目录

[1. 工具地址 2](#_Toc23241)

[2. 安装Git 2](#_Toc8040)

[3. 安装TortoiseGit 3](#_Toc23335)

[4. 安装语言包 7](#_Toc20164)

[5. 创建github账号 8](#_Toc29779)

[6. 创建一个新的仓库 9](#_Toc25505)

[7. 通过SSH连接Github 11](#_Toc4660)

[8. Git常规操作 13](#_Toc31673)

[9. Git其他指令 22](#_Toc24412)

[10. 附录 22](#_Toc5131)

# 工具地址

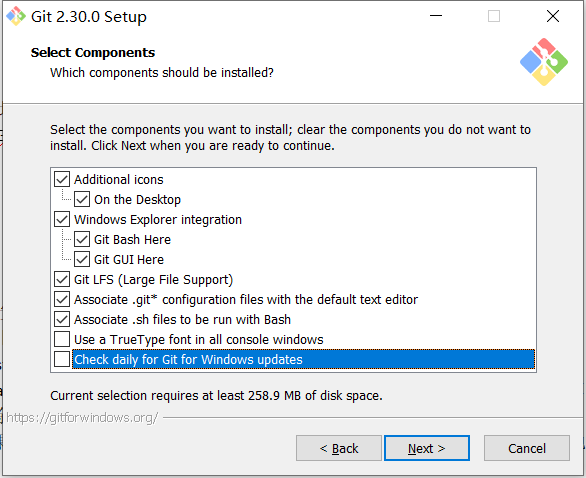
Git下载地址：<https://git-scm.com/downloads>

TortoiseGit 和语言包：<https://tortoisegit.org/download/>

