目录

[Github组织协作 2](#_Toc436925041)

[添加团队成员 3](#_Toc436925042)

[Pull Request 7](#_Toc436925043)

[Issues 11](#_Toc436925044)

[Wiki 24](#_Toc436925045)

[代码评注 39](#_Toc436925046)

[分析 44](#_Toc436925047)

# Github组织协作

[GitHub](http://github.com/)已经成为的一切开放源码软件的基石。开发人员喜欢它，基于它进行协作，并不断通过它开发令人惊叹的项目。除了​​代码托管，[GitHub](http://github.com/)的主要吸引力是使用它作为一个协作开发工具。在本教程中，让我们来看看一些最有用的GitHub的功能，特别是使团队工作更有效率，更高生产力，非常重要的，好玩的那些功能！

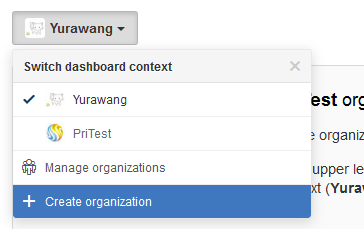
在这个世界上的软件项目，不可避免的是，我们必须和一个团队一起工作来交付软件。在本教程中，我们将探索一些软件开发团队最常用的工具。这些工具包括：

* **添加团队成员** – 成员及权限
* **Pull Request** – 发送代码变更和合并
* **Issues**– Github上的问题追踪
* **Wiki**– 文档记录
* **代码评注** – 代码行评论
* **分析** – 图形与网络

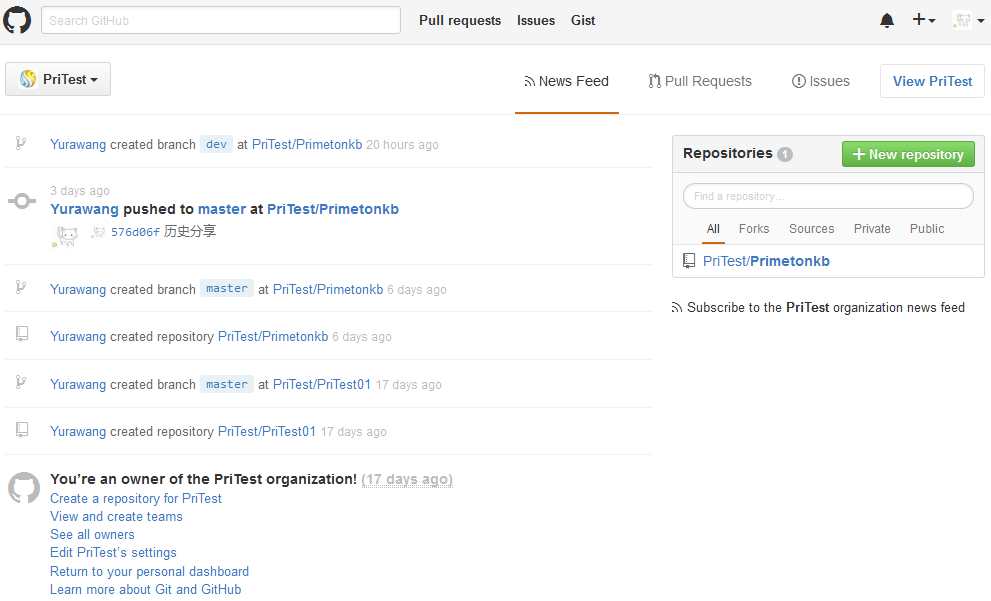
# 添加团队成员

**添加成员**

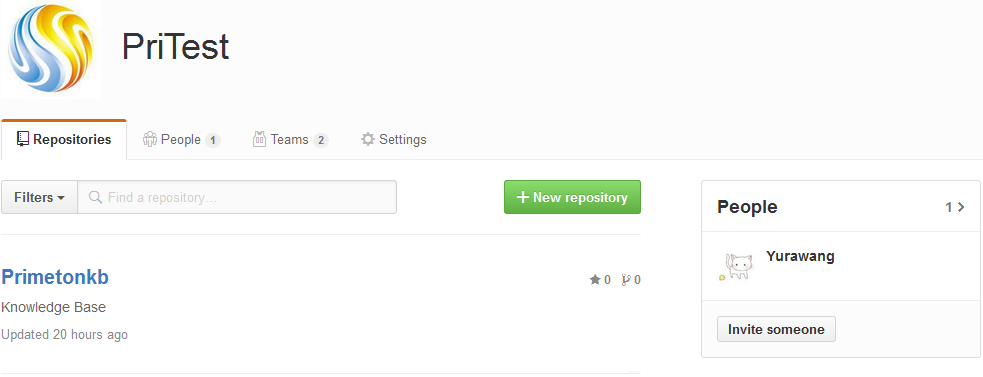
首先，登陆个人Github账户，切换到组织界面（针对Owner），如图，个人账户下方即为所属组织。



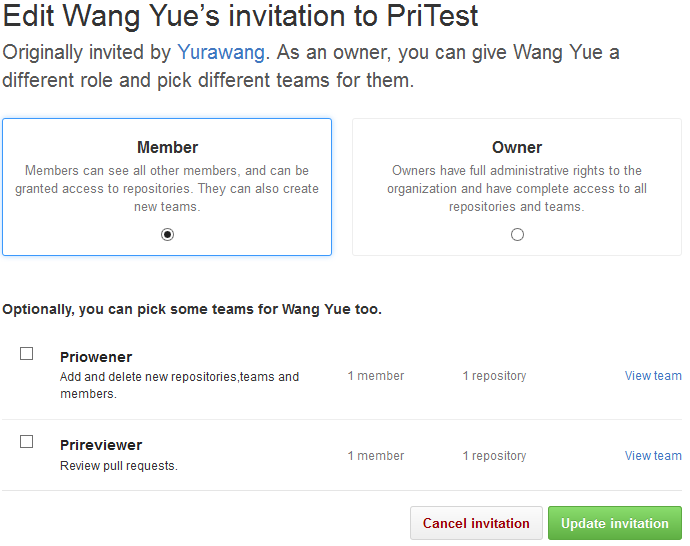
组织面板显示组织最新动态，如下图：



点击右上角**View Organization name**进入组织界面，有组织仓库、成员、团队、设置等选项：



选择右侧**Invite someone**通过用户名搜索添加组织成员，可以为新添成员选择不同权限和团队：



**拥有者（Owner）**：创建组织的用户理所当然是组织拥有者之一。还可以为组织指派更多的拥有者，多个组织拥有者的权限并无差别，都享有管理组织的最高权限，包括增删仓库、成员、团队等。

**成员（Member）**：可见组织所有成员，可新建团队，可被组织拥有者赋予更多权限。

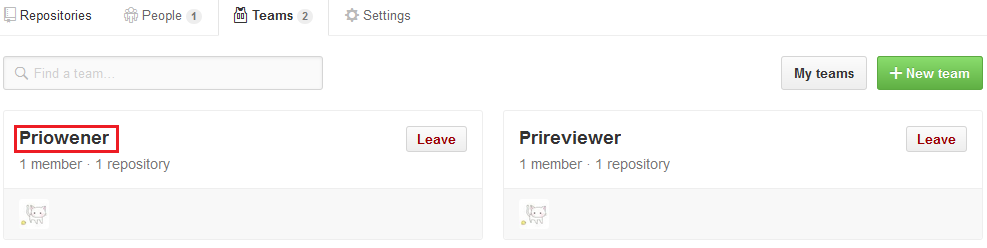
接受邀请后的成员及其权限显示在如下界面：



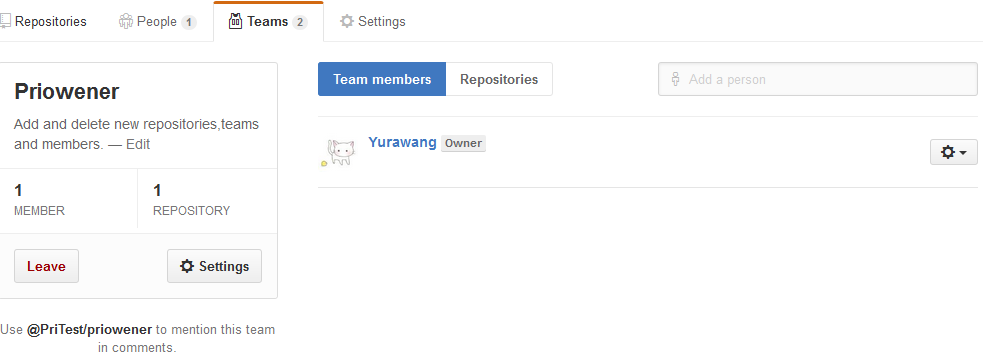
针对km组，部分成员可被添加为Owner，享有对组织的最高权限；其他普通成员可先被添加为Member，后期根据实际需求修改部分成员权限及分组。

**添加团队：**

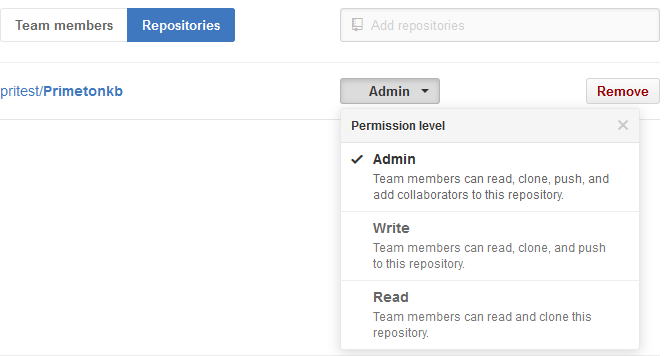
进入团队界面，显示自己所属团队，点击团队名称可查看团队具体信息，以**Priowener**为例。



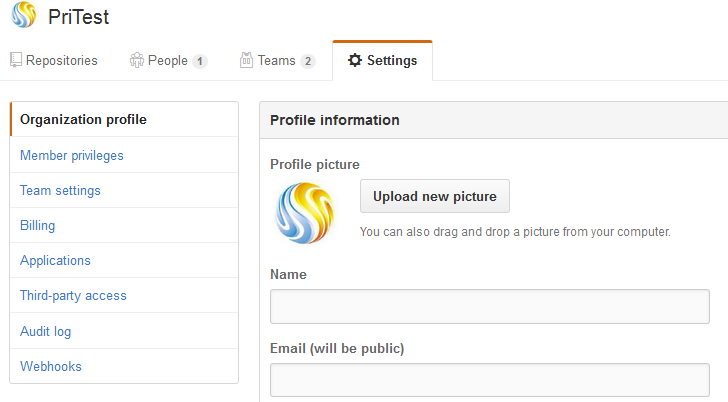
进入**Priowener**团队界面后默认显示**Team members**列表，可通过右侧**Add a person**搜索用户名并添加到该团队中，相应的，也可以通过成员列表后面的编辑按钮删除成员。



