**git remote add origin的使用**

code\_lee7个月前 (03-28)[git](https://www.54yk.com/?cate=17)1420

新建一个仓库以后，提示命令

Bash

echo "# Linux\_base" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin git@github.com:bykevin/Linux\_base.git

git push -u origin master

如果已经 commit ，直接执行后面的命令

Bash

git remote add origin git@github.com:bykevin/Linux\_base.git

git push -u origin master

[](https://www.zhihu.com/people/cxyxiaowu)

[**程序员吴师兄**](https://www.zhihu.com/people/cxyxiaowu)

**​**

3,463 人赞同了该回答

**作为一名在 GitHub 上收获 70k star 的程序员**，我想用两万字加一百张图片从头到尾的给你介绍 GitHub 的具体使用，通过这个 GitHub 教程，让你掌握 GitHub 的使用方法。



我的 GitHub 地址：[MisterBooo/LeetCodeAnimation](https://link.zhihu.com/?target=https%3A//github.com/MisterBooo/LeetCodeAnimation)

**篇幅较长，建议先收藏**，阅读时可以利用搜索功能快速定位到感兴趣的章节。

如果无法正常访问 GitHub，可以阅读我之前分享的一篇文章：

[GitHubPorn：提高国内访问 Github 速度的 9 种方法1708 赞同 · 45 评论文章](https://zhuanlan.zhihu.com/p/365490253)

一个对你很有帮助的 GitHub 项目：

[iCSToCS/CSBook​github.com/iCSToCS/CSBook](https://link.zhihu.com/?target=https%3A//github.com/iCSToCS/CSBook)

**一、初识 GitHub · 简介篇**

**1 前言**

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 于 2008 年 4 月 10 日正式上线，除了 Git 代码仓库托管及基本的 Web 管理界面以外，还提供了订阅、讨论组、文本渲染、在线文件编辑器、协作图谱（报表）、代码片段分享（Gist）等功能。目前，其托管版本数量非常之多，而且其中不乏知名开源项目，例如 Ruby on Rails、jQuery、python 等。

作为开源代码库以及版本控制系统，Github 拥有超过千万的开发者用户。随着越来越多的应用程序转移到了云上，Github 已经成为了管理软件开发以及发现已有代码的首选方法。

如前所述，作为一个分布式的版本控制系统，在 Git 中并不存在主库这样的概念，每一份复制出的库都可以独立使用，任何两个库之间的不一致之处都可以进行合并。

GitHub 可以托管各种 Git 库，并提供一个 web 界面，但与其它像 SourceForge 或 Google Code 这样的服务不同，GitHub 的独特卖点在于从另外一个项目进行分支的简易性。为一个项目贡献代码非常简单：首先点击项目站点的Fork的按钮，然后将代码检出并将修改加入到刚才分出的代码库中，最后通过内建的pull request机制向项目负责人申请代码合并。

GitHub 项目本身自然而然的也在 GitHub 上进行托管，只不过在一个私有的，公共视图不可见的库中。开源项目可以免费托管，但私有库则并非如此。在 GitHub，用户可以通过Explore轻而易举地找到海量的开源代码。因此，称之为程序员的 圣地 也不过吧？

**2 业界大神**

在 GitHub 上建立个人主页的业界大神多如牛毛，在此仅介绍 3 位，以供大家膜拜！

2.1 Linus Torvalds



GitHub 主页：[torvalds - Overview](https://link.zhihu.com/?target=https%3A//github.com/torvalds)

Linus Torvalds，全球著名的电脑程序员、黑客，Linux 之父。

2.2 John Resig



Github主页： [jeresig - Overview](https://link.zhihu.com/?target=https%3A//github.com/jeresig)

John Resig，jQuery 的创始人和技术领袖，著有《Pro JavaScript Techniques》（即《精通JavaScript》）等经典 JavaScript 书籍。

2.3 Jake Wharton



Github主页： [JakeWharton - Overview](https://link.zhihu.com/?target=https%3A//github.com/JakeWharton)

Jake Wharton， Android 之神，GitHub 全球排名第二位。

**3 开源项目及公司**

在 GitHub 上开源的著名项目非常之多，选择在 GitHub 上开源的世界顶级公司也很多，在此也各罗列 3 个，以供大家膜拜！

3.1 项目

Linux：[torvalds/linux](https://link.zhihu.com/?target=https%3A//github.com/torvalds/linux)

Swift：[apple/swift](https://link.zhihu.com/?target=https%3A//github.com/apple/swift)

Ruby：[ruby/ruby](https://link.zhihu.com/?target=https%3A//github.com/ruby/ruby)

3.2 公司

Google: [Google](https://link.zhihu.com/?target=https%3A//github.com/google)

Microsoft：[Microsoft](https://link.zhihu.com/?target=https%3A//github.com/microsoft)

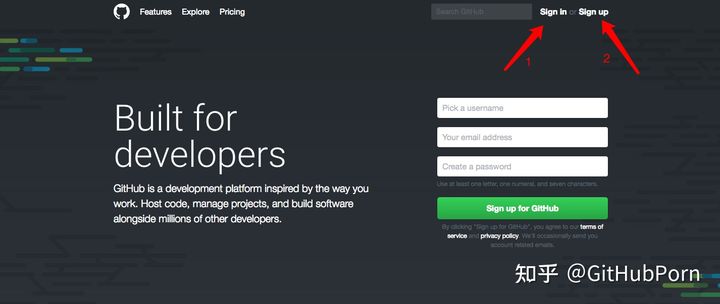
Alibaba：[Alibaba](https://link.zhihu.com/?target=https%3A//github.com/alibaba)

**二、敲开 GitHub 的大门 · 注册账号**

在 GitHub 里面有众多的业界大神、有丰富的学习资料、有著名的开源项目代码，我们也可以在 GitHub 中增长自己的技术能力、渲染自己的简历，甚至搭建自己的个人博客或者网站。

此外，最重要的，也是 GitHub 的核心亮点，那就是：我们可以在 GitHub 上同世界各地的伙伴协同开发项目，而且简单、容易的让人难以置信。那么，你准备好了吗？现在，就让我们一起敲开 GitHub 的大门，进入 GitHub 的世界吧！

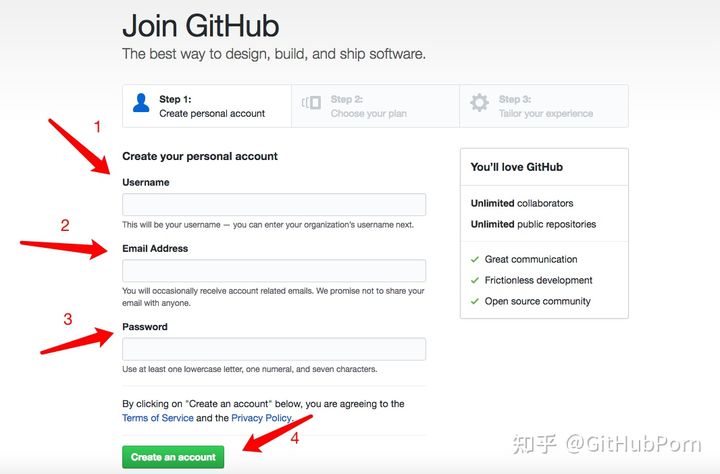
首先，进入 GitHub 的官网：[https://www.github.com](https://link.zhihu.com/?target=https%3A//www.github.com/)



