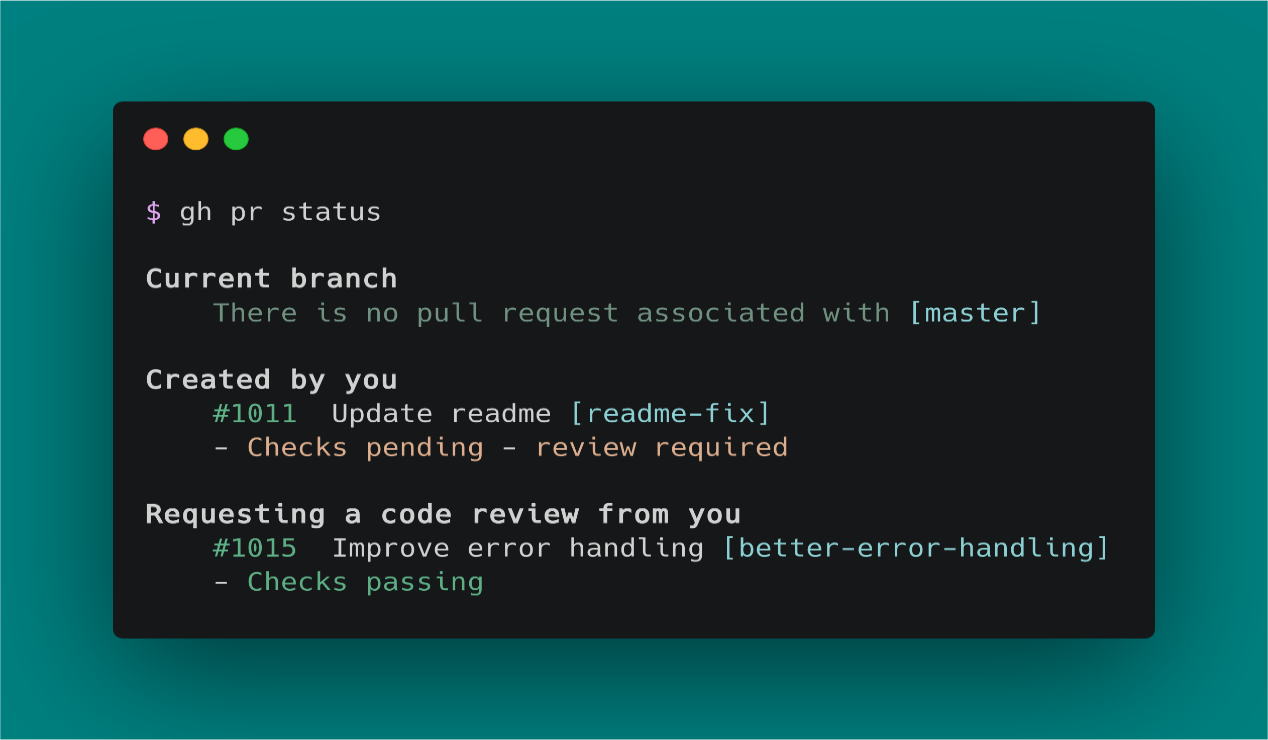
[**GitHub-Cli**](https://cli.github.com/)**的**[**安装**](https://github.com/cli/cli#installation/)**与**[**使用**](https://cli.github.com/manual)**(PR)**

初一（2）班 吴奕鸣 24号

综述

**** **Take GitHub to the command line!**

想必GitHub大家都不陌生。一开始我也只是偶尔上去安装个软件啥的，但是自从几个月前我接触到用GitHub作为服务器和二级域名来做网站后，我访问GitHub也就更加频繁了。每天都要重复登陆、收验证码实在是有些浪费时间了，再加上这么慢的网速，于是我开始使用Git+Hexo本地生成+上传的模式。可用了一段时间后，我却发现有比Git更快、更强、更高大上的终端命令行——**GitHub-Cli**（以下也简称gh）。