**Priowener**团队界面右侧还有**Repositories**选项，可以创建该团队与仓库的联系并设定对仓库的造作权限，在新建仓库时也可选择将仓库与特定团队相连。



更多组织设置可选择**Settings**选项：



目前组织有两个团队，分别为Priowener和Prireviewer，前者成员暂定组织创建者和少数成员，可以对组织仓库、人员、团队进行增删、权限设置等；后者为评审团队，暂定km组里总监以上级别成员，享有对仓库的读写权限。

# Pull Request

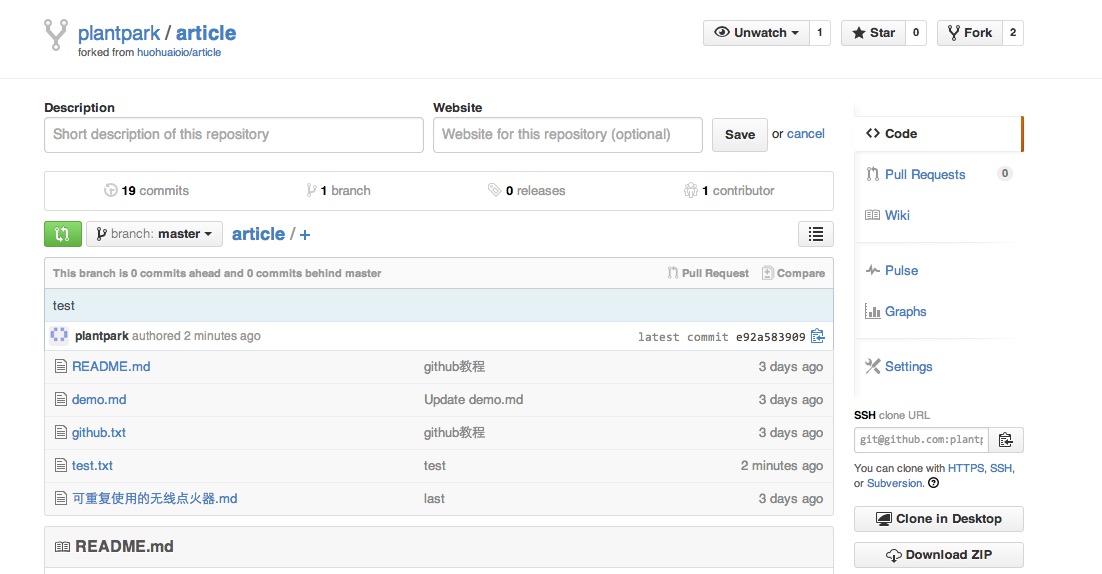
上一章节讲了组织的成员组成及相关权限，以km组为例，加入到组织之后，成员该如何进行相关文件的提交呢？

**首先**

团队已新建仓库。作为成员之一，可直接fork项目到自己的respository。



效果如下图所示：plantpark为成员的个人Github账号，article即为repository，下面一行小字显示“forked from huohuaioio/article”,至此，说明团队article项目下的代码已经被自己fork了。



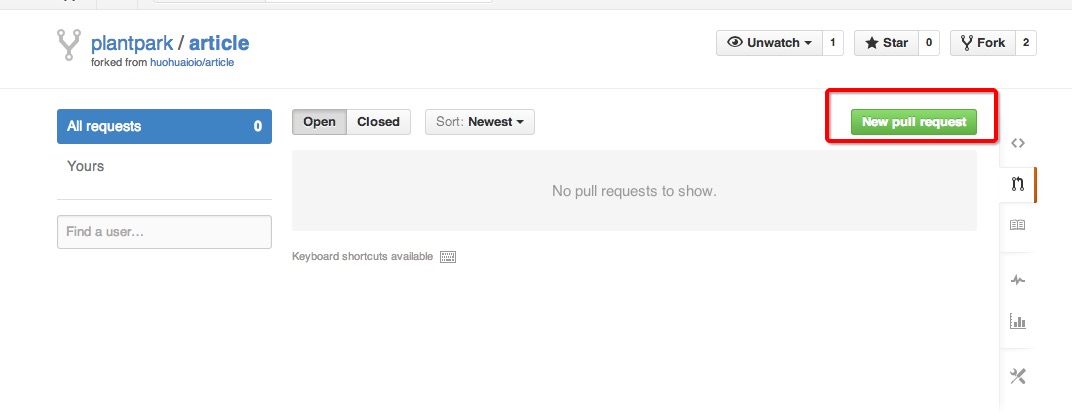
**其次**

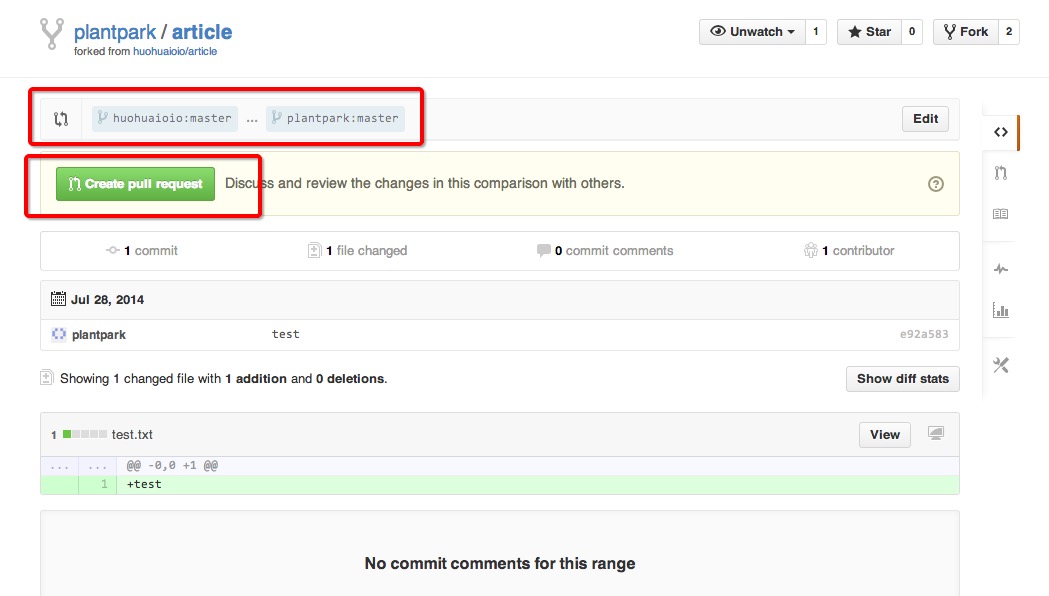
需要将位于Github中的article项目复制到本地。如何在本地设置Github账号以及key等操作这里不做赘述，请参见Github使用手册。

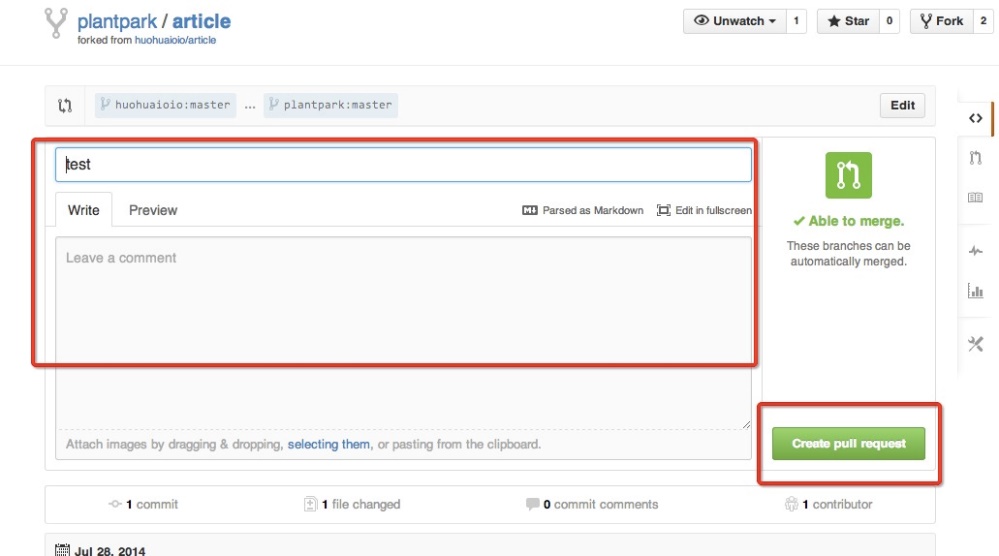


如图，依次新建一个文件夹，githubtest,名称随意。然后git init在这里进行初始化，git clone git@github.com:plantpark/article.git 这个地址，在自己fork后的respository右侧有显示。这样，Github的代码同步到了本地，下面即可在本地新建文件，nano test.txt。test.txt可以用第三方编辑器打开编辑，将文章的markdown代码放入其中，这样一篇文章算是完成了。下面要做的就是通过 push将代码同步到Github。git add \*,添加此文件夹下所有新建与更改的文件，一种偷懒的做法，git commit -m "test"，写入commit，以便备注每次的push。git push origin master，最后一步，将代码同步到Github。这样，在Github中即可看到自己刚刚提交的文件。