标注 1：**Sign in**，登录；

标注 2：**Sign up**，注册。

如上图所示，这是 GitHub 的官网首页，点击 **Sign up**进行注册：



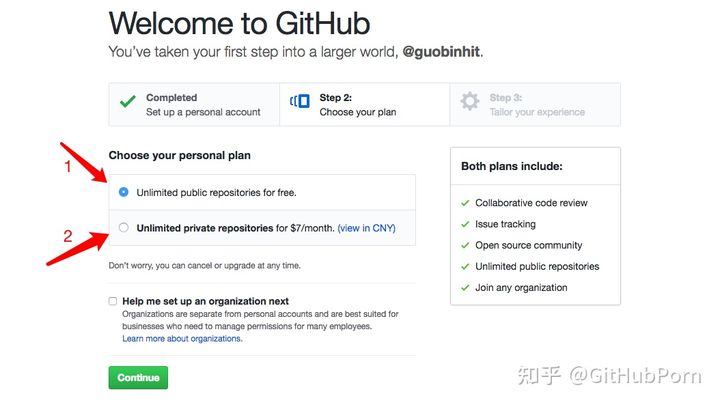
标注 1：Username，用户名；

标注 2：Email Address，邮箱地址；

标注 3：Password，密码；

标注 4：Create an account，创建账号按钮。

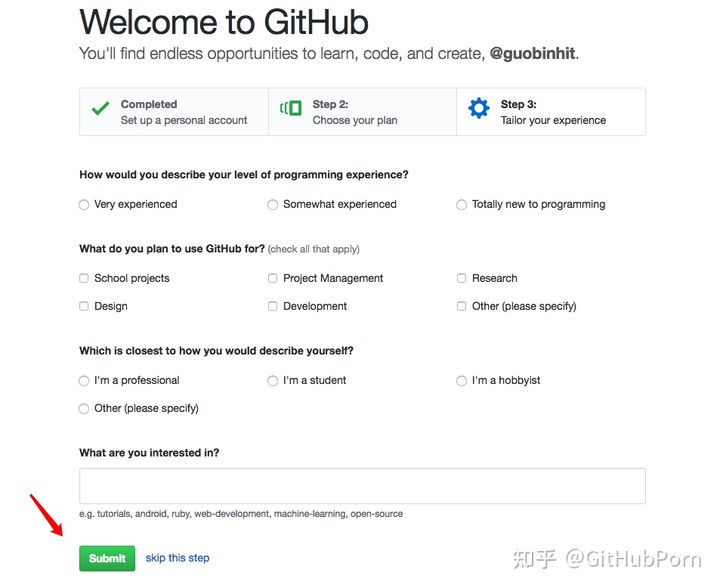
如上图所示，其中 **标注 1**比较重要，最好起对我们具有标识性的用户名，而且如果以后我们要在 GitHub 上搭建自己的个人博客，[其默认地址就是username.github.io](https://link.zhihu.com/?target=https%3A//www.cxyxiaowu.com/15727.html)；标注 2 是需要我们验证 GitHub 账号的邮箱，注册时只能填写一个，之后可以添加多个邮箱。上面的信息都填写完成后，点击 **Create an account**，进入如下界面：



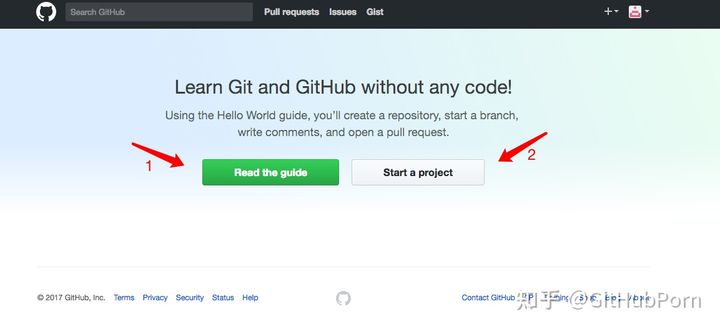
标注 1：公开的，免费仓库；

标注 2：私有的，付费仓库。

如上图所示，我们进入了注册 GitHub 账号流程的第 2 步，在这里有一点需要我们注意，那就是：GitHub 的仓库分为两种，一种是public repositories公开免费版，一种是private repositories私有付费版。其中，私有仓库一般是由企业或者不希望自己的仓库公开的个人用户购买，这也是 GitHub 的主要收入来源。在这里，我们选择免费版就可以，然后点击Continue，进入如下界面：



如上图所在，这是注册 GitHub 账号流程的第 3 步，也是最后一步。这个步骤的主要作用就是收集用户的个人信息，如果感兴趣的话，可以对上面的“ 问答表 ”进行勾选，然后点击Submit提交；如果不感兴趣的话，则可以直接点击skip this step跳过这一步。无论点击两者之中的那个按钮，都将进入如下界面：

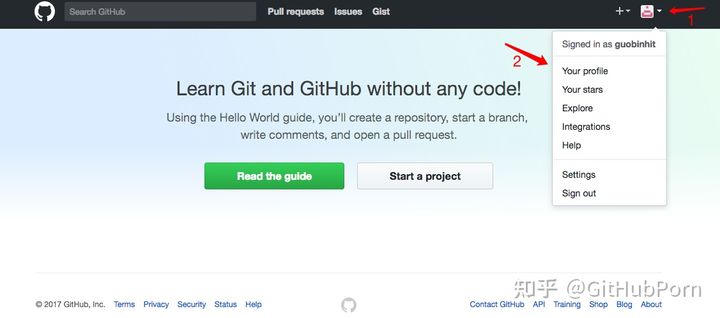


标注1：Read the guide，阅读指南；

标注2：Start a project，建立项目。

如上图所示，到这里，说明我们已经成功注册了 GitHub 的账号。在这个界面，GitHub 为我们提供了两个建议性的选择，一是Read the guide阅读 GitHub 指南，二是Start a project直接建立项目。

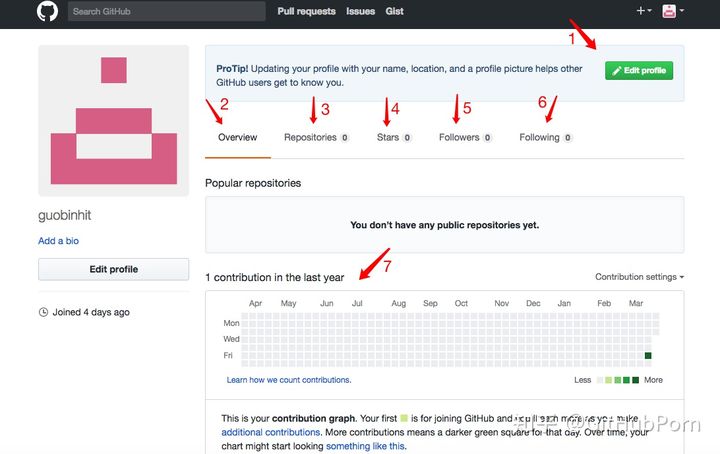
**三、GitHub 主页介绍及修改个人信息**



标注 1：View profile and more，更多选项视图；

标注 2：Your profile，个人简介。

如上图所在，我们依次点击 标注 1 所示的View profile and more和 标注 2 所示的Your profile，进入「个人简介」界面：



标注 1：Edit profile，修改个人简介；

标注 2：Overview，个人主页概览；

标注 3：Repositories，仓库；

标注 4：Star，点星记录；

标注 5：Followers，粉丝；

标注 6：Following，关注的 GitHub 账号；

标注 7：个人贡献历史记录。

如上图所示：

标注 1 表示的为Edit profile，这个选项当我们修改完个人信息之后，就会自动消失；

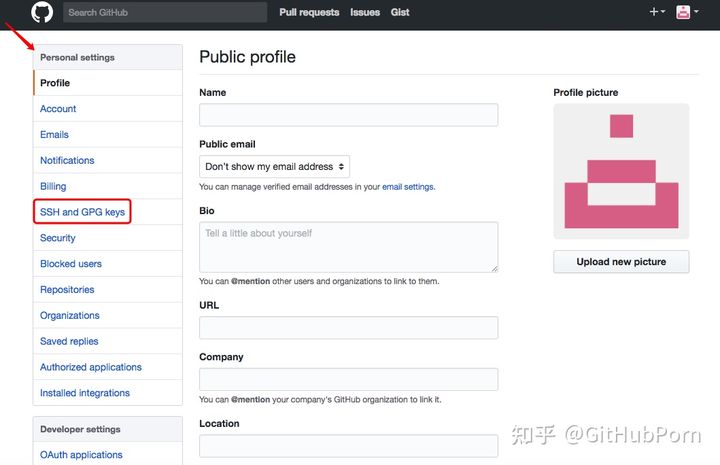
标注 2 表示的为Overview，展示了我们账号的主要内容，包括仓库和贡献等；

标记 3 表示的为Repositories，是我们建立的仓库，包括Fork来的项目，GitHub 也会自动为我们创建一个仓库；

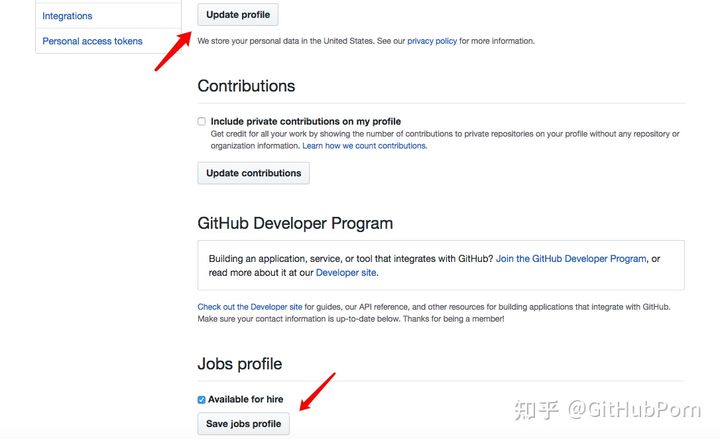
标注 4 表示为Star，收藏了我们的“点星”，或者说是“点赞”过的项目；

标注 7 表示的为我们最近一年来的contribution，用实心的小方格标记，小方格的颜色越深，表示我们的contribution越多。

在这里，我们点击Edit profile，编辑个人简历：

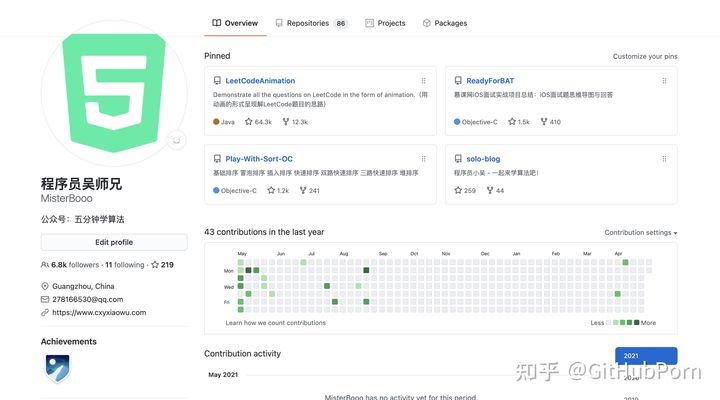


如上图所示，我们可以通过这个界面填写个人信息，包括 Name（昵称）、Bio（自我介绍）、URL（链接）、Company（公司）、Location（位置）以及 Upload new picture（上传头像）等等。在我们填写完个人信息之后，可以点击Update profile更新个人简介，同时可以勾选Available for hire，选择“雇主可见”，然后点击Save jobs profile，保存我们的求职简历。

