# GitHub的简单使用

## 实验介绍

### 关于本实验

Git已经成为程序员必备技能之一，而GitHub作为流行的Git仓库托管平台，不仅提供Git仓库托管，还是一个非常优秀的技术人员社交平台，技术人员可以通过开源的项目进行协作、交流，是现在优秀的工程师必须娴熟运用的方法。

### 实验目的

了解Git的基本操作与相关原理。

了解在Github上协作的基本流程。

了解Github项目的DevOps思想。

## 实验任务配置

### 注册GitHub

使用GitHub，第一步是注册GitHub账号。

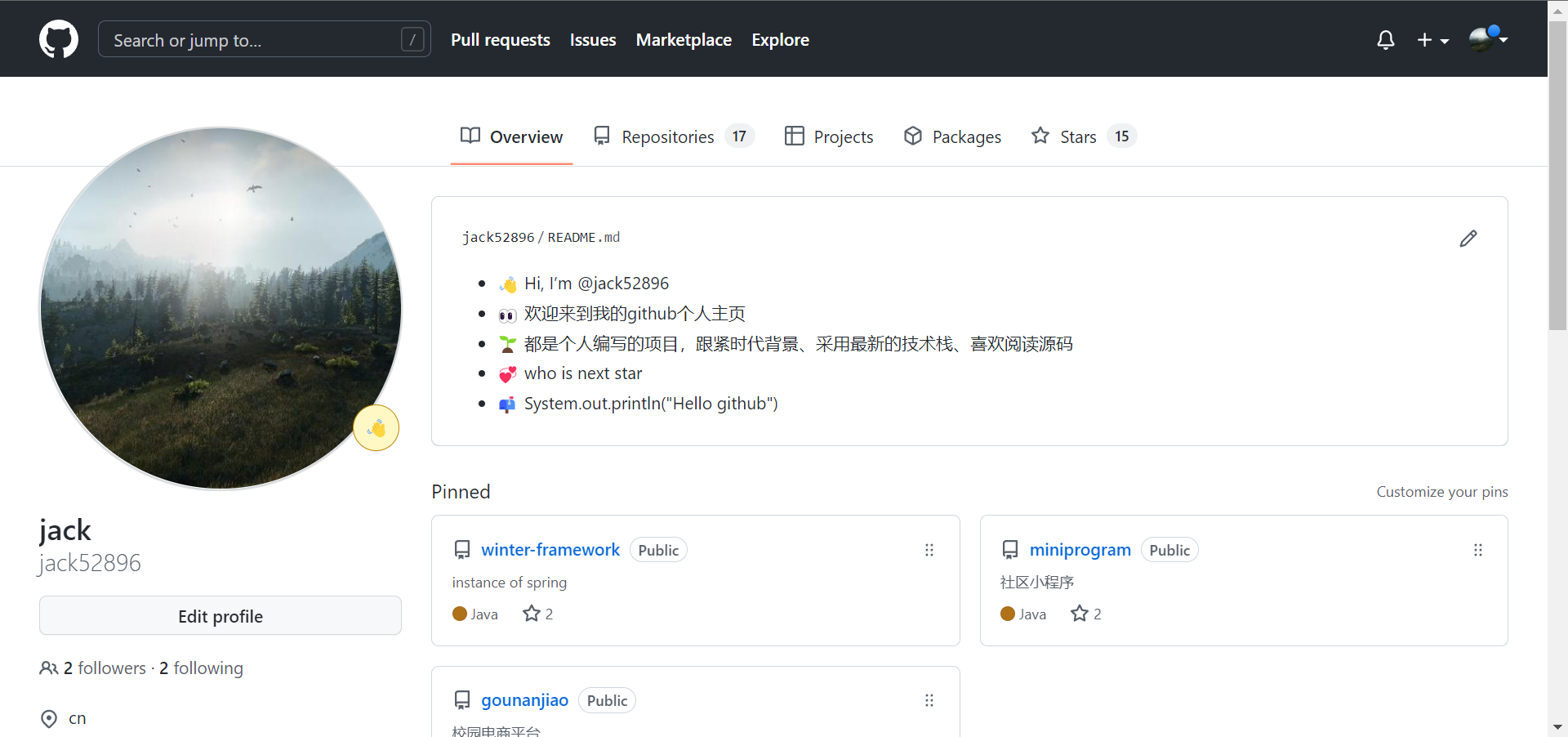
打开GitHub首页

打开浏览器，在地址栏中输入GitHub的网址：https://www.github.com/ 后按回车键。

注册

点击网页右上方的Sign Up按钮，进入注册页面。

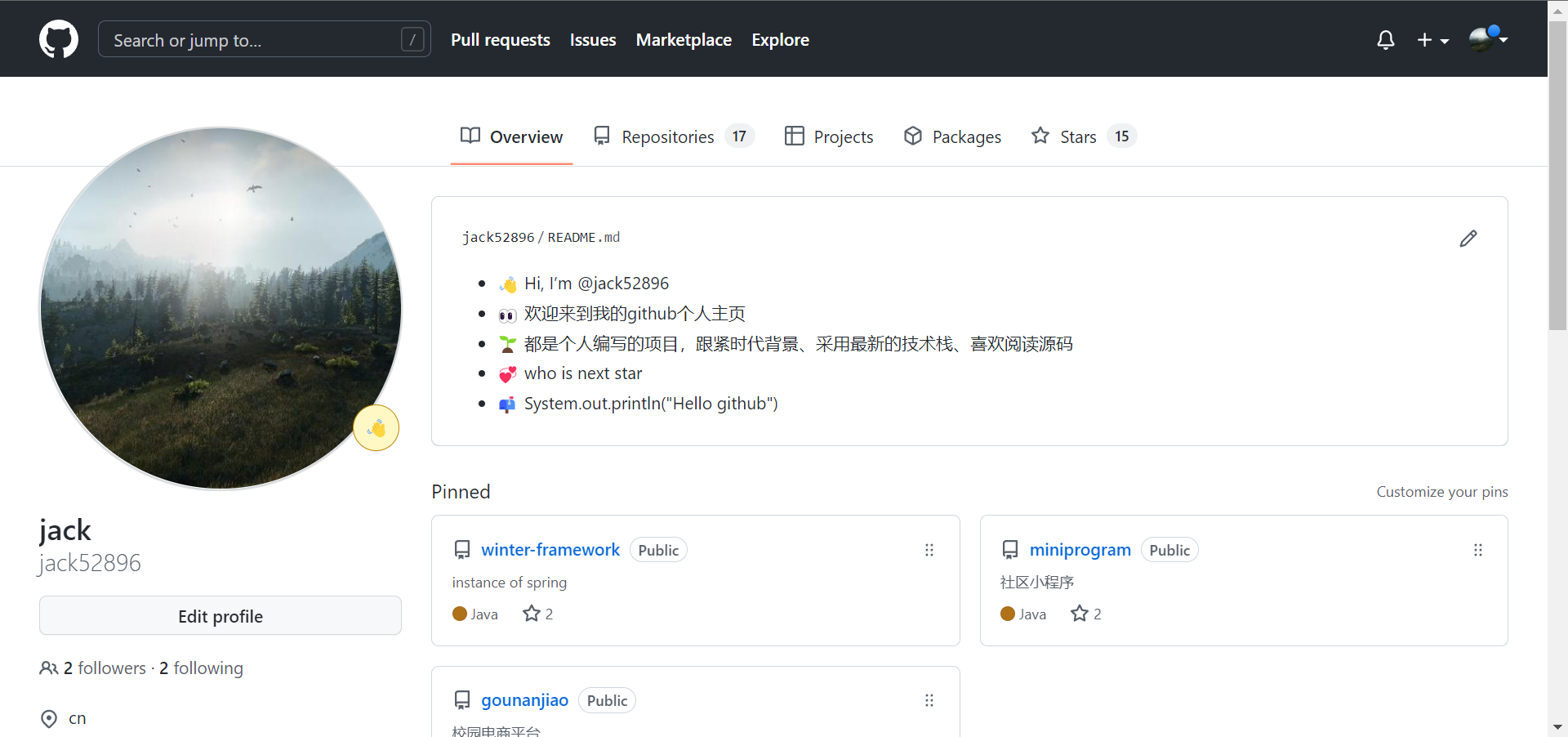
在新建账户页面中，根据页面提示输入昵称、邮箱、密码等信息，完成后点击Create Account按钮，如图1-1所示。



