# Git与GitHub

# 第1章：Git的起源

## 20180407230104746“自由主义教皇”林纳斯·托瓦兹

* Linux
* Git

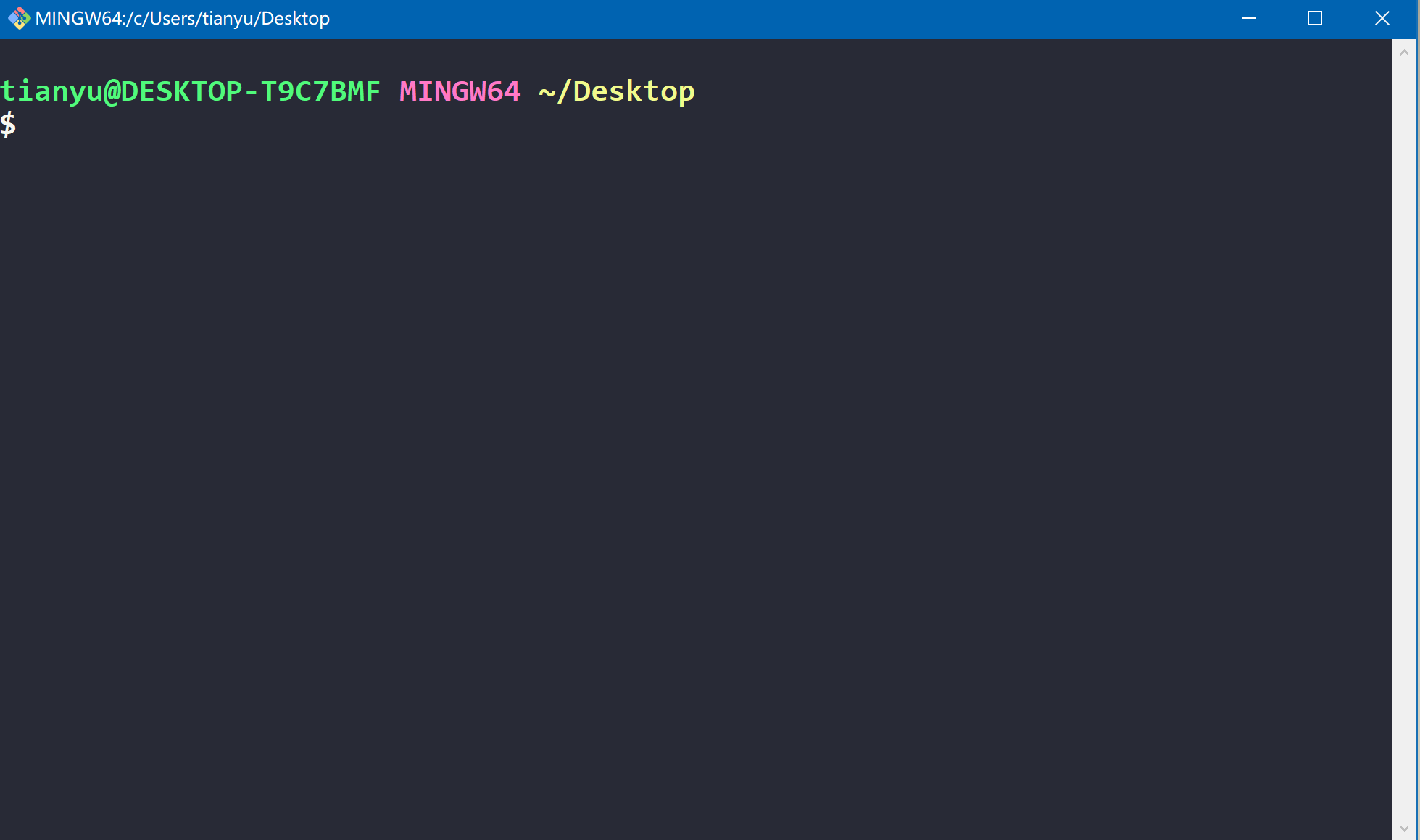
## Git是什么？

* Git是目前世界上最先进的分布式版本控制 （没有之一）。

# 第2章：Git的使用

## Git的安装

* Windows版的Git，从https://git-scm.com/download/win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"