# 安装Git

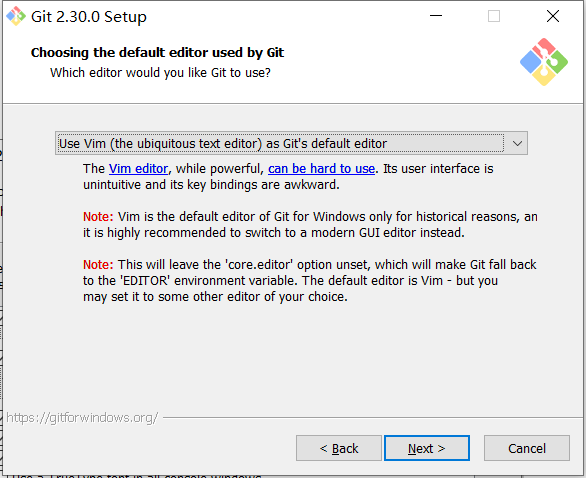
第一步：选择安装。

第二步：勾选必需的项，其他可选可不选。

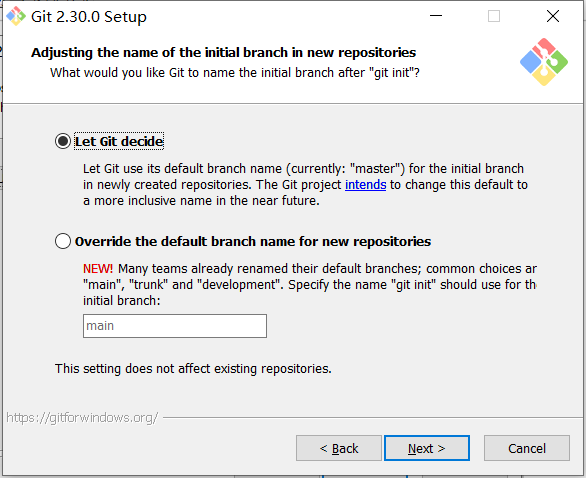


第三步：默认就行。

第四步：编辑器是在输入的时候需要使用的，默认的编辑器就行，如有个人偏好，请自行选择，不过需下载安装。



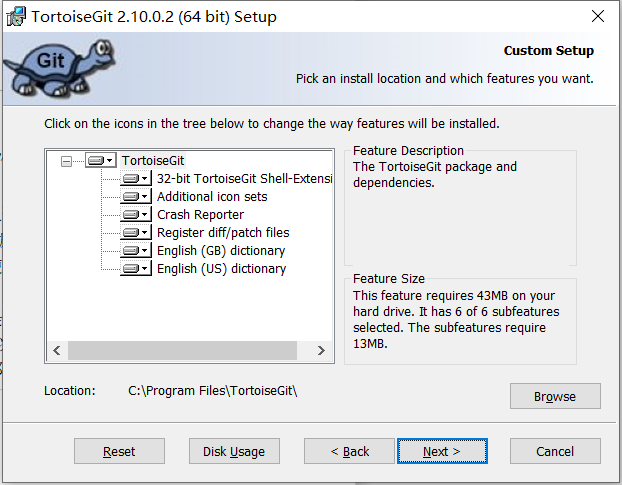
第五步：新建仓库时分支名称，默认为“master”，如需修改可自定义。



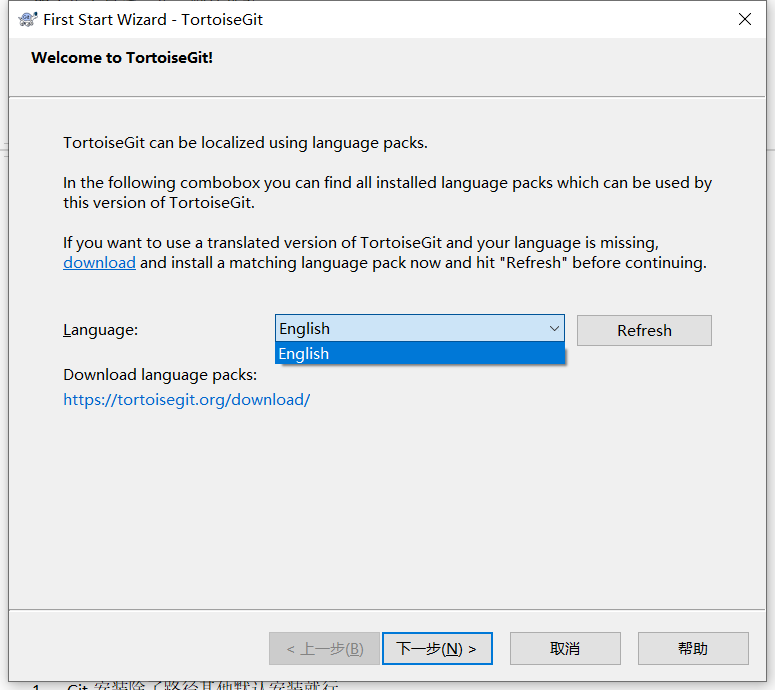
第六步至最后一步：默认选项。

# 安装TortoiseGit

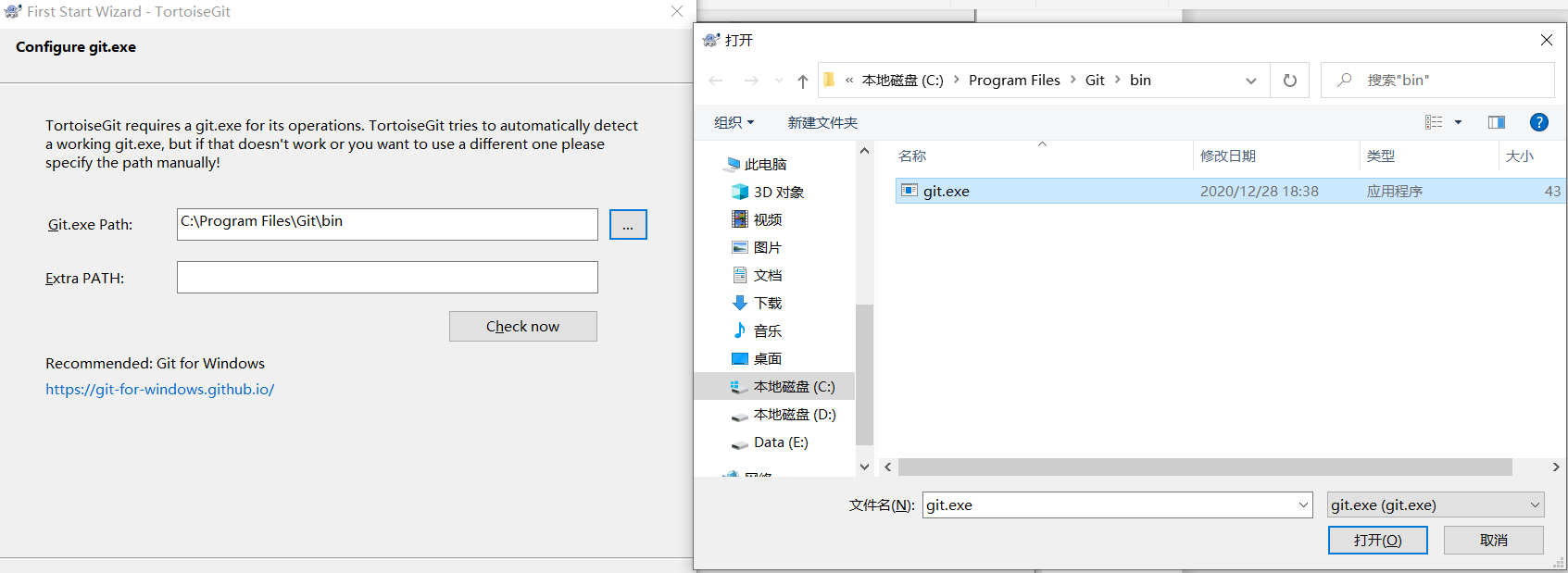
第三步：将所有的全部选中安装，再选择安装目录，点击下一步。



安装完成后，只有一个语言，选择English就行，然后下一步，再下一步。