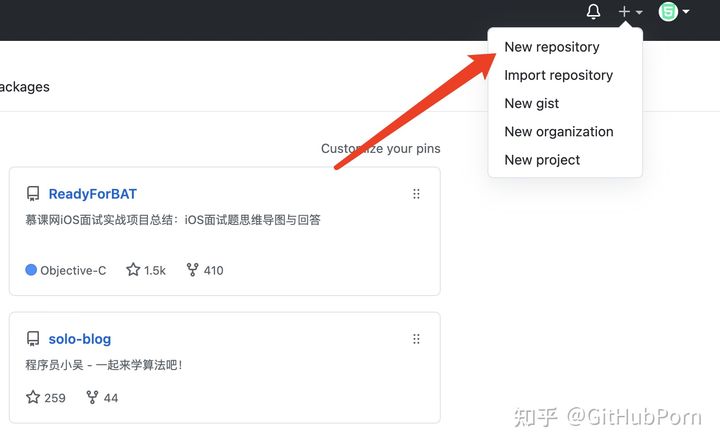


此外，在Personal settings中，还包含很多其他的选择，如在Account中，可以修改账号密码；在Emails中，可以修改绑定的邮箱等等。在这里，用红色框圈起的SSH and GPG keys非常重要，我们可以通过它连接到本地的 IDE，从而保证项目提交与检出的安全性。

接下来，展示一下我们刚刚修改完的 GitHub 账号主页：

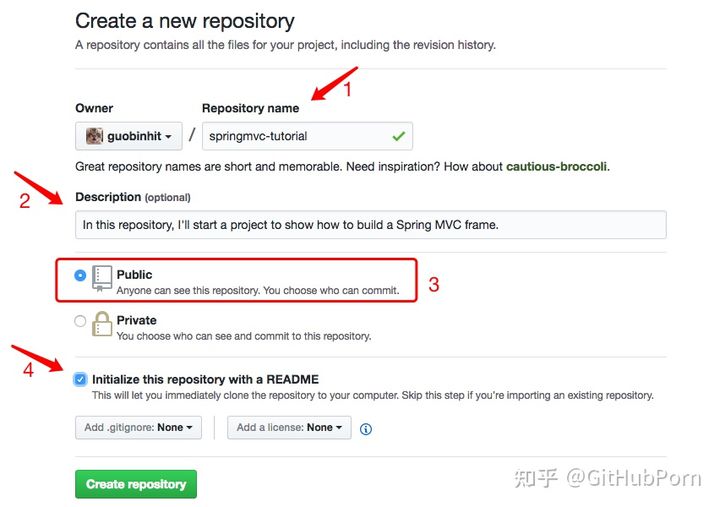


**四、创建 GitHub 仓库的步骤及方法**



如上图所示，此为博主的 GitHub 个人主页，点击Repositories，进入如下界面：

点击上图标注所示的绿色New按钮，进入下一步：



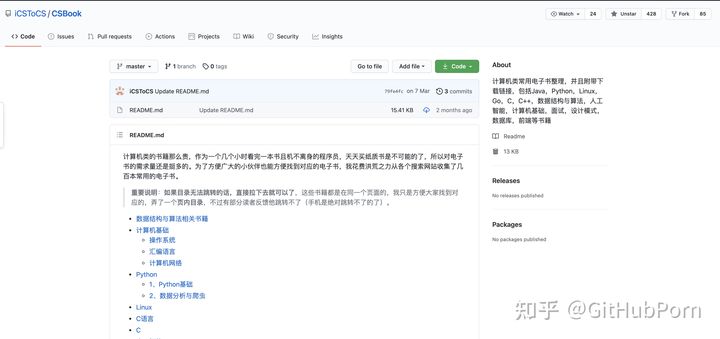
标注 1：Repository name，仓库名称；

标注 2：Description，可选描述，也就是写不写都可以；

标注 3：Public，默认的仓库类型；

标注 4：Initialize this repository with a README，初始化仓库的信息文件，建议勾选。

如上图所示，这是创建 GitHub 仓库的核心页面，里面包含了众多信息。为了方便演示，博主已经把各种所需的信息都填写完啦！接下来，点击绿色Create repository按钮即可：



如上图所示，我们已经把仓库创建成功啦！

仓库名为 CSBook ，包含 3 个commit，第一个 commit 是我们通过勾选Initialize this repository with a README，[创建了一个初始化提交文件README.md](https://link.zhihu.com/?target=https%3A//github.com/iCSToCS/CSBook)，[其中文件后缀为.md](https://link.zhihu.com/?target=https%3A//github.com/iCSToCS/CSBook)，表示文件为 Markdown 格式；包含 1 个branch，为master分支，即主分支；包含 1 个contributor，为贡献者，也就是我们自己。

**五、GitHub 术语解释**

为了大家进一步了解和使用 GitHub，我们一起来看看 GitHub 的常用术语，也可以说是基本概念：

**Repository**：简称Repo，可以理解为“仓库”，我们的项目就存放在仓库之中。也就是说，如果我们想要建立项目，就得先建立仓库；有多个项目，就建立多个仓库。

**Issues**：可以理解为“问题”，举一个简单的例子，如果我们开源一个项目，如果别人看了我们的项目，并且发现了bug，或者感觉那个地方有待改进，他就可以给我们提出Issue，等我们把Issues解决之后，就可以把这些Issues关闭；反之，我们也可以给他人提出Issue。

**Star**：可以理解为“点赞”，当我们感觉某一个项目做的比较好之后，就可以为这个项目点赞，而且我们点赞过的项目，都会保存到我们的Star之中，方便我们随时查看。在 GitHub 之中，如果一个项目的点星数能够超百，那么说明这个项目已经很不错了。

我整理了一些计算机的电子书，来点个赞吧：

[iCSToCS/CSBook​github.com/iCSToCS/CSBook](https://link.zhihu.com/?target=https%3A//github.com/iCSToCS/CSBook)

**Fork**：可以理解为“拉分支”，如果我们对某一个项目比较感兴趣，并且想在此基础之上开发新的功能，这时我们就可以Fork这个项目，这表示复制一个完成相同的项目到我们的 GitHub 账号之中，而且独立于原项目。之后，我们就可以在自己复制的项目中进行开发了。

**Pull Request**：可以理解为“提交请求”，此功能是建立在Fork之上的，如果我们Fork了一个项目，对其进行了修改，而且感觉修改的还不错，我们就可以对原项目的拥有者提出一个Pull请求，等其对我们的请求审核，并且通过审核之后，就可以把我们修改过的内容合并到原项目之中，这时我们就成了该项目的贡献者。

**Merge：**可以理解为“合并”，如果别人Fork了我们的项目，对其进行了修改，并且提出了Pull请求，这时我们就可以对这个Pull请求进行审核。如果这个Pull请求的内容满足我们的要求，并且跟我们原有的项目没有冲突的话，就可以将其合并到我们的项目之中。当然，是否进行合并，由我们决定。

**Watch：**可以理解为“观察”，如果我们Watch了一个项目，之后，如果这个项目有了任何更新，我们都会在第一时候收到该项目的更新通知。

**Gist：**如果我们没有项目可以开源或者只是单纯的想分享一些代码片段的话，我们就可以选择Gist。不过说心里话，如果不翻墙的话，Gist并不好用。

**六、Git 的安装流程及步骤**

现在，我们已经知道了如何创建 GitHub 账号、创建仓库、进行个性化设置等等，但是我们还要知道：GitHub 是基于版本控制系统 Git 之上的啊！

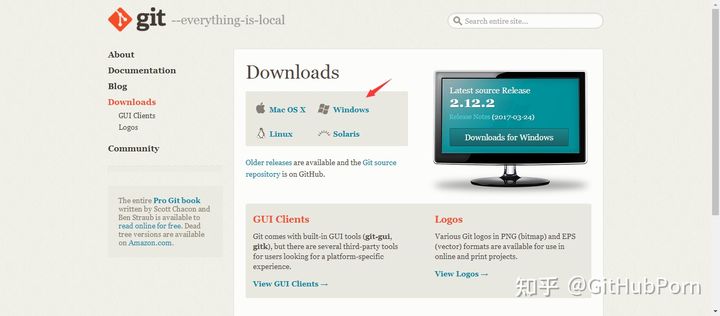
如果我们想要进行代码托管，想要进行团队协作，这都少不了一个工具，那就是：Git。