**最后**

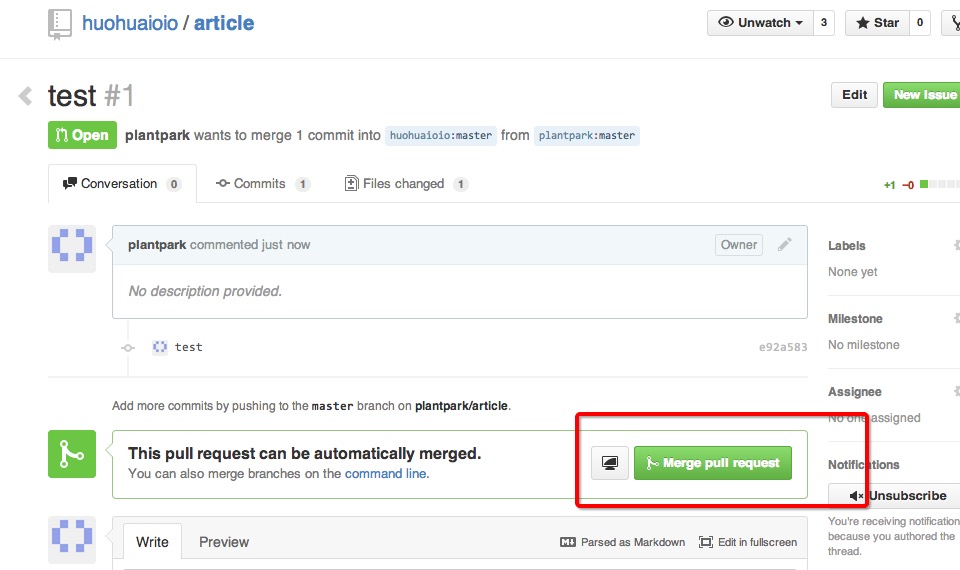






如图所示，在点击respository右侧的pull request，看到一个new pull request 按钮，点击它就能将自己的代码提交到forked的项目中。写入commit后就直接点creat pull request。

当然，这是作为团队成员自己提交项目代码完成了任务，但是，但变化还未显示到项目代码中，还需要一步，就是评审员在审核后将提交的pull request 进行merge，添加到项目中。



至此，一个团队协同合作的Github流程完整呈现出来了，当然，还有些其他的高级功能，这里暂不做表述，可在后面的文章进行详细解说

**注意：**

每个开发者的仓库都有自己的分支路线，而这些分支路线会通过代码汇总映射到源仓库中去。

以**Primetonkb**为例，存在于整个项目的开始、开发、迭代、终止过程中的永久性分支只有两个master和dev。

**master**：主分支从项目一开始便存在，它用于存放经过评审，已经完全稳定的文件。所以，应该随时保持master仓库代码的清洁和稳定，确保入库之前是通过完全reivew的。master分支是所有分支中最不活跃的，大概每个月或每两个月更新一次，每一次master更新的时候都应该用git打上tag，说明有新版本发布了。

**dev**：开发分支，一开始从master分支中分离出来，用于成员存放基本稳定代码。之前说过，每个成员的仓库相当于源仓库的一个镜像，每个成员自己的仓库上也有master和dev。成员把新文件做好以后，是存放到自己的dev中，当整理完以后，可以向管理者/评审者发起一个pull request，请求把自己仓库的dev分支合并到源仓库的dev中。

所有成员的新提交文件会在源仓库的dev分支中进行汇总，当dev中的文件经过评审，已经逐渐趋于稳定了。这时候，我们就可以把dev分支合并到master分支中，发布一个最新版本。

任何人不应该向master直接进行无意义的合并、提交操作。正常情况下，master只应该接受dev的合并，也就是说，master所有更新应该源于合并dev的更新。

还有哪些工具能帮助我们更方便地使用Github呢，下面让我们一起来看看吧~

# Issues

缺陷跟踪（Bug Tracking）是软件研发流程中重要的一环，集项目需求管理和缺陷管理于一身，通过对研发工作流的控制帮助团队建立规范的研发体系。GitHub提供 轻量级的缺陷跟踪模块，称为Issues。小巧、易用的Issues模块能与Pull Request紧密整合，是Pull Request工作流的有益补充。

一个小型、管理文档和网页的项目，使用Pull Request往往就足够了。试想如果贡献者能够直接修改代码（Fork and edit this file）并通过Pull Request贡献给项目核心开发者，那么为什么还要通过Issues模块报告错误并由他人来更改呢？但是对于大型项目需要做需求管理，或者参与代码开发 有难度，则非常有必要通过Issues模块启用缺陷跟踪系统，提供更多途径让贡献者参与到项目中来。

缺陷跟踪可以通过项目的管理页面开启或关闭，如图4-45所示。

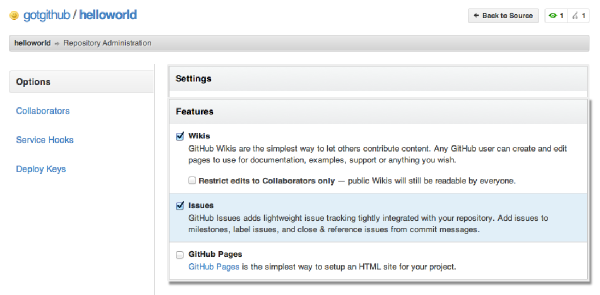
[](http://www.worldhello.net/gotgithub/images/project-admin-features-issue.png)

图4-45：开启或关闭Issues模块

**标签**

缺陷跟踪系统通常可用于管理多种不同类型的问题：需求、缺陷或其它，也可以通过项目不同模块、组件来为问题分类。GitHub在问题分类的实现上非常简单，通过标签（label）来为问题建立分类。

开启Issues模块后，项目的菜单中多出一个“Issues”项，点击则进入问题浏览界面，如图4-46所示。左侧的边栏是问题过滤器，由上至下 分为三个部分。最上面的过滤器根据问题的所有者对问题进行筛选，默认选择所有人的问题。中间的过滤器是根据里程碑对问题进行筛选，默认未选定任何里程碑 （初始尚未创建任何里程碑），不对问题进行里程碑过滤。最下面的过滤器依据问题标签对问题进行筛选，初始标签尚未创建。

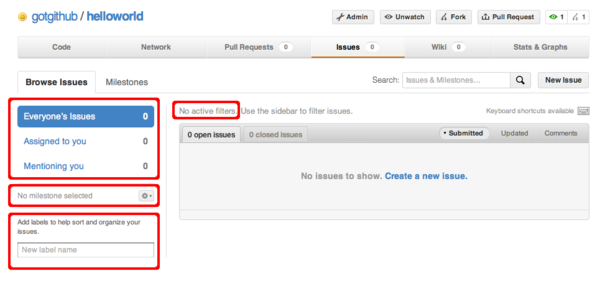
[](http://www.worldhello.net/gotgithub/images/issue-no-label.png)

图4-46：尚未定义标签

鼠标点击左侧边栏标签过滤器中的新建标签的文本框，显示如图4-47所示的新建标签界面。输入新的标签名，并为标签选择一个颜色，创建新的标签。

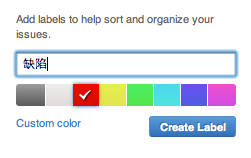
[](http://www.worldhello.net/gotgithub/images/issue-new-label.png)

图4-47：创建新标签

创建的标签显示在过滤器中，如图4-48所示。点击过滤器中的标签则对问题进行筛选，取消过滤器可以点击图中标记的“Clear active milestone and label filters”链接。

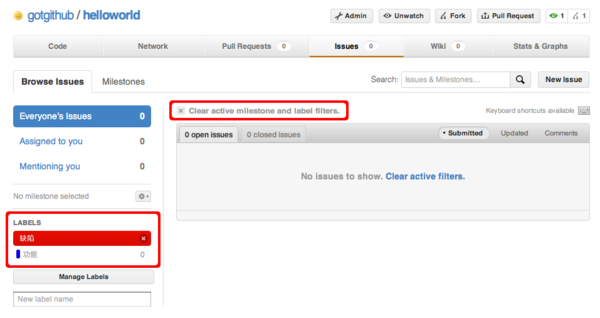
[](http://www.worldhello.net/gotgithub/images/issue-labels.png)

图4-48：过滤器中的标签选项

标签过滤器下方的“Manage Labels”按钮用于管理标签，下面的输入框用于创建新的标签。

**里程碑**

里程碑（Milestones）是项目进度管理的重要工具。在传统项目管理中，里程碑对应于一个项目开发计划、一个软件版本；在敏捷项目管理中，里程碑对应于一个Sprint（冲刺）；在软件代码的版本库中则对应于一个标签（tag）或分支（branch）。

在Issues模块中的“Milestones”页面用于里程碑管理。创建新的里程碑需要输入里程碑名称和里程碑的截止时间，如图4-49所示。

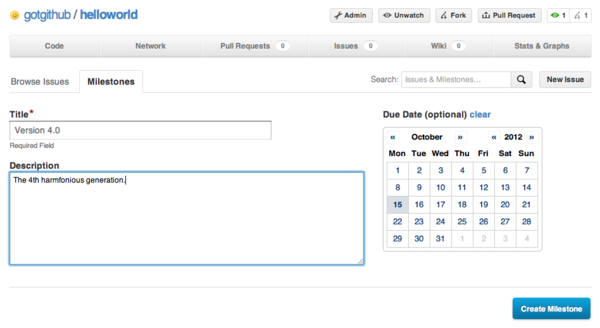
[](http://www.worldhello.net/gotgithub/images/issue-new-milestone.png)

图4-49：创建新里程碑

创建的里程碑以进度条形式显示在里程碑页面中，如图4-50所示定义了两个里程碑。这两个里程碑的时间跨度定义的太长，敏捷的项目管理从来不这么定义。

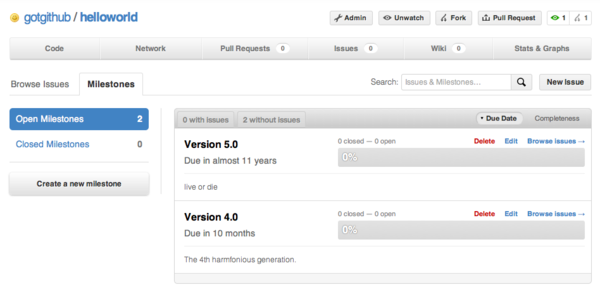
[](http://www.worldhello.net/gotgithub/images/issue-milestones.png)

图4-50：里程碑列表

**Issue的生命周期**

GitHub的Issues模块非常简单，对标签和里程碑进行简单的设置后，基本上就完成了Issues模块的配置工作，接下来就是如何创建和修改Issue，完成项目的缺陷跟踪和需求管理等，这才是Issues模块的主要工作。

每个Issue都有自己的生命周期，从问题的创建，到问题的指派，再到问题的解决，直至问题的关闭。图4-51就是以普通贡献者身份为项目创建Issue。

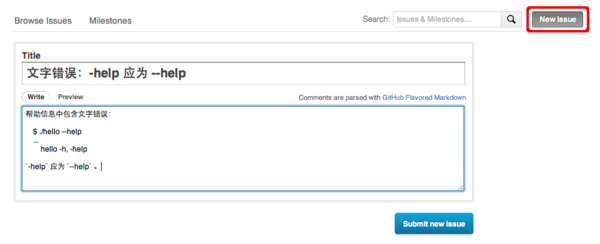
[](http://www.worldhello.net/gotgithub/images/issue-new-by-non-member.png)