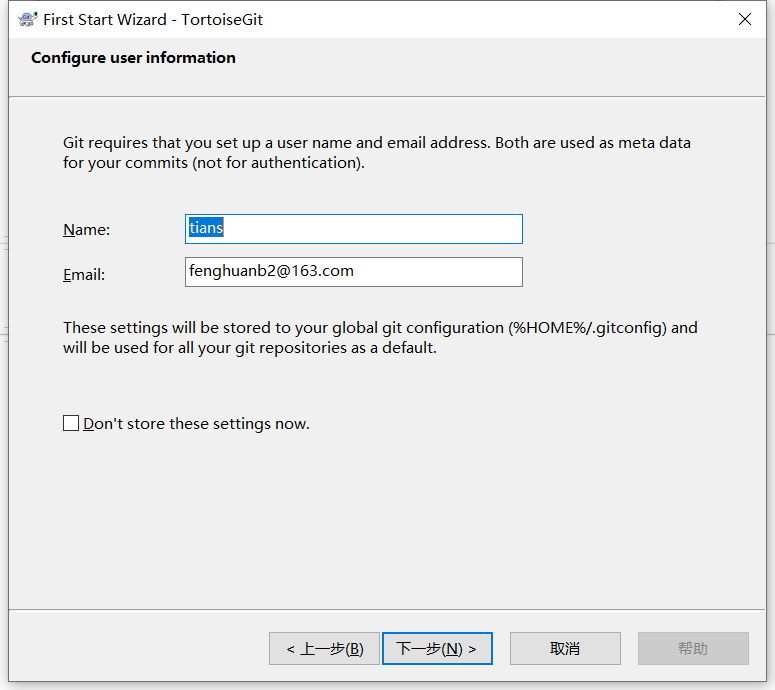


选择Git.exe的执行路径（如果默认选中了则无需更改）。选中后点击“Check now”，如果出现git版本则证明已成功。然后下一步。

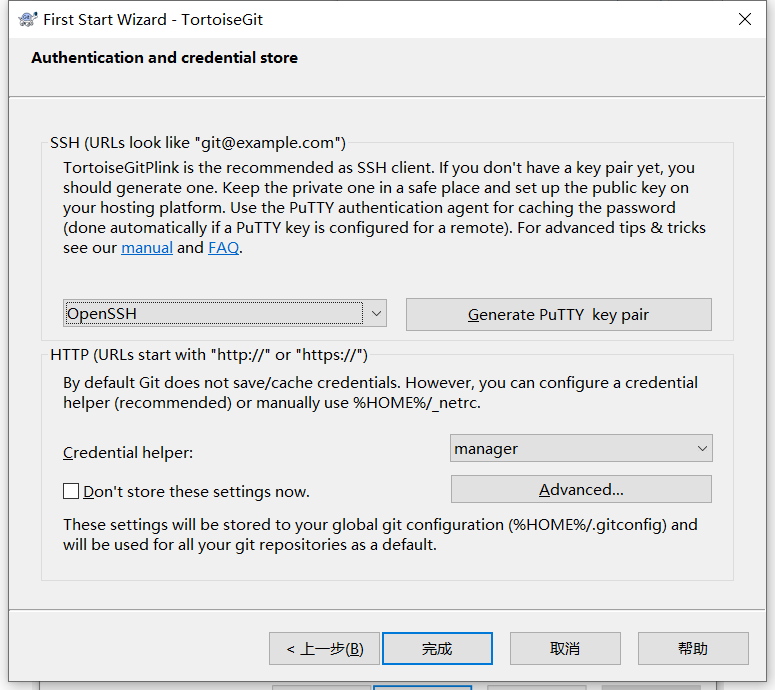




设置名称和邮箱。名称和邮箱将作为今后代码更改人的识别信息，设置完成下一步。

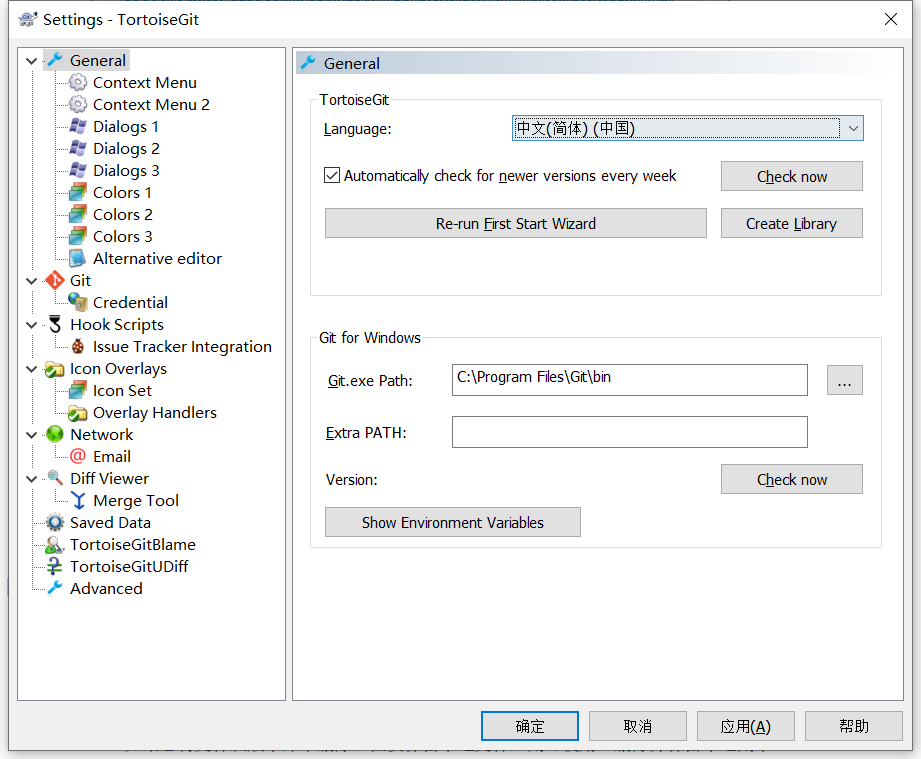


选择OpenSSH，点击“完成”按钮。



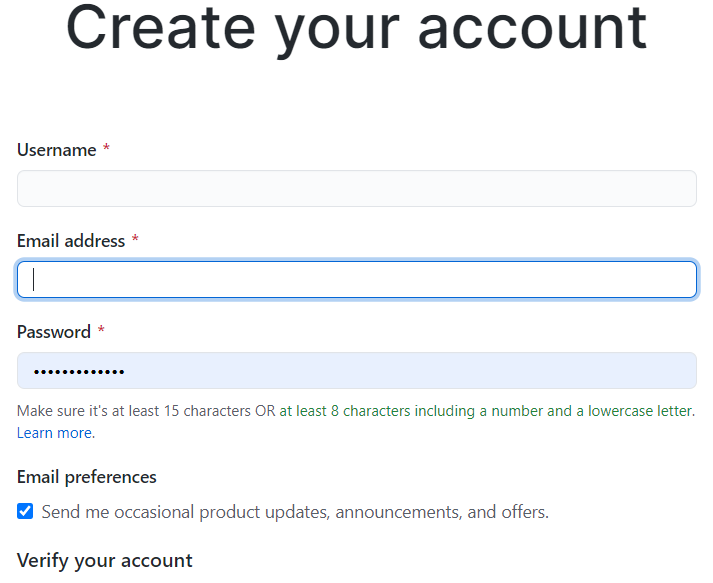
# 安装语言包

安装完成“TortoiseGit-LanguagePack-2.10.0.0-64bit-zh\_CN.msi”文件后，随便找个文件夹，右键-TortoiseGit-Settings。在设置窗口选中“General”，“Language”选择中文。