因此，在本篇博文中，我们就一起来了解一下 Git 的安装流程及步骤。

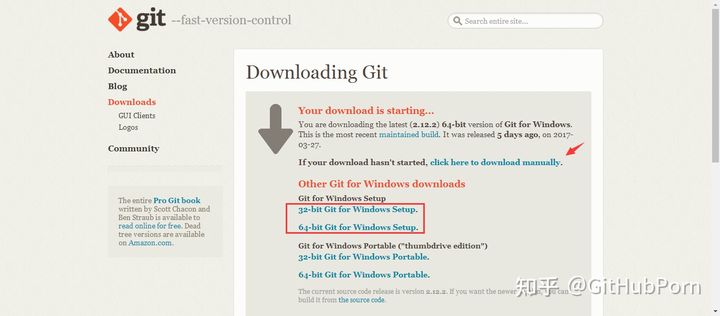
首先，进入 Git 的官网：git - -fast-version-control



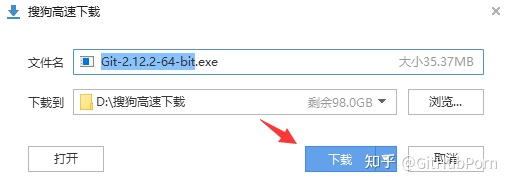
如上图所示，在 Git 的官网中点击Downloads，进入如下页面：



如上图所示，选择对应的操作系统，以博主为例，点击Windows，进入如下页面：



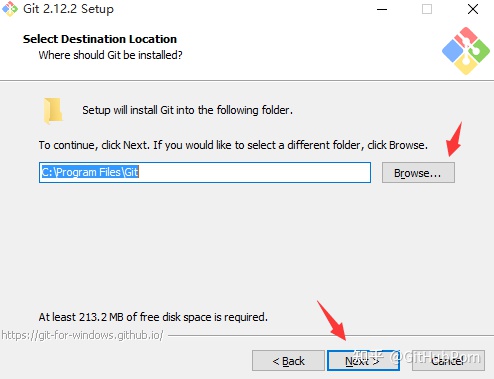
如上图所示，正常情况下，会自动弹出下载框，否则的话，手动点击红色箭头所示的**click here to download manually**亦可进入如下界面：



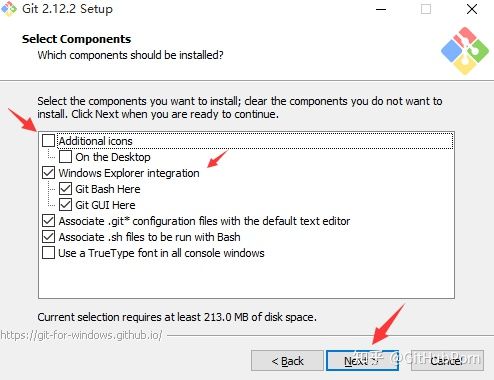
如上图所示，直接点击 下载 即可，下载完成后，双击打开，进入如下界面：



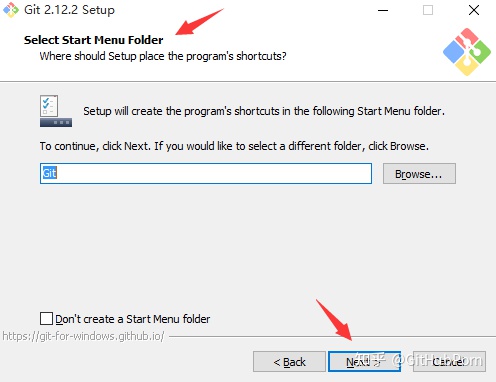
如上图所示，这是 Git 的安装界面，点击Next，进入如下界面：



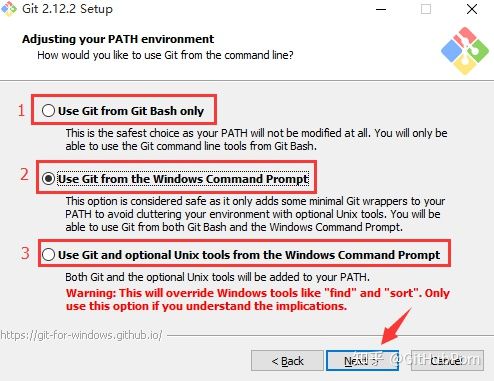
如上图所示，选择 Git 的安装目录，默认安装到C盘的Program Files目录下，想换的话，点击Browse进入更换。在这里，我们选择将其安装到D盘的Program Files目录下，选择完成后，点击Next，进入如下界面：



如上图所示，这里有一些可勾选的项，我们可以按自己的实际需求进行选择（后面同样如此），例如勾选Additional icons，将在 Git 安装完成后，在桌面创建一个图标，也就是打开 Git 的快捷方式。在这一步，建议大家选择默认即可，例如默认勾选的Windows Explorer integration，就可以让我们在点击鼠标右键的时候，快速选择打开Git GUI或者 Git Bash。选择完成后，点击Next，进入如下界面：



如上图所示，选择 开始菜单文件夹，默认即可，点击Next，进入如下界面：

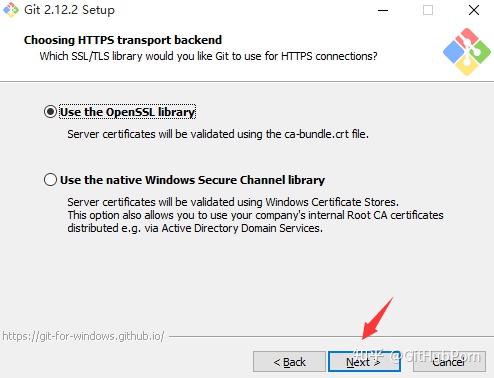


标注 1：仅使用 Git Bash 进行操作；

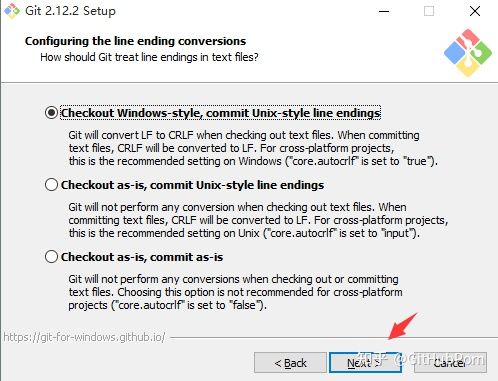
标注 2：在选择使用 Git Bash 进行操作的同时，也可以使用 Windows 命令行操作，建议选择此项；

标注 3：在选择使用 Git 的同时，也把 Unix 工具加入到了我们的配置之中，而且此操作会覆盖 Windows 的一些工具，强烈不建议选择此项。

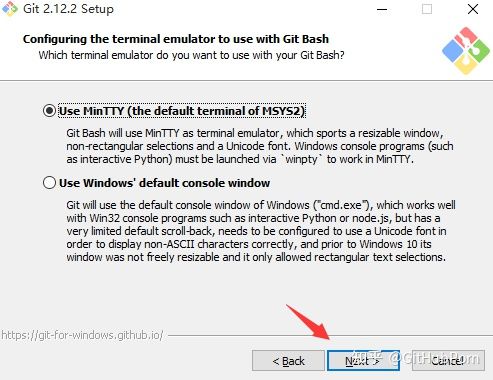
如上图所示，我们选择 标注2 所示的Use Git from the Windows Command Prompt，点击Next，进入如下界面：



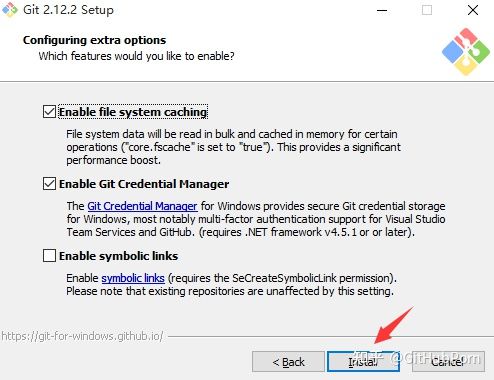
如上图所示，选择 HTTPS 传输后台，默认即可，点击Next，进入如下界面：



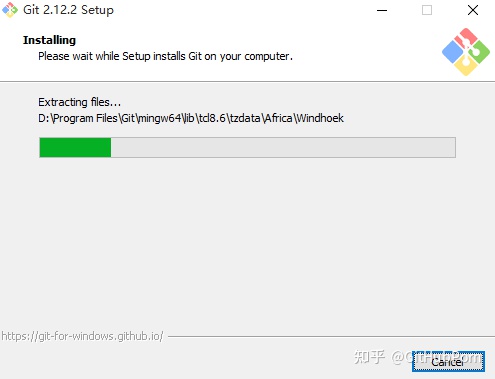
如上图所示，配置行结束标记，默认即可，点击Next，进入如下界面：



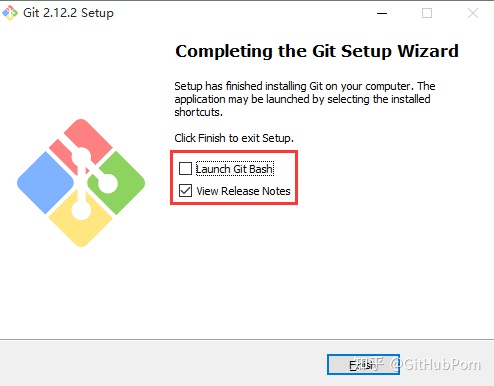
如上图所示，配置 Git Bash 的终端模拟器，默认即可，点击Next，进入如下界面：