Github是什么？



GitHub是一个**面向开源及私有软件项目的托管平台**，因为只支持Git作为唯一的版本库格式进行托管，故名GitHub。

GitHub（Git）的一大亮点是**同一项目不同的分支与分支间Pull Request**，即合并不同分支，常用于不同开发者之间的小型商讨、改动与更新。

该项目在2018年6月4日正式被微软收购。

1. Gh简介

GitHub-Cli,也就是**GitHub最新推出的命令行终端**，具有快速、高效等特性，给万千码农带来了极大的福利——更快地访问、克隆仓库，更快的提交更新，更高效的处理issue和PR。曾经必须要到GitHub网页端才能修改的元信息，现在在CMD上就可以操作。

十几天前gh已经推出了最新的2.4.0版本。这款终端使用GoLang编写，配合一定的批处理文件就可以完成一切任务。最新数据显示该仓库已经有了2万6千个Star。

Git是什么？

Git是一个开源的分布式版本控制系统，可以有效、高速地处理从很小到非常大的项目版本管理。也是Linus Torvalds为了帮助管理Linux内核开发而开发的一个开放源码的版本控制软件。显著的优点有适合分布式开发，速度快、灵活，任意两个开发者之间可以很容易的解决冲突。还可以离线工作。

[项目官网](https://cli.github.com/) [项目源码地址](https://github.com/cli/cli/)

1. Gh的安装

安装其实很简单，毕竟它不是GUI，只是一个类似Bash的东东嘛。下面以*Windows*为例：

1. 安装包安装（通用）

访问[这个链接](https://github.com/cli/cli/releases/tag/v2.4.0)，找到Assets。然后你会找到很多很多的安装包，在比较下面的地方有一个**gh\_2.4.0\_windows\_amd64.msi**的安装文件（貌似是64位的）。尝试下载它……很慢的话可以访问下面这个加速网址：[下载地址](https://github.com.cnpmjs.org/cli/cli/releases/tag/v2.4.0)*（实在不行的话那我也没辙了）*