备注：以上两步的名字和邮箱可随意配置，但最好使用自己的邮箱。

* + git config user.name 用于查看配置的姓名
  + git config user.email 用于查看配置的邮箱
  + 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。

## 2. 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
  + ESC + ：+ q! 不保存并退出
* 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. 工作区+版本区+暂存区

工作区（working Directory）：简单的理解——你在电脑里能看到的目录。

暂存区（stage）：介于工作区和版本区中间，工作区到版本区的“必经之路”

版本库（Repository）：工作区有一个隐藏目录.git，准确的来说这个不算工作区，而是Git的版本库。

第一步：git add把文件添加进暂存区。

第二步：git commit把暂存区的所有内容提交到当前版本库。

## 4. 创建版本库

1. git init命令：初始化版本库

创建成功会提示：**Initialized empty Git repository in c:/Users/xxx/Desktop/demogit/.git/**

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，不要去修改和删除这个文件里的内容。

1. git add x.xx命令：添加指定文件到暂存区中

没有任何提示，代表交成功了

若未初始化，会提示fatal: Not a git repository (or any of the parent directories): .git

失败会提示fatal: pathspec 'x.txt' did not match any files

可能会出现警告：warning: LF will be replaced by CRLF in xxxxx

原因： linux和window的换行符不一致导致的。

解决方式：执行命令git config --global core.autocrlf false

1. 怎么查看文件有没有添加成功呢？

* git status

1. **红色**表示在**工作区**。

2. 绿色表示在暂存区。

3. 没有任何显示代表所有文件位于版本区。

1. git commit -m 'xxx' 提交暂存区所有文件到版本区

* 提交成功会提示：

[master (root-commit) 88bbb64]

first commit1 file changed,

2 insertions(+)create mode 100644 x.txt

* 如果只输入git commit会出问题，这时需要ESC + ： q! 退出就好

## 差异对比(了解)

git diff : 比较暂存区与工作区

git diff --cached : 比较版本区与暂存区

git diff master : 比较版本区与工作区

## 6. 日志+版本号

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

## 7. 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 版本号  回退到指定Obfafd的commit id版本

git reset HEAD 用**版本库中**的文件去替换**暂存区**的全部文件。

git checkout -- x.txt  用**暂存区指定文件**去替换**工作区的指定文件**（危险）

git checkout HEAD x.txt 用**版本库中的文件**替换**暂存区**和**工作区的文件**（危险）

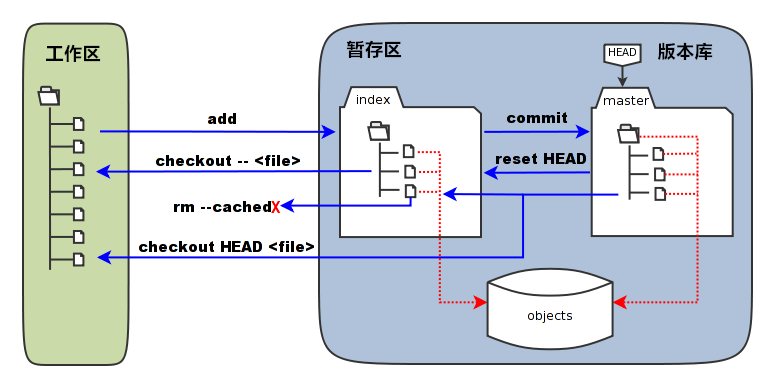
git rm --cached x.txt 从**暂存区**删除文件

## 8. 删除文件

git rm x.txt 删除文件

git rm -r xxxx 删除文件夹

**git完整图示如下：**



## 9. 分支

* 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dev分支到当前分支
* git branch -d dev  删除指定分支
* git diff branch1 branch2 显示出两个分支之间所有有差异的文件的详细差异
* git diff branch1 branch2 --stat 显示出两个分支之间所有有差异的文件列表
* git diff branch1 branch2 xxx 显示指定文件的详细差异