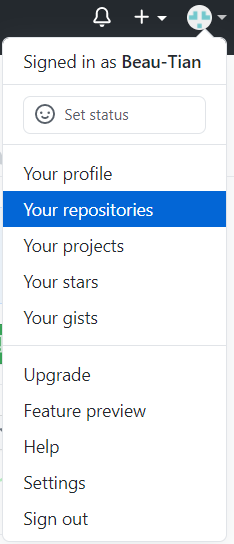
# 创建github账号

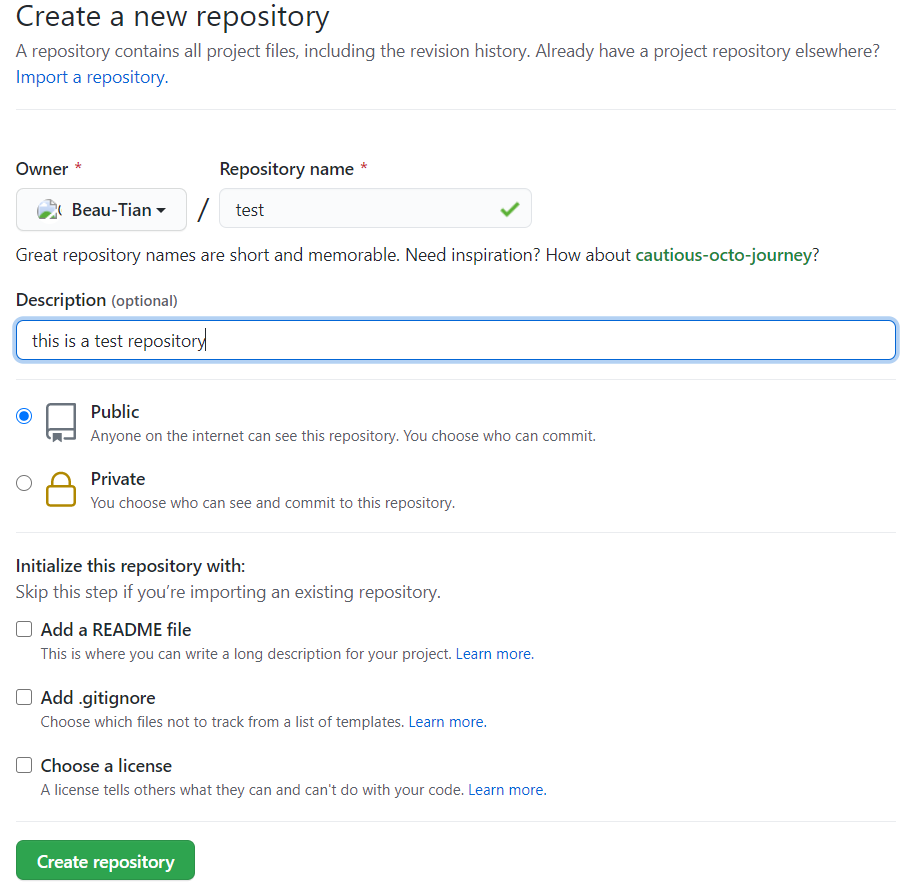
进入https://github.com/，点击“Sign up”，输入账号、邮箱、密码，然后下一步。后面的调查可答可不答，最后完成邮箱验证即可。



# 创建一个新的仓库

点击个人头像-Your repositories-New。Repository name是必填的，Description是可填的。public是公有仓库，任何人都可以访问，private是私有仓库，只有自己或者邀请加入的人才能访问。





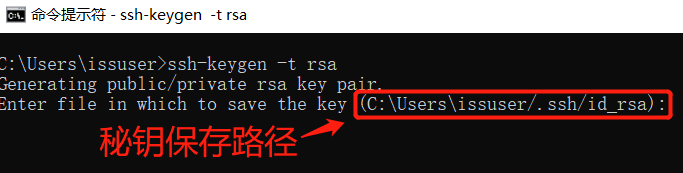
# 通过SSH连接Github

## SSH概述

SSH 为建立在应用层基础上的安全协议。SSH 是较可靠，专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。

## 生成RSA秘钥

cmd执行ssh-keygen -t rsa，全部默认选项，生成目录有两个文件，.pub文件结尾的是公钥。默认：C:\Users\用户\.ssh。



## github添加公钥

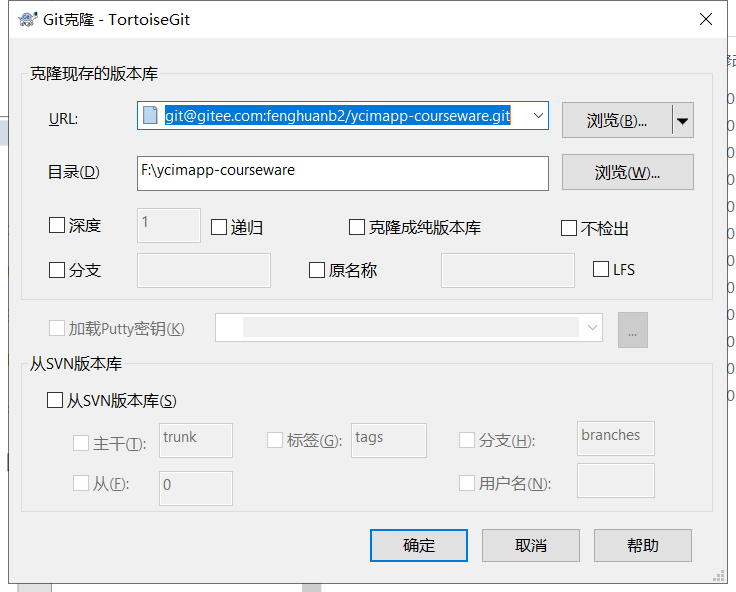
Github中点击头像-settings-SSH and GPG keys-New SSH key。输入Title，将.pub结尾的公钥文件内容复制到Key中，最后点击保存。