下载下来以后，直接运行安装即可。安装完成后，打开CMD运行：

$ gh –-version

如果CMD返回给你有用的信息（而不是“gh“不是内部或外部命令，也不是可运行的程序或批处理文件。），那么说明你已经安装好了！

1. Chocolatey安装（没装过的同志请往下看

使用过Chocolatey的同志直接运行下面的命令！

**$ choco find gh**

**$ choco install gh –y**

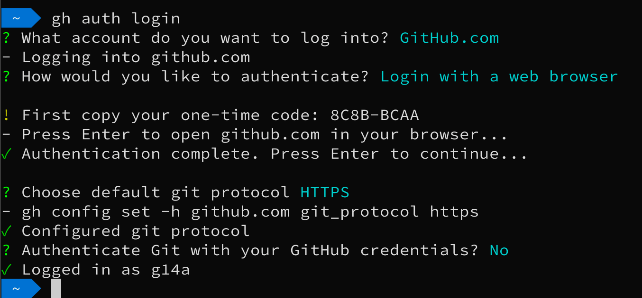
嗯……搞定了！

1. Gh登录时刻！

准备好你的GitHub账号，我们终于要登录了！登录也很简单，只要运行：

**$ gh auth login //这很简单吧？**

然后就会有下面的提示语：



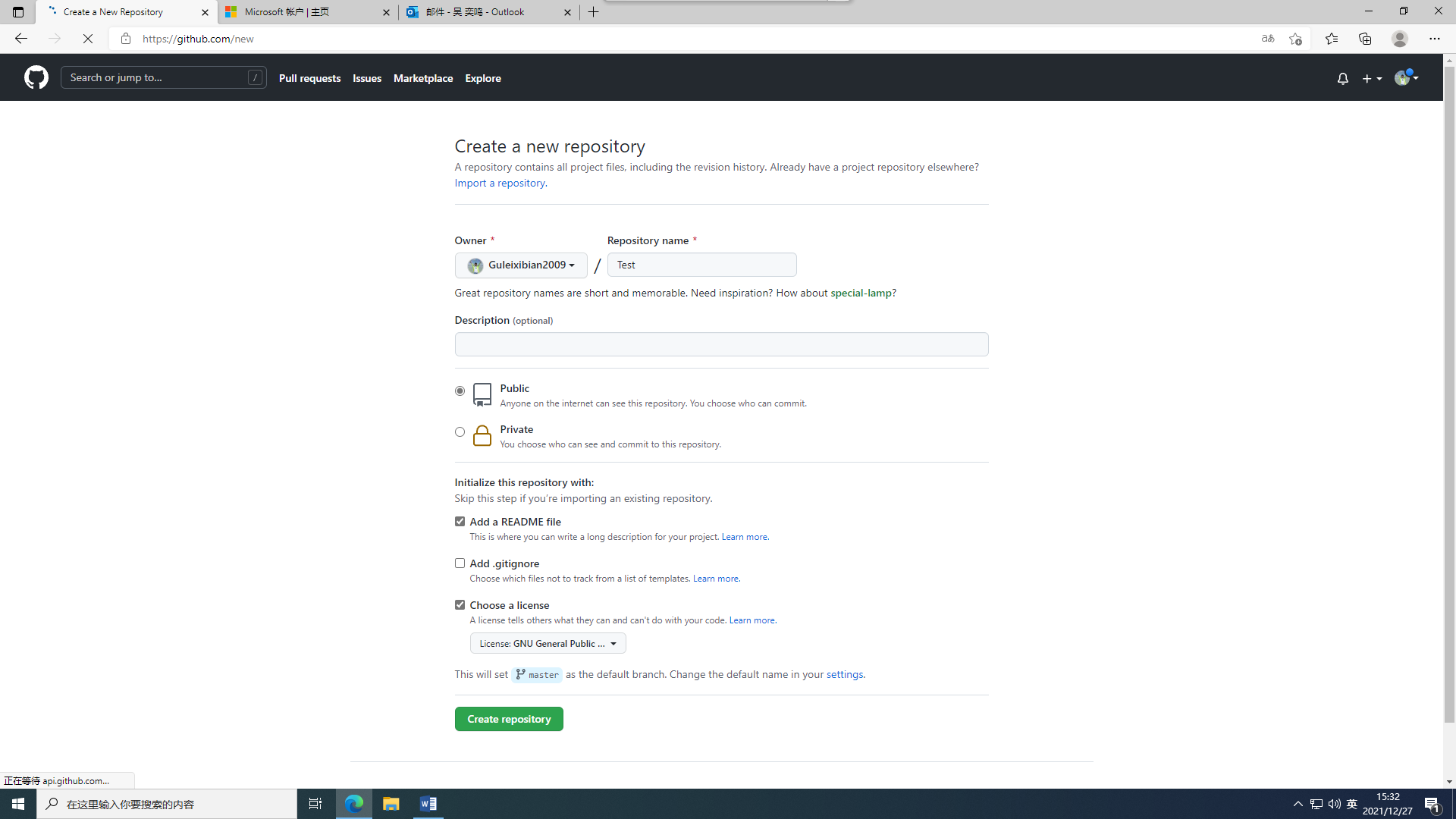
*（图片来源于网络……所以不是我的用户名*

两种登陆方法：你可以使用**浏览器登陆账号**后登录GitHub-Cli，也可以使用**粘贴个人验证码登录**。由于GitHub-Cli的内核也是用到Git的，所以它也会要求你配置Git，选择SSH或者HTTPS貌似都差不多。

如果最后它提示你类似**Logged in as …**说明你成功了！！

1. 关于你的仓库

使用“项目托管平台”你没有仓库怎么行？用你本地的Git或者在线创建一个仓库，然后你就可以操作了。



创建一个仓库后，把它拉取到本地。这个时候GitHub-Cli就能派上用场了：

**$ gh repo clone {username}/{repo\_name}**

如果是我的话，我可能会用：

**$ gh repo clone Guleixibian2009/Test**

现在你的电脑上应该就有了一个本地仓库了。现在让我们来更新它！

首先我们来**切换到一个新的分支**（*Git版本库中存放项目不同版本的一种机制*）。然后，做一点点修改，然后再提交回远程端（在这里就是GitHub！）：

**$ git checkout –b test //checkout -b 会创建并进入一个新的分支**

**$ mkdir Test12.17 //创建一个文件夹**

**$ cd Test12.17**

**$ echo “What makes unicorns cry?”>>test.txt //做一点点修改**

现在我们已经生成了一个**新**文件夹，一个**新**文件并进入了一个**新的**分支。然后，用大家都熟知的Git来进行以下操作：

**$ git add . //缓存所有修改**

**$ git commit –m “Update 12.17” //创建一个更新记录**

**$ git push –-set-upstream origin test //上传 *注意这里需要配置远程***

现在**貌似我们的更改已经被上传到GitHub上面去了**！不过很明显，我们提交到的是一个叫做test的分支，我们还需要把他提交到master，即主分支上去。

1. PRs

其实我们的确可以将更新直接提交到master上，但这不是重点——在GitHub上更多的是多人合作项目，这时将更新代码直接提交到主分支上貌似就有点风险了（万一有Bug呢）。然后，我们就可以尝试一种新的方式——**Pull Request**。（感觉前面铺垫好多）