图4-51：以普通贡献者身份创建问题

录入问题标题和描述后，点击“Submit new issue”按钮，完成问题创建。图4-52显示了新建立的问题，可以看出新建问题尚未设置标签。

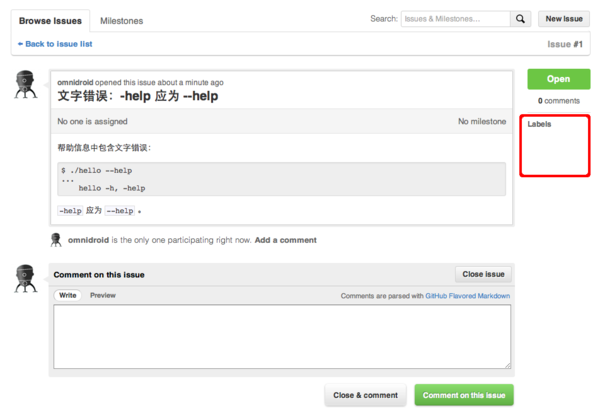
[](http://www.worldhello.net/gotgithub/images/issue-created.png)

图4-52：新创建的问题尚未添加标签等

普通贡献者创建问题时只能录入问题的标题和描述，而不能设置问题的指派（谁来负责）、添加标签和设置里程碑。如果希望问题通知到特定的开发者，可以在问题描述中以“@用户名”的方式通知到该用户[[1]](http://www.worldhello.net/gotgithub/04-work-with-others/050-issue.html#id6)，这也是众多社交软件通行的做法。

项目成员创建问题时，拥有更大权限，也有更多的可选项。如图4-53所示。

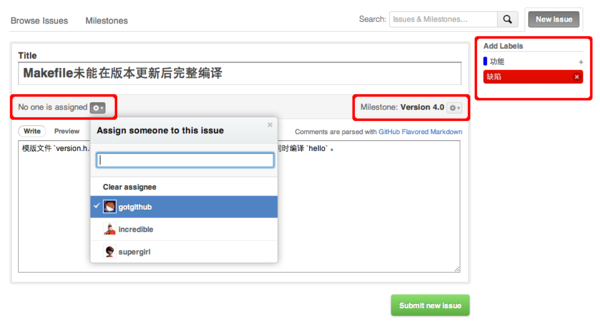
[](http://www.worldhello.net/gotgithub/images/issue-new-by-member.png)

图4-53：以项目成员身份创建问题

完成上述两个问题的创建后，问题浏览界面显示新创建的两个问题，一个以项目成员身份创建的问题已经被设置了“缺陷”的标签，而另外一个问题则没有设置任何标签。如图4-54所示。

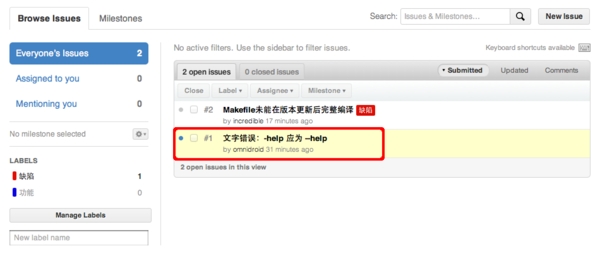
[](http://www.worldhello.net/gotgithub/images/issue-list.png)

图4-54：所有问题列表

以项目成员身份登录，在问题浏览界面即可为问题重新设定标签，指派负责人，设置里程碑，以及关闭问题等。如图4-55所示。

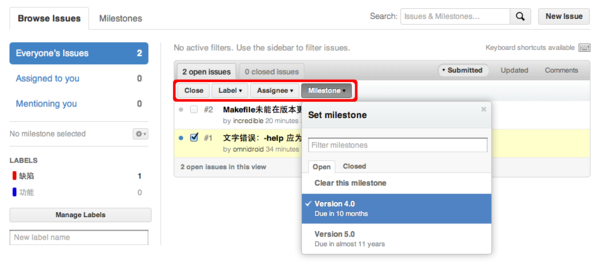
[](http://www.worldhello.net/gotgithub/images/issue-update.png)

图4-55：为问题添加指派、里程碑和标签

在问题浏览页面的过滤器中选择里程碑”Version 4.0“，可以看到两条问题都显示出来，这是因为这两条问题都属于该里程碑。里程碑的进度条显示进度为零，这是因为里程碑所包含的全部（两个）问题都处于打开状态，尚未解决。如图4-56所示。

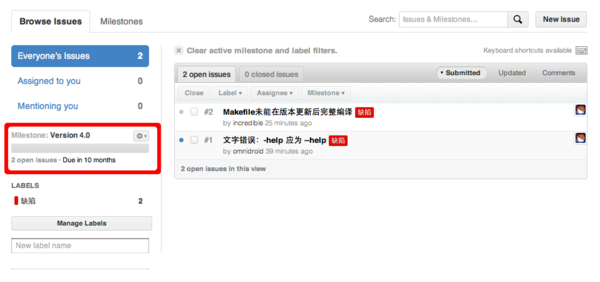
[](http://www.worldhello.net/gotgithub/images/issue-list-with-milestone.png)

图4-56：通过里程碑筛选问题

邮件通知功能是缺陷跟踪系统推动工作流的重要工具，GitHub的Issues模块也具有邮件通知功能。除了像其他缺陷跟踪系统在收到邮件通知后，访问Web界面参与问题的讨论外，还可以直接以邮件回复的功能参与到工作流中[[2]](http://www.worldhello.net/gotgithub/04-work-with-others/050-issue.html#id7)。

GitHub还支持版本库提交和问题建立关联，只要提交说明中出现“#xxx”（Issue编号）字样。如果在提交说明中的问题编号前出现特定关键字，还可以关闭问题。支持的关键字有：

* fixes #xxx
* fixed #xxx
* fix #xxx
* closes #xxx
* close #xxx
* closed #xxx

下面就以gotgithub/helloworld版本库为例，关闭编号为“#1”的问题。

* 克隆版本库，若本地工作区尚不存在。
* $ git clone git@github.com:gotgithub/helloworld.git

$ cd helloworld

* 编辑文件*src/main.c*，改正“问题#1”发现的文字错误。
* $ vi src/main.c
* $ git diff
* diff --git a/src/main.c b/src/main.c
* index 3daf9fe..f974b49 100644
* --- a/src/main.c
* +++ b/src/main.c
* @@ -19,7 +19,7 @@ int usage(int code)
* " say hello to the world.\n\n"
* " hello -v, --version\n"
* " show version.\n\n"
* - " hello -h, -help\n"
* + " hello -h, --help\n"
* " this help screen.\n\n"), \_VERSION);
* return code;

}

* 将修改添加至暂存区。

$ git add -u

* 提交，并在提交说明中用fixed #xxx关键字关闭相关问题。

$ git commit -m "Fixed #1: -help should be --help."

* 向GitHub版本库推送。

$ git push

推送完毕后，在问题浏览界面可以看到里程碑“Version 4.0”的进度已经完成了一半，即其中一个问题（#1）已经完成并关闭。如图4-57所示。

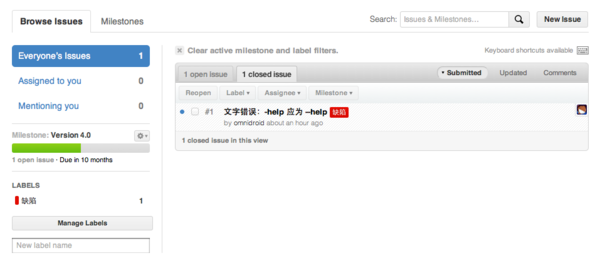
[](http://www.worldhello.net/gotgithub/images/issue-milestone-half-closed.png)

图4-57：关闭一个问题，里程碑完成50%

查看已经完成的问题（#1），可以看到其中关联到一个提交，该提交正是我们刚刚创建的。如图4-58所示。

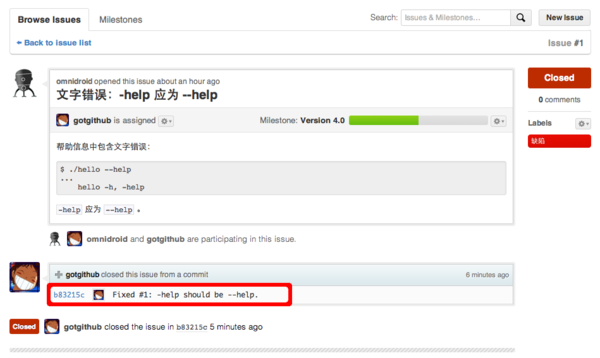
[](http://www.worldhello.net/gotgithub/images/issue-closed-by-commit.png)

图4-58：已关闭问题中的提交链接

点击关联的提交，显示如图4-59的提交界面，出现在提交说明中的问题编号也可点击，指向缺陷追踪系统中该问题的链接。

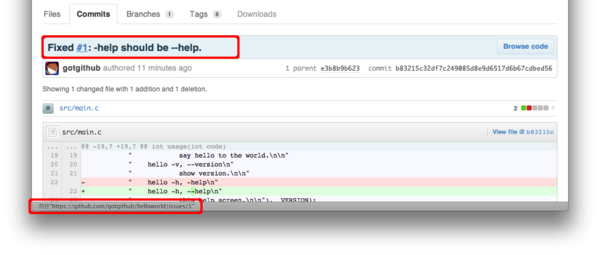
[](http://www.worldhello.net/gotgithub/images/commit-link-to-issue.png)

图4-59：提交中的问题链接

**Pull Requst也是Issue**

Pull Request和Issue一样，也是一种对项目的反馈，而且是更为主动的反馈。GitHub的Issues模块将Pull Request也纳入到问题的管理之中，完美地将Pull Request整合到问题追踪的框架之中。

为了弄清二者之间的关联，首先创建一个Pull Request。

以非项目成员（如用户 omnidroid）的账号访问gotgithub/helloworld项目，查看文件src/Makefile，点击“Fork and edit this file”按钮快速创建派生项目，如图4-60所示。

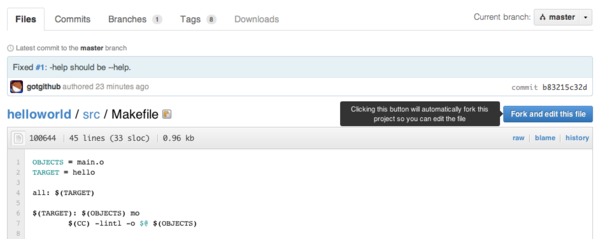
[](http://www.worldhello.net/gotgithub/images/fork-and-edit-btn-for-issue.png)