GitHub注册页面

### GitHub界面总览

GitHub界面的总体功能，如图1-2所示。



GitHub个人主页

图中标记的序号含义如下：

（1）进入GitHub主页的按钮。

（2）搜索框，用户可以在这里搜索一些开源项目。

（3）用户的头像。

（4）用户的昵称（账号名称），用户可以在设置中设置昵称。

（5）提示用户添加一个类似于个人描述或个性签名的东西。

（6）此处有三个数据：Followers，追随（关注）你的人；Following，你追随（关注）的人和Starred，你获得点赞（Star）的数量。

（7）Pull requests：即其他人向用户的仓库提交合并请求。

（8）Issues：即其他人对用户的项目提的问题。

（9）用户的项目仓库。

（10）用户每天向GitHub提交的贡献的分布图。

读者可以参考相关的专业书籍详细了解更多相关知识。

### 安装Git

下载Git安装包

在Git的官网（https://git-scm.com/）下载安装包并执行安装程序。图1-3显示的为Windows操作系统下安装Git后开始菜单中相关的快捷方式。

图形用户界面

中度可信度描述已自动生成

Windows操作系统下Git相关快捷方式

Git相关参数配置

安装完成后，启动命令行后首先设置Git的用户名：

git config --global user.name "YOUR NAME"