## 10. 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。
* 解决版本冲突最好的办法是借助IDE解决，简单且高效。

# 第3章：版本控制系统的区别

## 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，每个开发人员电脑里只有其中一个版本。

## 2. 分布式版本控制系统

代表有Git、BitKeeper

每个开发人员电脑里都有一个完整的版本库。同时，它也需要一台充当“中央服务器”的电脑，来方便“交换”大家的代码修改。

## 3. 集中式与分布式的区别

|  |  |  |
| --- | --- | --- |
| 类别  项目 | 集中式版本控制系统 | 分布式版本控制系统 |
| 主要区别 | 每个开发者只有应用代码库的一个版本。 | 每个开发者都有整个代码库的所有版本。 |
| 在离线状态下开发者无法进行版本管理开发。 | 在离线状态下开发者可以进行版本管理开发, 等到有网时再push到仓库中。 |

# 第4章：GitHub

## GitHub是什么？

* GitHub是一个Git项目托管网站。

## GitHub能做什么？

* 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远程仓库的使用

### 场景一（关联）：本地有仓库，要和远程仓库做关联

* git init
* git add .
* git commit -m “first commit”
* 在GitHub上创建一个远程仓库
* git remote add origin <https://github.com/xpromise/oa.git> (HTTPS)

备注：如果此步关联错了，解决办法如下。

暴力解决：删除.git文件夹，重新建立本地仓库。

优雅解决：git remote remove origin，再在重新关联仓库。

### 3.2. 场景二（推送）：本地有仓库有内容，要推送给远程库

* git push -u origin master （首次加-u）
* 根据提示输入用户名密码
* 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，在以后的推送时可以简化命令git push origin master。
* 备注：正常情况下，成功推送一次后，电脑会记住和账号与密码，下次推送时不会再提示输入。若在电脑不能够自动记住github的账户和密码，需执行以下命令解决：git config --global credential.helper store

### 3.2. 场景三（拉取）：本地有仓库有内容，获取远程库新内容

* 第一种拉取方式：git pull origin master

将远程仓库的master分支上代码版本复制/合并到本地master分支上

* 第二种拉取方式：git fetch origin master:tmp

新建一个tmp分支，将远程仓库的master分支上代码版本复制到tmp分支上，不会自动合并。

### 3.2. 场景四（克隆）：本地无仓库，要获取一个完整的远程库

备注：只在第一次获取远程库时才需要克隆

* git clone https://github.com/xxx.git (HTTPS)

