

# 尚硅谷技术课程系列之 Git



尚硅谷 JavaEE 教研组

# 第1章 Git 快速入门

## 1.1 Git 概述

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, **convenient staging areas**, and **multiple workflows**.

Git 是一个免费的，开源的分布式版本控制系统，可以快速高效地处理从小型或大型的各种项目。Git 易于学习，占用空间小，性能快得惊人。

## 1.2 SCM 概述

SCM (Software Configuration Management, 软件配置管理) 是一种标识、组织和控制修改的技术。它应用于整个软件生存周期。

作为评价一个大中型软件开发过程是否正确，合理，有效的重要手段，CMM(Capability Maturity Model )能力成熟度模型提供了不同等级的标准流程，对软件开发过程（流程）进行了约束和建议，而作为 CMM 2 级的一个关键域（Key Practice Area, KPA），SCM 软件在整个软件的开发活动中占有很重要的位置。

Git 软件比 Subversion、CVS、Perforce 和 ClearCase 等 SCM (Software Configuration Management 软件配置管理) 工具具有性价比更高的本地分支、方便的暂存区域和多个工作流等功能。

## 1.3 Git 安装

### 1.3.1 软件下载

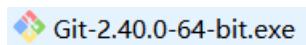
软件官网地址为：<https://git-scm.com/>

软件下载地址为：

<https://github.com/git-for-windows/git/releases/download/v2.40.0.windows.1/Git-2.40.0-64-bit.exe>

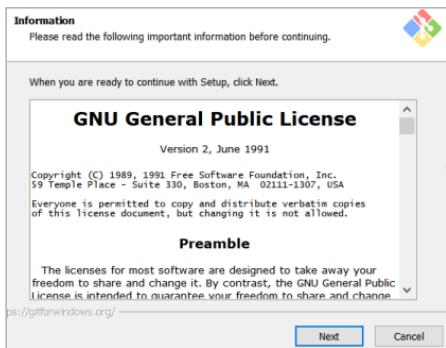


最早 Git 是在 Linux 上开发的，很长一段时间内，Git 也只能在 Linux 和 Unix 系统上跑。不过，慢慢地有人把它移植到了 Windows 上。现在，Git 可以在 Linux、Unix、Mac 和 Windows 这几大平台上正常运行了。由于开发机大多数情况都是 windows，所以本教程选择相对简单的 Windows 系统软件版本进行下载，此处我们下载 Windows 系统的 2.40.0 版本软件

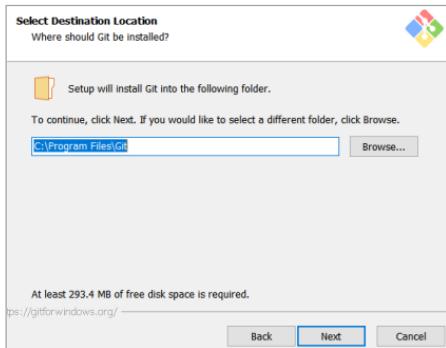


### 1.3.2 软件安装

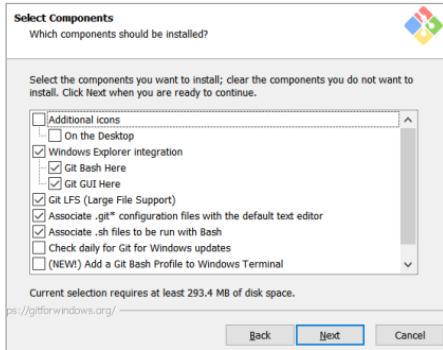
Git 软件安装的步骤虽然比较多，但是整个安装过程还是比较简单的，双击 exe 执行文件，按照引导界面的提示安装即可



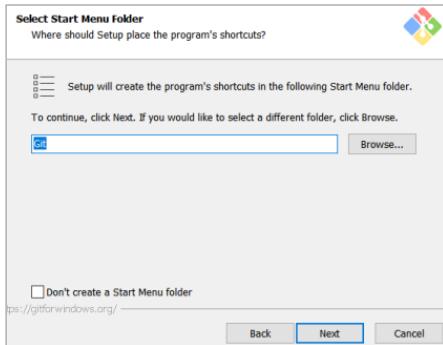
点击 Next 下一步，选择安装目录，默认安装在 c 盘中



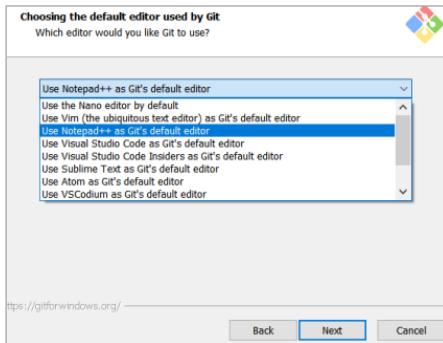
点击 Next 下一步，选择组件，此处默认即可



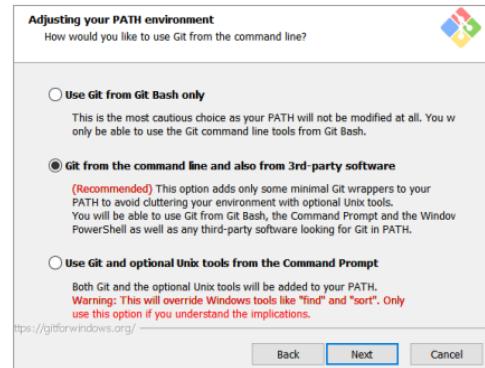
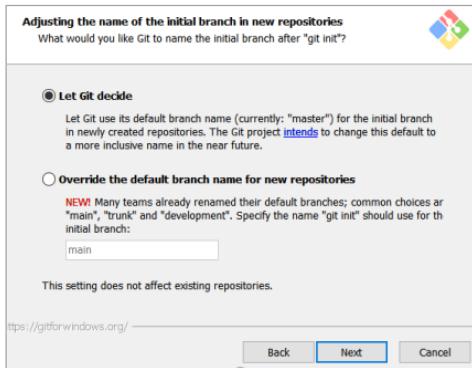
点击 Next 下一步，配置启动菜单，此处默认即可

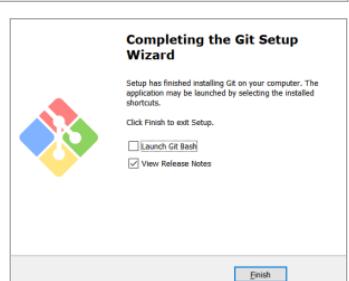
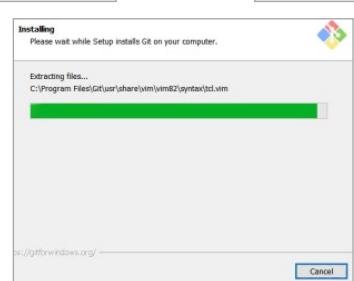
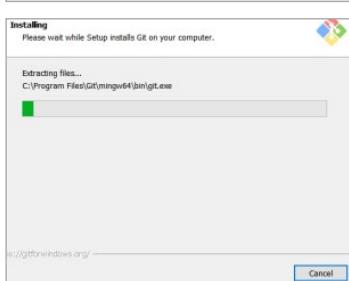
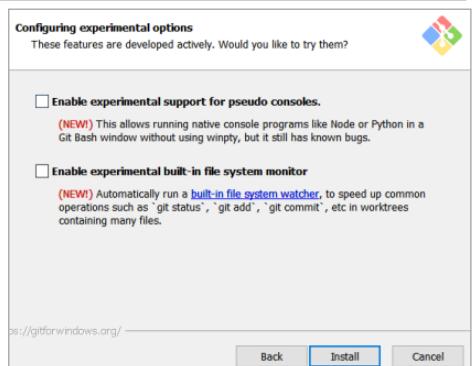
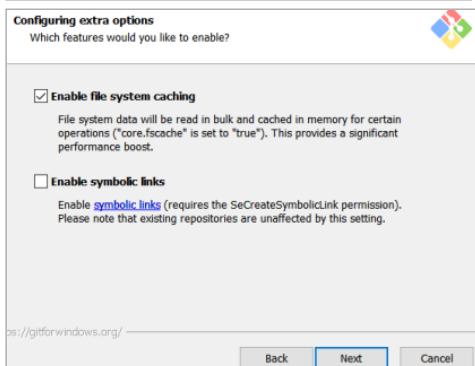
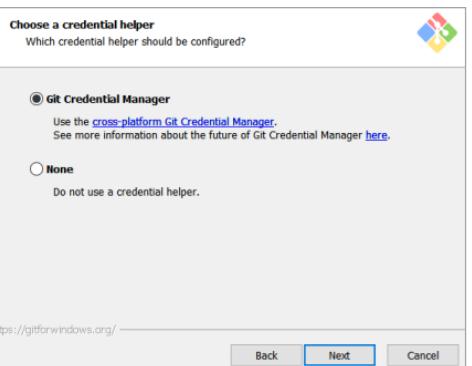
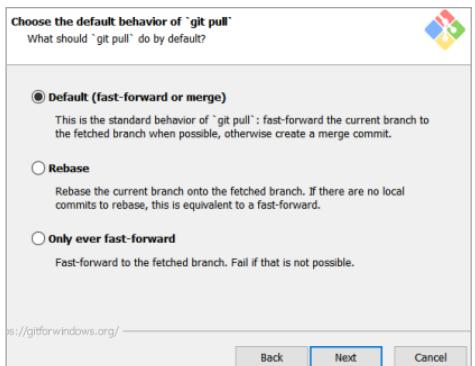
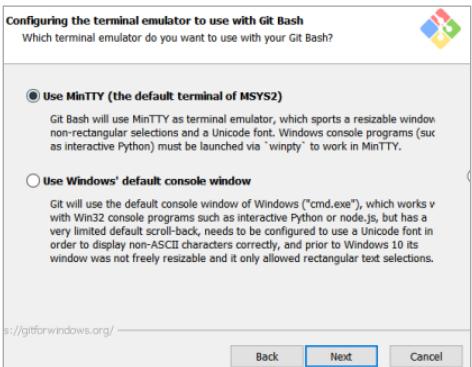
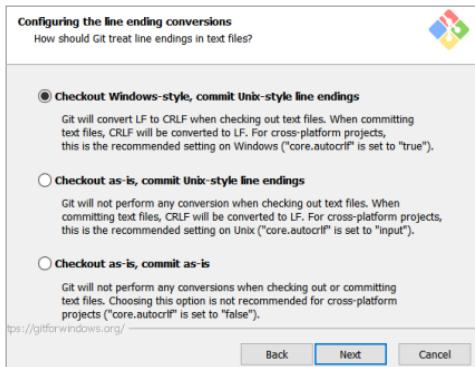
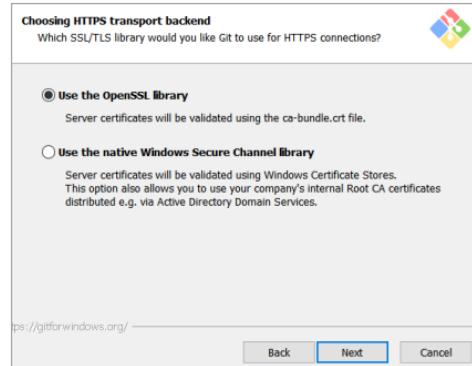
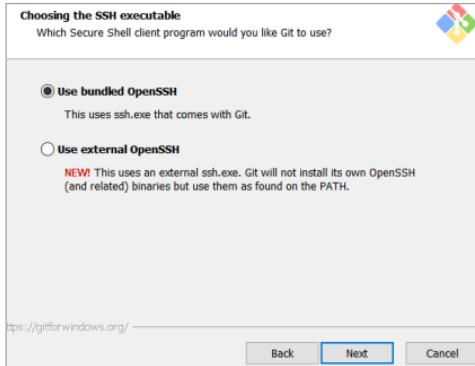


点击 Next 下一步，配置编辑器工具，这里选择自己习惯的编辑工具即可。



后续就不需要进行什么特殊配置了，所以连续默认选择 Next 下一步即可。





### 1.3.3 软件测试

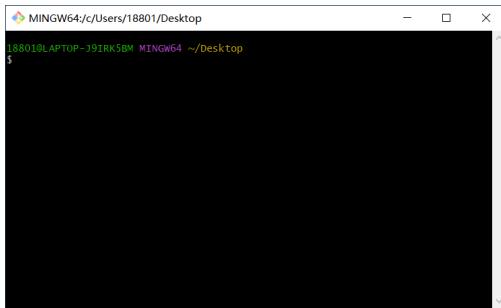
在 Windows 桌面空白处，点击鼠标右键，弹出右键菜单



Git 软件安装后，会在右键菜单中增加两个菜单

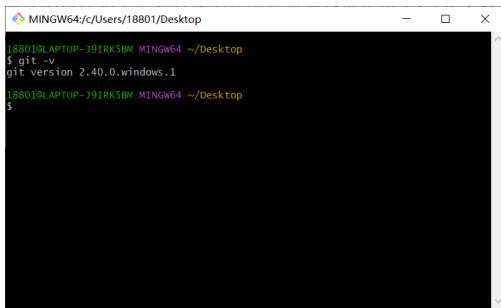
- Git GUI Here
- Git Bash Here

此处仅仅是为了验证 Git 软件安装的效果，所以选择 Git Bash Here 菜单，选择后，Windows 系统弹出 Git 软件的命令行黑窗口，



窗口弹出后，可以输入 Git 软件的操作指令。此时我们使用键盘输入操作指令：git -v 或 git --version，查看当前 Git 软件的安装版本。

```
git -v  
git --version
```



