# Git与GitHub

# 第1章：简介

## Git与GitHub的简介

* Git是一个分布式版本控制系统。
* GitHub是一个托管Git项目的网站。

## “自由主义教皇”林纳斯·托瓦兹

* Linux
* Git

# 第2章：版本控制系统

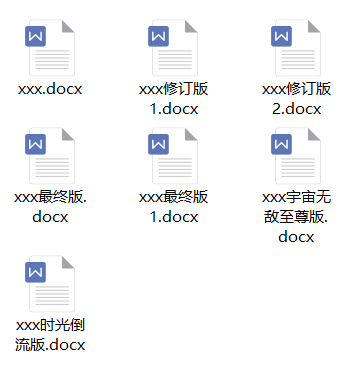
## 1. 什么是版本控制系统？

* 它是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统。
* 简单来说：它会记录你每次commit提交的代码的版本信息，你随时可以回退或者查看之前的历史版本。

## 2. 为什么要使用版本控制系统？

如果你用Microsoft Word写过长篇大论，那你一定有这样的经历：

想删除一个段落，又怕将来想恢复找不回来怎么办？有办法，先把当前文件“另存为……”一个新的Word文件，再接着改，改到一定程度，再“另存为……”一个新文件，这样一直改下去，最后你的Word文档变成了这样：



过了一周，你想找回被删除的文字，但是已经记不清删除前保存在哪个文件里了，只好一个一个文件去找，真麻烦。

看着一堆乱七八糟的文件，想保留最新的一个，然后把其他的删掉，又怕哪天会用上，还不敢删，真郁闷。

更要命的是，有些部分需要你的财务同事帮助填写，于是你把文件Copy到U盘里给她（也可能通过Email发送一份给她），然后，你继续修改Word文件。一天后，同事再把Word文件传给你，此时，你必须想想，发给她之后到你收到她的文件期间，你作了哪些改动，得把你的改动和她的部分合并，真困难。

正因为开发中我们也会遇到这些问题，所以我们需要引入版本控制来管理我们代码的历史版本。

## 3. 使用版本控制系统的优势

* 浏览所有开发的历史纪录。
* 掌握团队的开发进度。
* 可以做任意修改，可以回退之前正常的版本。
* 通过分支功能来发行软件的不同版本，例如：稳定版和开发版。

## 4. 版本控制系统的分类

### 4.1. 集中式版本控制系统

代表有SVN、CVS

集中式版本控制系统，版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。中央服务器就好比是一个图书馆，你要改一本书，必须先从图书馆借出来，然后回到家自己改，改完了，再放回图书馆。



集中式版本控制系统最大的毛病就是必须联网才能工作，如果在局域网内还好，带宽够大，速度够快，可如果在互联网上，遇到网速慢的话，可能提交一个10M的文件就需要5分钟，这还不得把人给憋死啊。

### 4.2. 分布式版本控制系统

代表有Git、BitKeeper

和集中式版本控制系统相比，分布式版本控制系统的安全性要高很多，因为每个人电脑里都有完整的版本库，某一个人的电脑坏掉了不要紧，随便从其他人那里复制一个就可以了。而集中式版本控制系统的中央服务器要是出了问题，所有人都没法干活了。



在实际使用分布式版本控制系统的时候，其实很少在两人之间的电脑上推送版本库的修改，因为可能你们俩不在一个局域网内，两台电脑互相访问不了，也可能今天你的同事病了，他的电脑压根没有开机。因此，分布式版本控制系统通常也有一台充当“中央服务器”的电脑，但这个服务器的作用仅仅是用来方便“交换”大家的修改，没有它大家也一样干活，只是交换修改不方便而已。

### 4.3. 集中式与分布式的优缺点

* 集中式的优缺点：
  + 优点：权限管理方便。中央服务器存储全部代码，代码安全度高。
  + 缺点：中央服务器宕机就系统瘫痪。没网就没法工作。在网络慢的时候程序员的时间都浪费在代码更新上。
* 分布式的优缺点：
  + 优点：每个电脑都有一套完整的代码库。没有网的情况下仍旧可以正常工作。有网的时候再同步代码。
  + 缺点：权限控制不严格。