如上图所示，配置补充功能，默认即可，点击Next，进入如下界面：



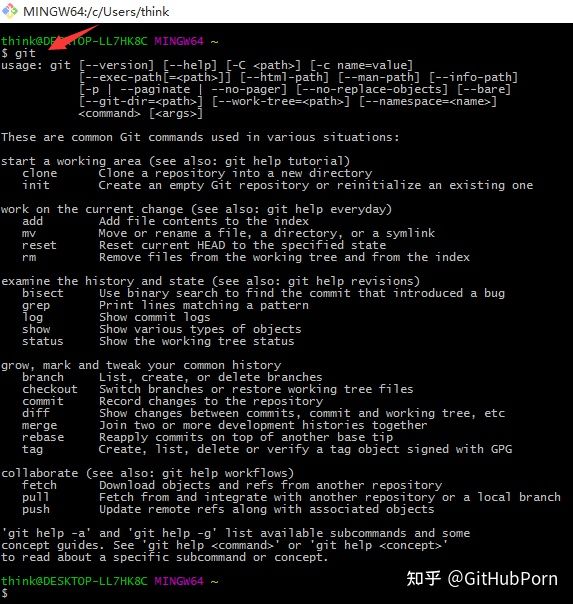
如上图所示，展示了 Git 安装中的界面，安装完成后，弹出如下窗口：



如上图所示，这表示 Git 已经安装完成了，至于图中的两个选择，则分别表示 打开 Git Bash 和 浏览 Git 版本信息，可以都选，也可以都不选，在这里，我们选择Launch Git Bash，进入如下界面：



如上图所示，我们打开了 Git Bash，输入git命令，将显示如下结果：

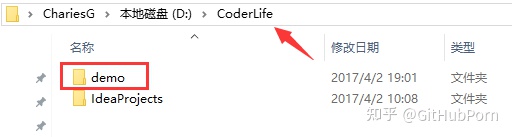


如上图所示，Git 已经准备就绪啦，接下来就是你的 show time 啦！

**七、Git 初体验及其常用命令介绍**

接下来介绍 Git 的命令操作，包含 init、add 等，在 Git 中，所有的命令都是以git开头，例如，git init其作用就是初始一个 Git 仓库。

为了方便演示，我们先在D盘的CoderLife目录下创建一个名为demo的子目录，并在其中新建一个名为hit.txt的文件，接下来我们的 Git 操作都是基于此目录和文件的。



此外，在这里还要强调一点，那就是：在我们进行任何的git操作之前，我们都得先切换到 Git 的仓库目录。

换言之，我们得到先进入到（我们定义的）Git 仓库的最顶层文件目录下，然后从此目录中进入 Git Bash，这样之后的操作才能顺利进行。

如果是 Linux 操作系统，则可以直接cd到仓库目录。

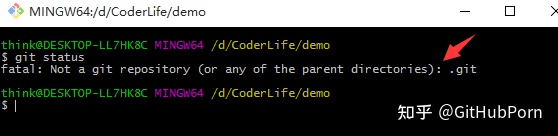
以博主为例，选择demo目录作为 Git 仓库，然后进入demo目录之中，点击鼠标右键，再选择Git Bash Here，即可打开 Git Bash 的命令行窗口。



如上图所示，Git 会自动定位到进入的位置，如我们选择的demo目录，这也是为什么我们需要先进入到 Git 仓库的最顶层目录下，然后再打开 Git Bash 的原因。下面，我们结合 Git 的常用命令演示一下 Git 的相关操作。

**第 1 个命令：git status**

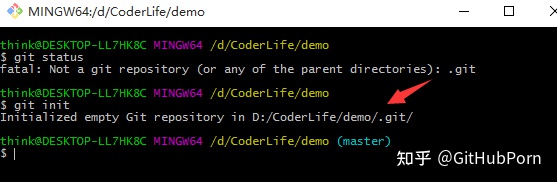
在命令行窗口的光标处，输入git status命令，查看仓库的状态：



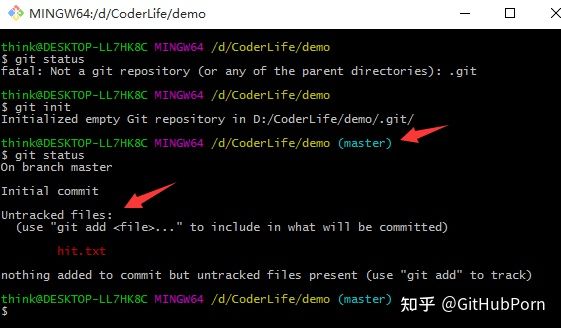
如上图所示，结果显示demo不是一个 Git 仓库，这是很正常的反应，因为我们还没有在计算机中声明demo为 Git 仓库，之前说demo是 Git 仓库只是我们口头上的说的，计算机当然不会认可。

**第 2 个命令：git init**

在命令行窗口的光标处，输入git init命令，初始化 Git 仓库：



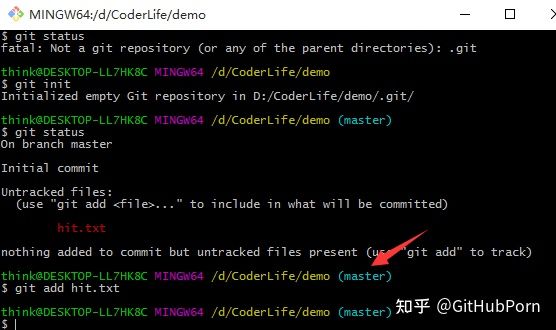
如上图所示，结果显示已经初始化demo为一个空的 Git 仓库啦！在这里大家可以会有些疑问，因为我们在建立demo目录的同时也在里面新建了一个名为hit.txt的文件，怎么初始化仓库之后，demo目录就变成空的了呢？这个问题稍后解惑，我们重新输入git status命令检查一下仓库的状态：



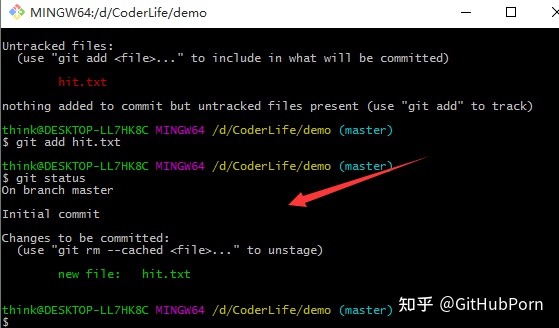
如上图所示，在我们初始化仓库之后，demo目录已经成为一个 Git 仓库了，并且默认进入 Git 仓库的master分支，即主分支。在这里，我们需要注意的是Untracked fies提示，它表示demo仓库中有文件没有被追踪，并提示了具体没有被追踪的文件为hit.txt，还提示了我们可以使用git add命令操作这个文件，简直不要太好。

**第 3 个命令：git add**

在命令行窗口的光标处，输入git add hit.txt命令，将hit.txt文件添加到 Git 仓库：



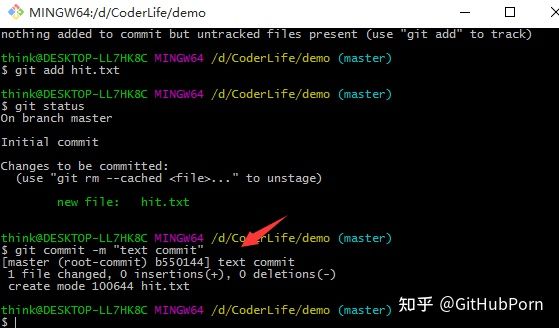
如上图所示，如果没有报错，就说明命令已经执行啦！接下来，输入git status命令查看仓库状态：



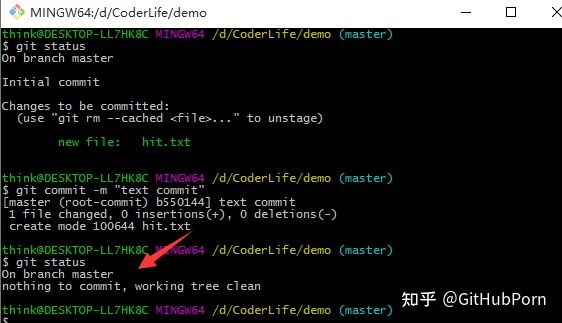
如上图所示，已经显示Initial commit初始化提交了，同时已经没有Untracked files提示了，这说明文件hit.txt已经被添加到 Git 仓库了，而在我们没有进行git add操作之前，文件hit.txt并不被 Git 仓库认可，因此才会出现提示初始化仓库为空的现象。在这里，需要声明一点，那就是：git add命令并没有把文件提交到 Git 仓库，而是把文件添加到了「临时缓冲区」，这个命令有效防止了我们错误提交的可能性。