输入指令回车后，如果黑窗口中打印出咱们安装的软件版本 2.40.0，Git 软件安装成功了。

## 第2章 Git 基础使用

### 2.1 Git 基础概念

Git 是一个免费的，开源的分布式版本控制软件系统，学习 Git 软件的具体操作前，我们需要对一些基础的概念和名词进行解释

#### 2.1.1 版本控制

一般情况下，一份文件，无论是 DOC 办公文档，还是编程源码文件，我们都会对文件进行大量的修改和变更。但是我们无法保证每一次的修改和变更都是正确并有效的，往往有的时候需要追溯历史操作，而版本控制（Revision control）是一种在开发的过程中用于管理我们对文件、目录或工程等内容的修改历史，方便查看更改历史记录，备份以便恢复以前的版本的软件工程技术。

没有进行版本控制或者版本控制本身缺乏正确的流程管理，在软件开发过程中将会引入很多问题，如软件代码的一致性、软件内容的冗余、软件过程的事物性、软件开发过程中的并发性、软件源代码的安全性，以及软件的整合等问题。

#### 2.1.2 分布式

在 Git 中，每个版本库都是一样重要的。所以就不存在像集中式版本控制软件中以谁为主得问题。任何一个库都可以当成主库。

这种方式可以更大限度地保证项目资源得安全。

#### 2.1.3 系统

一般软件系统指的是可以独立运行的软件应用程序。而 Git 软件就是专门用于对代码文件进行版本控制得应用程序。同时也提供客户端对系统所管理得资源进行访问。

#### 2.1.4 区域

Git 软件为了更方便地对文件进行版本控制，根据功能得不同划分了三个区域



- 存储区域：Git 软件用于存储资源得区域。一般指得就是.git 文件夹
- 工作区域：Git 软件对外提供资源得区域，此区域可人工对资源进行处理。

- 暂存区：Git 用于比对存储区域和工作区域得区域。Git 根据对比得结果，可以对不同状态得文件执行操作。

## 2.2 Git 基础指令

Git 软件是免费、开源的。最初 Git 软件是为辅助 Linux 内核开发的一套软件，所以在使用时，简单常用的 linux 系统操作指令是可以直接使用的

### 2.2.1 linux 系统操作指令

指令	含义	说明
cd 目录	change directory	改变操作目录
cd ..		退回到上一级目录
pwd	Print work directory	打印工作目录
ls	list directory contents	显示当前目录的文件及子文件目录
ll	ls -l 简化版本	更详细地显示当前目录的文件及子文件目录
mkdir 文件夹名称	make directory	新建一个文件夹
rm 文件	remove	删除文件
rm -r 文件夹	Remove	删除文件目录
touch 文件		如果创建的文件不存在，那么创建一个空文件
reset		清屏
clear		清屏
exit		退出终端窗口

### 2.2.2 Git 软件指令

#### 2.2.2.1 配置信息

作为一个工具软件来讲，一般都会有默认的配置文件来保存基础的配置信息，Git 软件的配置文件位置为：[Git 安装路径/etc/gitconfig](#)

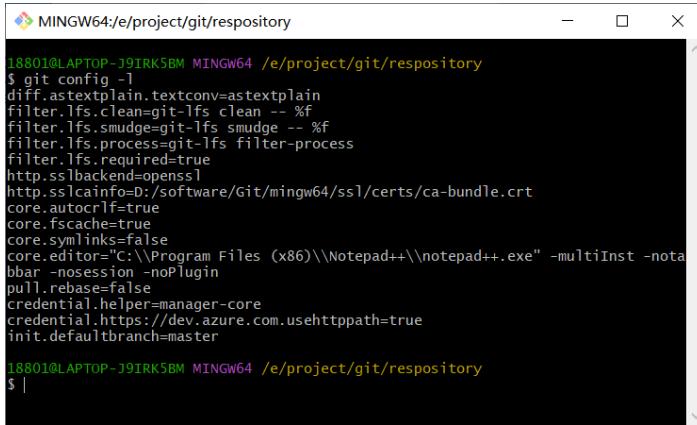
```

1 [diff "astextplain"]
2   textconv = astextplain
3 [filter "lfs"]
4   clean = git-lfs clean -- %f
5   smudge = git-lfs smudge -- %f
6   process = git-lfs filter-process
7   required = true
8 [http]
9   sslBackend = openssl
10  sslCAInfo = D:/software/Git/mingw64/ssl/certs/ca-bundle.crt
11 [core]
12  autocrlf = true
13  fscache = true
14  symlinks = false
15  editor = 'C:\\\\Program Files (x86)\\\\\\Notepad++\\\\notepad++.exe' -multiInst -notabbar -nosession -noPlugin
16 [pull]
17  rebase = false
18 [credential]
19  helper = manager-core
20 [credential "https://dev.azure.com"]
21  useHttpPath = true
22 [init]
23  defaultBranch = master
24

```

默认情况下，我们可以通过指令获取软件的配置信息：

```
git config -l
```



```
MINGW64:/e/project/git/repository
$ git config -l
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=D:/software/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.editor="C:\\Program Files (x86)\\Notepad++\\notepad++.exe" -multiInst -nota
bbar -noSession -noPlugin
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/repository
$ |
```

### 2.2.2.2 名称和邮箱

如果你是第一回使用 Git 软件，需要告诉 Git 软件你的名称和邮箱，否则是无法将文件纳入到版本库中进行版本管理的。这是因为在多人协作时，不同的用户可能对同一个文件进行操作，所以 Git 软件必须区分不同用户的操作，区分的方式就是名称和邮箱。

当然了，你可能会说我就用本地库就行了，不需要进行多人协作，是不是就可以不用配置呢。这是不行的，因为 Git 软件的设计初衷本身就是针对于 linux 系统的分布式开发协同工作，所以它天生就是用于分布式协同工作的，这里无论你是否使用这个功能，它本身就是这么设计的。所以是一定要配置的，否则就会出现如下提示：

```
Author identity unknown
*** Please tell me who you are.

Run

git config --global user.name "Your Name"
git config --global user.email "you@example.com"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got '18801@LAPTOP-J9IRK5BM.(none)')
```

当然了，配置的过程并不复杂，输入相关指令即可

```
git config --global user.name test
git config --global user.email test@atguigu.com
```

这里的--global 表示全局配置，后续的所有文件操作都会使用该用户名及邮箱。此时在操作系统的用户目录，会产生新的配置文件

 .gitconfig 2022/12/9 18:12 GITCONFIG 文件 1 KB

文件中就包含了刚刚增加的配置信息

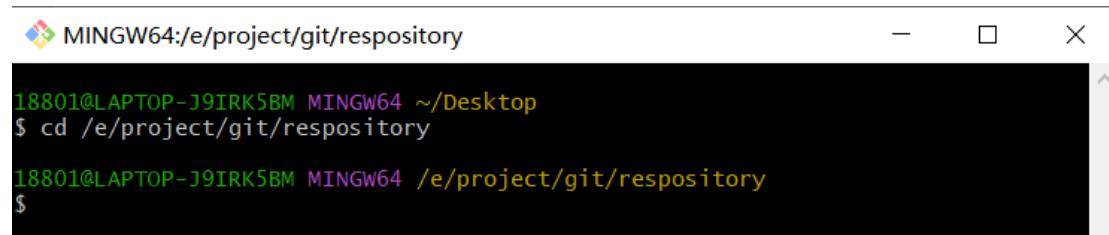
```
1 [user]
2   name = test
3   email = test@atguigu.com
4
```

### 2.2.2.3 初始化版本库

Git 软件主要用于管理文件的版本信息，但它只是一个软件，不可能安装后就直接将系统中所有的文件全部纳入到它的管理范畴中。并且，软件管理版本信息的主要目就是管理文件的修改和变更，如果将系统中所有文件都进行管理其实意义是不大的。所以一般情况下，我们需要指定某一个文件目录作为软件的管理目录。因为这个目录主要就作为 Git 软件的管理文件的版本变化信息，所以这个目录也称之为 Git 软件的版本仓库目录。

具体操作过程如下：

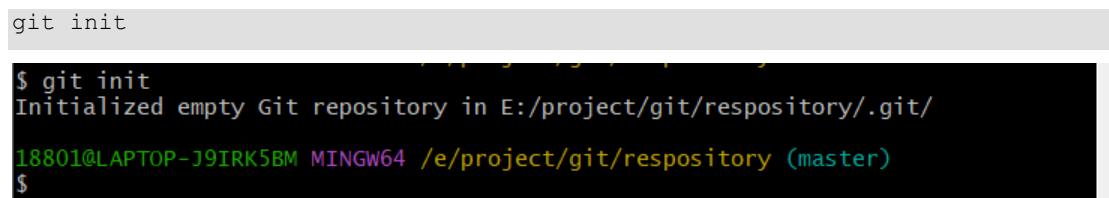
- 我们首先通过指令进入到指定文件目录



```
MINGW64:/e/project/git/respository
18801@LAPTOP-J9IRK5BM MINGW64 ~/Desktop
$ cd /e/project/git/respository

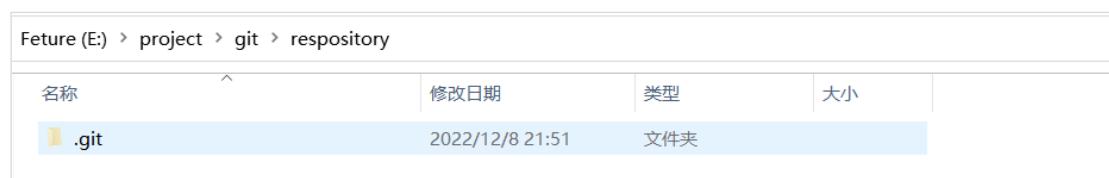
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository
$
```

- 执行指定的指令，创建文件版本库



```
git init
$ git init
Initialized empty Git repository in E:/project/git/respository/.git/
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$
```

- 版本库创建成功后，会在目录中创建.git 目录，用于管理当前版本库。



### 2.2.2.4 向版本库中添加文件

虽然创建了版本库，但是现在版本库中还没有任何的文件，所以这里我们先手动创建文件：test.txt



因为文件已经放置在版本库中了。所以可以通过软件的指令查看版本库状态



```
git status
```

```
MINGW64:/e/project/git/respository
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test.txt
nothing added to commit but untracked files present (use "git add" to track)
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$
```

此时会发现，test.txt文件属于 **untracked files**（未追踪文件），这里表示当前的test.txt文件虽然放置到了版本库的文件目录中，被Git软件识别到了，但是未纳入到版本库管理中。所以属于未追踪文件。通过这个现象可以认为，系统文件夹物理目录和版本库管理目录的含义是不一样的。只有文件被纳入到版本库管理后，Git软件才能对文件修改后的不同版本内容进行追踪处理，也就是所谓的 **tracked files** 了。那么如何将文件纳入到版本库的管理呢，这就需要我们执行以下命令了：

```
#这里的文件是需要提供扩展名的
git add test.txt
```

```
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test.txt
nothing added to commit but untracked files present (use "git add" to track)
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git add test.txt
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$
```

此时你再查看版本库状态，就会发现文件状态的变化。

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git add test.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   test.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
```

你会发现，此时文件状态为 **cached file**，这是什么意思呢？其实这也是Git管理文件时的一种状态：**暂存状态**。就是我们生活中常说的草稿状态。也就是说对于Git来讲，它认为此时

文件只是一种临时草稿状态，随时可能会进行修改或删除，并不算真正的操作完成状态。所以并不会把文件纳入到版本库管理中。

为什么会这样呢？其实这就涉及到版本的作用。生活中，我们学习时，一般会写学习笔记，虽然写完后不一定会看，但是该写的还是要写的。然后给这些笔记文件起名时，一般就会带着当天的时间或数字。比如【Java 学习笔记\_20220101.md】，或者【Java 学习笔记\_Ver1.1.md】，这里的时间或数字主要作用就是用于区分同一份笔记在不同时间节点记录的内容，这里的数字或时间我们就称之为版本。

那如果你只是随便写写，或写到一半，还没有写完的话，会专门给文件改个名称吗？应该不会，对不对，因为对于你来讲，这个笔记文件并没有记录完成，对吗。但是你非得说，你今天不想继续学习了，然后给文件改了一个名称，也不是不可以。对于 Git 软件来讲，道理是一样的。如果确定要把文件放置在版本库中，那么就需要执行确定提交指令

```
# commit 表示真正地纳入到版本库中
# -m 表示提交时的信息 (message)，是必须输入的。用于描述不同版本之间的差别信息
git commit -m "my first git file"
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git commit
Aborting commit due to empty commit message.