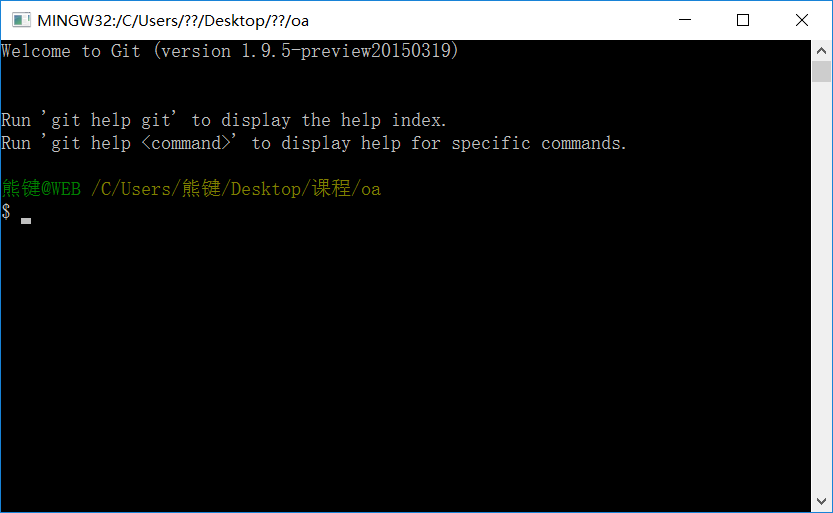
# 第3章：Git

## Git是什么？

* Git是目前世界上最先进的分布式版本控制系统（没有之一）。

## Git的安装

* Windows版的Git，从https://git-for-windows.github.io下载然后按默认选项安装即可。
* 安装完成后，右键打开菜单栏找到“Git”->“Git Bash”，蹦出一个类似命令行窗口的东西，就说明Git安装成功！



* 安装完成后，还需要最后一步设置，在命令行输入：
  + git config --global user.name "Your Name"
  + git config --global user.email "email@example.com"
  + 因为Git是分布式版本控制系统，所以，每个机器都必须自报家门：你的名字和Email地址。你也许会担心，如果有人故意冒充别人怎么办？这个不必担心，首先我们相信大家都是善良无知的群众，其次，真的有冒充的也是有办法可查的。
  + 注意git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

## Git的使用（重点）

### 3.1. Linux的命令

* mkdir xxx 新建文件夹
* vi x.txt 新建文件（Visual editor）
  + 输入 i 进入编辑模式
  + ESC + ：+ wq 保存并退出
* cd xxx 进入xxx目录
* cd .. 返回上一级目录
* ls 列出当前文件夹中所有文件
* pwd 显示当前目录
* cat x.txt 显示文件内容
* clear 清屏

### 3.2. 创建版本库

1. git init初始化版本库

* 创建成功会提示 Initialized empty Git repository in c:/Users/熊键/Desktop/0725git/.git/
* 没有初始化执行git命令，会提示fatal: Not a git repository (or any of the parent directories): .git
* 你还会发现目录上多出一个.git的文件夹,这个文件夹是Git来跟踪管理版本库的，不要去修改这个文件里的内容。

1. git add x.txt添加指定文件到仓库中

* 不会有任何提示，但是提交成功了
* 失败会提示fatal: pathspec 'x.txt' did not match any files

1. 怎么查看有没有添加成功呢？

* git status 红色表示在工作区，绿色表示在暂存区

1. git commit -m 'xxx'提交所有文件

* 提交成功会提示： [master (root-commit) 88bbb64] first commit  
    1 file changed, 2 insertions(+)  
    create mode 100644 x.txt
* 如果只输入git commit会出问题，ESC + ：+ wq 退出就好