**第 4 个命令：git commit**

在命令行窗口的光标处，输入git commit -m "text commit"命令，将hit.txt文件提交到 Git 仓库：



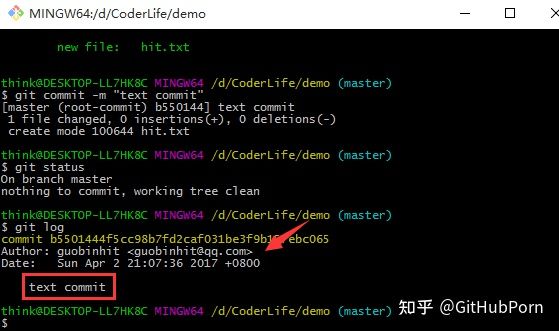
如上图所示，我们成功将文件hit.txt提交到了 Git 仓库，其中commit表示提交，-m表示提交信息，提交信息写在双引号""内。接下来，再输入git status命令查看仓库状态：



如上图所示，结果显示nothing to commit, working tree clean，这表示已经没有内容可以提交了，即全部内容已经提交完毕。

**第 5 个命令：git log**

在命令行窗口的光标处，输入git log"命令，打印 Git 仓库提交日志：



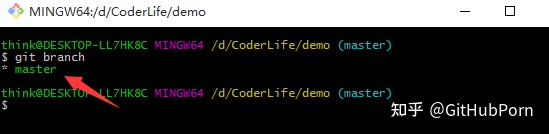
如上图所示，显示了我们的提交记录，提交记录的内容包括Author提交作者、Date提交日期和提交信息。

通过以上的操作，我们会发现一个现象，那就是：在每个git操作之后，我们基本都会输入git status命令，查看仓库状态。

这也从侧面说明了git status命令使用的频率之高，也建议大家在操作 Git 仓库的时候多使用git status命令，这能帮助我们实时了解仓库的状态，显然非常有用。

**第 6 个命令：git branch**

在命令行窗口的光标处，输入git branch命令，查看 Git 仓库的分支情况：



如上图所示，显示了仓库demo中的分支情况，现在仅有一个master分支，其中master分支前的\*号表示“当前所在的分支”，例如\* master就意味着我们所在的位置为demo仓库的主分支。输入命令git branch a，再输入命令git branch，结果如下图所示：



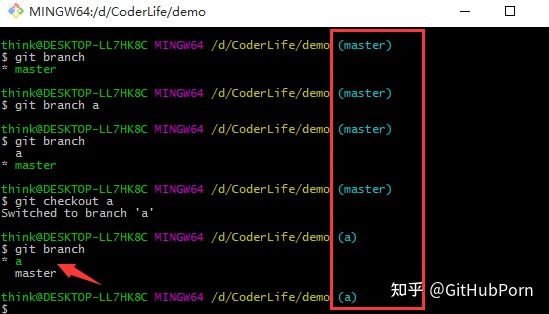
如上图所示，我们创建了一个名为a的分支，并且当前的位置仍然为主分支。

**第 7 个命令：git checkout**

在命令行窗口的光标处，输入git checkout a命令，切换到a分支：



如上图所示，我们已经切换到a分支啦！也可以通过命令git branch查看分支情况：



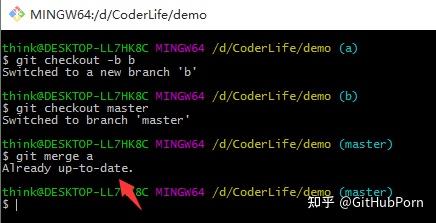
在这里，我们还有一个更简单的方法来查看当前的分支，即通过观察上图中用红色框圈起来的部分。此外，我们也可以在创建分支的同时，直接切换到新分支，命令为git checkout -b，例如输入git checkout -b b命令：



如上图所示，我们在a分支下创建b分支（b为a的分支），并直接切换到b分支。

**第 8 个命令：git merge**

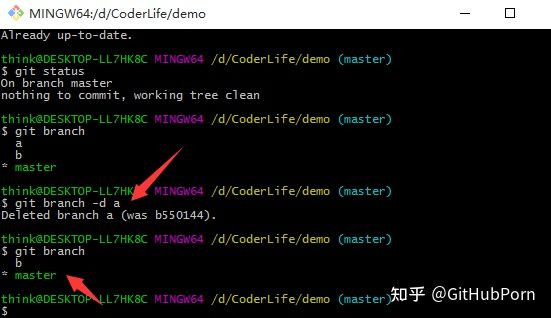
切换到master分支，然后输入git merge a命令，将a分支合并到master分支：



如上图所示，我们已经将a分支合并到主分支啦！此外，在这里需要注意一点，那就是：在合并分支的时候，要考虑到两个分支是否有冲突，如果有冲突，则不能直接合并，需要先解决冲突；反之，则可以直接合并。

**第 9 个命令：git branch -d & git branch -D**

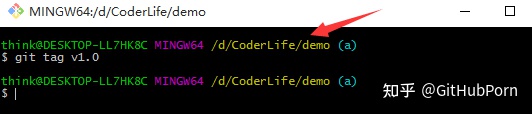
在命令行窗口的光标处，输入git branch -d a命令，删除a分支：



如上图所示，我们已经将分支a删除啦！不过有的时候，通过git branch -d命令可以出现删除不了现象，例如分支a的代码没有合并到主分支等，这时如果我们一定要删除该分支，那么我们可以通过命令git branch -D进行强制删除。

**第 10 个命令：git tag**

在命令行窗口的光标处，输入git tag v1.0命令，为当前分支添加标签：



如上图所示，我们为当前所在的a分支添加了一个v1.0标签。通过命令git tag即可查看标签记录：



如上图所示，显示了我们添加标签的记录。通过命令git checkout v1.0即可切换到该标签下的代码状态：



如上图所示，我们已经成功切换到a分支的v1.0标签啦！

**八、利用 SSH 完成 Git 与 GitHub 的绑定**

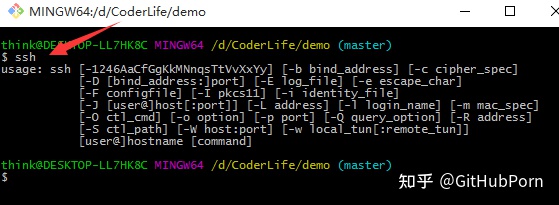
现在，我们已经对 GitHub 有了一定的了解，包括创建仓库、拉分支，或者通过Clone or download克隆或者下载代码；我们也下载并安装了 Git，也了解了其常用的命令。

But，无论是 GitHub，还是 Git，我们都是单独或者说是独立操作的，并没有将两者绑定啊！也就是说，我们现在只能通过 GitHub 下载代码，并不能通过 Git 向 GitHub 提交代码。

因此，在本篇博文中，我们就一起完成 Git 和 GitHub 的绑定，体验通过 Git 向 GitHub 提交代码的能力。不过在这之前，我们需要先了解 SSh（安全外壳协议），因为在 GitHub 上，一般都是通过 SSH 来授权的，而且大多数 Git 服务器也会选择使用 SSH 公钥来进行授权，所以想要向 GitHub 提交代码，首先就得在 GitHub 上添加 SSH key配置。

**第 1 步：生成 SSH key**

我们要想生成SSH key，首先就得先安装 SSH，对于 Linux 和 Mac 系统，其默认是安装 SSH 的，而对于 Windows 系统，其默认是不安装 SSH 的，不过由于我们安装了 Git Bash，其也应该自带了 SSH. 可以通过在 Git Bash 中输入ssh命令，查看本机是否安装 SSH：



如上图所示，此结果表示我们已经安装 SSH 啦！接下来，输入ssh-keygen -t rsa命令，表示我们指定 RSA 算法生成密钥，然后敲三次回车键，期间不需要输入密码，之后就就会生成两个文件，分别为id\_rsa和id\_rsa.pub，即密钥id\_rsa和公钥id\_rsa.pub. 对于这两个文件，其都为隐藏文件，默认生成在以下目录：

Linux 系统：~/.ssh

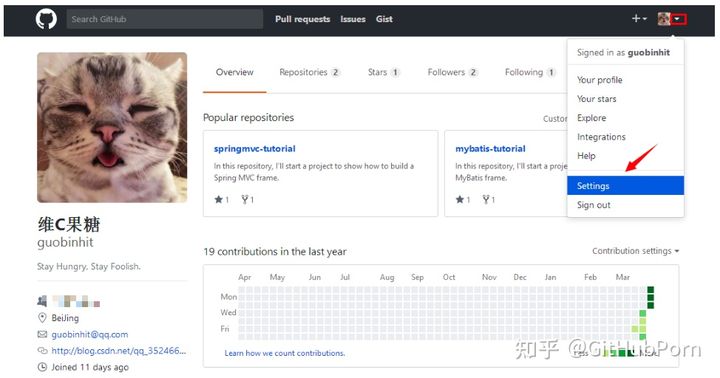
Mac 系统：~/.ssh

Windows 系统：C:\Documents and Settings\username\\.ssh

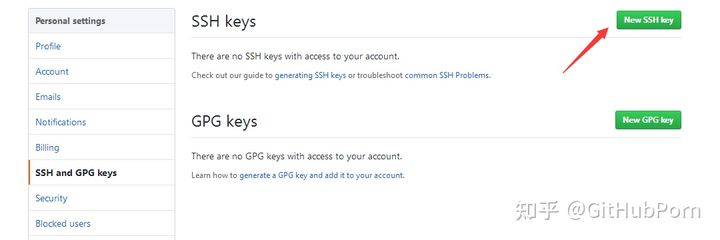
Windows 10 ThinkPad：C:\Users\think\.ssh

密钥和公钥生成之后，我们要做的事情就是把公钥id\_rsa.pub的内容添加到 GitHub，这样我们本地的密钥id\_rsa和 GitHub 上的公钥id\_rsa.pub才可以进行匹配，授权成功后，就可以向 GitHub 提交代码啦！