然后设置Git用户的电子邮箱地址：

git config --global user.email your@email.com

如图1-4所示。

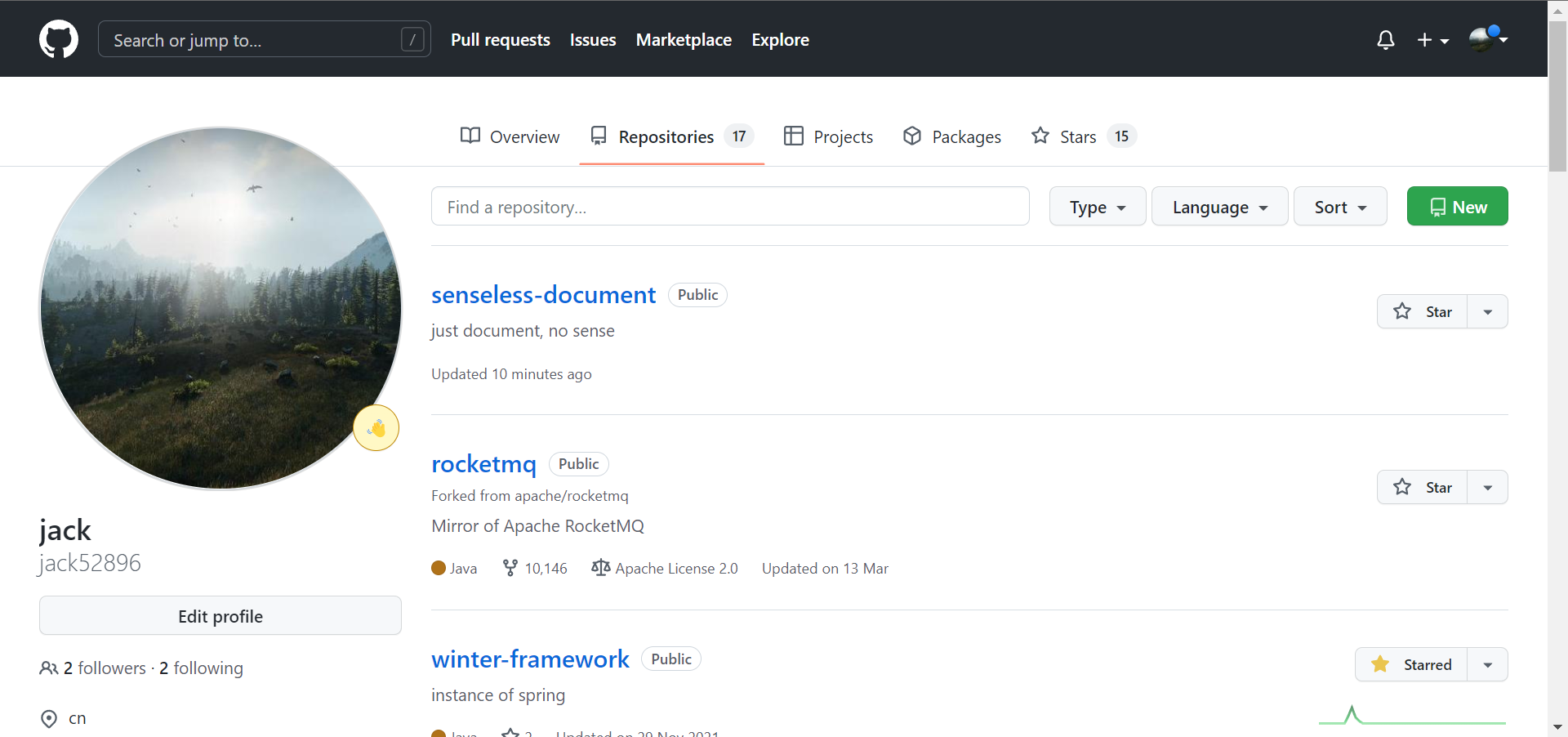
文本

描述已自动生成

配置Git用户名与邮箱

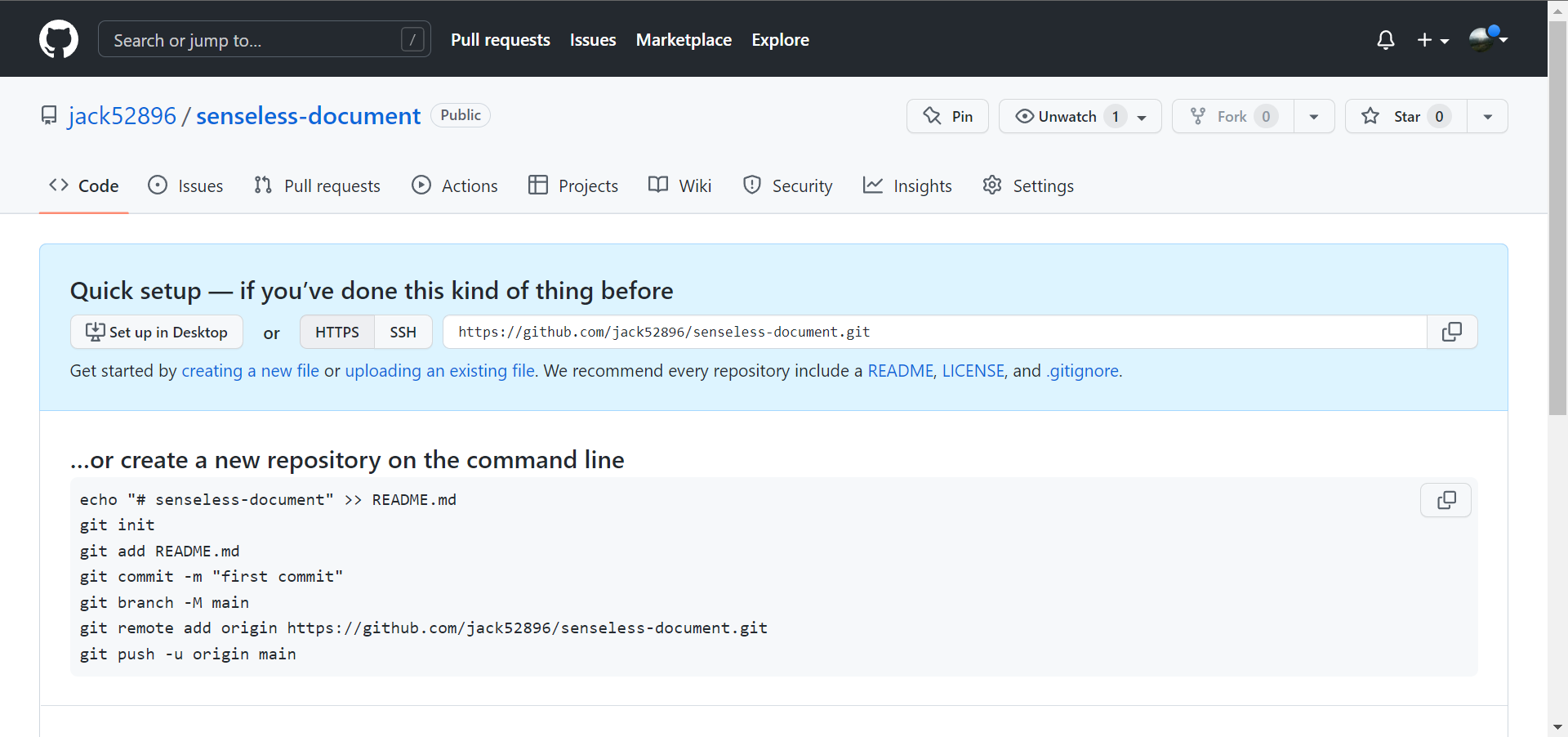
### 在GitHub上新建仓库

在任意的页面右上角单击+，然后单击新建仓库“New”，如图1-5所示。



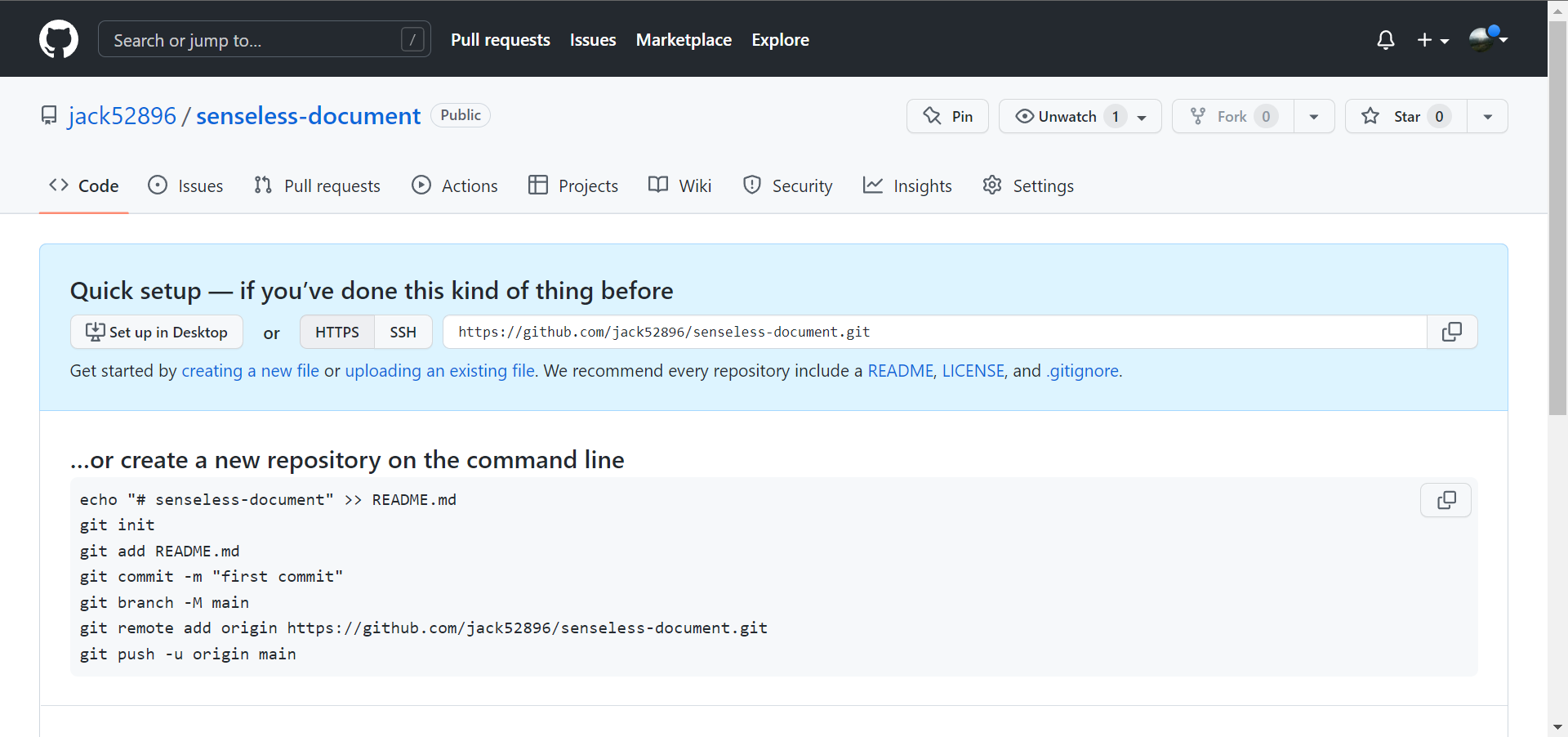
新建仓库

为仓库创建一个简短便于记忆的名字。例如“hello-world”，如图1-6所示。