### 3.3. 理解工作区+版本库+暂存区

* 工作区（working Directory）：简单的理解你在电脑里能看到的目录。
* 版本库（Repository）：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。
* 暂存区（stage）：Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。
  + 第一步是用git add把文件添加进去，实际上就是将工作区文件添加到暂存区
  + 第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支（版本库）。



### 3.4. 日志+版本号+对比不同

* git diff HEAD -- filename 查看工作区和暂存区里面最新版本的区别
* git log 显示从最近到最远的所有提交日志
* git reflog 显示每次提交（commit）的commit id

### 3.5. 版本回退+版本穿梭+版本撤销

* git reset --hard HEAD^ 版本回退（回退一次提交）
* git reset --hard Obfafd 回退到指定Obfafd的commit id版本
* git checkout -- x.txt 将文件在工作区的修改全部撤销

### 3.6. 删除文件、删完提交

* rm x.txt 或 手动删除
* git rm x.txt 确认删除
* git checkout -- x.txt 撤销删除

### 3.7. 分支（查看+新建+切换+合并+删除）

* git checkout -b dev 创建dev分支，并切换到dev分支
* git branch 查看当前分支
* git checkout master 切换分支
* git merge dev 合并指定dev分支到当前分支
* git branch -d dev 删除指定分支

### 3.8. 版本冲突

* 合并分支时，如果在同一个文件，在同一个地方，都修改了或新增内容会引起版本冲突。
* 解决版本冲突最好办法是手动解决，去文件里直接修改。

# 第4章：GitHub

## GitHub是什么？

* GitHub是一个Git项目托管网站，主要提供基于Git的版本托管服务。

## GitHub能干嘛？

* 能够分享你的代码或者其他开发人员配合一起开发。
* GitHub是一个基于Git的代码托管平台，Git并不像SVN那样有一个中心服务器。目前我们使用到的Git命令都是在本地执行，你就需要将数据放到一台其他开发人员能够连接的服务器上。

## GitHub怎么使用？

* https://github.com/

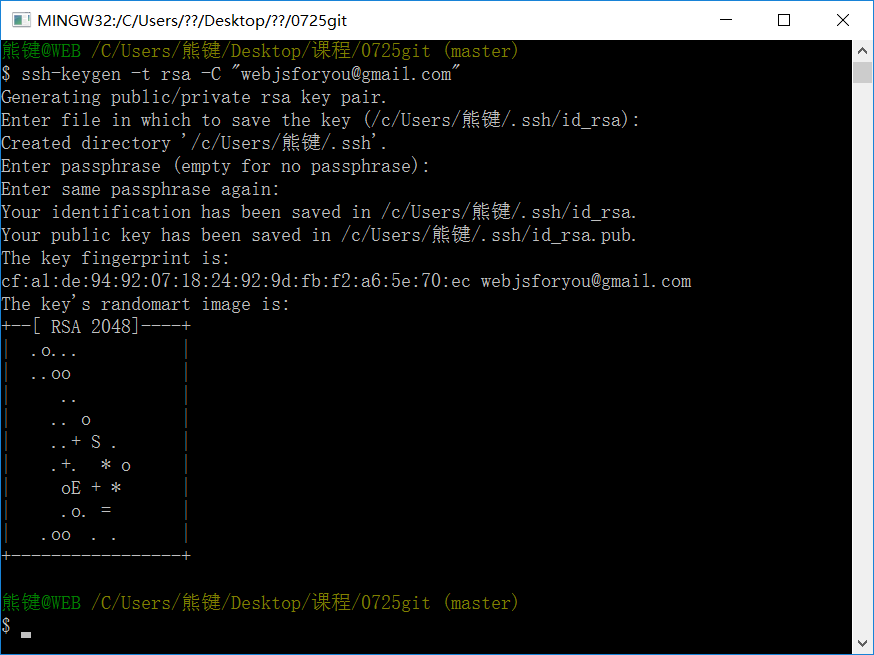
## 配置SSH与使用

1. 输入cd ~/.ssh，由于你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以我们需要配置验证信息。