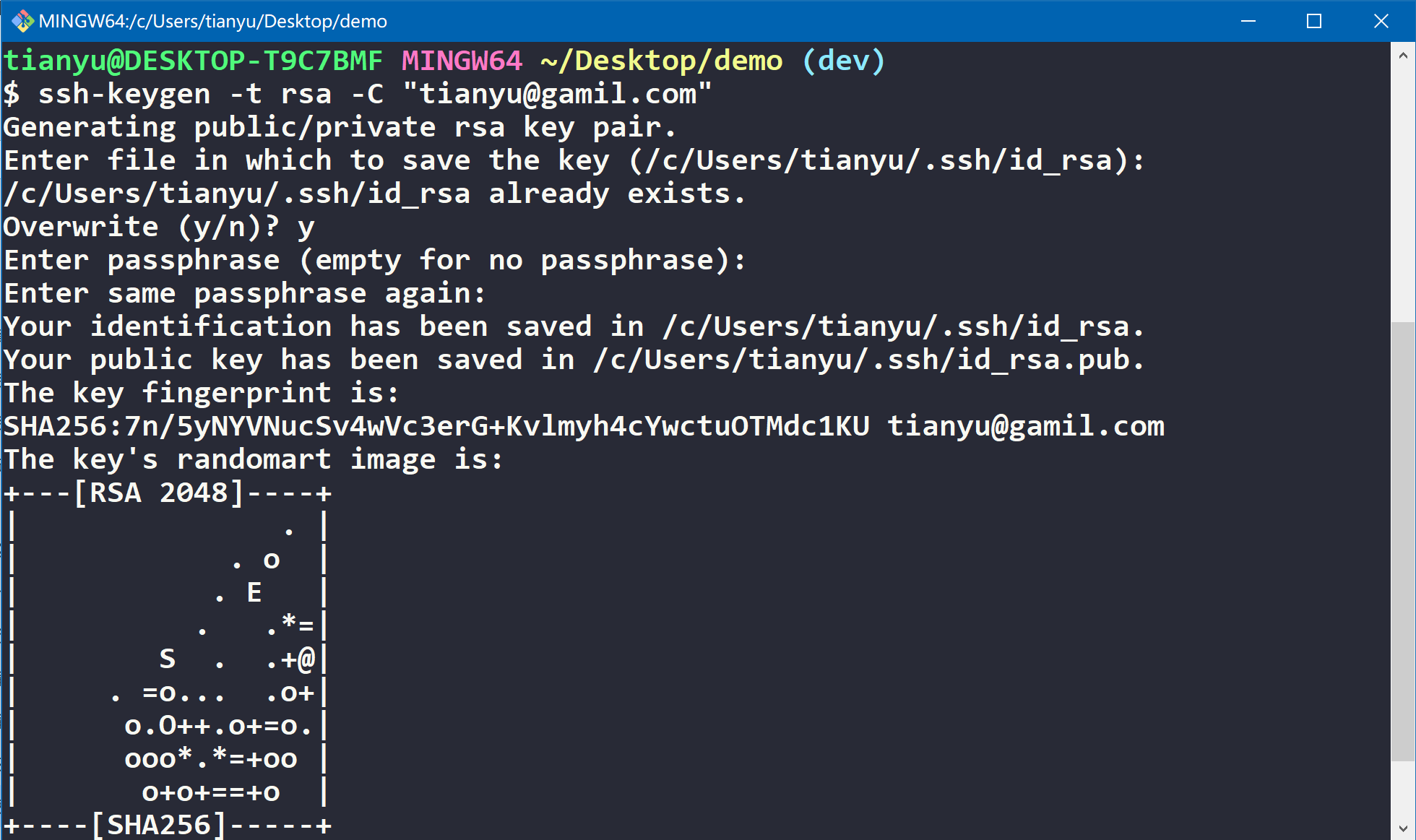
## 使用SSH方式与github交互

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

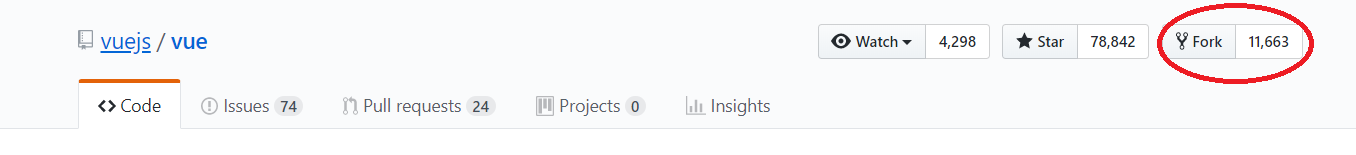


1. 使用

* git remote add origin [git@github.com:xpromise/oa.git](mailto:git@github.com:xpromise/oa.git) (SSH)
* git clone git@github.com:xpromise/oa.git (SSH)