## 链接Github

Cmd/git bash：git clone ssh-url。

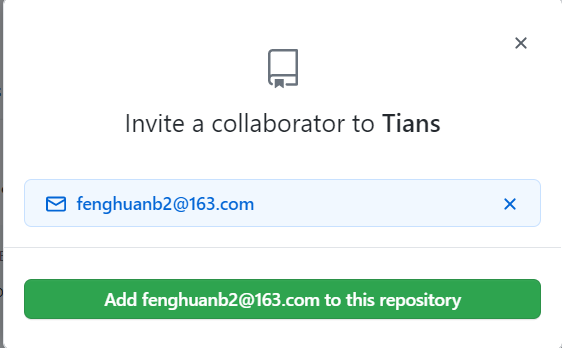
TortoiseGit：工作目录-右键-Git克隆-输入URL，然后确定即可。



# 邀请项目开发者

当我们需要和他人协同开发时，就需要他成为项目中的成员。

登录进入github-选择需要协同开发的仓库-Settings-Manage access-Invite a collaborator-输入被邀请人的账号/邮箱-选中-发送邀请-等待被邀请人接受邀请-可以向仓库推送文件。



# Git常规操作

## 初始化

初始化一个本地仓库。

Cmd/git bash：git init

TortoiseGit：工作目录-右键-Git在这里创建版本库-确定即可。

初始化会生成一个隐藏的.git文件夹，便是本地仓库。

## 添加

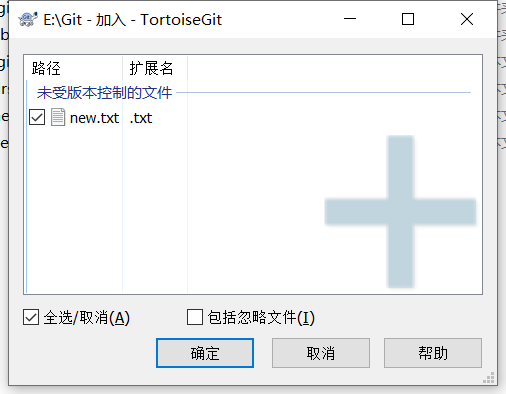
添加一个文件到暂存区。

Cmd/git bash：git add filename。

例：添加多个文件：git add 1.txt 2.txt

添加所有文件：git add .