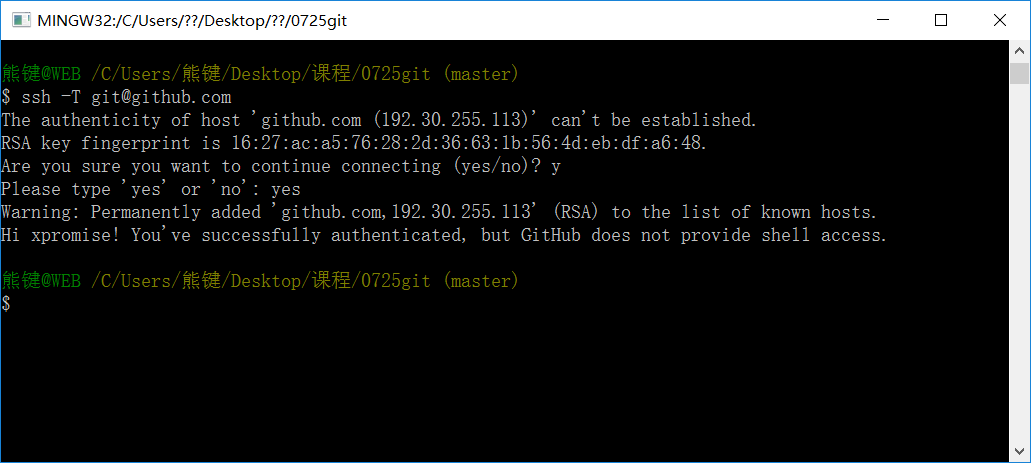
* No such file or directory 表示第一次

1. 创建SSH Key：ssh-keygen -t rsa -C ["xxx@xxx.com"](mailto:%22xxx@xxx.com%22)

* 它会在~/下生成.ssh文件夹，进去打开id\_rsa.pub文件复制里面全部的内容
* 将其复制到你的GitHub里的SSH keys中



1. 测试连通性：ssh -T [git@github.com](mailto:git@github.com)



## GitHub远程仓库的使用

* 现在的情景是，我在本地创建了一个oa项目后，我又想在GitHub创建一个oa项目，并且让这两个仓库进行远程同步

1. 现在本地新建一个Git项目

* git init
* git add .
* git commit -m “first commit”

1. 到GitHub上新建一个同名的空项目

* New repository

1. 本地和GitHub上的仓库进行关联

* git remote add origin [git@github.com:xpromise/oa.git](mailto:git@github.com:xpromise/oa.git) (SSH)
* git remote add origin <https://github.com/xpromise/oa.git> (HTTPS)

1. 把本地库的内容推送到远程git push命令，实际上是把当前分支master推送到远程

* git push -u origin master （首次）
* 由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送到远程新的master分支，还会把本地master分支和远程的master分支关联起来，在以后的推送或者拉取时可以简化命令git push origin master。

## Git的交互模型

### 6.1. 创建本地仓库

* 创建.gitignore配置文件
* git init
* git add .
* git commit -m "xxx"

### 6.2. 创建GitHub远程仓库

* New repository
* 指定仓库名称
* 创建

### 6.3. 将本地仓库推送到远程仓库

* git remote add origin https://github.com/xxx.git 关联远程仓库
* git push origin master 将本地仓库推送到远程仓库