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git commit -m "my first git file"
[master (root-commit) 1c3aa0e] my first git file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
```

再查看 Git 状态

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git status
On branch master
nothing to commit, working tree clean

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$
```

提交后，Git 会对当前的操作进行 Hash 计算，通过计算后的值将数据保存下来，保存的位置为版本库.git 文件目录的 objects 中，我们可以通过指令查看当前提交

```
git show
```

```

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git show
commit 1c3aa0e5d65bd7d6eda82094c984f5dbd946fd66 (HEAD -> master)
Author: test <test@atguigu.com>
Date:   Fri Dec 9 19:51:04 2022 +0800

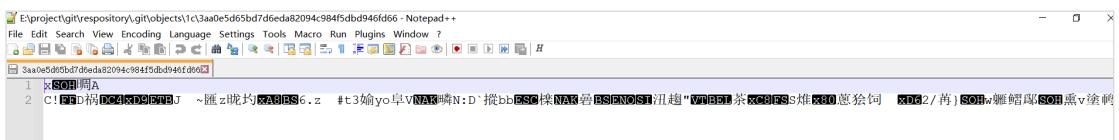
    my first git file

diff --git a/test.txt b/test.txt
new file mode 100644
index 0000000..e69de29

```

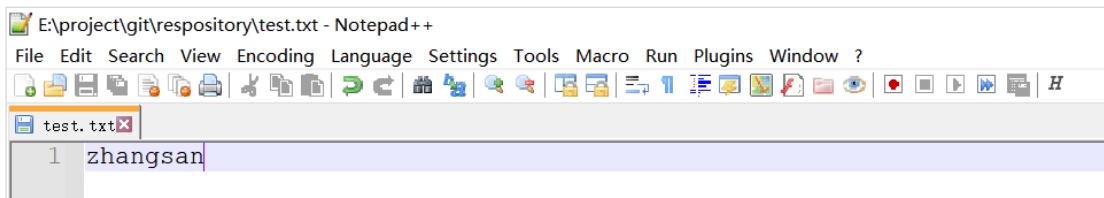
Structure (E) > project > git > repository > .git > objects > 1c				
名称	修改日期	类型	大小	
3aa0e5d65bd7d6eda82094c984f5dbd946fd66	2022/12/9 19:51	文件	1 KB	

由于文件内容进行了转换处理，直接打开你是看不懂的



## 2.2.2.5 修改版本库文件

现在文件已经被纳入到版本库中，因为咱们的文件是空的，所以这里我们增加一些内容



此时，Git 版本库中的文件和本地的文件就有了不同。我们可以查看状态

```

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   test.txt

no changes added to commit (use "git add" and/or "git commit -a")

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ 

```

**modified** 表示文件已经修改了，我们可以把这一次的修改提交到版本库中

```

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git commit -a -m "update file"
[master f2f113f] update file
 1 file changed, 1 insertion(+)

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ 

```

原则上来讲，这里的操作顺序依然应该是

```
# 先增加，再提交  
git add test.txt  
git commit
```

但是这里我们简化了一下操作

```
git commit -a -m "update file"
```

这个指令操作中多了一个-a 的参数，等同于将增加，提交两步操作融合成了一步。

提交成功后，我们来展示以下当前 Git 软件库

```
$ git show  
commit f2f113f0f794f7409d4ee06cec0247fa2bfabc4a (HEAD -> master)  
Author: test <test@atguigu.com>  
Date:   Fri Dec 9 22:17:33 2022 +0800  
  
        update file  
  
diff --git a/test.txt b/test.txt  
index e69de29..45887f0 100644  
--- a/test.txt  
+++ b/test.txt  
@@ -0,0 +1 @@  
+zhangsan  
\ No newline at end of file  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
```

## 2.2.2.6 查看版本库文件历史

版本库中的文件我们已经修改并提交了，那么文件的版本信息就会发生变化，那我们如何来查看这个变化呢？这里我们可以采用 log 指令进行查看

```
git log
```

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)  
$ git log  
commit f2f113f0f794f7409d4ee06cec0247fa2bfabc4a (HEAD -> master)  
Author: test <test@atguigu.com>  
Date:   Fri Dec 9 22:17:33 2022 +0800  
  
        update file  
  
commit 1c3aa0e5d65bd7d6eda82094c984f5dbd946fd66  
Author: test <test@atguigu.com>  
Date:   Fri Dec 9 19:51:04 2022 +0800  
  
        my first git file  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
```

如果感觉看着不舒服，也可以美化一下显示方式：

```
git log --pretty=oneline
```

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)  
$ git log --pretty=oneline  
f2f113f0f794f7409d4ee06cec0247fa2bfabc4a (HEAD -> master) update file  
1c3aa0e5d65bd7d6eda82094c984f5dbd946fd66 my first git file
```

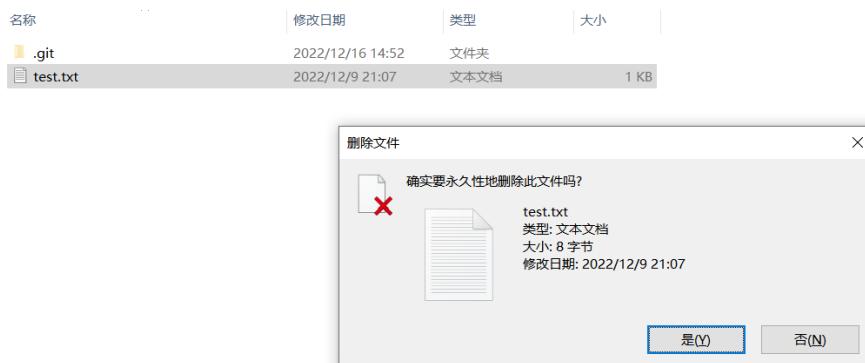
也可以使用简单方式查看

```
git log --oneline  
f2f113f update file  
1c3aa0e my first git file
```

### 2.2.2.7 删除文件

一般情况下，Git 软件就是用于管理文件的版本变更，但是在一些特殊的场景中，文件可能作废或不再使用，那么就需要从版本库中删除，记住，这里说的并不是从物理文件目录中删除，而是从版本库中删除。

#### ➤ 将本地文件从目录中删除



#### ➤ 查看 Git 版本库状态信息

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)  
$ git status  
On branch master  
Changes not staged for commit:  
  (use "git add/rm <file>..." to update what will be committed)  
    (use "git restore <file>..." to discard changes in working directory)  
      deleted:   test.txt  
  
no changes added to commit (use "git add" and/or "git commit -a")  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)  
$ |
```

此时 Git 软件会识别出来，版本库中有一份文件和当前用于临时操作文件的暂存区内的文件状态不一致：版本库中文件还在，但是操作区内的文件已经没有了。所以软件提供了两个选择：一个是将版本库中的文件也进行（提交）删除操作。另外一个就是从版本库中恢复文件。

#### ➤ 使用指令从版本库中恢复文件

```
git restore test.txt  
  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)  
$ git restore test.txt  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)  
$ |
```

Feture (E) > project > git > repository				
名称	修改日期	类型	大小	
.git	2022/12/17 8:04	文件夹		
test.txt	2022/12/17 8:04	文本文档	1 KB	

➤ 如果想要真正删除文件，那么也要将版本库中同时删除

Feture (E) > project > git > repository				
名称	修改日期	类型	大小	
.git	2022/12/17 8:04	文件夹		
test.txt	2022/12/17 8:04	文本文档	1 KB	

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      deleted:    test.txt

no changes added to commit (use "git add" and/or "git commit -a")

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git commit -a -m 'delete test.txt'
[master 72feb47] delete test.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 test.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ |
```

此时查看 Git 日志

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git log --pretty=oneline
72feb4743a73b981114ca10c5db1fa3cc9593776 (HEAD -> master) delete test.txt
f2f113f0f794f7409d4ee06cec0247fa2bfabc4a update file
1c3aa0e5d65bd7d6eda82094c984f5bdb946fd66 my first git file

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ |
```

### 2.2.2.8 恢复历史文件

如果版本库中一份文件中已经被删除了，那么此时这份文件还能找回来吗？其实原则上来讲，已经不行了，因为文件删除本身也是一种变更操作，也算是版本库管理的一部分。所以想要将已经删除的那份文件从版本库中取出来，已经是不可能了。但是，要注意的是，版本库管理的是文件不同版本的变更操作，这个不同版本的概念还是非常重要的。也就是说，最后的那个删除的文件版本已经没有了，但是之前版本的文件其实还是存在的。所以如果我们

能将文件恢复到某一个版本，那么那个版本的文件就依然存在。

➤ 查看版本库信息

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git log --oneline
72feb47 (HEAD -> master) delete test.txt
f2f113f update file
1c3aa0e my first git file

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$
```

➤ 将版本库文件重置到某一个版本

```
# 这里的 f2f113f 就是版本 Hash 值，用于唯一确定版本库中此版本的标记
# 当然了这是一个简短版，完整的比较长
# 如果不记得具体的版本值，版本值也可以使用 HEAD 值，比如最新的上一个版本：HEAD^
# 如果后退更多的版本，可以使用 HEAD~N
git reset --hard f2f113f

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git reset --hard f2f113f
HEAD is now at f2f113f update file

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$
```

➤ 被删除的文件回来了

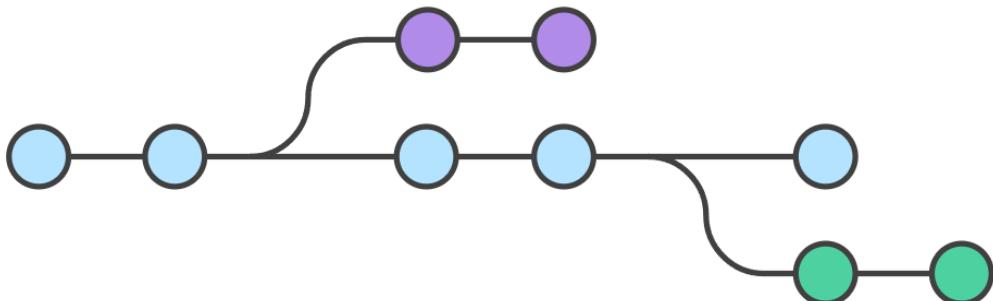
名称	修改日期	类型	大小
.git	2022/12/17 16:54	文件夹	
test.txt	2022/12/17 16:54	文本文档	1 KB

## 第3章 Git 进阶使用

在之前的操作中，所有的操作都是基于一条主线完成的。就好比，咱们学习的时候，记学习笔记，今天学点，那么就写一点，明天学点，再写一点，最后，完全学完了，这个笔记也就记全了。但实际上，有些文件可能在不同的场合需要同时使用不同的内容，而且还不能冲突，比如项目的配置文件，我需要本地进行测试，同时还要部署到服务器上进行测试。本地和服务器上的环境是不一样的，所以同一个配置文件就需要根据环境的不同，进行不同的修改。本地环境没问题了，修改配置文件，提交到服务器上进行测试，如果测试有问题，再修改为本地环境，重新测试，没问题了，再修改为服务器配置，然后提交到服务器上进行测试。依次类推，形成迭代式开发测试。

从上面的描述上看，就会显得非常繁琐，而且本质上并没有太重要的内容，仅仅是因为环境上的变化，就需要重新修改，所以如果将本地测试环境和服务器测试环境区分开，分别进行文件版本维护，是不是就会显得更合理一些。这个操作，在 Git 软件中，我们称之为 branch，分支。

这里的分支感觉上就是树上的分叉一样，会按照不同的路线生长下去。有可能以后不再相交，当然，也可能以后会不断地纠缠下去，都是有可能的。



### 3.1 Git 分支

#### 3.1.1 主干分支

默认情况下，Git 软件就存在分支的概念，而且就是一个分支，称之为 master 分支，也称之为干线分支。

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
```

这就意味着，所有文件的版本管理操作都是在 master 这一个分支路线上进行完成的。

不过奇怪的是，为什么之前的操作没有体现这个概念呢，那是因为，默认的所有操作本身就

都是基于 master 分支完成的。而 master 主干分支在创建版本库时，也就是 git init 时默认就会创建。

### 3.1.2 其他分支

就像之前说的，如果仅仅是一个分支，在某些情况并不能满足实际的需求，那么就需要创建多个不同的分支。

#### 3.1.2.1 创建分支

```
# git branch 分支名称
git branch b1
git branch b2

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git branch b1

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git branch b2

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ |
```

现在我们创建了 2 个分支，不过这两个分支都是基于 master 主干分支为基础的。

#### 3.1.2.2 查看分支

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ git branch -v
* b1      f2f113f update file
  b2      f2f113f update file
  master  f2f113f update file
  test    f2f113f update file

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ |
```

#### 3.1.2.3 切换分支

我们将工作线路切换到 b1

```
# git checkout 分支名称
git checkout b1

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git checkout b1
Switched to branch 'b1'

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ |
```

此时我们添加新的文件 b1.txt

Feture (E) > project > git > repository				
名称	修改日期	类型	大小	
.git	2022/12/18 18:26	文件夹		
b1.txt	2022/12/18 23:03	文本文档	1 KB	
test.txt	2022/12/17 16:54	文本文档	1 KB	

然后提交到版本库

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ git add b1.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ git commit -m 'b1 branch commit'
[b1 beb0e6c] b1 branch commit
 1 file changed, 1 insertion(+)
 create mode 100644 b1.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$
```

此时，查看分支信息，会发现不同分支的版本进度信息发生了改变

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ git branch -v
* b1      beb0e6c b1 branch commit
  b2      f2f113f update file
  master   f2f113f update file
  test     f2f113f update file

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$
```

如果此时切换回到主干分支的话，那么 b1.txt 文件就不存在了，因为对应版本信息不一样。

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ git checkout master
Switched to branch 'master'

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ ll
total 1
-rw-r--r-- 1 18801 197610 8 Dec 17 16:54 test.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ |
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git checkout b1
Switched to branch 'b1'

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ ll
total 2
-rw-r--r-- 1 18801 197610 9 Dec 18 23:26 b1.txt
-rw-r--r-- 1 18801 197610 8 Dec 17 16:54 test.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ |
```

### 3.1.2.4 删除分支

如果觉得某一个分支建立的不太理想或已经没有必要在使用了，那么是可以将这个分支删除的。

```
# git branch -d 分支名称
Git branch -d b2

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ git branch -d b2
Deleted branch b2 (was f2f113f).