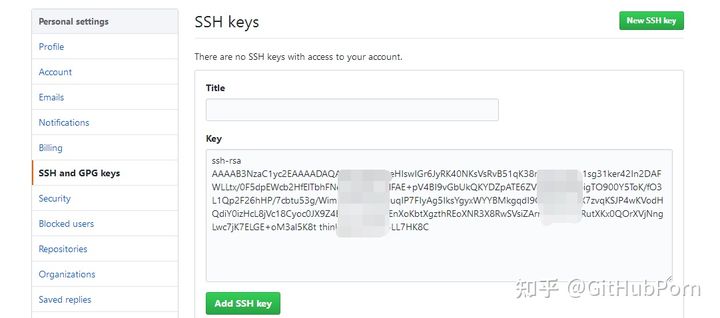
**第 2 步：添加 SSH key**



如上图所示，进入我们的 GitHub 主页，先点击右上角所示的倒三角▽图标，然后再点击Settins，进行设置页面；点击我们的头像亦可直接进入设置页面：



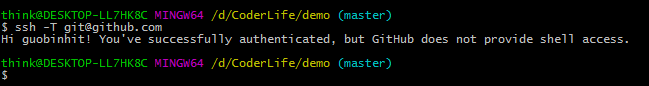
如上图所示，进入Settings页面后，再点击SSH and GPG Keys进入此子界面，然后点击New SSH key按钮：



如上图所示，我们只需要将公钥id\_rsa.pub的内容粘贴到Key处的位置（Titles的内容不填写也没事），然后点击Add SSH key 即可。

**第 3 步：验证绑定是否成功**

在我们添加完SSH key之后，也没有明确的通知告诉我们绑定成功啊！不过我们可以通过在 Git Bash 中输入ssh -T git@github.com进行测试：



如上图所示，此结果即为Git 与 GitHub 绑定成功的标志。

**九、通过 Git 将代码提交到 GitHub**

到这一步我们已经完成了本地 Git 与远程 GitHub 的绑定，这意味着我们已经可以通过 Git 向 GitHub 提交代码啦！

但是在进行演示之前，我们需要先了解两个命令，也是我们在将来需要经常用到的两个命令，分别为 push 和 pull 。

push：该单词直译过来就是“推”的意思，如果我们本地的代码有了更新，为了保持本地与远程的代码同步，我们就需要把本地的代码推到远程的仓库，代码示例：

git push origin master

pull：该单词直译过来就是“拉”的意思，如果我们远程仓库的代码有了更新，同样为了保持本地与远程的代码同步，我们就需要把远程的代码拉到本地，代码示例：

git pull origin master

此外，在之前我们讲到过pull request，在这里，估计大家就能更好的理解了，它表示：如果我们fork了别人的项目（或者说代码），并对其进行了修改，想要把我们的代码合并到原始项目（或者说原始代码）中，我们就需要提交一个pull request，让原作者把我们的代码拉到 ta 的项目中，至少对于 ta 来说，我们都是属于远程端的。

一般情况下，我们在push操作之前都会先进行pull操作，这样不容易造成冲突。

**提交代码**

对于向远处仓库（GitHub）提交代码，我们可以细分为两种情况：

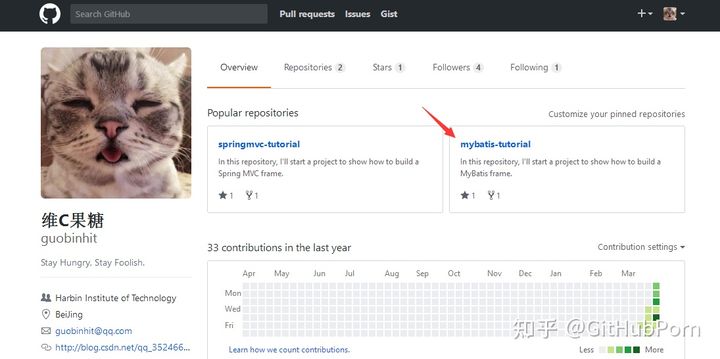
第一种：本地没有 Git 仓库，这时我们就可以直接将远程仓库clone到本地。通过clone命令创建的本地仓库，其本身就是一个 Git 仓库了，不用我们再进行init初始化操作啦，而且自动关联远程仓库。我们只需要在这个仓库进行修改或者添加等操作，然后commit即可。

接下来，以博主的 GitHub 账号中的 CSBook 项目为例，进行演示。

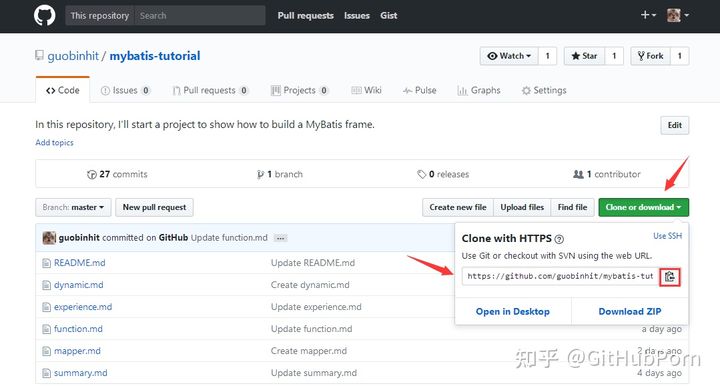
**CSBook：**计算机类常用电子书整理，并且附带下载链接，包括Java，Python，Linux，Go，C，C++，数据结构与算法，人工智能，计算机基础，面试，设计模式，数据库，前端等书籍。

[iCSToCS/CSBook​github.com/iCSToCS/CSBook](https://link.zhihu.com/?target=https%3A//github.com/iCSToCS/CSBook)

首先，进入 GitHub 个人主页：



如上图所示，点击 mybatis-tutorial 项目：



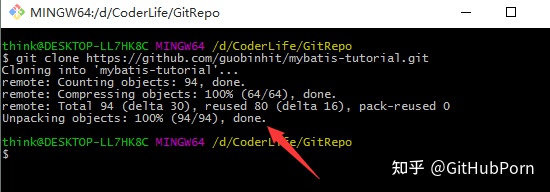
如上图所示，进入mybatis-tutorial项目后，点击Clone or download，复制上图所示的地址链接。然后，进入我们准备存储 Git 仓库的目录，例如下面我们新建的GitRepo目录， 从此目录进入 Git Bash：



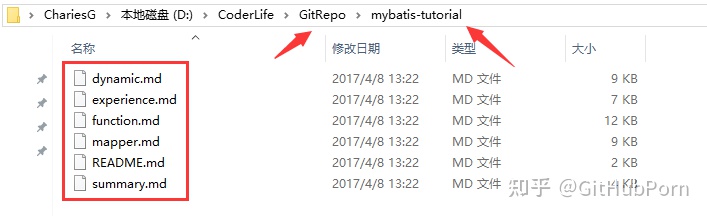
接下来，输入

git clone https://github.com/guobinhit/mybatis-tutorial.git

命令，其中clone后面所接的链接为我们刚刚复制的远程仓库的地址：



如上图所示，我们已经把远程的mybatis-tutorial仓库clone到本地啦！下面，我们看看clone到本地的仓库内容与远程仓库的内容，是否完全一致：



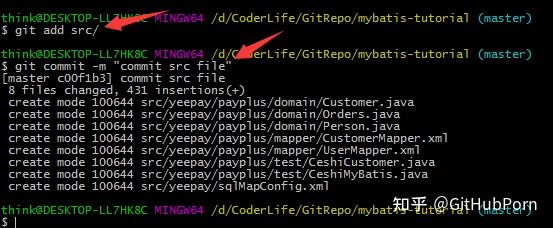
如上图所示，显示我们已经把远程仓库mybatis-tutorial的内容都clone到本地啦！接下来，为了方便演示，我们直接把之前重构的「史上最简单的 MyBatis 教程」里面的mybatis-demo项目的代码复制过来：