TortoiseGit：工作目录-右键-TortoiseGit-添加-选中要添加的文件-确定。



## 提交

提交是将新添加/修改的文件从暂存区中提交到版本库中。

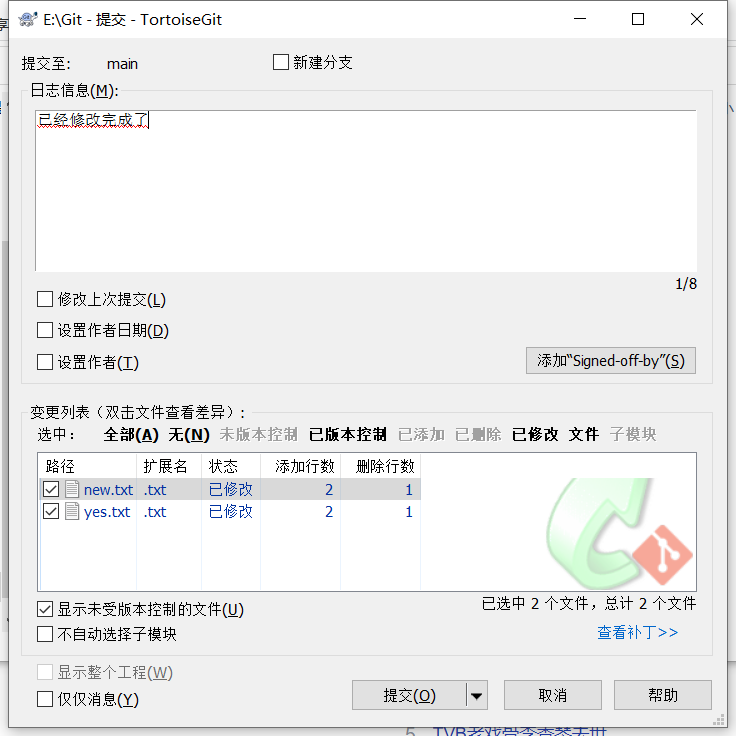
Cmd/git bash：git commit file -m message

file：要提交的文件，可为空，如果为多个文件，则替换为-a。

-m：输入提交日志，如果没有-m，执行后则会进入编辑器输入提交日志。

message：提交时的日志。

TortoiseGit：工作目录-右键-Git提交->master(分支名)-输入提交日志-提交。

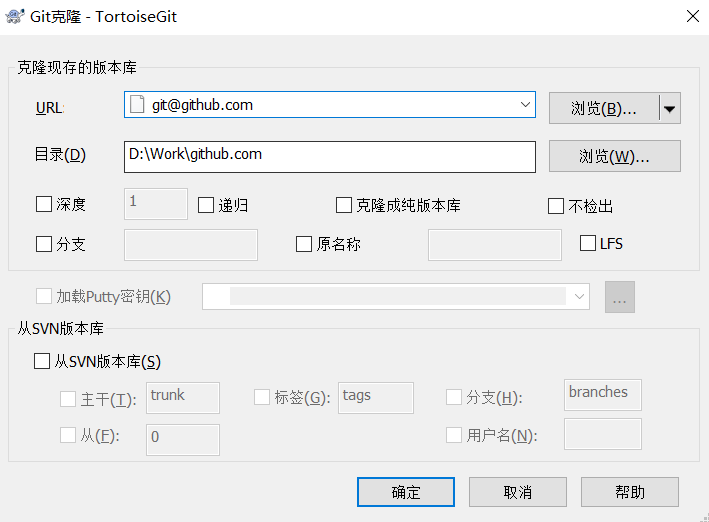


## 克隆

克隆是从远程服务器克隆一个一模一样的版本库到本地。

Cmd/git bash执行命令为：git clone url

TortoiseGit工作目录-右键-Git克隆-填写Url、选择文件夹路径，点击“确定”按钮即可。



## 拉取

拉取是从远程服务器上拉取指定分支，并和本地分支进行合并，相当于从服务器获取最新，并更新本地，让本地分支与服务器分支保持一致。

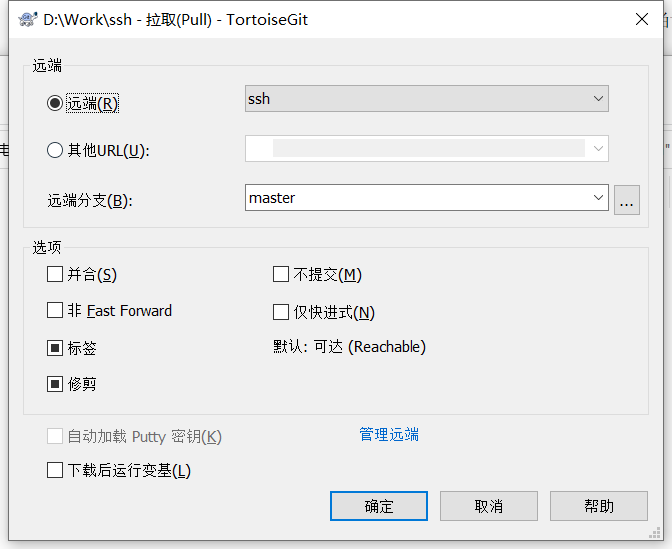
Cmd/git bash命令：git pull remote master:branch

remote：添加Github链接时设置的名称。

master：远程分支。

branch：要合并的本地分支，如果省略则表示有追踪关系（TortoiseGit分支查看器可查看）的分支或名称相同的分支。

TortoiseGit拉取：工作目录-右键-TortoiseGit-拉取-选择远端、远端分支，然后确定即可。



## 推送

推送是将本地分支推送到服务器，并和指定分支进行合并。

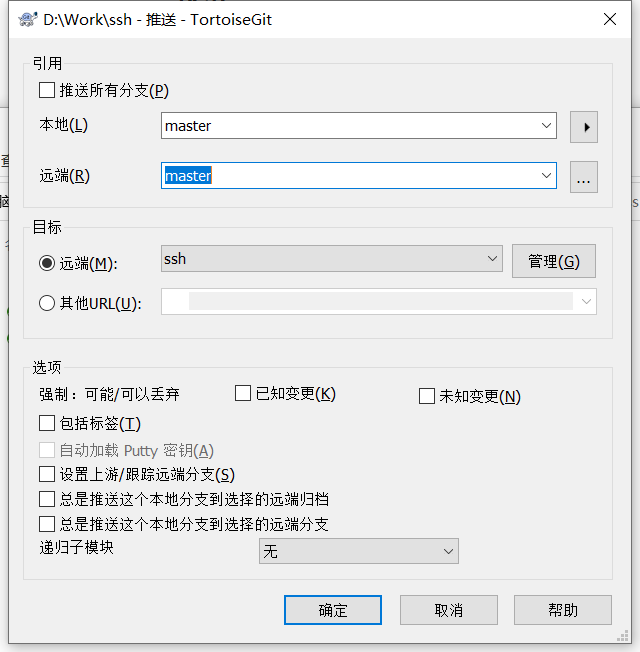
Cmd/git bash命令：git push remote master:branch