图4-60：在线编辑并创建派生项目

通过GitHub提供的在线编辑功能修改src/Makefile文件，修改完毕后撰写提交说明，点击“Propose File Change”按钮提交。如图4-61所示。

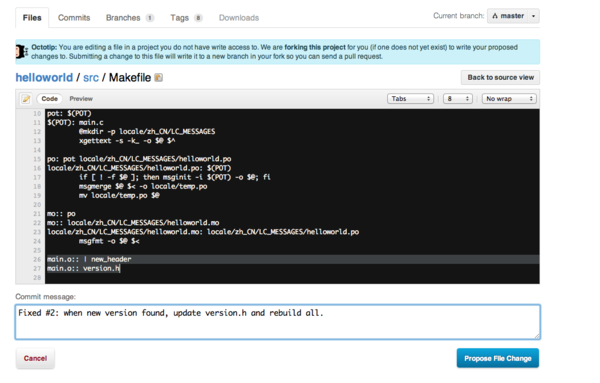
[](http://www.worldhello.net/gotgithub/images/fork-and-edit-form-for-issue.png)

图4-61：在线编辑并提交

在提交说明中特意使用了“Fixed #2”关键字，以便该提交被上游版本库接纳后能够关闭关联的问题。

当完成提交后，GitHub会自动开启创建新的Pull Request对话框，如图4-62所示。

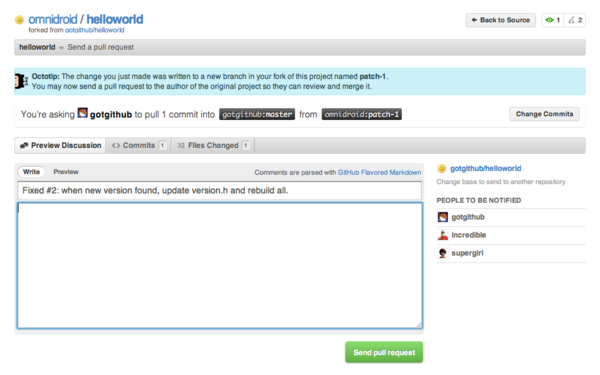
[](http://www.worldhello.net/gotgithub/images/new-pull-request-for-issue.png)

图4-62：创建Pull Request

Pull Request创建完毕后，除了在菜单项“Pull Requests”中有显示外，在“Issues”的问题浏览页面中也会显示。如图4-63所示，新建立的Pull Request的编号不是从1开始创建，而是接着问题的编号顺序创建，所以当Pull Request出现在问题列表中时，如果不注意后面的山型的分支图标，根本意识不到这不是一个普通的问题（Issue），而是一个Pull Request。

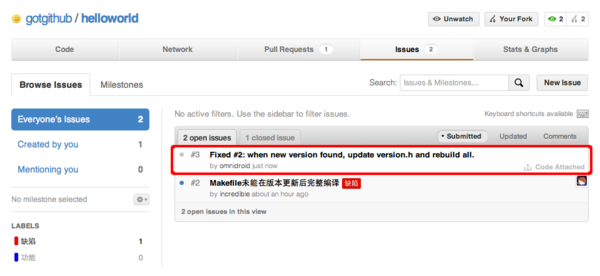
[](http://www.worldhello.net/gotgithub/images/issue-list-with-pull-request.png)

图4-63：Pull Request也显示在Issues中

显示在问题浏览界面中的Pull Request和问题一样，可以为其设置标签、指派负责人、设置里程碑。如图4-64所示。

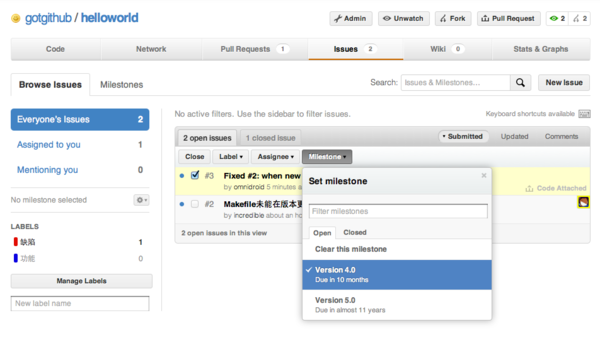
[](http://www.worldhello.net/gotgithub/images/pull-request-update-as-issue.png)

图4-64：可以像更新其他Issue那样更新Pull Request

当Pull Request归类到里程碑“Version 4.0”中时，在过滤器打开里程碑“Version 4.0”，可以看到本来已经完成50%的进度，由于新增了一个“问题”（Pull Request），导致进度降低了。如图4-65所示。

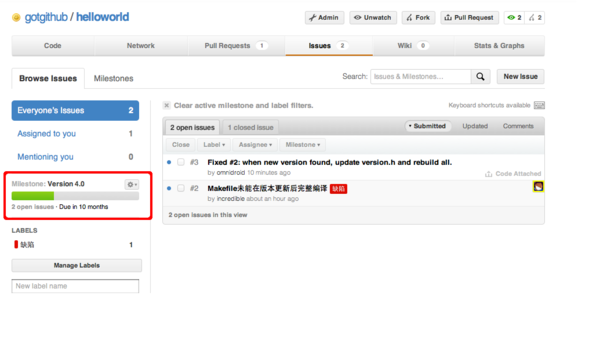
[](http://www.worldhello.net/gotgithub/images/milestone-progress-with-pull-request.png)

图4-65：里程碑进度调整

点击编号为“#3”的问题（Pull Request），会进入到Pull Request页面。点击页面中的“Merge pull request”按钮实现Pull Request的合并。如图4-66所示。

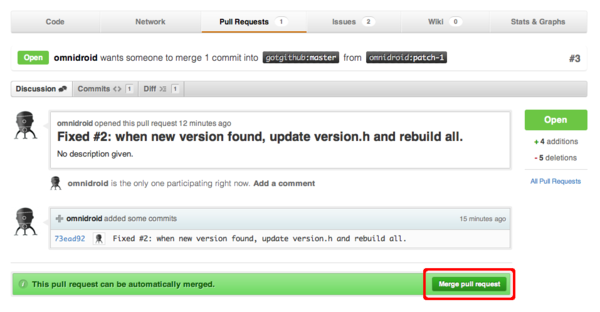
[](http://www.worldhello.net/gotgithub/images/merge-pull-request-for-issue.png)

图4-66：在线合并Pull Request

点击“Confirm Merge”确认合并，如图4-67所示。

[](http://www.worldhello.net/gotgithub/images/merge-pull-request-for-issue-confirm.png)

图4-67：确认合并Pull Request

完成合并后，查看该Pull Request，可以看到该Pull Request已经关闭。如图4-68所示。

[](http://www.worldhello.net/gotgithub/images/pull-request-close-for-issue.png)

图4-68：Pull Request自动关闭

如果再回到问题浏览界面，能够猜到现在里程碑“Version 4.0”的进度是多少么？

由于关闭了编号为“#3”的Pull Request，以及所合并的Pull Request中对应提交的提交说明的指令同时关闭了编号为“#2”的问题，所以现在里程碑“Version 4.0”关联的所有问题均已关闭。里程碑也显示已关闭，即里程碑完成度为100%。

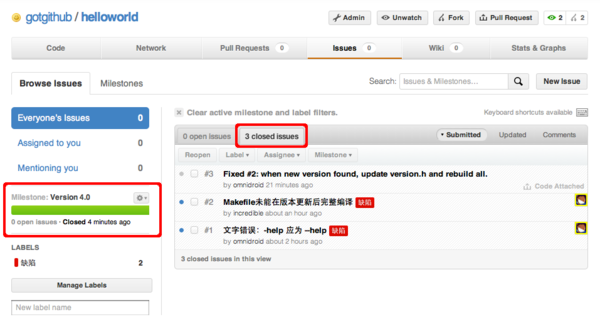
[](http://www.worldhello.net/gotgithub/images/milestone-closed.png)

图4-69：里程碑关闭

# Wiki

维基是Web协同著作平台，可以让任何浏览网页的人都能够方便地参与网页的编辑和创建。这源自于维基如下魔力：

* 快速更改。修改网页无需复杂的后台修改和网页部署流程，浏览的网页直接提供编辑按钮，任何查看网页的用户均可在线编辑网页。
* 简洁语法。编写网页不需要学习复杂的HTML，取而代之的是易学易用的格式化文本（维基语法），有的维基还提供图形化编辑界面。
* 版本控制。熟悉Git的人，可以把维基看作是Web的版本控制。历次修改都记录在案，历史修订可进行比较，可恢复到历史版本等。
* 维基链接。页面链接使用[[页面名称]]语法，可以非常方便地创建新页面，并实现页面间的互联。

GitHub提供了维基模块，方便项目团队创建社区驱动和维护的项目文档。

**维基初始化**

GitHub的维基模块可以通过项目管理页面控制开启或关闭，默认开启，如图4-70所示。因为GitHub提供了项目展示的多种途径，一些小项目如果觉得用README文件构建项目说明，或者用gh-pages分支维护项目主页就足够了，大可关闭维基模块。

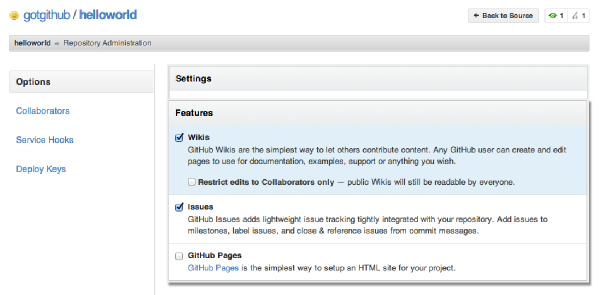
[](http://www.worldhello.net/gotgithub/images/project-admin-features-wiki.png)

图4-70：开启或关闭Wiki模块

项目启用维基后，进入维基页面，如图4-71所示，会发现维基页面并没有自动创建，还需要进行初始化。

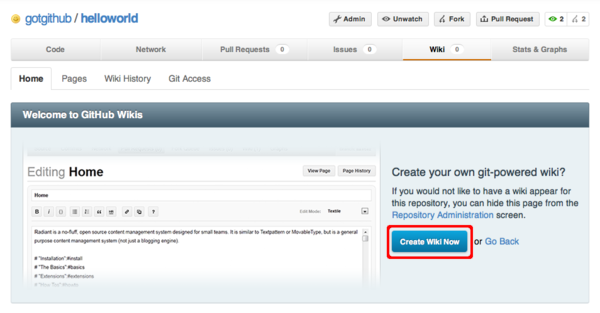
[](http://www.worldhello.net/gotgithub/images/wiki-not-exist.png)