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ |
```

```

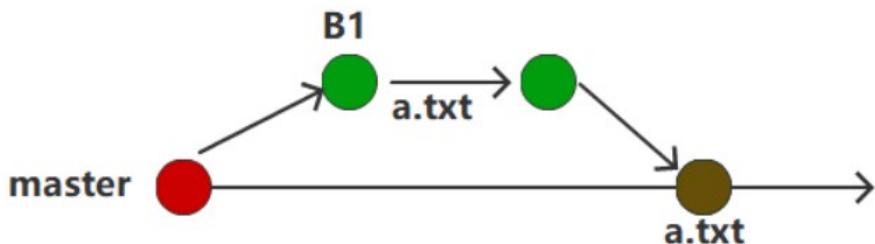
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ git branch -v
* b1    beb0e6c b1 branch commit
  master f2f113f update file
  test   f2f113f update file

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ |

```

## 3.2 Git 合并

无论我们创建多少个分支，都是因为我们需要在不同的工作环境中进行工作，但是，最后都应该将所有的分支合在一起。形成一个整体。作为项目的最终结果。



### 3.2.1 主干分支

首先我们先将主干分支的所有文件清空掉

```

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ ll
total 0

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ |

```

在当前主干分支中创建一份文件 master.txt，并提交

```

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ ll
total 1
-rw-r--r-- 1 18801 197610 6 Dec 20 08:50 master.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git add master.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git commit -m 'master.txt'
[master (root-commit) a643ee3] master.txt
 1 file changed, 1 insertion(+)
 create mode 100644 master.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ |

```

### 3.2.2 其他分支

基于主干分支的内容，我们创建其他分支，并直接切换到新的分支

```

# git checkout -b 分支名称
git checkout -b new_branch

```

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git checkout -b new_branch
Switched to a new branch 'new_branch'

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (new_branch)
$ ll
total 1
-rw-r--r-- 1 18801 197610 6 Dec 20 08:50 master.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (new_branch)
$ |
```

在新的分支中添加新文件 branch.txt

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (new_branch)
$ git add branch.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (new_branch)
$ git commit -m 'branch.txt'
[new_branch 76dead0] branch.txt
 1 file changed, 1 insertion(+)
 create mode 100644 branch.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (new_branch)
$ |
```

此时切换回主干分支，只有 master.txt 文件。

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (new_branch)
$ git checkout master
Switched to branch 'master'

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ ll
total 1
-rw-r--r-- 1 18801 197610 6 Dec 20 08:50 master.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ |
```

再切换回 new\_branch 分支，branch 文件就又回来了。

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git checkout new_branch
Switched to branch 'new_branch'

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (new_branch)
$ ll
total 2
-rw-r--r-- 1 18801 197610 6 Dec 20 08:55 branch.txt
-rw-r--r-- 1 18801 197610 6 Dec 20 08:50 master.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (new_branch)
$ |
```

### 3.2.3 合并分支

这里我们将 new\_branch 分支的文件内容合并到主干分支中。首先先切换回主干分支

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (new_branch)
$ git checkout master
Switched to branch 'master'

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ |
```

然后执行分支合并指令

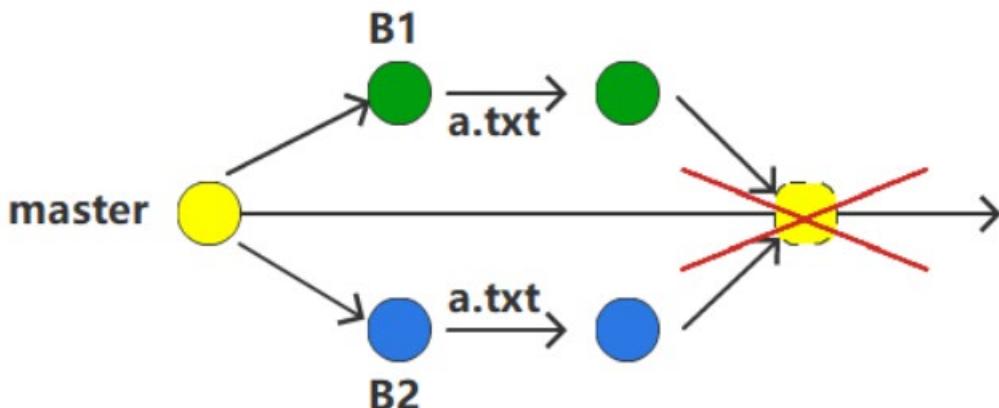
```
# git merge 分支名称  
git merge new_branch  
  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)  
$ git merge new_branch  
Updating a643ee3..76dead0  
Fast-forward  
 branch.txt | 1 +  
 1 file changed, 1 insertion(+)  
 create mode 100644 branch.txt  
  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)  
$
```

此时再次查看文件，就会发现 branch.txt 文件已经可以看到。

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)  
$ ll  
total 2  
-rw-r--r-- 1 18801 197610 6 Dec 20 08:58 branch.txt  
-rw-r--r-- 1 18801 197610 6 Dec 20 08:50 master.txt
```

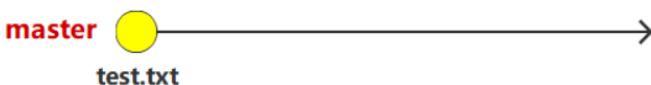
### 3.3 Git 冲突

在多分支并行处理时，每一个分支可能是基于不同版本的主干分支创建的。如果每隔分支都独立运行而不进行合并，就没有问题，但是如果在后续操作过程中进行合并的话，就有可能产生冲突。比如 B1, B2 的两个分支都是基于 master 分支创建出来的。B1 分支如果和 B2 分支修改了同一份文件的话，那么在合并时，以哪一个文件为准呢，这就是所谓的冲突。



接下来，咱们就演示一下。

#### 3.3.1 主干分支



首先我们先将主干分支的所有文件清空掉

```
l8801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ ll
total 0

l8801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$
```

主干分支添加文件 test.txt，文件内容为空

名称	修改日期	类型	大小
.git	2022/12/20 9:14	文件夹	
test.txt	2022/12/20 11:11	文本文档	0 KB

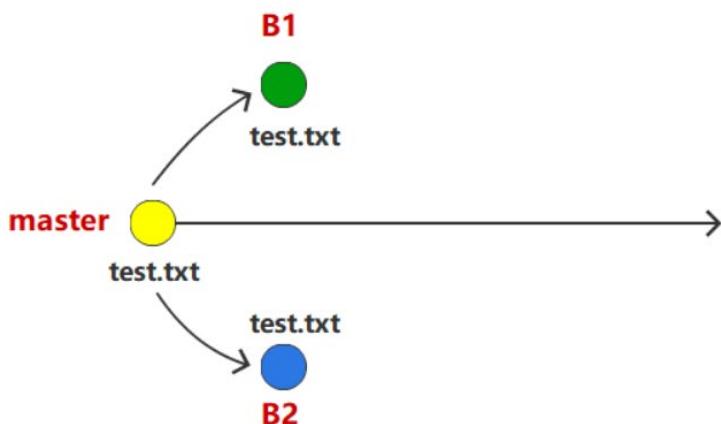
```
l8801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git add test.txt

l8801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git commit test.txt -m 'new file'
[master (root-commit) 514ab43] new file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.txt

l8801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ |
```

### 3.3.2 其他分支

基于主干分支，创建两个分支 B1, B2

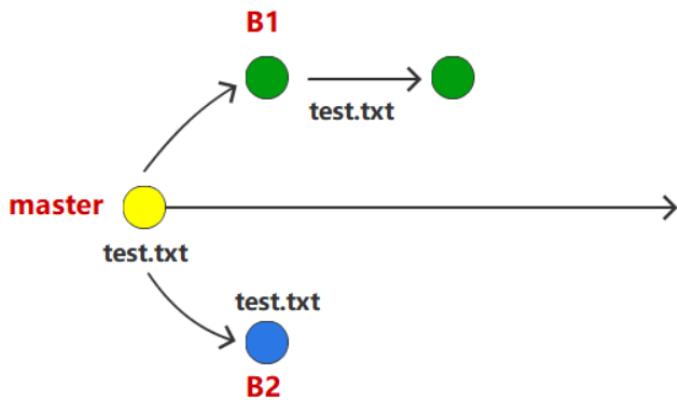


```
l8801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git branch b1

l8801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git branch b2

l8801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git branch
  b1
  b2
* master
```

### 3.3.3 切换分支-B1



切换到 B1 分支，修改文件内容

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git checkout b1
Switched to branch 'b1'

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ |
```



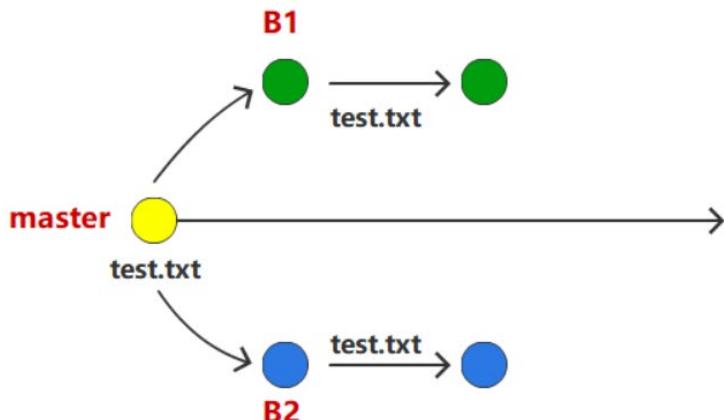
提交修改后的文件

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ git add test.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ git commit test.txt -m 'b1 branch'
[b1 a42f681] b1 branch
1 file changed, 1 insertion(+)

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ |
```

### 3.3.4 切换分支-B2



切换到 B2 分支，查看文件内容

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b1)
$ git checkout b2
Switched to branch 'b2'

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b2)
```

名称	修改日期	类型	大小
.git	2022/12/20 15:51	文件夹	
test.txt	2022/12/20 15:51	文本文档	0 KB

修改文件内容：



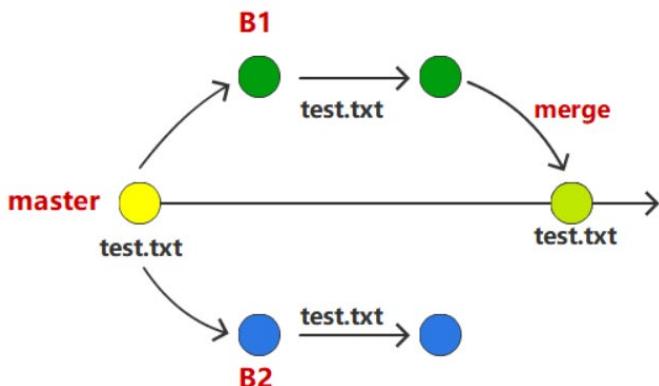
提交文件

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b2)
$ git add test.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b2)
$ git commit test.txt -m 'b1 branch'
[b2 51f2a74] b1 branch
 1 file changed, 1 insertion(+)

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b2)
$
```

### 3.3.5 合并分支-B1



切换到 master 主干分支，此时 test.txt 文件内容为空

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (b2)
$ git checkout master
Switched to branch 'master'

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ ll
total 0
-rw-r--r-- 1 18801 197610 0 Dec 20 16:05 test.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$
```

将 B1 分支合并到主干分支中

```

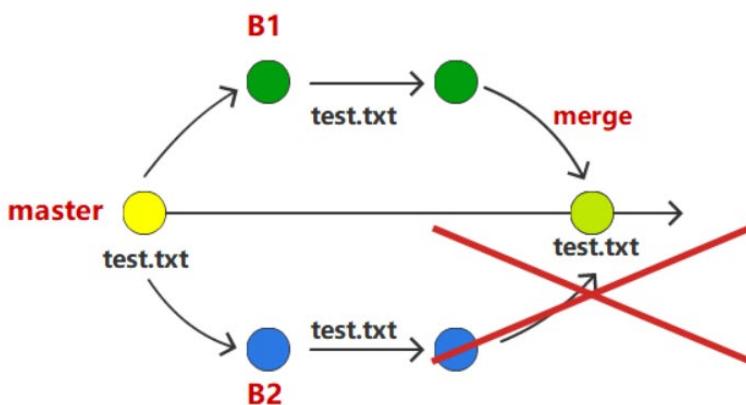
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git merge b1
Updating 514ab43..a42f681
Fast-forward
 test.txt | 1 +
 1 file changed, 1 insertion(+)

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ ll
total 1
-rw-r--r-- 1 18801 197610 2 Dec 20 16:07 test.txt

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ 