remote：添加Github链接时设置的名称。

master：本地分支。

branch：远程分支，如果省略则表示有追踪关系的分支或名称相同的分支。

TortoiseGit推送：工作目录-右键-TortoiseGit-推送-选择本地分支和远端分支，点击确定即可。



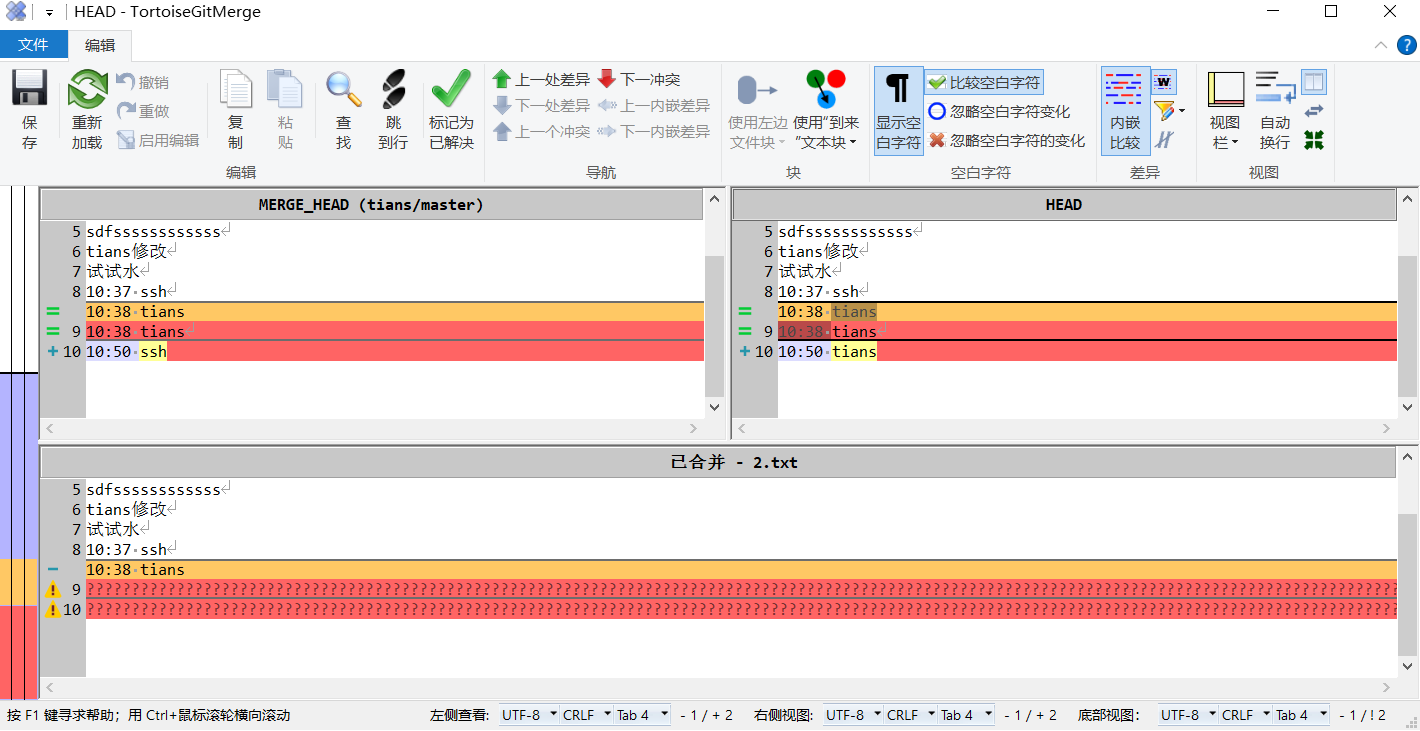
## 解决冲突

当A修改完new.txt文件并提交到远程仓库后，B也修改完了new.txt并提交到本地仓库，这时B从远程仓库中pull时，A和B的new.txt内容不一致会造成冲突，因为git不知道要用谁的，所以这时需要手动解决冲突。

**编辑冲突的方法：**

方法1：打开冲突文件直接编辑，编辑完成后保存。

方法2：选中冲突文件-右键-TortoiseGit-编辑冲突，编辑完成后保存，右键-TortoiseGit-解决冲突。

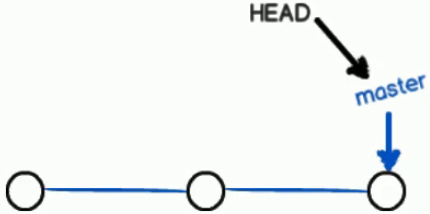


## 分支

为了便于理解，大家暂时可以认为分支就是当前工作目录中代码的一份副本。

我们每次提交时，Git都会把他们串成一条时间线，这条时间线就是分支，每个项目都有一个主分支，即“master”分支。可以从主分支创建出N个子分支，子分支意味着可以把你的工作从开发主线上分离开来。

head（头）指针指向master，master指向提交点，所以Head指向当前分支。

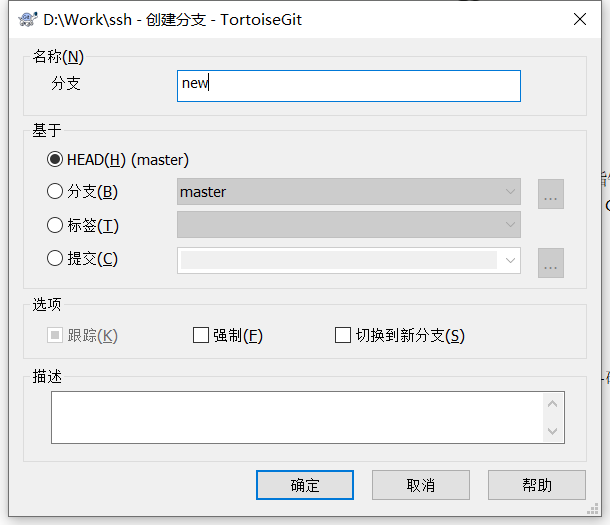


当我们创建新的分支时，如：new，Git会新建一个叫做new的指针，指向master相同的提交点，再把Head指向new，就表示当前分支在new上了。所以Git创建分支非常的快，因为只是改变指针指向。

