·······	53
	创建 fix-B 分支 ······	
	推进至 feature-A 分支合并后的状态······	
	消除冲突	
	查看冲突部分并将其解决	
	提交解决后的结果 ·····	
	git commitamend——修改提交信息·······	
	git rebase -i——压缩历史·······	
	创建 feature-C 分支······	
	修正拼写错误	
	更改历史····································	
4.4	推送至远程仓库	
	git remote add——添加远程仓库·······	
	git push——推送至远程仓库·······	
	推送至 master 分支·······	
	推送至 master 以外的分支 ····································	
4.5	从远程仓库获取	
	git clone——获取远程仓库 ····································	65

	获取远程仓库	65
	获取远程的 feature-D 分支······	66
	向本地的 feature-D 分支提交更改·······	
	推送 feature-D 分支······	67
	git pull——获取最新的远程仓库分支····································	67
4.6	帮助大家深入理解 Git 的资料	68
	Pro Git	68
	LearnGitBranching	69
	tryGit	69
4.7	小结	70
O	第5章 详细解说GitHub的功能	71
5.1	键盘快捷键·····	72
5.2	工具栏	73
	关于 UI	73
	1 LOGO	73
	2 Notifications·····	73
	3 搜索窗□	73
	4 Explore····	73
	5 Gist	74
	6 Blog ····	
	7 Help·····	
	8 头像、用户名	
	9 Create a new	
	10 Account settings	
	11 Sign out ·····	
5.3	控制面板	75
	关于 UI	75
	News Feed · · · · · · · · · · · · · · · · · ·	76
	2 Pull Requests·····	
	3 Issues ·····	
	4 Stars ·····	
	5 Broadcast······	76

	Repositories you contribute to	76
	7 Your Repositories ·····	76
5.4	个人信息	77
	关于 UI	77
	1 用户信息	77
	2 Popular Repositories	
	3 Repositories contributed to	78
	4 Public contributions ·····	78
	5 Contribution Activity	78
	6 Repositories·····	78
	7 Public Activity ·····	79
5.5	仓库	80
	关于 UI·····	80
	● 用户名(组织名)/仓库名	
	Watch/Star/Fork · · · · · · · · · · · · · · · · · · ·	80
	3 Code	81
	4 Issue	81
	Pull Requests	81
	6 Wiki	82
	7 Pulse·····	82
	3 Graphs ·····	82
	Network	82
	Settings	
	SSH clone URL	
	Clone in Desktop ·····	
	Download ZIP	
	a commits·····	
	b branches	
	G releases ·····	
	d contributors	
	Compare & review	
	f branch	
	9 path	
	• Fork this project and Create a new file · · · · · · · · · · · · · · · · · · ·	
	文件的相关操作	
	(Column) 专栏:通过部分名称搜索文件 ····································	85

	查看差别·····	88
	查看分支间的差别	
	查看与几天前的差别	
	查看与指定日期之间的差别	
5.6	Issue	87
	简洁且表现力丰富的描述方法	88
	语法高亮	
	添加图片	90
	添加标签以便整理	90
	添加里程碑以便管理······	91
	Column 专栏:了解贡献时的规则! ·······	
	Tasklist 语法······	92
	通过提交信息操作 Issue·······	
	在相关 Issue 中显示提交·······	
	Close Issue ····	
	将特定的 Issue 转换为 Pull Request······	
5.7	Pull Request	94
	Column 专栏: 获取 diff 格式与 patch 格式的文件 ····································	
	Conversation	
	Column 专栏:引用评论····································	
	Commits	
	Column 专栏:在评论中应用表情······	
	Files Changed ·····	98
5.8	Wiki	99
	Pages	100
	History	101
	Column 专栏:在 Wiki 中显示侧边栏····································	101
5.9	Pulse	102
	active pull requests	103
	active issue	
	commits	104
	Releases published	
	Unresolved Conversations	
	OTHESON CO CONTRETS ALIONS	10-