```

### 3.3.6 合并分支-B2



因为 B2 分支也对文件进行了修改，所以如果此时合并 B2 分支,就会提示冲突

```

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
$ git merge b2
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master|MERGING)
$ 

```

查看文件内容差异

```

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master|MERGING)
$ git diff
diff --cc test.txt
index 8a02e44,42fbb5a..0000000
--- a/test.txt
+++ b/test.txt
@@@ -1,1 -1,1 +1,5 @@@
- B1
- B2
++<<<<< HEAD
++B1
=====
++B2
++>>>>> b2

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master|MERGING)
$ 

```

```
1 <<<<< HEAD
2 B1
3 =====
4 B2
5 >>>>> b2|
6
```

这里的冲突，软件是无法判断该如何出来处理的，所以需要人工进行判断，将冲突的文件内容进行修正。

```
test.txt
1 B1
2 B2
3
```

重新提交到 master 主干分支中。

```
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master|MERGING)
$ git add test.txt

# git commit 文件名称 -i -m 注释

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master|MERGING)
$ git commit test.txt -i -m 'merge'
[master 3a9ac89] merge

18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/respository (master)
```

再查看一下 Git 软件的操作日志

```
# git log --graph
* commit 3a9ac89628493db0547a7983e17aa8ad5a71c469 (HEAD -> master)
  Merge: 5fafbf8 a7fb5b2
  Author: test <test@atguigu.com>
  Date:   Tue Dec 20 18:02:31 2022 +0800

    merge

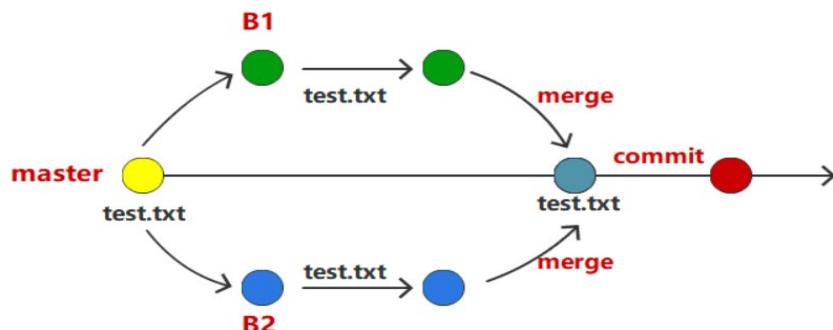
* commit a7fb5b22296acfde3488f83bf4428bb43da2e050 (b2)
  Author: test <test@atguigu.com>
  Date:   Tue Dec 20 17:56:24 2022 +0800

    b2

* commit 5fafbf86c02ffcab9e32ee8e068c0ed994164352 (b1)
  Author: test <test@atguigu.com>
  Date:   Tue Dec 20 17:55:37 2022 +0800

    b1

* commit a3ff54d9aa89f551e1558eab9aeef35295babf3d
  Author: test <test@atguigu.com>
```



# 第4章 Git 服务器集成

## 4.1 Git 远程服务器

在之前的操作中，所有的操作都是基于本地机器完成的。如果在公司中，一个项目是共用一个版本库的。那么所有的开发人员都应该对同一个版本库进行操作。因为 Git 软件本身就是用于 Linux 系统开发所设计的版本管理软件，所以项目中搭建的共享版本库也应该以 linux 系统为主。那么接下来，咱们就演示一下在 CentsOS 服务器中搭建 Git 服务器。

### 4.1.1 下载 Git 软件 (linux 版本)

官网下载地址：<https://mirrors.edge.kernel.org/pub/software/scm/git/git-2.38.1.tar.gz>

将下载后的压缩文件上传到 Linux 系统中

Name	Size (KB)	Last modified	Owner	Group	Access
..					
clickhouse		2021-07-07 08:49	root	root	drwxr-xr-x
mysql		2021-02-04 17:34	root	root	drwxr-xr-x
apache-flume-1.9.0-bin.tar.gz	66 345	2021-02-04 17:34	root	root	-rw-r--r--
apache-hive-3.1.2-bin.tar.gz	305 517	2021-02-04 17:35	root	root	-rw-r--r--
apache-phoenix-5.0.0-HBase-... .tar.gz	426 629	2021-07-10 23:00	root	root	-rw-r--r--
apache-zookeeper-3.5.7-bin.t... .tar.gz	9 093	2021-02-04 22:23	root	root	-rw-r--r--
elasticsearch-7.8.0-linux-x86_... .tar.gz	311 633	2021-03-06 22:42	root	root	-rw-r--r--
elasticsearch-8.1.0-linux-x86_... .tar.gz	503 716	2022-03-14 23:42	root	root	-rw-r--r--
flink-1.13.1-bin-scala_2_12.tgz	297 571	2021-06-05 06:50	root	root	-rw-r--r--
git-2.38.1.tar.gz	10 227	2022-12-20 19:01	root	root	-rw-r--r--
hadoop-3.1.3.tar.gz	330 152	2021-02-04 17:35	root	root	-rw-r--r--
hbase-2.0.5-bin.tar.gz	130 428	2021-02-04 17:36	root	root	-rw-r--r--
jdk-8u212-linux-x64.tar.gz	190 442	2021-02-04 17:36	root	root	-rw-r--r--
kafka-eagle-bin-2.0.5.tar.gz	79 061	2021-06-02 23:51	root	root	-rw-r--r--
kafka-eagle-web-2.0.5-bin.tar.gz	79 050	2021-10-10 21:44	root	root	-rw-r--r--
kafka_2.11-2.4.1.tgz	68 515	2021-02-04 17:36	root	root	-rw-r--r--
kafka_2.12-2.8.0.tgz	69 865	2021-05-07 21:52	root	root	-rw-r--r--

### 4.1.2 安装 Git 软件

#### 4.1.2.1 解压 Git

```
# 将压缩文件解压到自定义位置
tar -zxvf git-2.38.1.tar.gz -C /opt/module/
# 可以更改名字，变得简短一些，好操作
cd /opt/module
mv git-2.38.1/ git
```

#### 4.1.2.2 安装依赖

解压后，我们需要编译源码，不过在此之前需要安装编译所需要的依赖，耐心等待安装完成，中途出现提示的时候输入 y 并按回车。

```
# 
yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel gcc
perl-ExtUtils-MakeMaker
```

```

已安装:
expat-devel.x86_64 0:2.1.0-15.el7_9
openssl-devel.x86_64 1:1.0.2k-25.el7_9

作为依赖被安装:
gdbm-devel.x86_64 0:1.10-8.el7
keyutils-libs-devel.x86_64 0:1.5.8-3.el7
libdb-devel.x86_64 0:5.3.21-25.el7
libverto-devel.x86_64 0:0.2.5-4.el7
perl-ExtUtils-Install.noarch 0:1.58-299.el7_9
perl-Git.noarch 0:1.8.3.1-23.el7_8
systemtap-sdt-devel.x86_64 0:4.0-13.el7

作为依赖被升级:
curl.x86_64 0:7.29.0-59.el7_9           e2fsprogs.x86_64 0:1.42.9-19.el7
gettext.x86_64 0:0.19.8.1-3.el7          gettext-common-devel.noarch 0:0.19.8.1-3.el7
libcom_err.x86_64 0:1.42.9-19.el7        krb5-libs.x86_64 0:1.15.1-55.el7_9
libkadm5.x86_64 0:1.15.1-55.el7_9       libcurl.x86_64 0:7.29.0-59.el7_9
libsepol.x86_64 0:2.5-10.el7            libselinux-devel.x86_64 0:2.5-15.el7
nss.x86_64 0:3.79.0-4.el7_9             libselinux-python.x86_64 0:2.5-15.el7
nss-sysinit.x86_64 0:3.79.0-4.el7_9    libss.x86_64 0:1.42.9-19.el7
openssl-libs.x86_64 1:1.0.2k-25.el7_9  libssh2.x86_64 0:1.8.0-4.el7
                                         libssh2.x86_64 0:1.8.0-4.el7
                                         libss.x86_64 0:3.79.0-4.el7_9
                                         nss-softokn.x86_64 0:3.79.0-4.el7_9
                                         nss-util.x86_64 0:3.79.0-1.el7_9
                                         openssl.x86_64 1:1.0.2k-25.el7_9

 zlib-devel.x86_64 0:1.2.7-20.el7_9

```

#### 4.1.2.3 删除旧版 Git

安装编译源码所需依赖的时候，yum 操作会自动安装旧版本的 Git。

```
[root@linux1 module]# git --version
git version 1.8.3.1
```

我们这里需要卸载这个旧版的 Git

```
# 删除旧版本的 Git
yum -y remove git

Running transaction
正在删除 : gettext-devel-0.19.8.1-3.el7.x86_64
正在删除 : perl-Git-1.8.3.1-23.el7_8.noarch
正在删除 : git-1.8.3.1-23.el7_8.x86_64
验证中   : git-1.8.3.1-23.el7_8.x86_64
验证中   : perl-Git-1.8.3.1-23.el7_8.noarch
验证中   : gettext-devel-0.19.8.1-3.el7.x86_64

1/3
2/3
3/3
1/3
2/3
3/3

删除:
git.x86_64 0:1.8.3.1-23.el7_8

作为依赖被删除:
gettext-devel.x86_64 0:0.19.8.1-3.el7
perl-Git.noarch 0:1.8.3.1-23.el7_8

完毕!
[root@linux1 module]#
```

#### 4.1.2.4 编译、安装 Git

```
# 进入到 Git 软件的解压目录
cd /opt/module/git

# 编译时, prefix 设定为 Git 软件安装目录
make prefix=/usr/local/git all

# 安装 Git
make prefix=/usr/local/git install

remote_curl_aliases="git-remote-https git-remote-ftp git-remote-ftps" && \
for p in $remote_curl_aliases; do \
  rm -f "$execdir/$p" && \
  test -n "" && \
  ln -s "git-remote-http" "$execdir/$p" || \
{ test -z "" && \
  ln "$execdir/git-remote-http" "$execdir/$p" 2>/dev/null || \
  ln -s "git-remote-http" "$execdir/$p" 2>/dev/null || \
  cp "$execdir/git-remote-http" "$execdir/$p" || exit; } \
done
[root@linux1 git]#
```

#### 4.1.2.5 配置环境变量

修改 linux 系统中/etc/profile 文件，配置环境变量

```
# 配置环境变量
export PATH=$PATH:/usr/local/git/bin
```

```
# 刷新环境，让环境变量立即生效  
source /etc/profile
```

#### 4.1.2.6 建立链接文件

```
# git 安装路径是/usr/local/git，不是默认路径  
ln -s /usr/local/git/bin/git-upload-pack /usr/bin/git-upload-pack  
ln -s /usr/local/git/bin/git-receive-pack /usr/bin/git-receive-pack
```

#### 4.1.2.7 测试安装

```
# 获取 git 软件版本  
git --version  
[root@linux1 ~]# git --version  
git version 2.38.1  
[root@linux1 ~]#
```

### 4.1.3 创建 Git 用户

因为 Git 服务器需要安装在 linux 系统上，当使用远程客户端操作时，就需要提供相应的 Git 账号进行提交的，如果你的仓库文件的用户不是 git 的话，是 root 用户或者别的用户，那么你 git push ,它是不允许的，因为你的 git 用户没有权限。你可以给这个文件创立 git 用户，或者修改文件夹的权限让所有用户都可以更改

```
# 增加用户  
adduser git  
# 设定密码  
passwd git  
  
[root@linux1 git]# adduser git  
[root@linux1 git]# passwd git  
更改用户 git 的密码。  
新的 密码：  
无效的密码： 密码少于 8 个字符  
重新输入新的 密码：  
passwd: 所有的身份验证令牌已经成功更新。
```

### 4.1.4 SSH 免密登录

#### 4.1.4.1 服务器端操作

```
# 进入用户目录  
cd /home/git  
# 在 git 用户根目录创建.ssh 目录  
sudo mkdir .ssh  
sudo touch .ssh/authorized_keys  
# 设定.ssh 目录，authorized_keys 的权限  
sudo chmod -R 700 /home/git/.ssh  
sudo chmod 600 /home/git/.ssh/authorized_keys
```

#### 4.1.4.2 客户端端操作

```
# 在客户端生成 ssh 密钥  
# 默认生成的密钥用户就是当前用户，需要和之前的全局配置保持一致  
user.name=18801@LAPTOP-J9IRK5BM
```

```
user.email=18801@LAPTOP-J9IRK5BM
# 按照提示三次回车即可
ssh-keygen -t rsa
```

```
Your identification has been saved in /c/Users/18801/.ssh/id_rsa
Your public key has been saved in /c/Users/18801/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:6YicFhr1jSg1lqnY/7ZFY1SjgkcaV5zBuIiXJ02InrI 18801@LAPTOP-J9IRK5BM
The key's randomart image is:
+---[RSA 3072]---+
| .. o=ooo |
| ..oB. +o . |
| .o*Boo.o |
| .+o==+=. . |
| o * o o S |
| .o * + = . |
| ... * . o |
| E . . . |
| .o. |
+---[SHA256]---+
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/remote-rep
$
```