### 6.4. push本地的更新

* git add .
* git commit -m "xxx"
* git push origin master

### 6.5. 克隆GitHub上的项目

* git clone https://github.com/xxx.git 将远程仓库克隆到本地

### 6.6. pull远程的更新

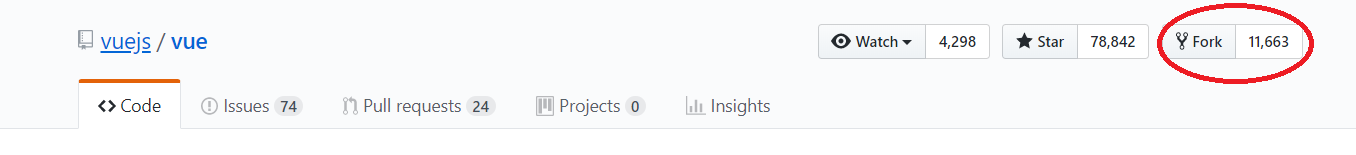
* git pull origin master 将远程仓库拉到本地来

### 6.7. 撤消本地修改

* git status 查看变化
* git checkout -- xxx 撤消指定文件的修改

## GitHub其他功能

### 7.1. Fork



现在的情景是，用叉子把别人的东西（copy on cut）叉到你碗里，就是把别人的项目clone一份，但是owner变成自己，这样你就可以在遵守open source license的前提下任意修改这个项目了。