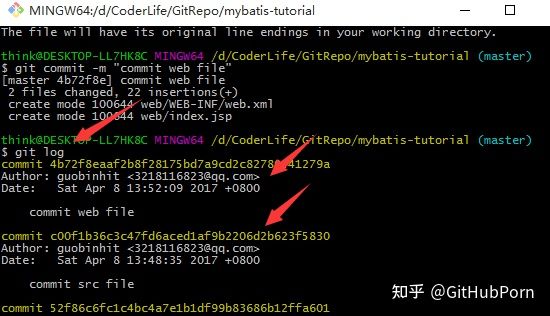
如上图所示，我们已经把mybatis-demo项目里面的主要内容src目录和web目录复制过来啦！接下来，从此目录进入 Git Bash，然后输入git status命令查看仓库状态：



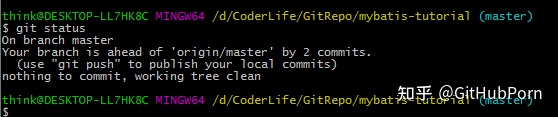
如上图所示，mybatis-tutorial已经是一个 Git 仓库了，而且在输入git status命令后显示有两个文件未被追踪，也就是我们刚刚复制过来的两个文件没有提交。通过「Git 初体验及其常用命令介绍」，我们已经知道了在真正提交代码之前，需要先进行git add操作：



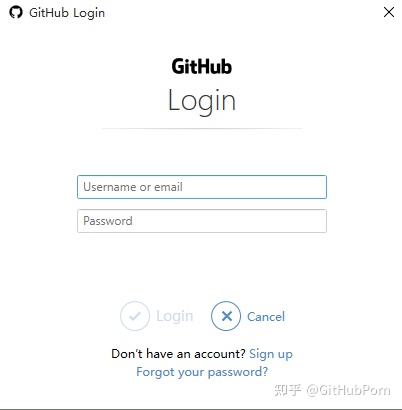
如上图所示，我们已经将src目录add并commit到mybatis-tutorial仓库啦！接下来，我们将web目录提交到仓库，然后输入git log命令查看仓库日志：



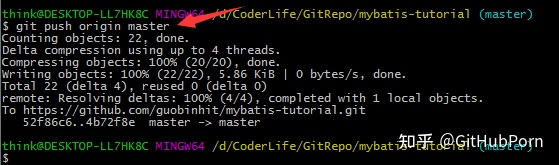
再输入git status命令查看仓库状态：



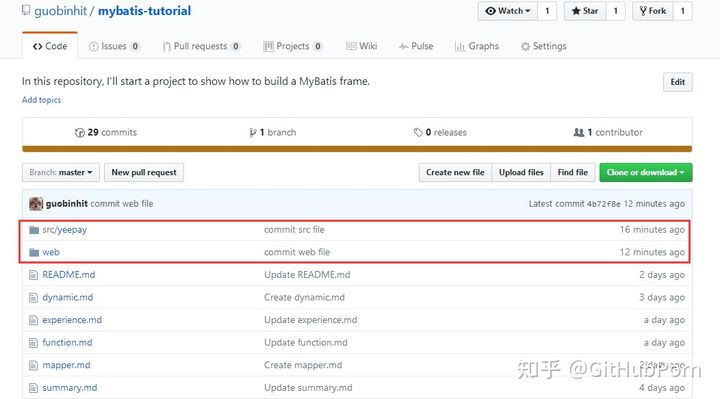
如上图所示，我们已经将mybatis-tutorial仓库里面新添加的两个目录都提交啦！下面，我们将本地仓库的内容push到远程仓库，输入git push origin master命令：



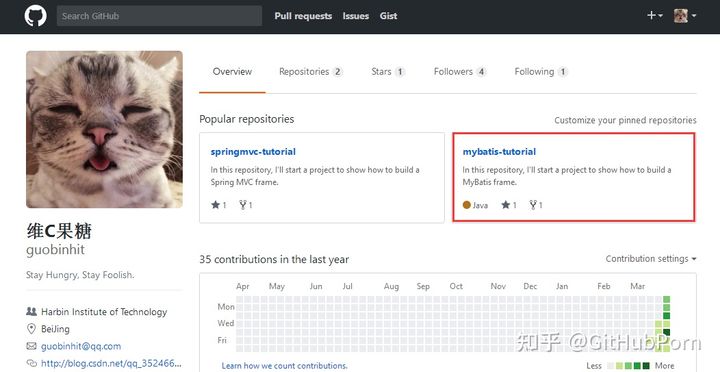
如上图所示，在第一次向远程仓库提交代码的时候，需要输入账号及密码进行验证，验证成功后，显示如下结果：



然后，刷新 GitHub 中mybatis-tutorial仓库：



如上图所示，我们已经将项目（仓库）中新添加的内容提交到了远程仓库。接下来，返回 GitHub 个人主页：



观察上图，我们会发现一个现象，那就是：mybatis-tutorial仓库的概要中新增了一个Java语言的标记。对于这个仓库语言的标记，其来源有两个，一是在我们创建仓库时就指定语言；二是在我们提交或者新建代码后由 GitHub 自动识别该语言。

以上介绍了向 GitHub 提交代码时的第一种情况，即：

第一种：本地没有 Git 仓库，这时我们可以直接将远程仓库clone到本地。通过clone命令创建的本地仓库，其本身就是一个 Git 仓库了，不用我们再进行init初始化操作啦，而且自动关联远程仓库。我们只需要在这个仓库进行修改或者添加等操作，然后commit即可。

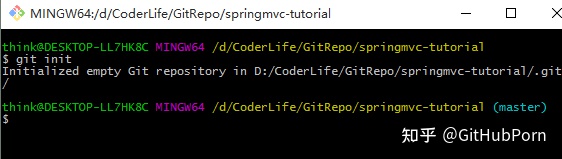
接下来，我们继续介绍向 GitHub 提交代码时可能遇到的第二种情况，即：

第二种：本地有 Git 仓库，并且我们已经进行了多次commit操作。

仍然以博主的开源项目为例，不过这次换成springmvc-tutorial项目进行演示。首先，建立一个本地仓库，命名为springmvc-tutorial：



如上图所示，进入该仓库，进入init初始化操作：



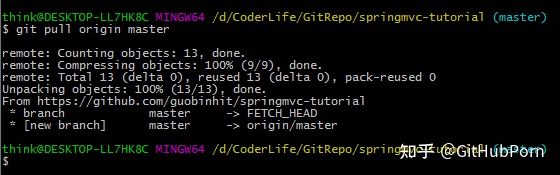
然后，输入

git remote add origin https://github.com/guobinhit/springmvc-tutorial.git

命令，关联远程仓库（在此，默认大家都知道如何获取远程仓库的地址），其中origin为远程仓库的名字：



输入git pull origin master命令，同步远程仓库和本地仓库：



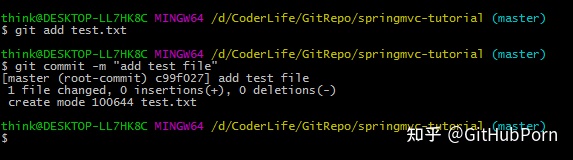
再回到本地springmvc-tutorial仓库，看看我们是否已经把远程仓库的内容同步到了本地：



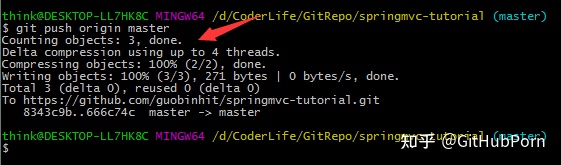
如上图所示，显然我们已经把远程springmvc-tutorial仓库里面仅有的README.md文件同步到了本地仓库。接下来，在本地仓库新建一个名为test.txt的测试文件：



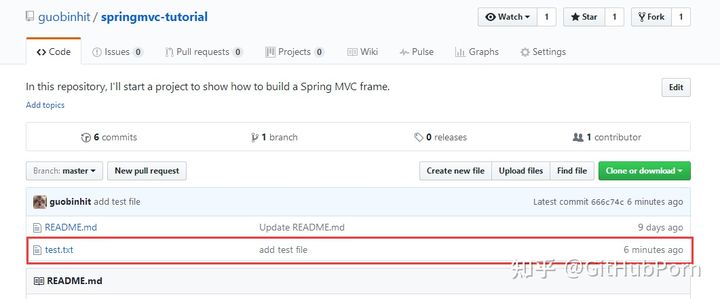
输入git add和git commit命令，将文件test.txt添加并提交到springmvc-tutorial仓库：



再输入git push origin master命令，将本地仓库修改（或者添加）的内容提交到远程仓库：



如上图所示，我们已经将本地仓库的内容同步到了远程仓库。下面，我们进入远程springmvc-tutorial仓库的页面，看看我们的提交结果：



如上图所示，我们已经将「通过 Git 将代码提交到 GitHub」的第二种情况演示完毕。

此外，在这个例子中，我们将远程仓库命名为origin，本地仓库名为springmvc-tutorial，其实两者的名字咱们可以随意取，一般来说，我们习惯性将远程仓库命名为origin，不过在需要关联多个远程仓库的时候，就需要我们再取别的名字啦！

最后，再强调一遍：在我们向远程仓库提交代码的时候，一定要先进行pull操作，再进行push操作，防止本地仓库与远程仓库不同步导致冲突的问题，尤其是第二种提交代码的情况，很容易就出现问题。