在用户根目录的.ssh 文件夹内， id\_rsa.pub 就是我们的公钥

名称	修改日期	类型	大小
id_rsa	2022/12/21 9:11	文件	3 KB
id_rsa.pub	2022/12/21 9:11	PUB 文件	1 KB
known_hosts	2022/12/21 0:48	文件	1 KB
known_hosts.old	2022/12/20 23:25	OLD 文件	1 KB

将文件中的内容复制到服务器端的.ssh/authorized\_keys 文件中

```
authorized_keys
1 ssh-rsa AAAAB3NzaC1yc2EAAAQABgQDK9fPayMENSxt1bTkUMInJdqW4UWt2d7072ApBa4O5cl0NUydRTvE7mrPUbAuRMoAhRztrvgF+9U3+Ix1gMpWDCCWPwF
2
```

## 4.1.5 创建 Git 版本库

### 4.1.5.1 创建文件目录

```
# 进入用户目录
cd /home/git
# 创建版本库目录
mkdir git-rep
# 设定文件所属用户
sudo chown git:git git-rep
```

### 4.1.5.2 初始化版本库

```
# 进入仓库目录
cd /home/git/git-rep
# 初始化仓库，和前面的 git init 略有不同
git init --bare test.git
# 设定文件所属用户
sudo chown -R git:git test.git
```

## 4.1.6 远程访问 Git 版本库

### 4.1.6.1 将远程仓库克隆到本地

```
# 将远程仓库克隆到本地，形成本地仓库
```

```
# 克隆远程仓库 => 用户@主机名:仓库地址  
git clone git@linux1:/home/git/git-rep/test.git
```

#### 4.1.6.2 提交文件到本地仓库

```
# 增加文件  
git add client.txt  
# 提交文件  
git commit -m 'client'  
  
$ git add client.txt  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/remote-rep/test (master)  
$ git commit -m 'client'  
[master (root-commit) 241bf27] client  
1 file changed, 1 insertion(+)  
create mode 100644 client.txt  
  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/remote-rep/test (master)  
$ |
```

#### 4.1.6.3 将本地仓库同步到远程仓库

```
# 同步远程仓库  
# 远程仓库默认有个别名叫 origin, 将本地仓库的文件推送 (push) 到远程仓库  
# git push 远程仓库别名 分支名称  
git push origin master  
  
$ git push origin master  
Enumerating objects: 3, done.  
Counting objects: 100% (3/3), done.  
Writing objects: 100% (3/3), 219 bytes | 219.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
To 192.168.10.101:/home/tiger/gitrepo/test.git  
 * [new branch]      master -> master  
  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/remote-rep/test (master)  
$ |
```

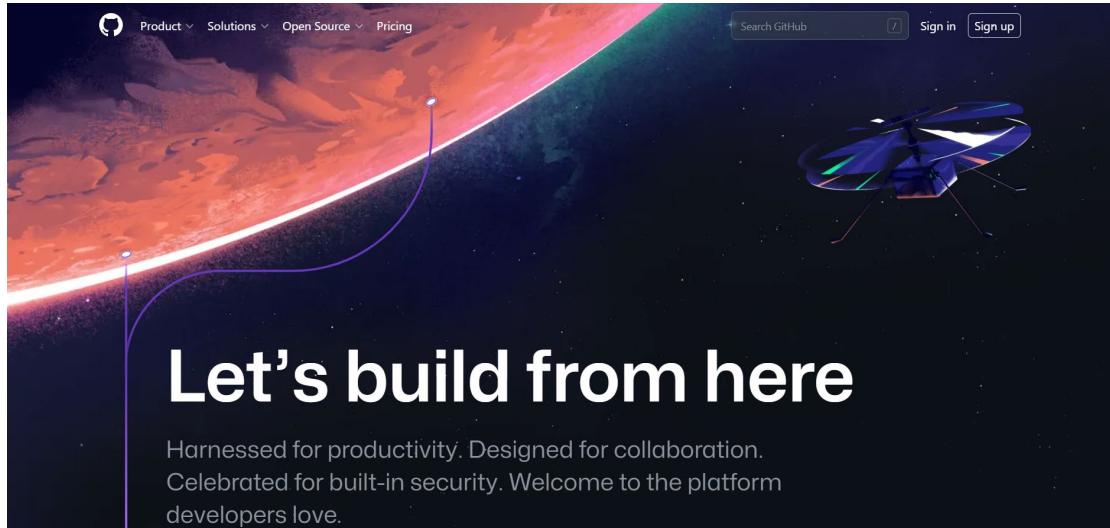
#### 4.1.6.4 查看远程仓库

```
# 服务器端切换用户  
su git  
# 进入仓库  
cd /home/git/git-rep/test.git  
# 切换到主干分支  
git checkout master  
# 查看 git 日志  
git log  
  
commit 241bf27ef625d3975ae90ee7edb24c13b338ba0c (HEAD -> master)  
Author: 18801@LAPTOP-J9IRK5BM <18801@LAPTOP-J9IRK5BM>  
Date:   Wed Dec 21 16:56:08 2022 +0800  
  
        client
```

## 4.2 GitHub 服务器

公司中，我们可以搭建中央服务器让项目组开发人员共享代码，但是如果我们的开发人

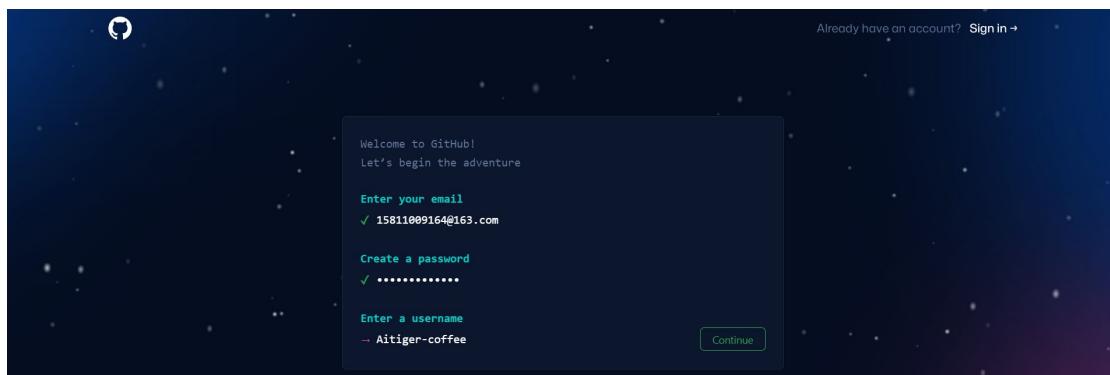
员都是通过互联网进行协作，而不是在同一个地方，那么开发时，程序文件代码的版本管理就显得更加重要，这就需要搭建一个互联网的版本库，让不同地点的人都可以进行访问。这里我们不用自己搭建。因为 GitHub 网站已经帮助我们提供了共享版本库功能。所以我们接下来就讲解一下，如何使用 GitHub 网站所提供的功能使用 Git。



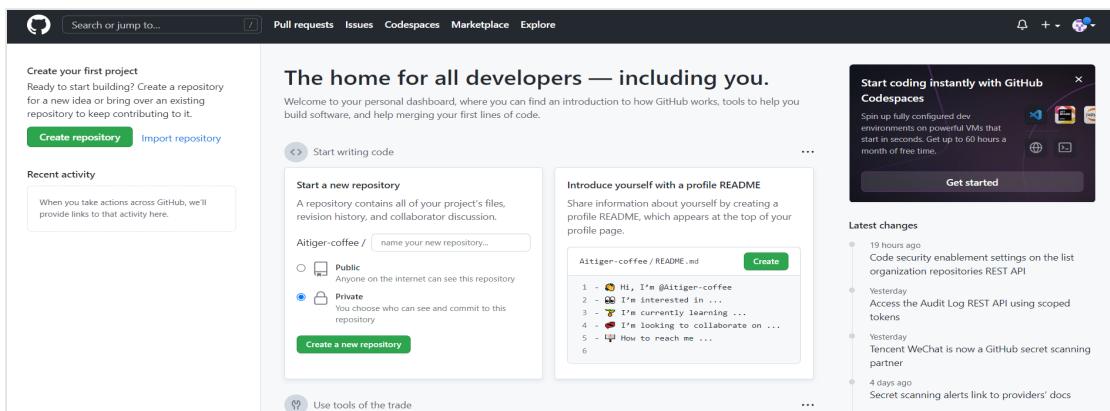
#### 4.2.1 注册网站会员

GitHub 官网地址：<https://github.com/>

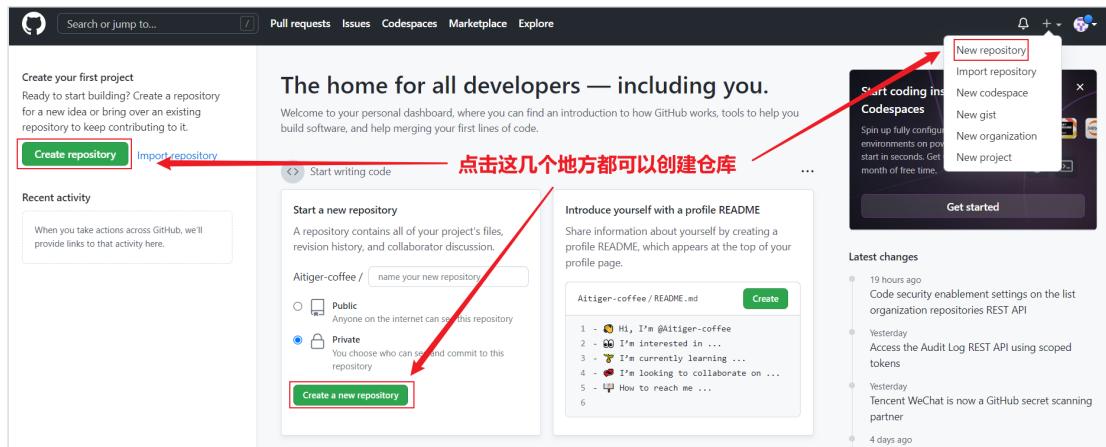
填写你的邮箱地址和密码，姓名



一顿操作，注册完毕后，进入你的主页



## 4.2.2 创建新的仓库

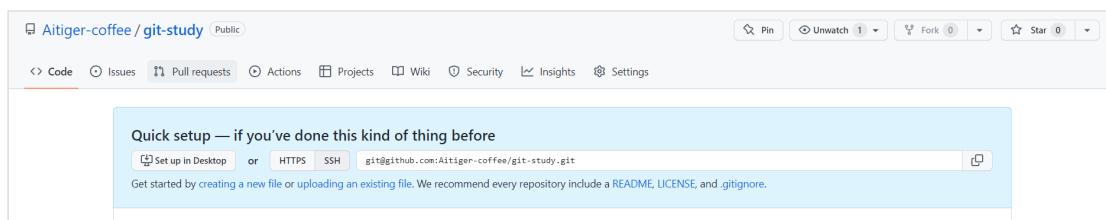


输入仓库的相关信息

The screenshot shows the "Create a new repository" form with the following fields and options:

- "Owner" dropdown: Aitiger-coffee
- "Repository name" input: git-study (highlighted by a red arrow)
- "Description (optional)" input: 用于学习的Git仓库 (highlighted by a red arrow)
- "Visibility" radio buttons:
  - Public (highlighted by a red arrow)
  - Private (highlighted by a red arrow)

点击创建按钮，创建新的仓库。



## 4.2.3 本地仓库的基本操作指令

```
# create a new repository on the command line
echo "# git-study" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:Aitiger-coffee/git-study.git
git push -u origin main

# push an existing repository from the command line
```

```
git remote add origin git@github.com:Aitiger-coffee/git-study.git  
git branch -M main  
git push -u origin main
```

#### 4.2.4 SSH 免密操作

github 支持两种同步方式“https”和“ssh”。如果使用 https 很简单基本不需要配置就可以使用，但是每次提交代码和下载代码时都需要输入用户名和密码。ssh 模式比 https 模式的一个重要好处就是，每次 push、pull、fetch 等操作时，不用重复填写遍用户名密码。前提是必须是这个项目的拥有者或者合作者，且配好了 ssh key。

##### 4.2.4.1 本地生成 SSH 密钥

```
# ssh-keygen -t rsa -C GitHub账号  
ssh-keygen -t rsa -C 15811009164@163.com  
Your identification has been saved in /c/Users/18801/.ssh/id_rsa  
Your public key has been saved in /c/Users/18801/.ssh/id_rsa.pub  
The key fingerprint is:  
SHA256:4ccwPizlSSRVkMV6YU3tYNcaXHW2h4pHiJYGa7N+qTM 15811009164@163.com  
The key's randomart image is:  
+---[RSA 3072]---+  
| 0.+*oo.o o* |  
| =.o+.+ =o+ |  
| + @o.o.+.+ |  
| . %.*.o .o . |  
| o S.+ o |  
| . . + . |  
| . o |  
| Eo |  
| .o |  
+---[SHA256]---+  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/remote-rep
```