5.10	Graphs	105
	Contributors	105
	Commit Activity ·····	106
	Code Frequency	106
	Punchcard	108
5.11	Network	108
5.12	Settings	109
	Options	109
	• Settings ·····	109
	2 Features····	
	3 GitHub Pages····	
	4 Danger Zone	
	Collaborators	111
	Webhooks & Services	112
	Deploy Keys·····	112
5.13	Notifications	112
5.14	其他功能	114
	GitHub Pages	114
	GitHub Jobs ·····	114
	GitHub Enterprise	114
	GitHub API	115
5.15	小结	115
	Column 专栏:在 Mac 的通知中心查看 GitHub 的 Notifications ····	115
	第6章 尝试Pull Request	117
V	第0早 云Mruli Nequest	117
6.1	Pull Request 的概要	118
	什么是 Pull Request ····································	118
	Pull Request 的流程 ···································	118
6.2	发送 Pull Request 前的准备	119
	查看要修正的源代码	120

	Fork120	
	clone 120	
	branch 121	
	为何要在特性分支中进行作业121	
	确认分支121	
	创建特性分支121	
	添加代码	
	提交修改122	
	创建远程分支 123	
6.3	发送 Pull Request 123	
6.4	让 Pull Request 更加有效的方法 126	
	在开发过程中发送 Pull Request 进行讨论126	
	明确标出 "正在开发过程中"127	
	不进行 Fork 直接从分支发送 Pull Request ······128	
6.5	仓库的维护 128	
	仓库的 Fork 与 clone129	
	给原仓库设置名称129	
	获取最新数据130	
6.6	小结 ————————————————————————————————————	
0	第7章 接收Pull Request ······························	
7.1	采纳 Pull Request 的方法 132	
7.2	采纳 Pull Request 前的准备 ·······133	
	代码审查133	
	查看图片的差别134	
	2-up134	
	Swipe135	
	Onion Skin	
	Difference 136	
	在本地开发环境中反映 Pull Request 的内容 136	
	(4) 左(V) 5 (V) 5	

	获取发送方的远程仓库	137
	创建用于检查的分支	138
	合并	138
	删除分支·····	139
	Column 专栏:如何提升代码管理技术 ·······	139
7.3	采纳 Pull Request	139
	合并到主分支 ·····	140
	push 修改内容······	141
7.4	小结	142
	(Column) 专栏:请协助我们共同创建互相学习的场所······	142
0	第8章 与GitHub相互协作的工具及服务	143
8.1	hub 命令	144
	概要	144
	安装	144
	安装	145
	确认运行情况	145
	设置别名	145
	实现 shell 上的功能补全 ·······	146
	~/.config/hub·····	146
	命令	146
	hub clone ·····	146
	hub remote add·····	147
	hub fetch·····	147
	hub cherry-pick ·····	147
	hub fork·····	148
	hub pull-request·····	148
	hub checkout ·····	148
	hub create ·····	
	hub push ·····	149
	hub browse ·····	
	hub compare ·····	150
	Column 专栏:让 GitHub Enterprise 支持 hub 命令····································	151

8.2	Travis CI	151
	概要	151
	实际尝试	152
	编写配置文件	152
	检测配置文件是否有问题	152
	与 GitHub 集成 ······	153
	将 Travis CI 的结果添加至 README.md······	155
8.3	Coveralls	156
	概要	156
	安装	157
	注册	157
	添加对象仓库	158
	编写配置文件 ·····	158
	添加 gem·····	
	查看报告	160
8.4	Gemnasium	160
8.5	Code Climate	161
8.6	Jenkins	162
	概要·····	162
	安装	164
	创建 bot 账户 ···································	165
	bot 账户的权限设置····································	165
	对象为个人账户时	165
	对象为 Organization 账户时······	165
	检查设置	167
	给 Jenkins 设置 SSH 密钥 ·······	
	初次使用 Jenkins 时······	
	已经在使用 Jenkins 时 ······	168
	GitHub pull request builder plugin 的安装····································	169
	Git plugin 的设置······	
	Github Pull Requests Builder 的设置······	
	Github server api URL·····	
	Access Token·····	171

	Admin list · · · · · · · · · · · · · · · · · · ·	172
	job 的创建与设置·······	172
	GitHub project·····	172
	源码管理	172
	构建触发器	
	构建	
	通知结果·····	174
	测试执行中的状态	
	Failed	
	All is well ·····	
	commit status	
	通过评论进行控制	
	执行任务	
	添加至 White list ·····	
	重新执行任务	
	变更指定评论 ····································	
8.7	小结	177
	Column 专栏:用 Coderwall 生成 GitHub 上的个人信息 ····	178
_		
	第9章 使用GitHub的开发流程	179
9.1	团队使用 GitHub 时的注意事项 ····································	180
	一切从简	180
	项目管理工具与 GitHub 的区别····································	
	项目管理工具与 GitHub 相异的原因	
	不 Fork 仓库的方法 ····································	
9.2	GitHub Flow——以部署为中心的开发模式·········	
3.2		
9.3	GitHub Flow 的流程	184
	随时部署,没有发布的概念	184
	进行新的作业时要从 master 分支创建新分支 ····································	185
	在新创建的分支中进行提交	
	定期 push····································	
	•	
	使用 Pull Request····································	187