为仓库命名

为仓库添加描述（可选），例如“My first repository on GitHub”，如图1-7所示。



为仓库添加描述

选择仓库类型。仓库类型分为公有（Public）或者私有（Private），具体如下。

Public：公有仓库对于刚入门的新手来说是不错的选择。这些仓库在GitHub上对于每个人是可见，用户可以从协作型社区中受益。

Private：私有仓库需要更多地步骤。它们只对于用户来说是可用的，这个仓库的所有者属于用户和及其指定要分享的合作者。

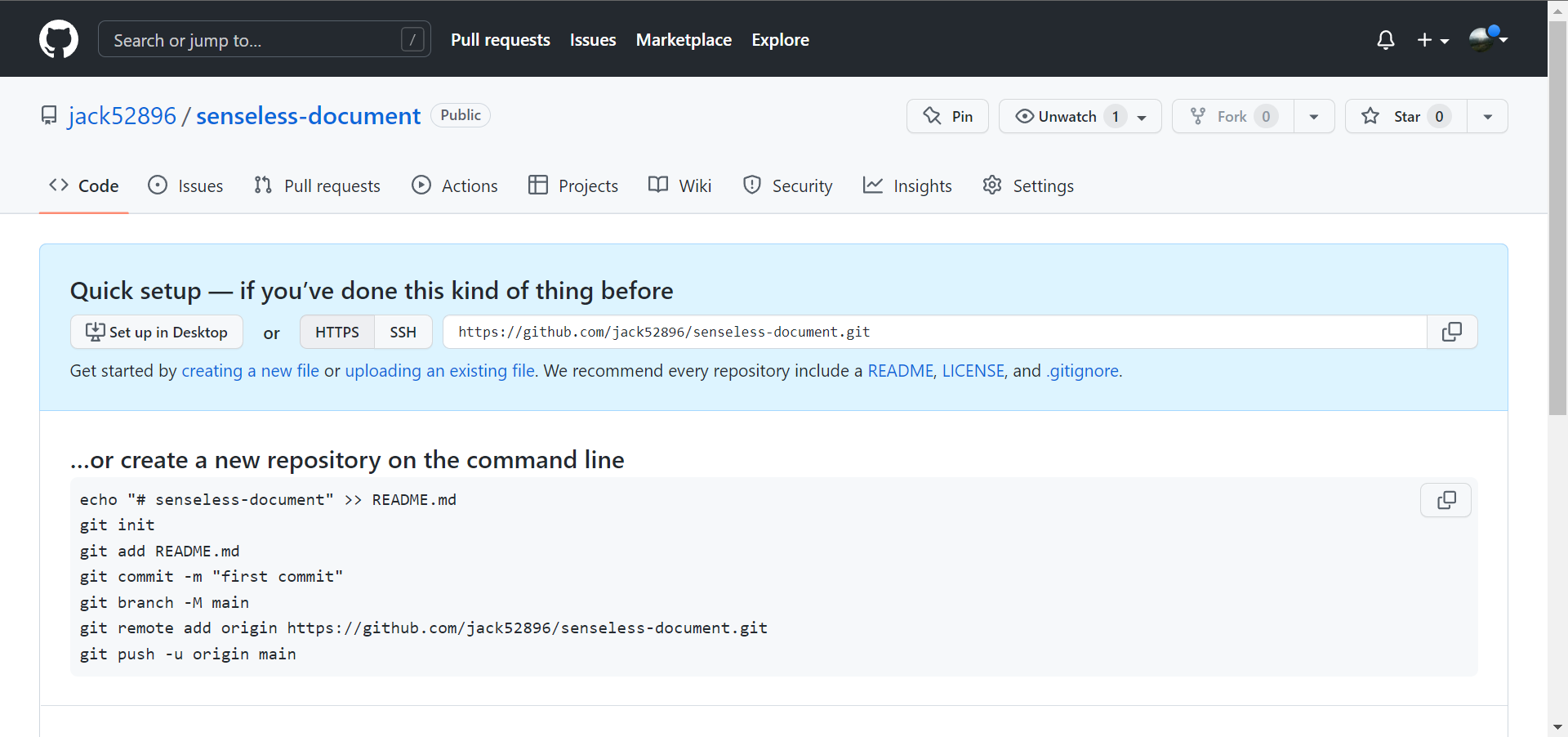
本例选择“Public”，如图1-8所示。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

设定仓库类型

选择初始化文件，选择“Initialize this repository with a README”，如图1-9所示。



选择初始化文件

单击“Create repository”，完成创建一个仓库。

至此第一个仓库成功创建，并且通过README文件进行了初始化。

### 提交第一个更改

一个提交就像项目里的文件在一个特定时间点上的快照一样。

当创建了一个新仓库，通过README文件初始化。README文件里有关于这个项目的详细解释，或者添加一些关于如何安装或者使用该项目的文档。README文件的内容会自动地显示在仓库的首页。