图4-71：尚未初始化的维基界面

点击“Create Wiki Now”按钮，自动创建维基首页，如图4-72所示。

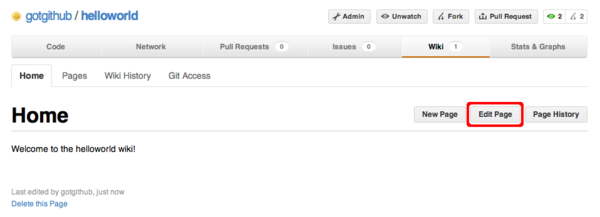
[](http://www.worldhello.net/gotgithub/images/wiki-home-init.png)

图4-72：自动创建的维基首页

**使用维基**

自动创建的维基首页只有非常简单的信息，点击编辑按钮，修改维基首页。编辑界面如图4-73所示。

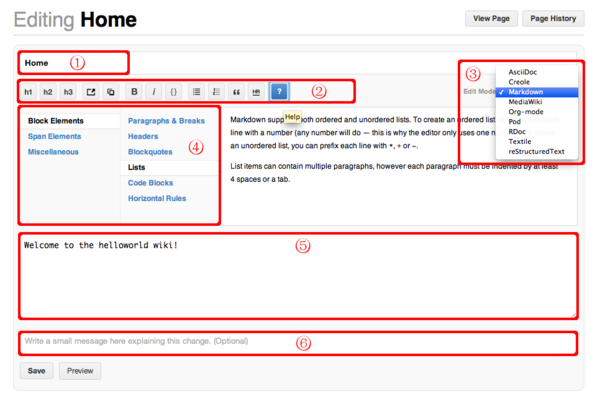
[](http://www.worldhello.net/gotgithub/images/wiki-edit-init.png)

图4-73：编辑维基首页

编辑界面有6个部分需要重点说明：

1. 页面名称。首页的页面名称为“Home”，不能随意更改，否则无法找到首页，或者页面之间的跳转会失效。
2. 工具条。从左至右分别是设置一级标题、二级标题、三级标题、插入链接、图片、字体加粗、斜体、代码块、列表、编号列表、引用、水平分割线等。
3. 页面语法格式。默认采用Markdown语法，还可以选用AsciiDoc、Creole等语法。注意如果改变语法格式，该维基页面的内容需要手工进行调整，而且页面的实际存储文件的文件扩展名会改变。
4. 语法帮助。当按下工具条中的帮助按钮会显示本语法帮助表格。
5. 维基内容编辑框。整个维基页面的内容都在这个编辑框中。鼠标拖动该编辑框右下角可以对编辑框大小进行缩放。
6. 可选的修改说明。修改页面时提供说明便于跟踪对页面的历史修订。

对初始创建的首页进行更改，如图4-74所示。

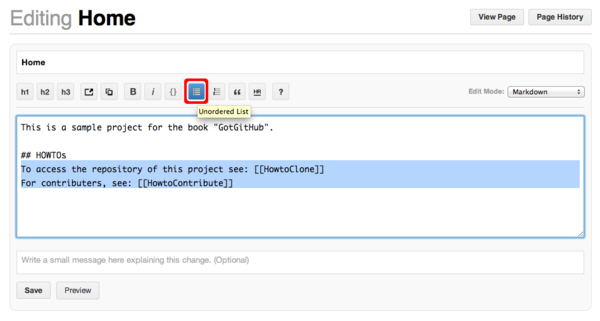
[](http://www.worldhello.net/gotgithub/images/wiki-edit-format.png)

图4-74：修改选中文本样式

图4-74中，在维基内容编辑框中选择两行文本，然后点击工具栏中的列表按钮，为选中内容应用列表样式。应用新样式后的效果如图4-75所示。然后填写提交说明，点击“Save”按钮保存更改。

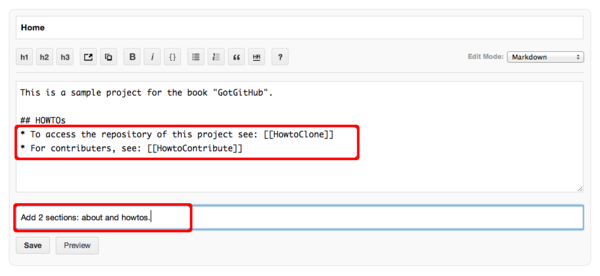
[](http://www.worldhello.net/gotgithub/images/wiki-edit-save.png)

图4-75：填写提交说明保存更改

注意图4-75的维基内容中有[[页面名称]]样式的语法，这个语法是维基特色的页面链接语法，指向另外的维基页面。实际页面输出如图4-76所示。

[](http://www.worldhello.net/gotgithub/images/wiki-home-simple-wiki-links.png)

图4-76：保存更改后的维基首页

如果对维基语法[[页面名称]]生成的链接标题不满意，还可以用[[链接标题|页面名称]]格式创建维基链接。对首页重新做一次修改，修改如下：

* 将[[HowtoClone]]改为[[how to clone|HowtoClone]]。
* 将[[HowtoContribute]]改为[[how to contribute|HowtoContribute]]。

修改后的首页效果如图4-77所示。

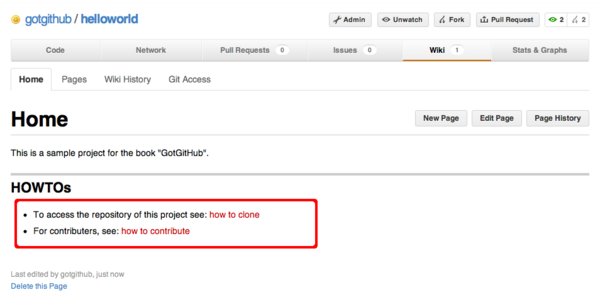
[](http://www.worldhello.net/gotgithub/images/wiki-home-update-wiki-links.png)

图4-77：修改维基链接标题后的首页

无论怎样更改维基页面都不怕内容丢失，因为维基记录了每一次修订历史，并可以回退任意一次修改。点击维基页面中的“Page History”按钮，查看页面修订历史，如图4-78所示。

[](http://www.worldhello.net/gotgithub/images/wiki-page-history.png)

图4-78：页面修订历史

首页的修订历史记录着维基初始化以来所有的修改，包括修改者、修改时间、提交说明，以及一个可点击的对象ID。点击对象ID查看对应版本的页面。还可以对不同版本的页面进行比较，选中两个版本点击“Compare Revisions”按钮，如图4-79所示。

[](http://www.worldhello.net/gotgithub/images/wiki-page-compare.png)

图4-79：页面版本间比较

在页面版本间的比较界面中，提供回退此次修改的按钮。点击“Revert Changes”按钮（图4-79所示），可以回退对首页的修改。查看首页的修订历史，会看到回退记录也显示其中，如图4-80所示。

[](http://www.worldhello.net/gotgithub/images/wiki-reverted.png)

图4-80：包含回退记录的页面修订历史

在维基中创建新页面有多种方法，可以点击页面中的“New Page”按钮，也可以像我们之前做的那样先在页面中用[[页面名称]]格式嵌入维基链接，然后在生成的页面中可以看到指向新页面的链接，当然这些链接所指向的页面并不存在。

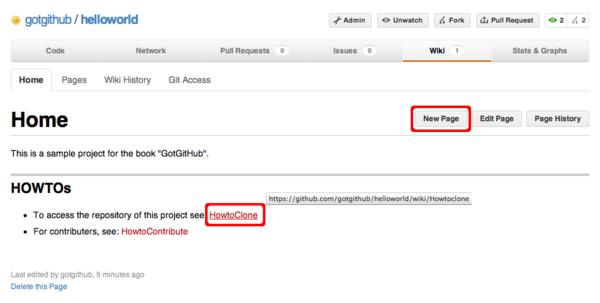
[](http://www.worldhello.net/gotgithub/images/wiki-newpage.png)

图4-81：页面中的维基链接

如图4-81所示，点击页面中指向不存在维基页面的链接，会自动开启创建新页面的对话框，如图4-82所示。

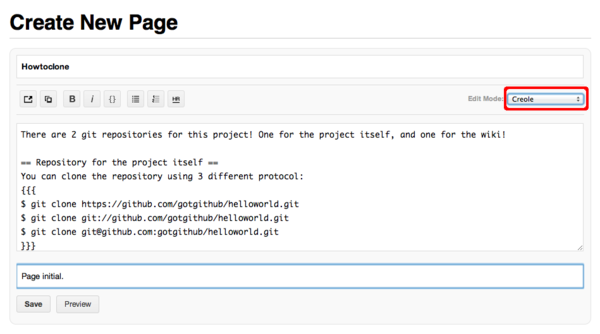
[](http://www.worldhello.net/gotgithub/images/wiki-newpage-edit.png)

图4-82：创建新维基页面

输入维基页面的内容，然后填写提交说明，点击“Save”按钮，保存新页面。生成的新页面如图4-83所示。

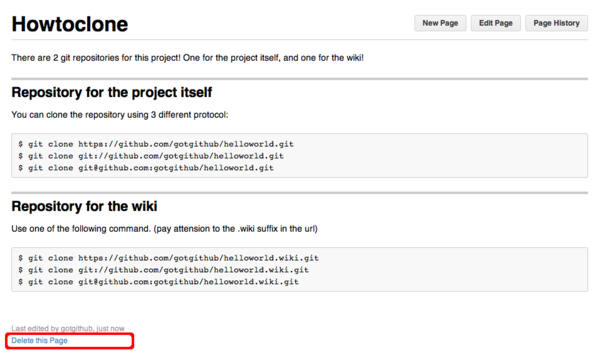
[](http://www.worldhello.net/gotgithub/images/wiki-newpage-created.png)

图4-83：生成的新页面

如果当前用户对页面具有写权限，则在页面左下角会看到一个删除本页面的链接。点击“Delete this page”链接并经确认后会删除页面。然后继续在维基中操作，如创建另外一个新页面HowtoContribute。