## GitHub其他功能

### 6.1. Fork



将别人的项目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。

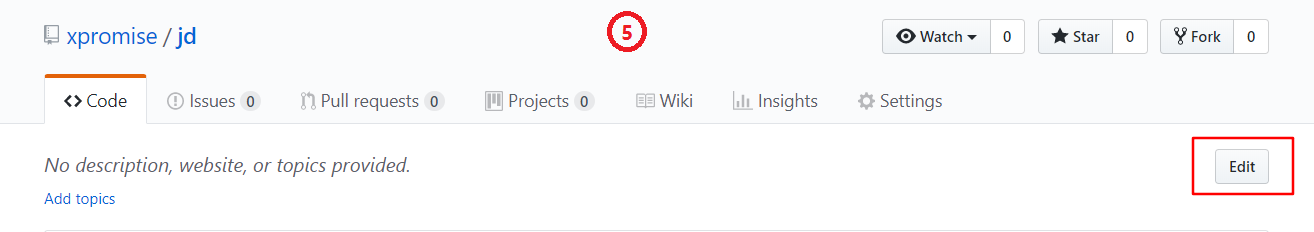
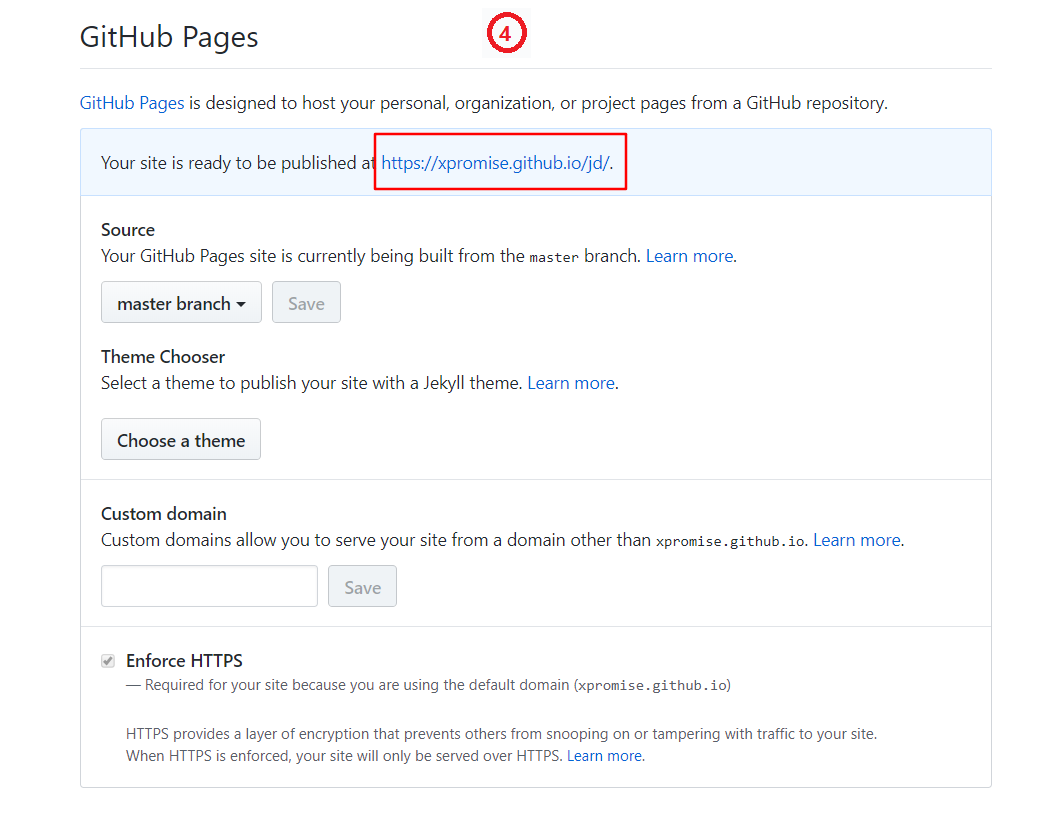
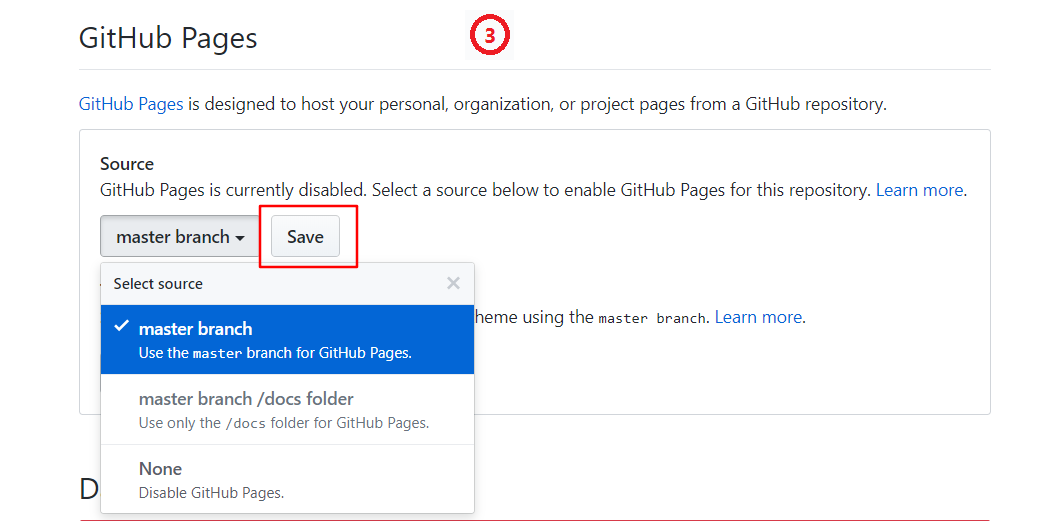