Pull Request 是一种**通知机制**。你修改了他人的代码，将你的修改通知原来的作者，希望他合并你的修改，这就是 Pull Request。Pull Request 本质上是一种**软件的合作方式**，是将涉及不同功能的代码，纳入主干的一种流程。这个过程中，还可以进行讨论、审核和修改代码。

所以，你学废了吗……

1. 创建PR

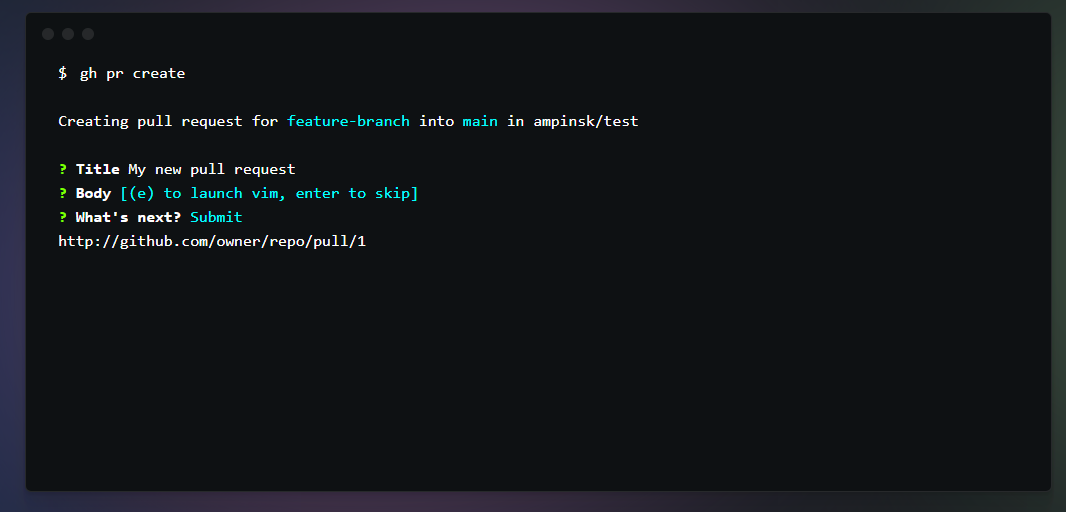
现在我们来尝试创建一个新PR。当然，我们是个人项目，所以PR仅起到演示的效果。

如图所示，创建一个PR我们需要运行：

**$ gh pr create //等同于 $ gh pr cr**

同时我们会需要提供例如标题、基本信息等等等等。当输出类似以下的提示时，你就成功了。

**https://github.com/Guleixibian2009/test/pull/1**

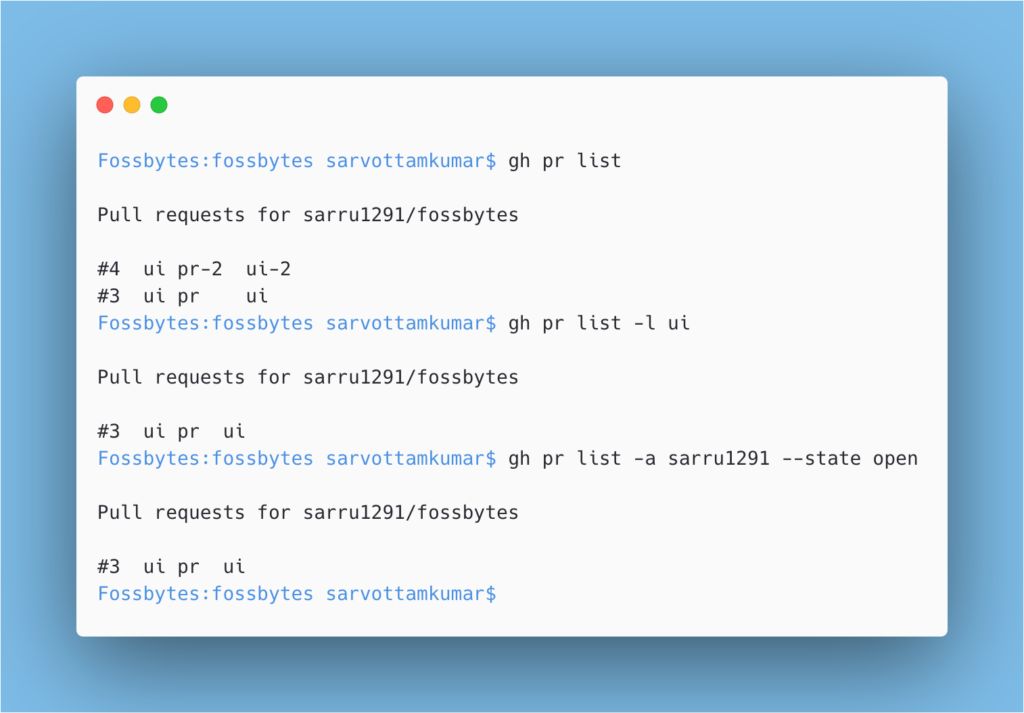


1. PR的合并

很明显，单独创建一个PR是没有用的。我们需要把他**合并到主分支里**！首先我们看一下到底有没有这个PR：

**$ gh pr list**

这个命令可以帮助我们查看所有的PR。比如下面的仓库就有2个进行中的PR

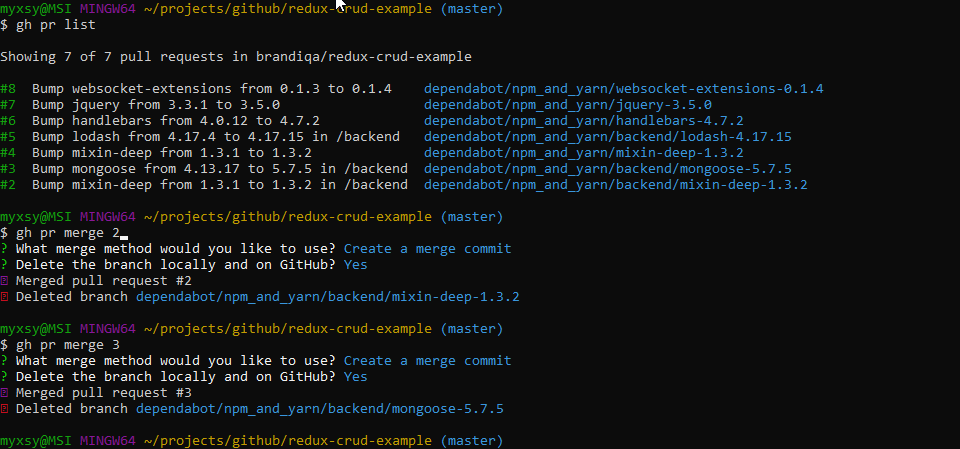


Em……只需要注意最上面那一行命令就可以了，后面两条是添加了几个限定条件。（看起来好复杂）

很明显我们的确有进行中的PR。为了将其合并入主分支，我们来Merge刚刚那个PR。

**$ gh pr merge 1 //1为PR编号**

示意图如下：



当然在合并PR时我们会需要选择一些基本信息，一般情况下选择Create a merge commit。如果你不想删除这个test分支，你可以在第二个选项中选择No。最后它会有提示语：

Merged pull request #1

然后，我们怎么知道自己的PR真的真的已经被合并了呢？我们可以*切换回原分支*master。

$ git checkout master //不需要 -b ，因为不是新分支

$ gh repo sync //这个指令可以同步当前分支

如果你的master分支中多了一开始创建的文件，那么，这个PR就是真的被合并了。

最后，如果你还想了解GitHub-Cli更多有关命令，你可以运行：

**$ gh <command> <subcommand> --help**

**//此处 <command>/<subcommand> 要用具体的命令代替**

**//如： gh repo clone --help**

就可以获得有关命令的更多信息了！

**写在最后**

经过