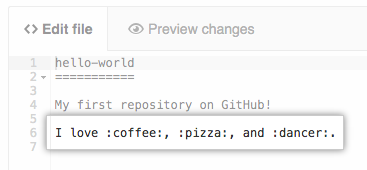
以下提交一个对README文件的修改。

在仓库的文件列表，单击README.md文件，如图1-10所示。

仓库中的README文件

在文件内容的右上方，单击  按钮。

在Edit file标签上，输入一些关于用户的信息，如图1-11所示。



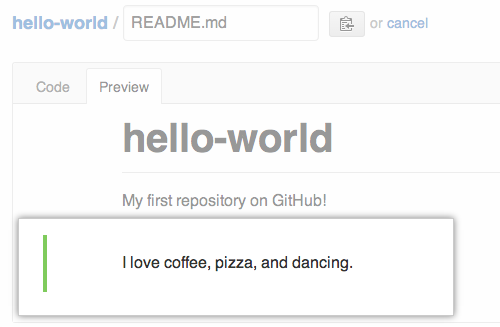
修改README文件

在新内容的上方，单击“Preview”，如图1-12所示。



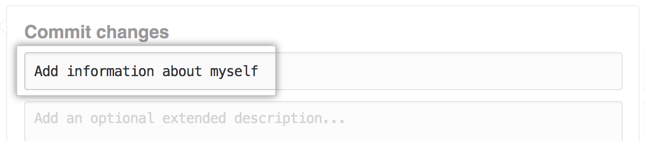
预览修改

检查用户对这个文件进行的更改，会看到新的内容被绿色标记，如图1-13所示。



检查修改

在页面的底部，即 "Commit changes" 下方，输入一些简短、有意义的提交信息来解释用户对这个文件所进行的修改，如图1-14所示。



添加提交注释

单击“Commit changes”，完成提交。

### Fork一个示例仓库

Fork是对一个仓库的复制。复制一个仓库允许自由试验各种改变，而不影响原始的项目。

一般来说，Fork 被用于更改别人的项目（贡献代码给已经开源的项目）或者使用别人的项目作为用户想法的初始开发点。

一种方式是更改别人的项目。使用Fork提出改变的一个很好的例子是漏洞修复。与其记录一个发现的问题，不如采用如下方式。

⦁ Fork这个仓库；

⦁ 进行修复；

⦁ 向这个项目的拥有者提交一个pull requset；

⦁ 如果这个项目的拥有者认同这些成果，他们可能会将这些修复更新到原始的仓库中。

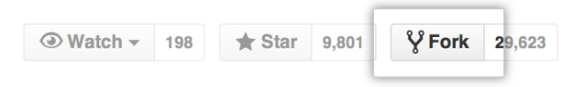
另一种方式是使用别人的项目作为用户想法的初始开发点。开源的核心是共享代码，我们可以制作更好、更可靠的软件。