### 创建分支

Cmd/git bash：git branch name

TortoiseGit：工作目录-右键-TortoiseGit-创建分支-输入分支名称-确定。



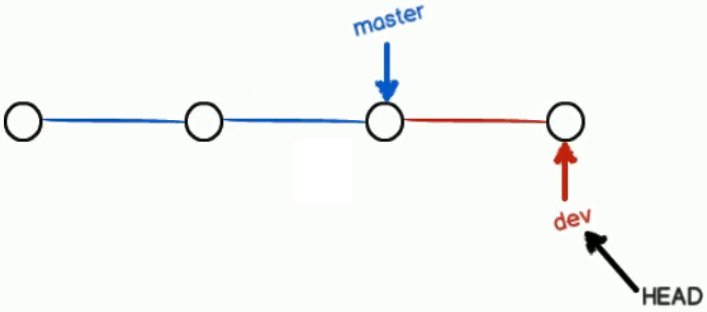
### 切换分支

Cmd/git bash：git checkout name

name：分支名称。

TortoiseGit：工作目录-右键-TortoiseGit-切换/检出。

切换分支，提交后，当前指针往前走一步，master指针不变。如：当前分支为dev，切换为dev分支后，提交后，dev指针往前走一步，master指针原地不动。



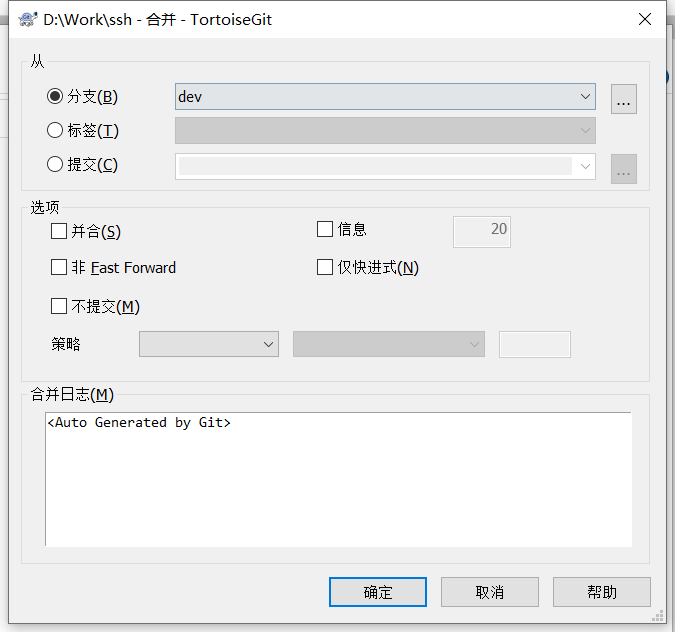
### 合并分支

分支与分支之间可随意合并。

Cmd/git bash：git merge branch

branch：要合并的分支。

TortoiseGit：工作目录-右键-TortoiseGit-合并-选择要合并的分支-确定。



### 删除分支

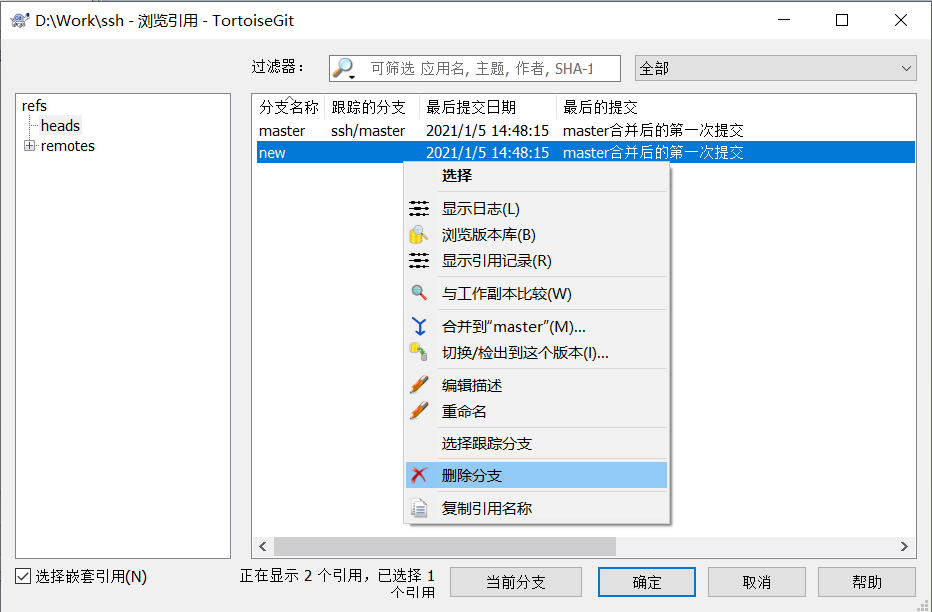
Cmd/git bash：git branch -d branch

branch：要删除的分支。

TortoiseGit：工作目录-TortoiseGit-切换/检出-点击分支选项后的“...”按钮。



然后选中要删除的分支，右键-删除分支。



# Git其他指令

1. 查看日志：git log [filename]
2. 查看状态：git status [-s]
3. 从暂存区将文件移除：git rm --cache filename
4. 重置状态：git reset [filename]
5. 恢复：git restore [filename]

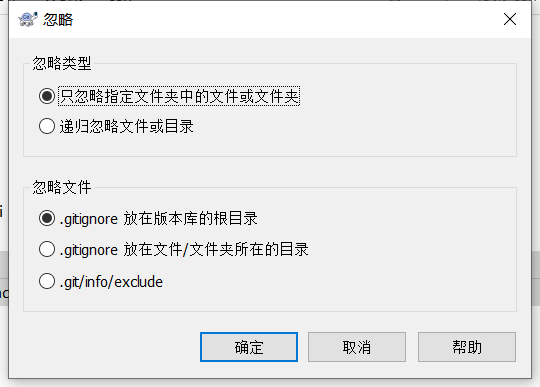
# Git忽略清单

将不需要被git管理的文件名字添加到此文件中，在执行git命令的时候，git就会忽略这些文件，文件名叫：.gitignore。

TortoiseGit：选中要修改的文件-TortoiseGit-删除并添加到忽略列表-根据名称删除和忽略x项。



选择忽略规则，点击确定，.gitignore文件将自动生成。



## **忽略清单语法规则**

* ! 开头的模式标识否定，该文件将会再次被包含，****如果排除了该文件的父级目录，则使用 ! 也不会再次被包含****。可以使用反斜杠进行转义
* / 结束的模式只匹配文件夹以及在该文件夹路径下的内容，但是不匹配该文件
* / 开始的模式匹配项目跟目录
* 如果一个模式不包含斜杠，则它匹配相对于当前 .gitignore 文件路径的内容，如果该模式不在 .gitignore 文件中，则相对于项目根目录
* \*\* 匹配多级目录，可在开始，中间，结束
* ? 通用匹配单个字符
* [] 通用匹配单个字符列表

## 常用匹配示例

* bin/: 忽略当前路径下的bin文件夹，该文件夹下的所有内容都会被忽略，不忽略 bin 文件
* /bin: 忽略根目录下的bin文件
* /\*.c: 忽略 cat.c，不忽略 build/cat.c
* debug/\*.obj: 忽略 debug/io.obj，不忽略 debug/common/io.obj 和 tools/debug/io.obj
* \*\*/foo: 忽略/foo, a/foo, a/b/foo等
* a/\*\*/b: 忽略a/b, a/x/b, a/x/y/b等
* !/bin/run.sh: 不忽略 bin 目录下的 run.sh 文件
* \*.log: 忽略所有 .log 文件
* config.php: 忽略当前路径的 config.php 文件