如果对之前删除页面Howtoclone的操作后悔，可以通过下面方法找回。

* 访问菜单中的“Wiki History”项，显示整个维基的修订记录（不是某个页面的修订记录）。如图4-84所示。

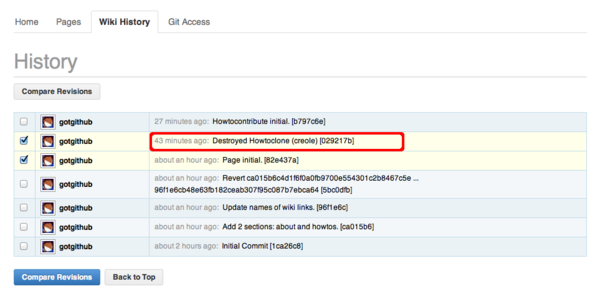
[](http://www.worldhello.net/gotgithub/images/wiki-history.png)

图4-84：维基修订记录

* 从图4-84可见上面第二条记录就是删除HowtoClone页面的操作，选择该记录及前一次记录，执行版本比较，如图4-85所示。

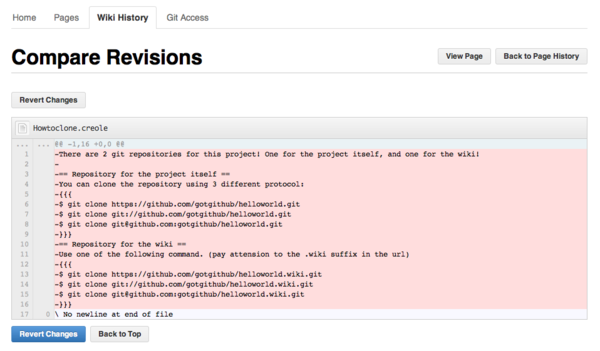
[](http://www.worldhello.net/gotgithub/images/wiki-compare-drop-page.png)

图4-85：版本比较

* 点击“Revert Changes”按钮，可以取消页面删除动作。更新后的维基修订历史如图4-86所示。

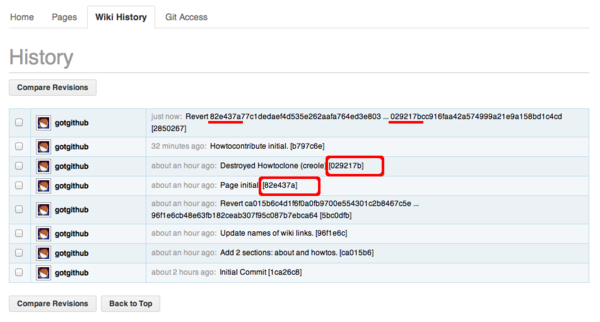
[](http://www.worldhello.net/gotgithub/images/wiki-page-restored.png)

图4-86：还原修订后的维基修订记录

* 查看维基页面列表，可以看到页面Howtoclone已经被找回。

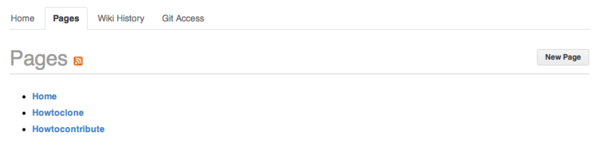
[](http://www.worldhello.net/gotgithub/images/wiki-pages.png)

图4-87：维基页面列表

**维基与Git**

随着对GitHub维基的深入使用，可能会遇到下面的问题：如何嵌入图片？多人编辑时如何避免冲突？解决这几个问题的办法就是用Git操作维基。在 之前查看维基修订历史，以及进行版本间比较时可能已经看出和Git是如何的相似，实际上GitHub的维基页面就是用Git版本库实现的。

在维基页面访问菜单中的“Git Access”项，会看到用Git访问维基页面的方法。如图4-88所示。

[](http://www.worldhello.net/gotgithub/images/wiki-git-access.png)

图4-88：用Git访问维基

对于项目gotgithub/helloworld来说，用Git克隆其维基，用如下命令：

$ git clone git@github.com:gotgithub/helloworld.wiki.git

进入到刚刚克隆的helloworld.wiki工作区中，查看包含的文件，会看到有三个文件。

$ cd helloworld.wiki

$ ls

Home.md Howtoclone.creole Howtocontribute.md

三个文件对应于三个维基页面，文件名就是维基的页面名称，而扩展名对应于采用的维基语法。以.md扩展名结尾的页面采用Markdown语法，而以.creole结尾的文件采用Creole标准维基语法。

下面就通过Git在维基版本库中添加一个图片。添加图片的操作只通过GitHub维基的Web界面是很难实现的，而使用Git则易如反掌。

* 创建一个名为images目录。这个目录并非必须，只是为了易于管理。
* $ mkdir images

$ cd images

* 在images目录中添加图片。

下面的操作从GitHub官方版本库中下载图片octocat.png并进行适当缩放。

$ wget https://github.com/github/media/raw/master/octocats/octocat.png

$ mogrify -resize '200' octocat.png

* 将图片添加到暂存区并提交。
* $ git add octocat.png

$ git commit -m "add sample image."

* 将本地提交推送到GitHub远程版本库。

$ git push

完成推送后，访问下面的网址可以看到刚刚上传的图片：

https://github.com/gotgithub/helloworld/wiki/images/octocat.png

接下来在维基页面中引用图片。嵌入图片的Markdown语法是：![Alt text here](Image URL here)。当然可以通过编辑本地版本库gotgithub/helloworld.wiki.git中的文件，但通过GitHub维基编辑界面嵌入图片无需记忆复杂的语法。如图4-89所示。

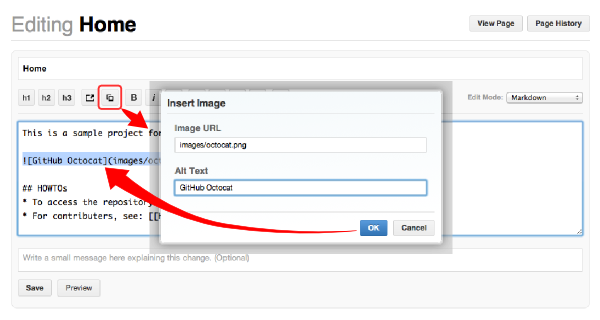
[](http://www.worldhello.net/gotgithub/images/wiki-insert-image.png)

图4-89：在维基页面中嵌入图片

点击“Preview”按钮，可以在保存前查看效果。在图4-90所示的预览界面中可以看到修改后的效果。

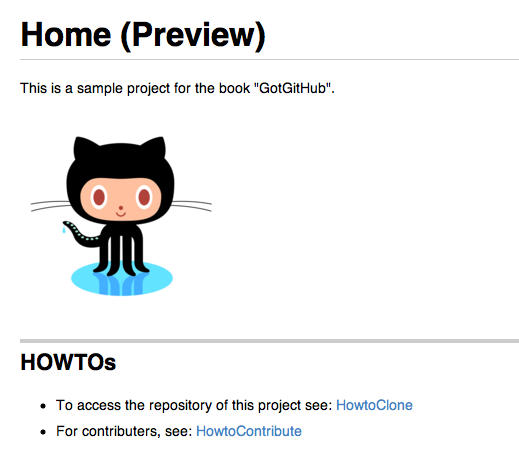
[](http://www.worldhello.net/gotgithub/images/wiki-insert-image-preview.png)

图4-90：预览效果

多人同时编辑一个维基页面会引起冲突，先提交的用户会成功，其他用户的编辑界面马上会显示冲突警告，并且保存按钮也被置灰，如图4-91所示。

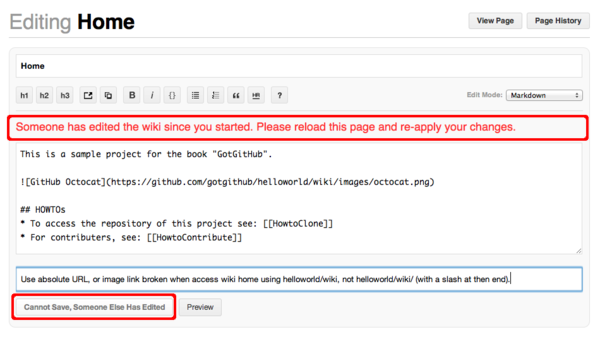
[](http://www.worldhello.net/gotgithub/images/wiki-edit-conflict.png)

图4-91：编辑冲突

GitHub的维基编辑界面没有提供冲突解决的工具，而利用Git本身强大的冲突解决功能可以很容易地解决这一难题。

例如用户gotgithub编辑维基首页Home遇到编辑冲突，为防止数据丢失，先将编辑框中的维基文本拷贝并粘贴到一个临时文件中，如文件中/path/to/draft.md。然后进行如下操作将draft.md中内容合并到维基页面中。

* 如果本地已经克隆维基版本库，则执行下面命令更新。

$ cd helloworld.wiki

$ git pull

* 如果没有，则克隆维基版本库。

$ git clone git@github.com:gotgithub/helloworld.wiki.git

$ cd helloworld.wiki

* 用Git命令查看版本库的历史，以便找出发生冲突的原始版本。

从下面的输出可以看出我们编辑的版本是基于提交fbb4bb4，由于用户incredible先于我们完成了对维基页面的修改以致发生了冲突。

$ git log -3 --pretty=short

commit 5ff5d998bb6cf99337813915282df94701d17ea0

Author: incredible <incredible@ossxp.com>

Add a note as image link broken if url without a end slash.

commit fbb4bb4f330bacf765d51736359b0a3e81ed945b

Author: gotgithub <gotgithub@gmail.com>

Insert image in page.

commit 94182c2b57ebce1f1bf8a310f78df87ae8e8219a

Author: gotgithub <gotgithub@gmail.com>

add sample image.

* 基于提交fbb4bb4建立分支，如分支mywiki。

$ git checkout -b mywiki fbb4bb4

* 将保存的draft.md覆盖欲修改的文件，如Home.md。

$ cp /path/to/draft.md Home.md

* 提交修改。

$ git add -u

$ git commit -m "Use absolute image link."

* 切换到master分支。

$ git checkout master

* 合并我们在mywiki分支的修改。

$ git merge mywiki

Auto-merging Home.md

CONFLICT (content): Merge conflict in Home.md

Automatic merge failed; fix conflicts and then commit the result.

* 调用图形工具解决冲突。

$ git mergetool

* 提交并查看合并后的提交关系图。

$ git commit -m "merge with incredible's edit."

$ git log --oneline --graph -4

\* d33b55a merge with incredible's edit.

|\

| \* 121c3b2 Use absolute image link.

\* | 5ff5d99 Add a note as image link broken if url without a end slash.

|/

\* fbb4bb4 Insert image in page.