事实上，当用户在GitHub上创建一个仓库时，可以选择自动包含一个许可文件，这个文件决定用户是否希望将项目分享给其他人。

Fork一个仓库分为两步：

⦁ 在GitHub上，定位到octocat/Spoon-Knife仓库。

⦁ 在页面右上角，单击“Fork”按钮，如图1-15所示。



Fork一个仓库

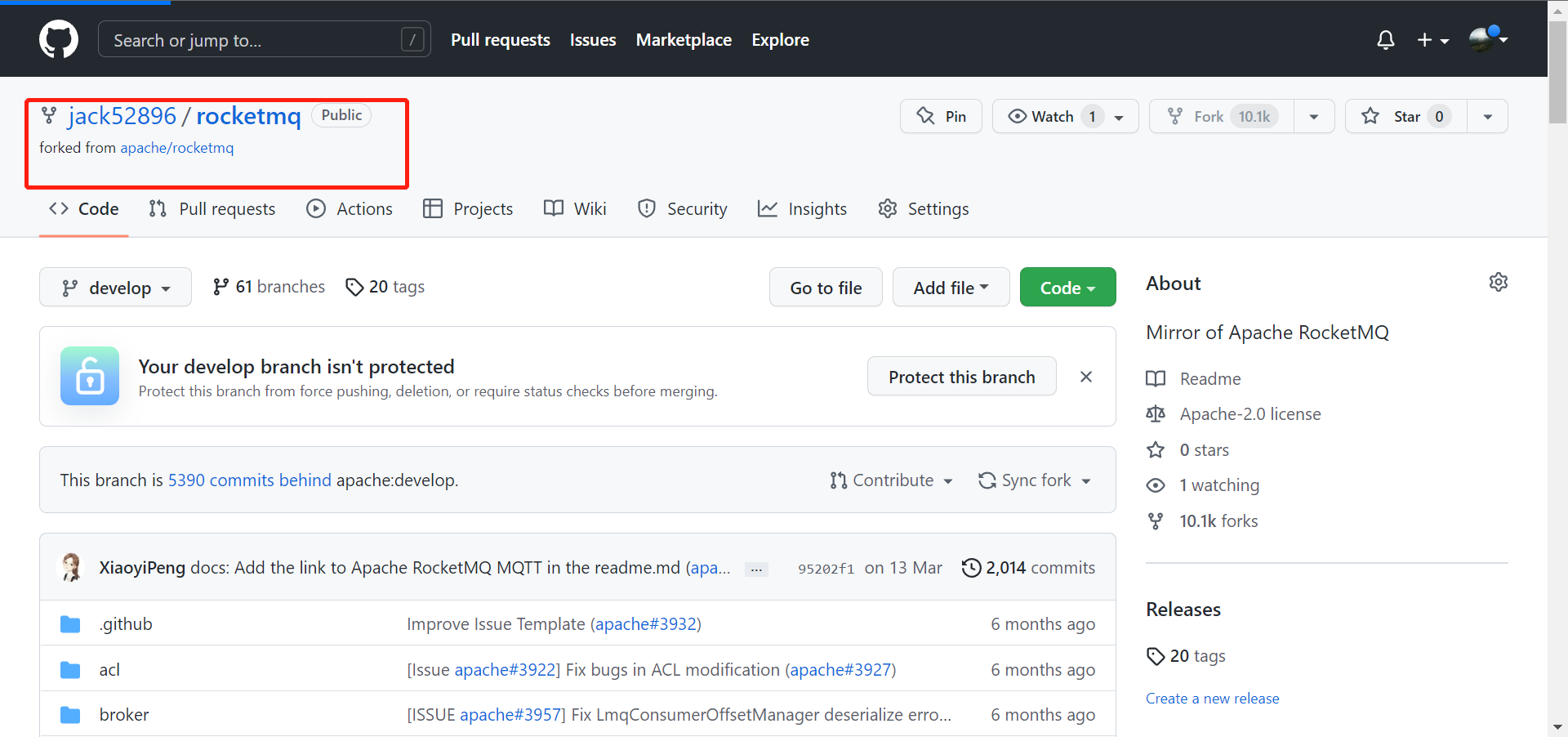
这样就已经Fork这个原始的octocat/Spoon-Knife仓库。

### 同步Fork仓库

在Fork一个项目为了提交更改向Upstream或原始仓库的情况下，已经很好地实现了将Upstream仓库定期同步到Fork。要做到这一点，需要在命令行中使用Git，可以使用刚Fork的octocat/Spoon-Knife仓库去练习设置Upstream仓库。

在GitHub上，定位到Fork的Spoon-Knife仓库。

在Fork的仓库页面的右侧边栏，单击Code下拉菜单，复制HTTPS的URL，如图1-16所示：



复制Clone用的仓库地址

打开命令提示符，输入git clone，然后粘贴在步骤2复制的URL：

$ git clone https://github.com/YOUR-USERNAME/Spoon-Knife

按回车键，本地克隆就创建好了。

$ git clone https://github.com/YOUR-USERNAME/Spoon-Knife

Cloning into 'Spoon-Knife'...

remote: Counting objects: 10, done.

remote: Compressing objects: 100% (8/8), done.

remove: Total 10 (delta 1), reused 10 (delta 1)

Unpacking objects: 100% (10/10), done.

如果Fork一个项目是为了提出更改这个原始的仓库，可以配置Git将原始的或者upstream的变化更改到本地。

在GitHub上，定位到octocat/Spoon-Knife仓库。

在这个仓库页面的右侧边栏，单击复制图标复制这个仓库的URL，如图1-17所示。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

原始仓库的Clone链接

打开命令提示符，进入到步骤2本地创建的Fork的本地目录。

输入git remote -v，按回车键，将会看到Fork当前配置的远程仓库：

$ git remote -v

origin https://github.com/YOUR\_USERNAME/YOUR\_FORK.git (fetch)

origin https://github.com/YOUR\_USERNAME/YOUR\_FORK.git (push)

输入git remote add upstream，然后粘贴步骤6复制的URL并按下回车键。

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

验证Fork里新指明的这个Upstream仓库，再次输入git remote -v。将会看到Fork的URL作为原始的地址，而原始的仓库的URL作为upstream。

$ git remote -v

origin https://github.com/YOUR\_USERNAME/YOUR\_FORK.git (fetch)

origin https://github.com/YOUR\_USERNAME/YOUR\_FORK.git (push)

upstream https://github.com/ORIGINAL\_OWNER/ORIGINAL\_REPOSITORY.git (fetch)

upstream https://github.com/ORIGINAL\_OWNER/ORIGINAL\_REPOSITORY.git (push)

通过以上操作，可以利用几个Git命令保持Fork与upstream的仓库的同步。