	务必让其他开发者进行审查	187
	合并后立刻部署	187
9.4	实践 GitHub Flow 的前提条件	188
	部署作业完全自动化	188
	使用部署工具 ·····	189
	通过 Web 界面进行部署的工具 ······	
	导入开发时的注意事项	
	重视测试·····	
	让测试自动化 ·····	
	编写测试代码,通过全部测试 ······	
	维护测试代码	
9.5	模拟体验 GitHub Flow	191
	Fizzbuzz 的说明······	191
	添加新功能	192
	创建新的分支	192
	如果尚未 clone 仓库······	
	如果之前 clone 过仓库 ······	
	创建特性分支 ······	
	实现新功能	
	创建 Pull Request······	196
	接收反馈·····	196
	修正缩进	197
	添加测试	199
	培育 Pull Request····································	202
	Pull Request 被合并······	202
9.6	团队实践 GitHub Flow 时的几点建议····································	203
	减小 Pull Request 的体积····································	204
	准备可供试运行的环境	204
	不要让 Pull Request 中有太多反馈 ····································	205
	不要积攒 Pull Request ····································	206
9.7	GitHub Flow 的小结	206
9.8	Git Flow——以发布为中心的开发模式	207

	便于理解的标准流程	207
	有时显得过于复杂 ······	209
9.9	导入 Git Flow 前的准备	209
	安装 git-flow	209
	- Mac 下的安装······	209
	Linux 下的安装······	210
	确认运行状况 ·····	210
	仓库的初始设置	
	创建仓库	
	进行 git flow 的初始设置······	
	在远程仓库中也创建 develop 分支········	
9.10	模拟体验 Git Flow	
	master 分支与 develop 分支的区别·······	
	master 分支······	
	develop 分支 ······	
	在 feature 中进行的工作·······	
	创建分支	
	在分支中进行作业 ·····	
	发送 Pull Request	
	通过代码审查提高代码质量······	
	更新本地的 develop 分支 ·······	219
	在 release 分支中进行的工作·······	220
	Column) 专栏:设置默认分支······	
	创建分支	
	分支内的工作	
	进行发布与合并	
	查看版本标签	
	更新到远程仓库	
	在 hotfix 分支中进行的工作····································	
	创建分支······	
	创建标签和进行发布······	
0.46	从 hotfix 分支合并至 develop 分支 ···································	
9.11	Git Flow 的小结	
	Column 专栏:版本号的分配规则	232

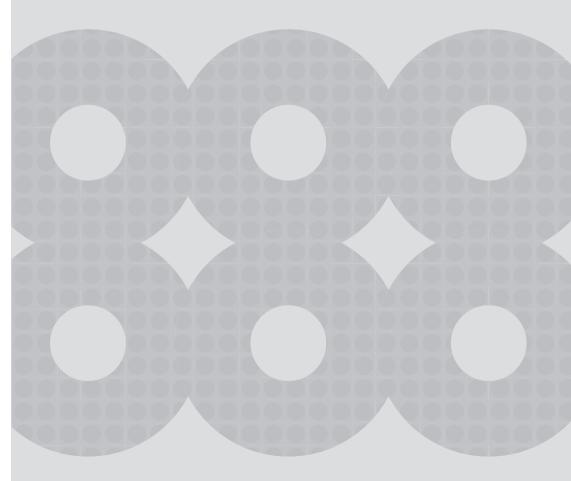
0	第10章	将GitHub应用到企业	233
10.1	将世界	标准的开发环境引入企业现场	234
	企业引入	∖ GitHub 的好处·······	234
	使用 Org	ganization	235
		- hub 的安全性·······	
	注意维护	户时间	235
	查看故障	章信息	236
10.2	GitHub	Enterprise	237
	概述		238
	引入的始	子处	238
	引入的學	×端 ·······	239
		、··· ∖ GitHub Enterprise 的几种情况·······	
		· Annual	
		umn 专栏:将 GitHub 的仓库作为 Subversion 仓库使用·····	
	希望	9维护与故障时间可控	240
10.3	能实现	Git 托管的软件	241
	Col	umn) 专栏:Bitbucket	241
10.4	小结 …		242
0	附录A	支持GitHub的GUI客户端	243
A .1	GitHub	for Mac, GitHub for Windows	244
A.2	Source	Tree	246
0	附录B	通过Gist轻松实现代码共享	247
B.1	Gist 的	特点	248
B.2	创建 Gi	st	248
	UI 讲解·		249
		ist description	