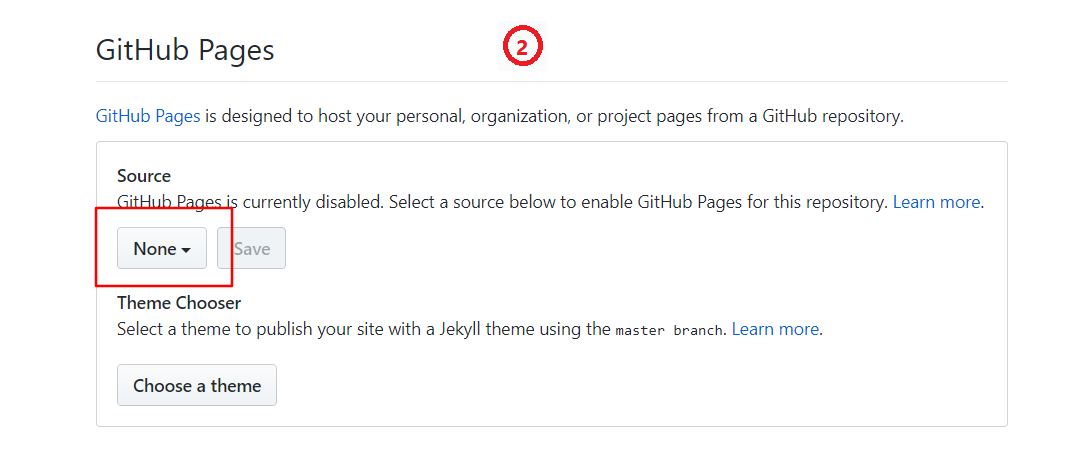
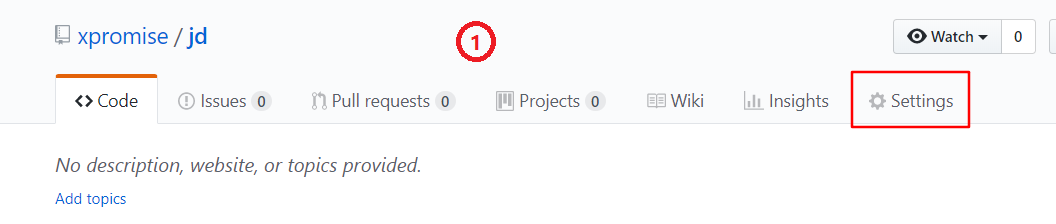
相当于你在原项目的主分支上又建立了一个分支，你可以在该分支上任意修改如果想将你得修改合并到元项目中式，可以pull request，这样原项目的作者就可以将你修改的东西合并到原项目的主分支上去，这样你就为开源项目贡献了代码，开源项目就会在大家的努力下不断地壮大和完善。