##### 4.2.4.2 集成用户公钥

执行命令完成后，在 window 本地用户.ssh 目录 C:\Users\用户名\.ssh 下面生成如下名称的公钥和私钥：

Windows-SSD (C:) > 用户 > 18801 > .ssh			
名称	修改日期	类型	大小
id_rsa	2022/12/21 22:26	文件	3 KB
id_rsa.pub	2022/12/21 22:26	PUB 文件	1 KB
known_hosts	2022/12/21 0:48	文件	1 KB
known_hosts.old	2022/12/20 23:25	OLD 文件	1 KB

按照操作步骤，将 id\_rsa.pub 文件内容复制到 GitHub 仓库中

3. 输入SSH公钥

2. 选择SSH

1. 选择配置

### SSH keys / Add new

Title: 15811009164@163.com

Key type: Authentication Key

Key:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAABgQDoVC5IT9jX0ZoNdggW9c2/QRLWL+0IWwx1rjRfSzZ0x2XHeuWPcRUHDljA6O0
BNCBpMxMvgPud66MuflAnL6AD10GboR5OPtXcd8Gel0xRh0QnY0qDvTna9BuUjEklLoPfCA6G5hfR4+QmRqla2tYxtQ0jt
3nzEFhHluWSwy4w7cNjQr6Wi/HbsRIOuKaHgkLp0HmHHbh58t/JVqGSwlgKx2xmjSjZXi5Tn25Sc4bOCgzAyJz0Z5bAi9
5Z55qRBZnN8e6hb7z8FOpa9wfCj9osE84D9zkWdBTbPmljPYLUw5t/K3c56ElKpuflEuwpnwOFRlx6N/gxO9x7hPBmxobe
V/G6f6c++Zj2BNGqQXuNLFv5iPrACbUgx9LwaM7HRU+ujWE6+uMUTzs7PndwnVOzd0Goq6PSIGwqxNZQt85teBkwctvi
Y/Svu9aejEKWTsi8Iue1CG9h8VWmX3lVBHaID5kj5N+MD8bnWJg+vq96nEutLlu8uNm= 15811009164@163.com
```

Add SSH key

点击 Add 按钮，增加 SSH 公钥信息

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

Authentication Keys

	15811009164@163.com SHA256:4ccwP1z1SSRVkMV6YU3tYNcaXHN2h4pH1JYGa7II+qTM Added on Dec 21, 2022 Never used — Read/write	Delete
--	--	--------

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

## 4.2.5 设定全局用户

```
git config --global user.name '15811009164'
# 这里的邮箱地址需要为 GitHub 网站的注册账号
git config --global user.email '15811009164@163.com'
```

## 4.2.6 创建本地库以远程地址

```
# 初始化本地仓库
git init
# 设置远程仓库
git remote add origin git@github.com:Aitiger-coffee/git-study.git
```

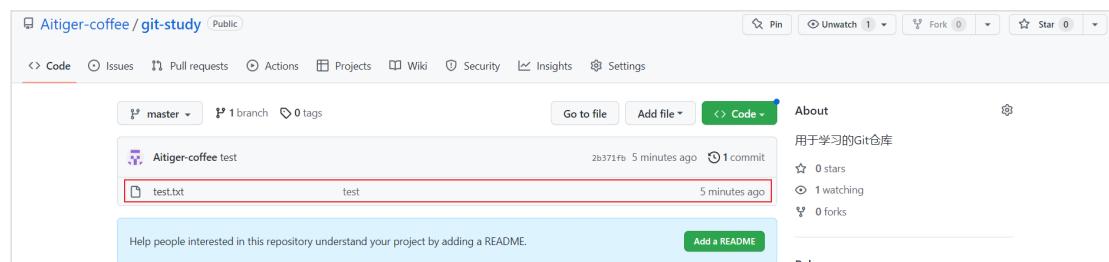
#### 4.2.7 新增，提交本地仓库文件

```
# 新增文件  
git add test.txt  
# 提交文件  
git commit test.txt  
  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/remote-rep (master)  
$ git add test.txt  
  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/remote-rep (master)  
$ git commit -m 'test'  
[master (root-commit) 2b371fb] test  
 1 file changed, 1 insertion(+)  
 create mode 100644 test.txt
```

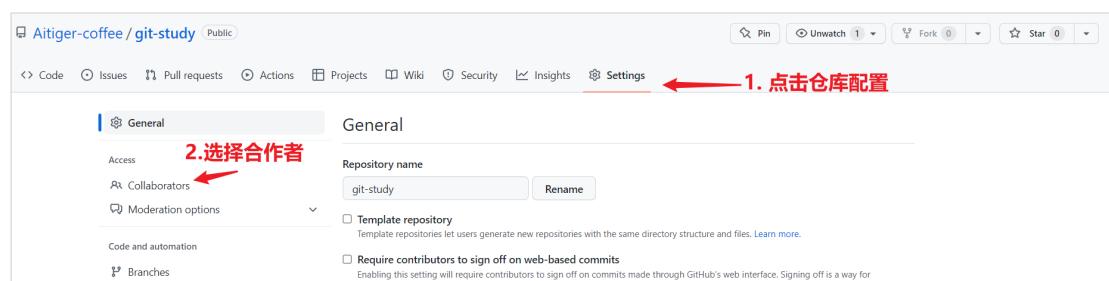
#### 4.2.8 推送到 GitHub 远程仓库

```
# 推送文件  
git push origin master  
  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/remote-rep (master)  
$ git push origin master  
The authenticity of host 'github.com (20.205.243.166)' can't be established.  
ED25519 key fingerprint is SHA256:+DiY3vvvV6TuJJhbpZisF/zLDA0zPMsvHdkr4UvCOqU.  
This key is not known by any other names  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes  
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.  
Enumerating objects: 3, done.  
Counting objects: 100% (3/3), done.  
Writing objects: 100% (3/3), 201 bytes | 201.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
To github.com:Aitiger-coffee/git-study.git  
 * [new branch]      master -> master  
  