xxiv | 目录

	2 name this file	249
	3 language ·····	250
	4 ACE Editor	250
	5 文件	250
	6 Add another File ·····	251
	7 Create Secret Gist ·····	251
	8 Create Public Gist ·····	251
B .3	查看 Gist	252
	Gist 的菜单······	252
	Gist Detail	253
	2 Revisions ·····	253
	3 Download Gist ·····	253
	4 Clone this gist·····	253
	S Embed this gist	253
	6 Link to this gist ·····	253
	文件的菜单 ·····	254
B.4	Your Gists	254
B.5	小结	255

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[®]。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。

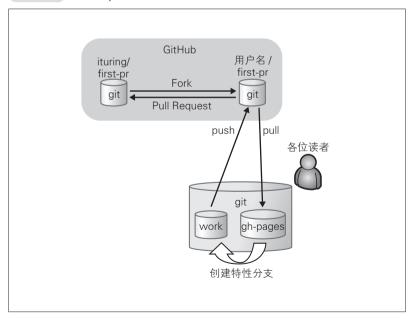
只要 Pull Request 被顺利采纳,我们就会成为这个项目的 Contributor (贡献者),我们编写的这段代码也将被全世界的人使用。这正是社会化 编程和开源开发的一大乐趣。

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 $^{\circ}$ 。该网站的源代码已经在 GitHub 上公开 $^{\circ}$ 。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。



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
              ←当前分支
* qh-pages
 remotes/origin/HEAD -> origin/gh-pages
 remotes/origin/qh-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。



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
e@ -39,6 +39,8 @@

清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。
+ 这本书读着很有趣。(@HIROCASTER)
+
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, 会跳转至杳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

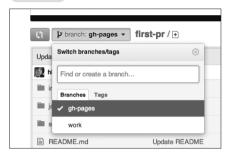
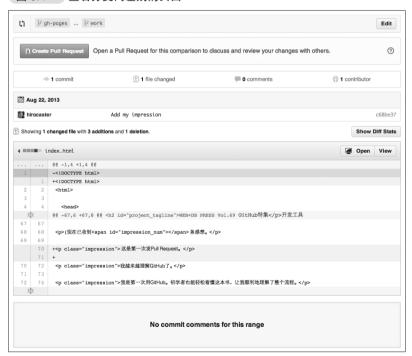


图 6.4 查看分支间差别的页面



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。

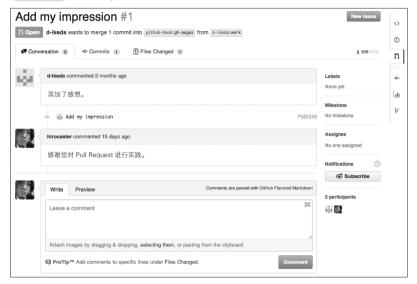
.

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 $^{\circ}$ 。

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。

[WIP] Add feature user management #1 n Open hirocaster wants to merge 1 commit into master from add-user-manage (1) Conversation 1 - Commits 1 Files Changed 1 3 || || || n hirocaster commented 15 days ago 添加用户的管理功能。 Ö ☑ 添加 Lin - 编辑 - 删除 的作业正在进行中。 Add feature user add #1 c561007 Add more commits by pushing to the add-user-manage branch on github-book/images

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 $^{\circ}$ 。关于这一开发流程的具体内容将在第 9 章详细说明。

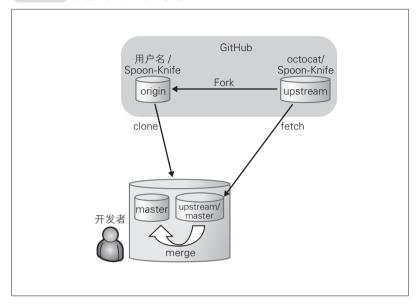
6.5 仓库的维护

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)。

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

将仓库更新至最新状态 图 6.8



● 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。