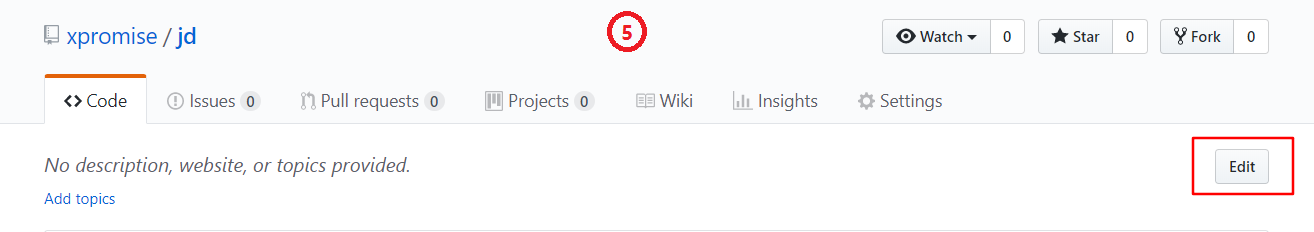
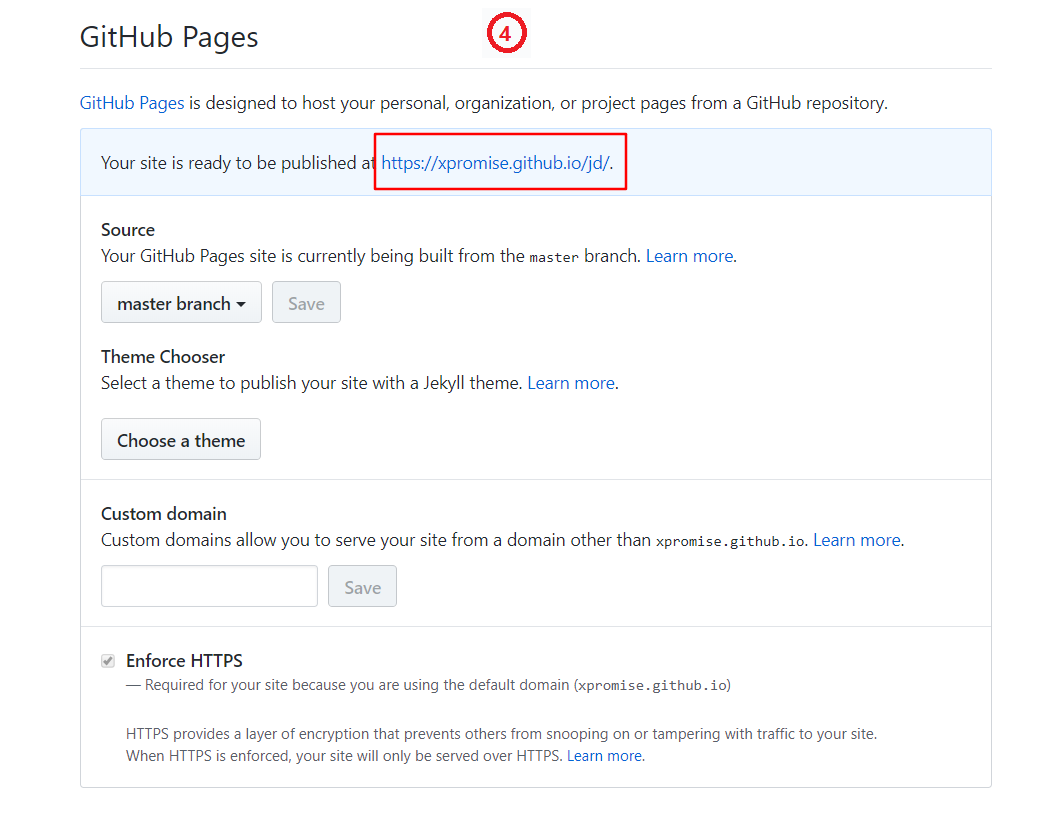
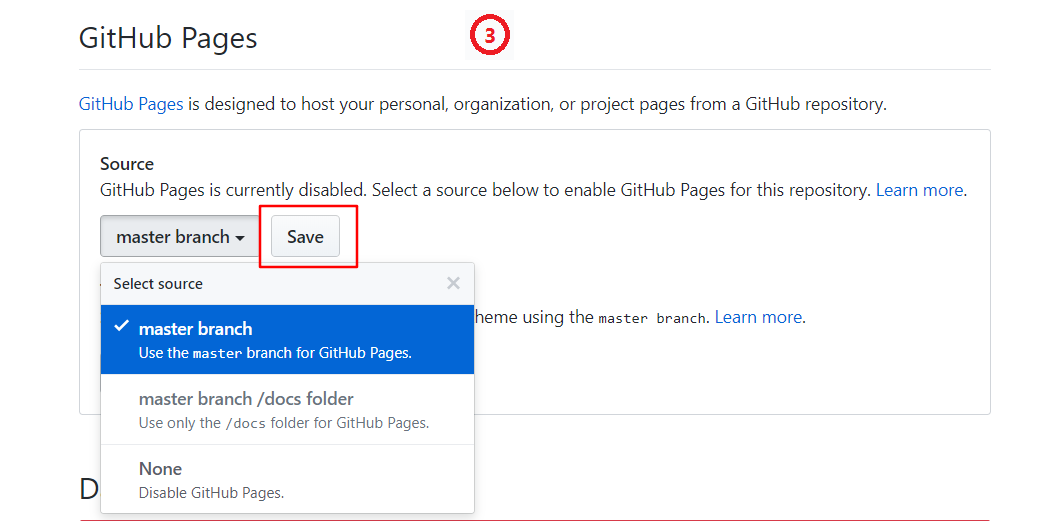
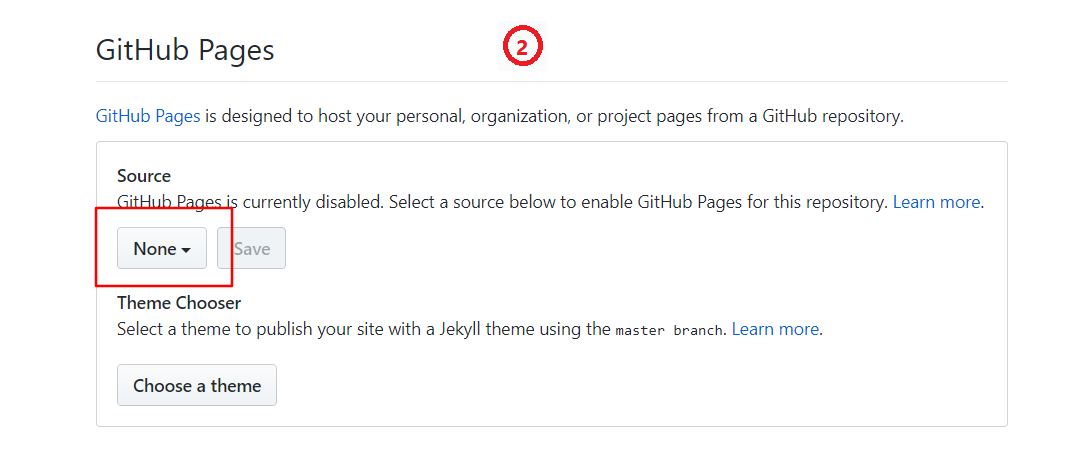
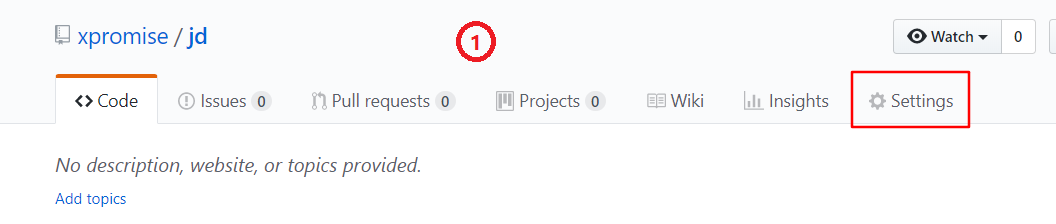
相当于你在原项目的主分支上又建立了一个分支，你可以在该分支上任意修改如果想将你得修改合并到元项目中式，可以pull request，这样原项目的作者就可以将你修改的东西合并到原项目的主分支上去，这样你就为开源项目贡献了代码，开源项目就会在大家的努力下不断地壮大和完善。