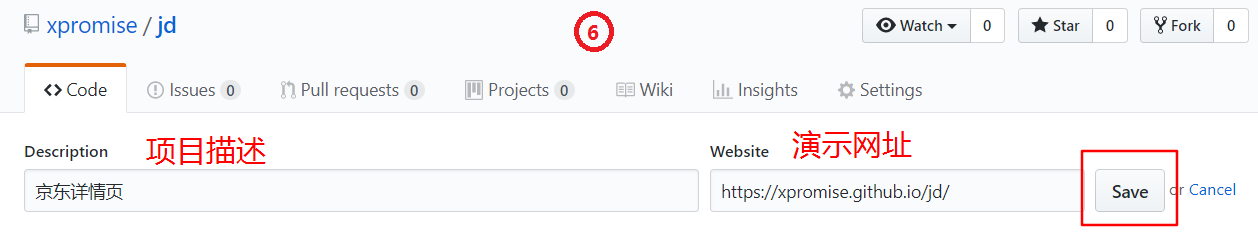
### 6.2. GitHub Pages

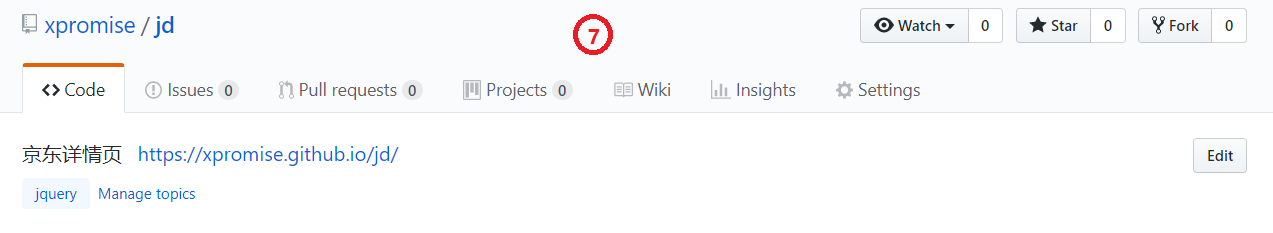
1. 推送本地代码到GitHub仓库

* 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

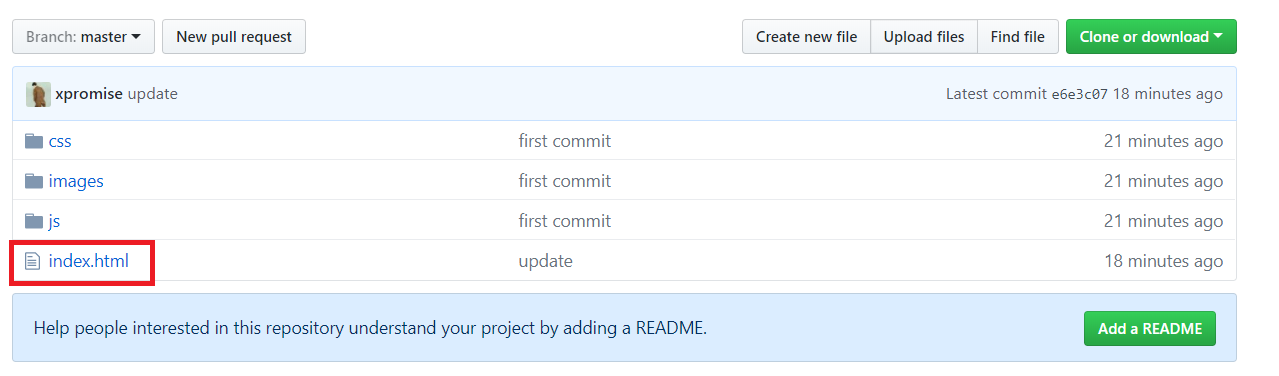






1. 需要注意的问题

* 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。



### 6.3. 博客

参考扩展文档