* 查看用户incredible的修改。

$ git show --oneline HEAD^1

5ff5d99 Add a note as image link broken if url without a end slash.

diff --git a/Home.md b/Home.md

index 6ada8e8..0bca3ec 100644

--- a/Home.md

+++ b/Home.md

@@ -1,5 +1,7 @@

This is a sample project for the book "GotGitHub".

+\*\*Note\*\*: if can not see the following image, add a slash('/') at the end of the URL.

+

![GitHub Octocat](images/octocat.png)

## HOWTOs

* 查看用户gotgithub的修改。

$ git show --oneline HEAD^2

121c3b2 Use absolute image link.

diff --git a/Home.md b/Home.md

index 6ada8e8..cdb9167 100644

--- a/Home.md

+++ b/Home.md

@@ -1,6 +1,6 @@

This is a sample project for the book "GotGitHub".

-![GitHub Octocat](images/octocat.png)

+![GitHub Octocat](https://github.com/gotgithub/helloworld/wiki/images/octocat.png)

## HOWTOs

\* To access the repository of this project see: [[HowtoClone]]

* 将本地合并后的版本库推送到GitHub。

$ git push

再来看看推送后GitHub的维基修订历史，和本地版本库看到的历史是一致的，如图4-92所示。

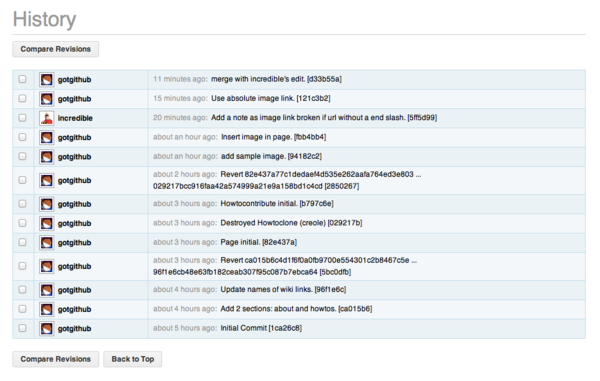
[](http://www.worldhello.net/gotgithub/images/wiki-pushed-history.png)

图4-92：推送后的维基修订历史

GitHub维基背后的引擎名为Gollum，GitHub已将其开源，项目网址：<https://github.com/github/gollum> 。安装Gollum，在克隆的维基版本库中运行gollum就可以在本地启动维基服务。

# 代码评注

针对项目的每一次Pull Request就相当于一次代码评审，评审以讨论的形式显示在Pull Request中。

在Pull Request中还能够看到对应的提交（一个或多个），并可以直接针对提交进行代码评注。对于采用集中式协同的项目，即使较少使用 Pull Request，也同样可以使用代码评注。代码评注会触发通知邮件给项目的开发者。

代码评注有两种形式，一种是针对整个提交的评注，另外一种是对代码进行逐行评注。

**提交评注**

查看项目的提交历史，从中选择一个提交，如图4-37所示。

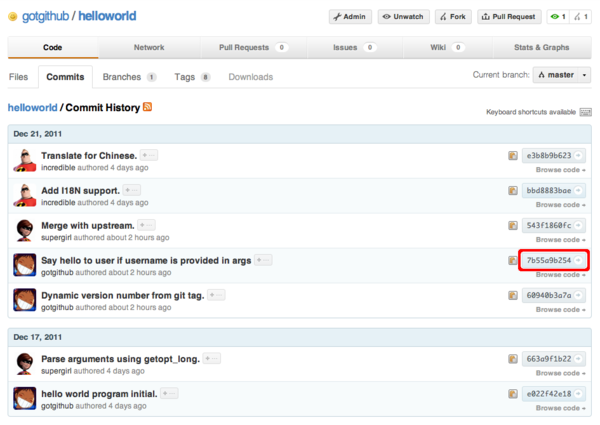
[](http://www.worldhello.net/gotgithub/images/commit-history.png)

图4-37：helloworld项目提交历史

如图4-38是查看提交的界面。除了提交说明、提交者信息之外，还显示提交所修改的文件和改动差异。在查看提交页面的最下方显示一个提交评注对话框，可以在其中写下评注。评注可以使用 Markdown 语法。

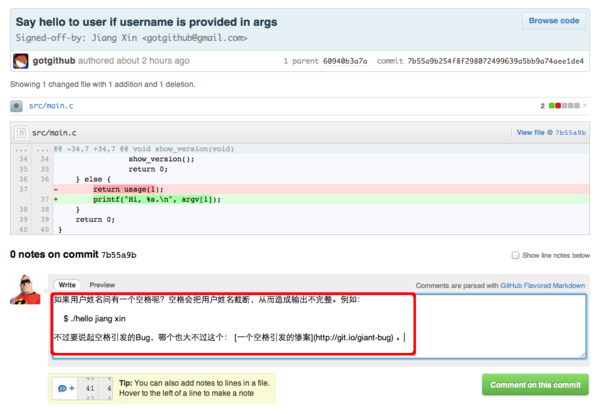
[](http://www.worldhello.net/gotgithub/images/commit-note.png)

图4-38：添加提交评注

添加评注后，所评注的提交的作者会收到通知邮件，提醒针对自己的提交有了新的评论。通知邮件如图4-39所示。

[](http://www.worldhello.net/gotgithub/images/commit-note-email-notify.png)

图4-39：提交评注的通知邮件

通过Web界面可以看到添加在提交下方的评注，并可以撰写新的评注展开讨论。评注者本人或提交的作者还可以编辑甚至删除评注。如图4-40所示。

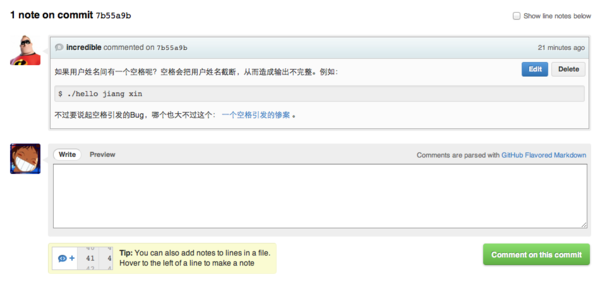
[](http://www.worldhello.net/gotgithub/images/commit-note-admin.png)

图4-40：提交评注

GitHub还支持Git自身提供的评注功能，如图4-41所示的是提交<http://git.io/git-notes>[[2]](http://www.worldhello.net/gotgithub/04-work-with-others/040-code-review.html#id8)的评注，这个评注并非通过GitHub添加的，而是由git-note命令提交的评注。这种评注针对一个特定提交只能有一个，GitHub只能显示不能编辑和删除。关于如何通过命令行查看git-note格式的评注，参见《Git权威指南》第570页“41.5 Git评注”。

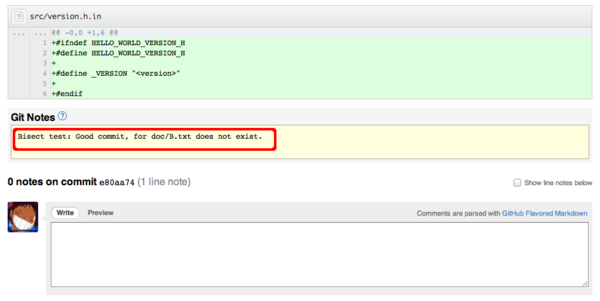
[](http://www.worldhello.net/gotgithub/images/commit-note-by-git-note.png)

图4-41：git-note评注

**逐行评注**

还是以gotgithub/helloworld版本库中的提交为例，看一下GitHub支持的逐行评注功能，即针对提交中的任意一行添加评注。浏览提交，如图4-42所示，当鼠标置于任意一行代码时，在该行代码的左侧会显示一个添加注释的图标。

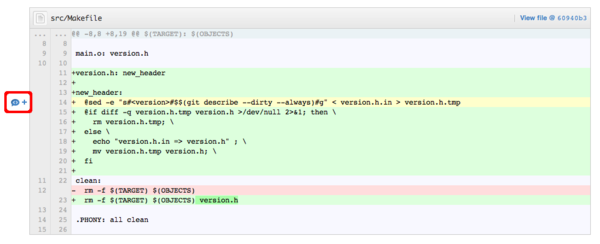
[](http://www.worldhello.net/gotgithub/images/commit-line-note-btn.png)

图4-42：添加逐行评注按钮

点击该图标（用于添加逐行评注的图标），会显示如图4-43所示的添加逐行评注对话框。该评注对话框出现在两行代码之间，在其中写下评注。

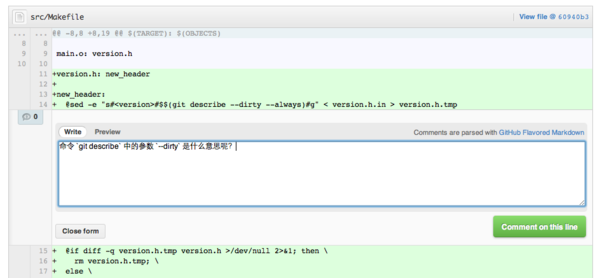
[](http://www.worldhello.net/gotgithub/images/commit-line-note-form.png)

图4-43：添加逐行评注

添加评注后，项目的开发人员同样会收到邮件通知。针对同一行代码的多次评论按时间顺序依次显示，图4-44展示了多个行间评注，其中一个评注还使用 Markdown 语法嵌入了一个图片。

[](http://www.worldhello.net/gotgithub/images/commit-line-note.png)

图4-44：逐行评注和提交评注

# 分析

有两个工具-图形和网络，让我们能洞察存储库的变化。[Github Graphs](https://github.com/blog/1093-introducing-the-new-github-graphs) 提供了代码库的合作者，以及代码提交的直观展现，而[Github Network](https://github.com/blog/39-say-hello-to-the-network-graph-visualizer)可视化直观地展现了每一个贡献者和他们在所有分支上的代码提交。这些分析和图形非常强大，尤其是当在团队中工作。

**图(Graphs)**

图提供了详细的分析，包括：

* **贡献者**：有哪些代码提交者？他们增加或者删除了多少代码行？
* **代码提交活动**：在过去的一年中，这些代码提交主要发生在哪些周？
* **代码频率**：在整个项目的生命周期的不同阶段，提交了多少代码行？
* **记录卡**：代码提交通常发生在每一天的什么时候？

**网络(Network)**

[GitHub网络(Network)](https://github.com/blog/39-say-hello-to-the-network-graph-visualizer)是一个非常强大的工具，让我们能看到每一个贡献者的代码提交，以及这些提交与其他的提交有什么关联。当我们作为一个整体观看这个网络的可视化展现时，我们能看到每一个库，每一个分支，和每一个提交，