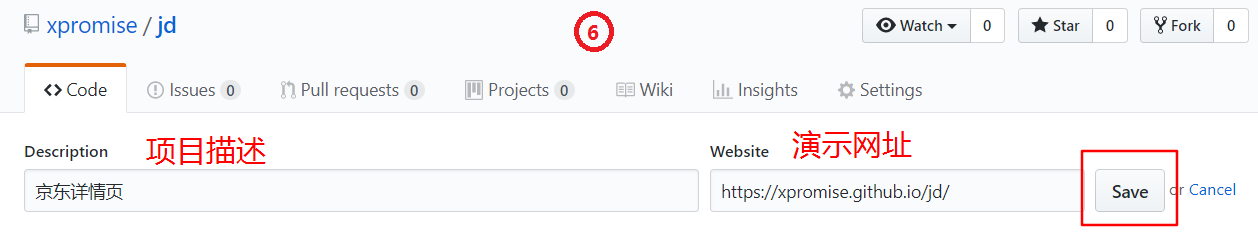
### 7.2. GitHub Pages

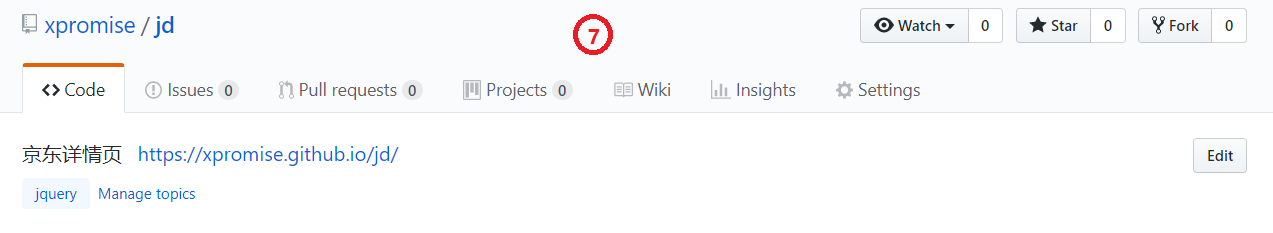
1. 推送本地代码到GitHub仓库

* 创建本地仓库
* git init
* git add .
* git commit -m "xxx"
* 创建GitHub远程仓库
* 本地仓库关联远程仓库
* git remote add origin [git@github.com:xpromise/jd.git](mailto:git@github.com:xpromise/jd.git)
* 推送本地更新
* git push -u origin master

1. 配置GitHub Pages选项







1. 需要注意的问题

* 目录第一层需要有一个index.html文件作为启动文件，命名不可更改。