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/remote-rep (master)
```

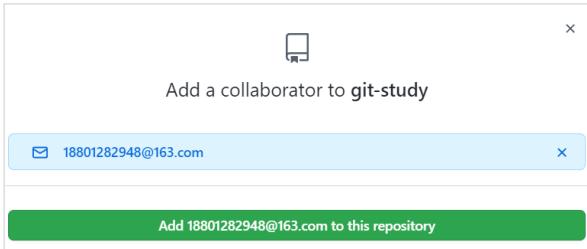
#### 4.2.9 查看 GitHub 远程仓库



#### 4.2.10 增加合作伙伴

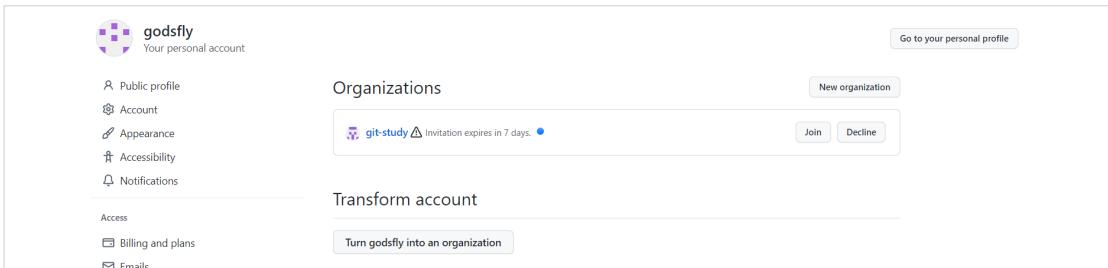


选择合作账号,发出合作申请

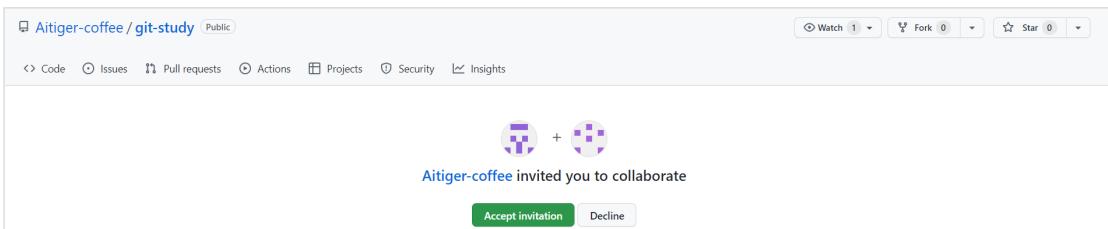


#### 4.2.11 合作伙伴确认

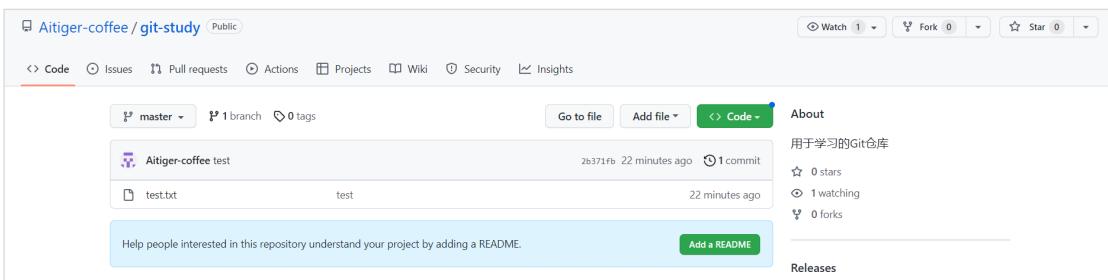
合作伙伴收到确认后, 点击 Join 按钮继续



点击 Accept Invitation 按钮, 进行确认

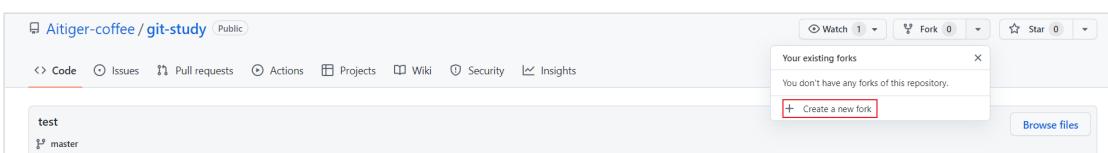


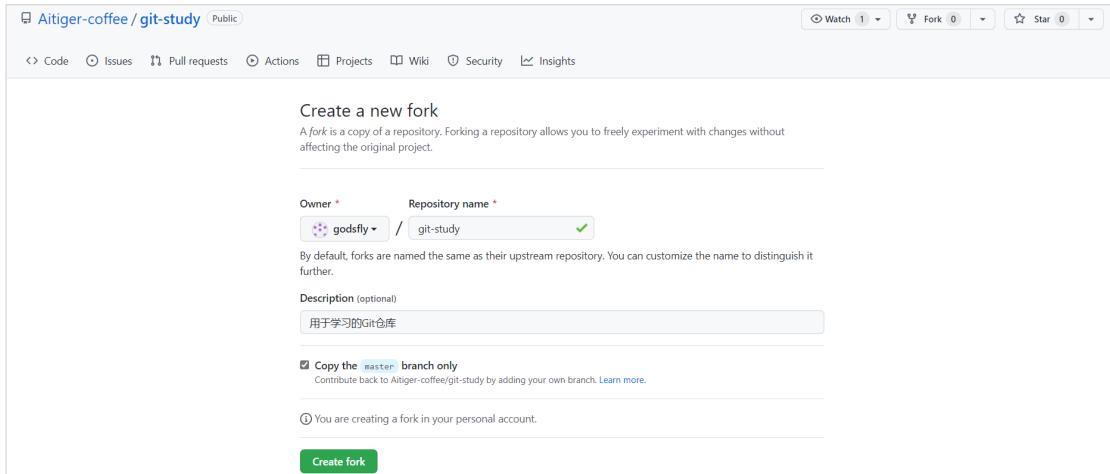
此时已经可以合作开发了。



#### 4.2.12 远程仓库 fork 操作

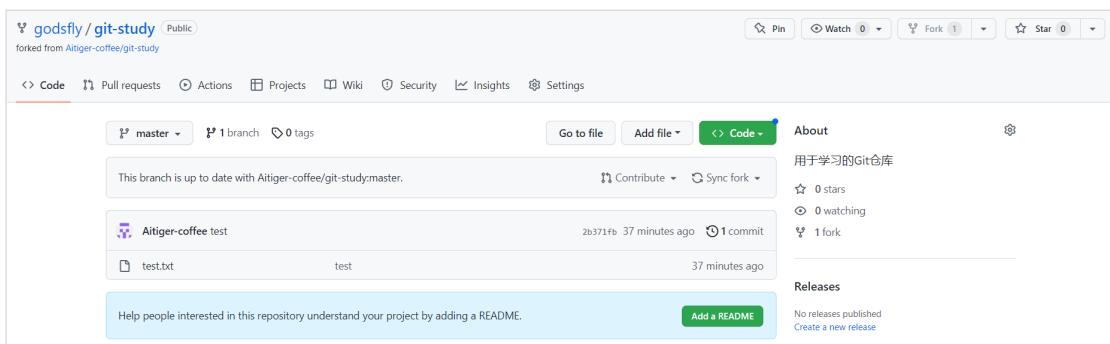
如果项目存在大量合作伙伴, 对于版本库的管理明显是一个特别大的风险, 所以如果不想要选择大量的合作伙伴, 但依然有人想要对项目代码进行维护, 更新和扩展的话, 此时, 我们就可以使用 fork 功能。





The screenshot shows the GitHub fork creation interface for the repository 'Aitiger-coffee/git-study'. The 'Create a new fork' section is displayed, with the owner set to 'godsfly' and the repository name set to 'git-study'. A description is provided: '用于学习的Git仓库'. The 'Copy the master branch only' checkbox is checked. A note indicates that forks are named the same as their upstream repository. The 'Create fork' button is at the bottom.

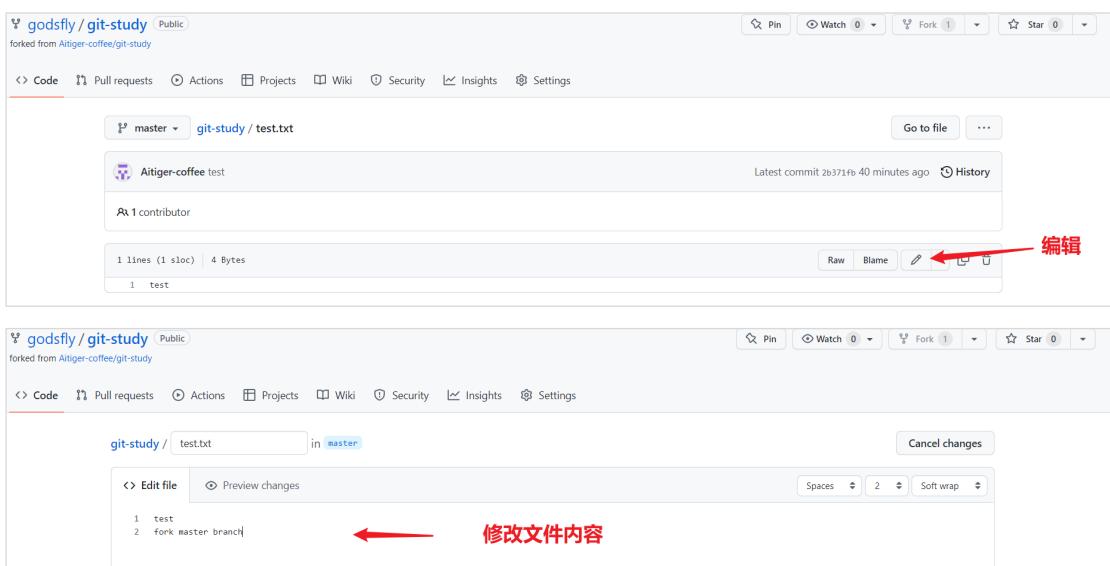
点击 Create fork 按钮即可



The screenshot shows the forked GitHub repository 'godsfly/git-study'. It displays the 'Code' tab, showing 1 branch ('master') and 0 tags. The repository is described as '用于学习的Git仓库'. A commit from 'Aitiger-coffee' is shown, with a file named 'test.txt' containing the content 'test'. There is a 'Add a README' button at the bottom.

这样就等同于创建了一个自己的远程仓库。但是这个远程仓库等同于是一个分支远程仓库，你可以随便操作，并不会影响源仓库，但是如果你的修改，更新想要融合到源仓库中，就需要提交申请了。

➤ 我们这里首先将文件改一下。



The screenshot shows the 'godsfly/git-study' repository with the 'Code' tab selected. The 'test.txt' file is open for editing. A red arrow points to the 'Edit' button in the toolbar. The file content is '1 test'. In the second screenshot, a red arrow points to the file content '2 fork master branch', with the text '修改文件内容' (Modify file content) written below it.

➤ 发送提交申请

## ➤ 合并修改请求

3. 查看文件更新情况

4. 合并更新请求

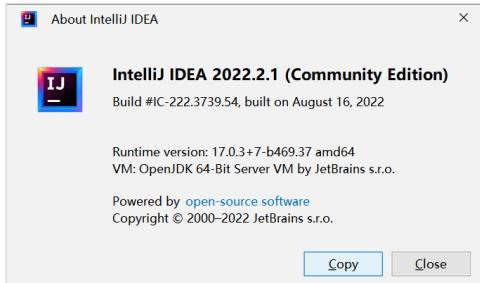
5. 确认合并

### ➤ 修改请求确认

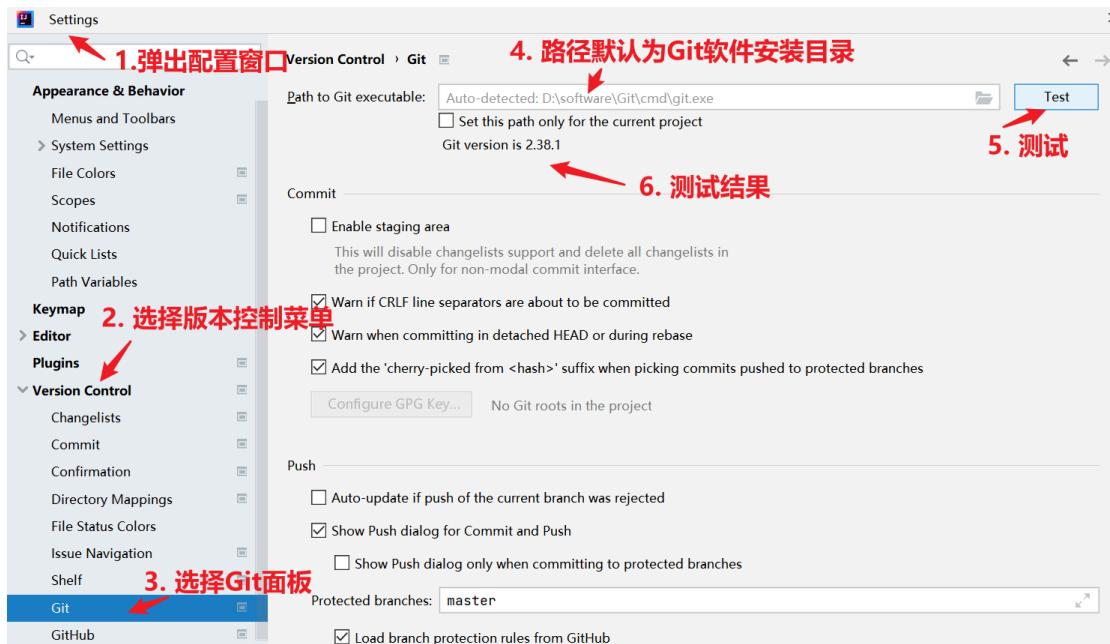
主干分区文件已经修改了

## 4.3 IDEA 集成 GitHub

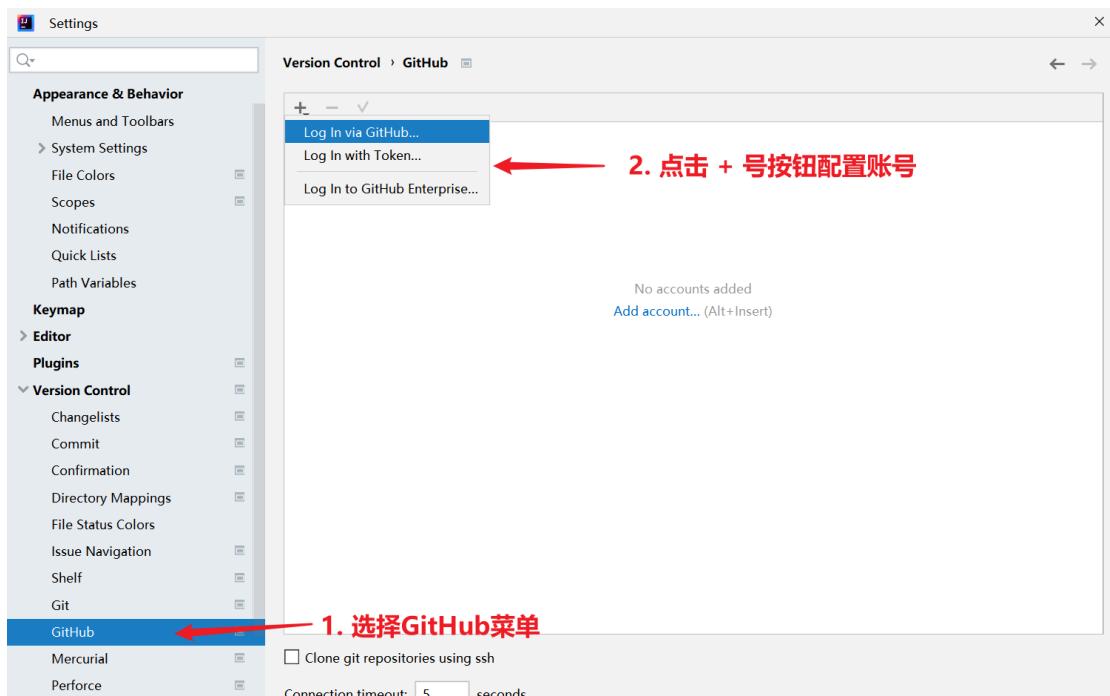
实际的开发中，代码都是采用 IDE 进行开发，所以我们这里介绍一下 IDEA 软件是如何集成 GitHub 远程仓库进行代码版本控制的。这里采用的 IDEA 版本为 2022.2.1, 其他版本的 IDEA 软件会略有差别



### 4.3.1 配置 Git 软件



### 4.3.2 配置 GitHub 账号

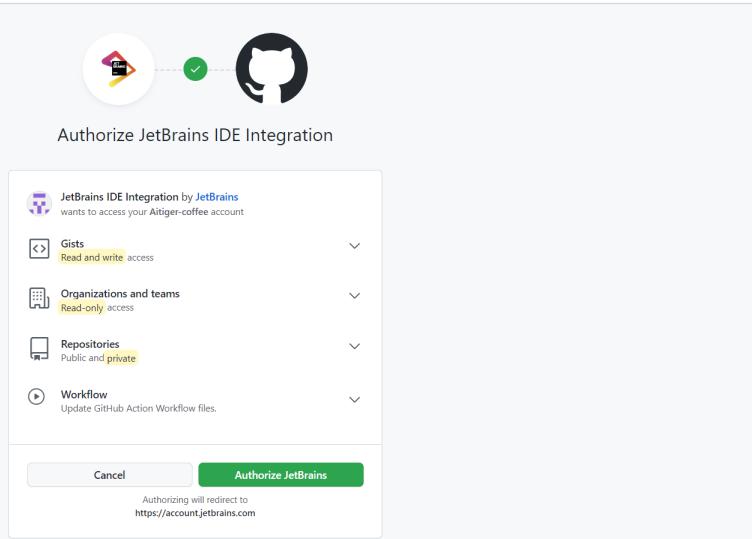


继续点授权按钮

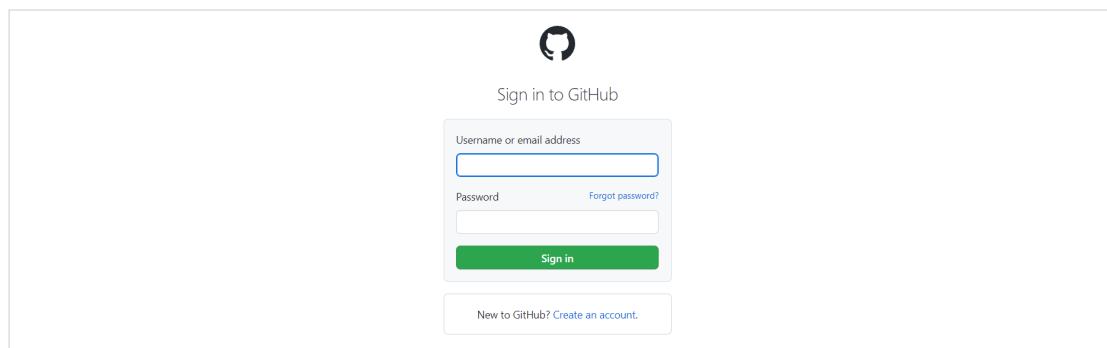
Please continue only if this page is opened from a JetBrains IDE.

[Authorize in GitHub](#)

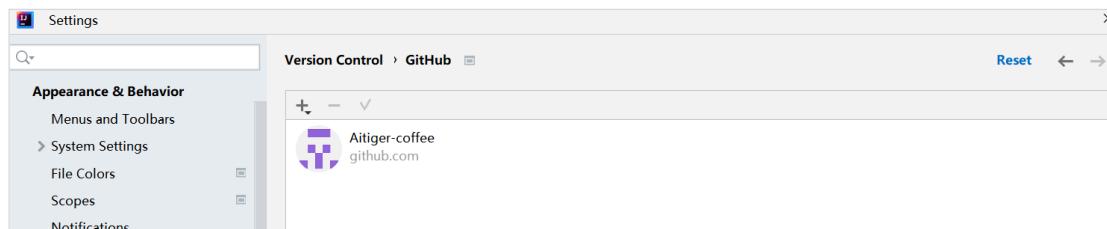
继续点击授权按钮



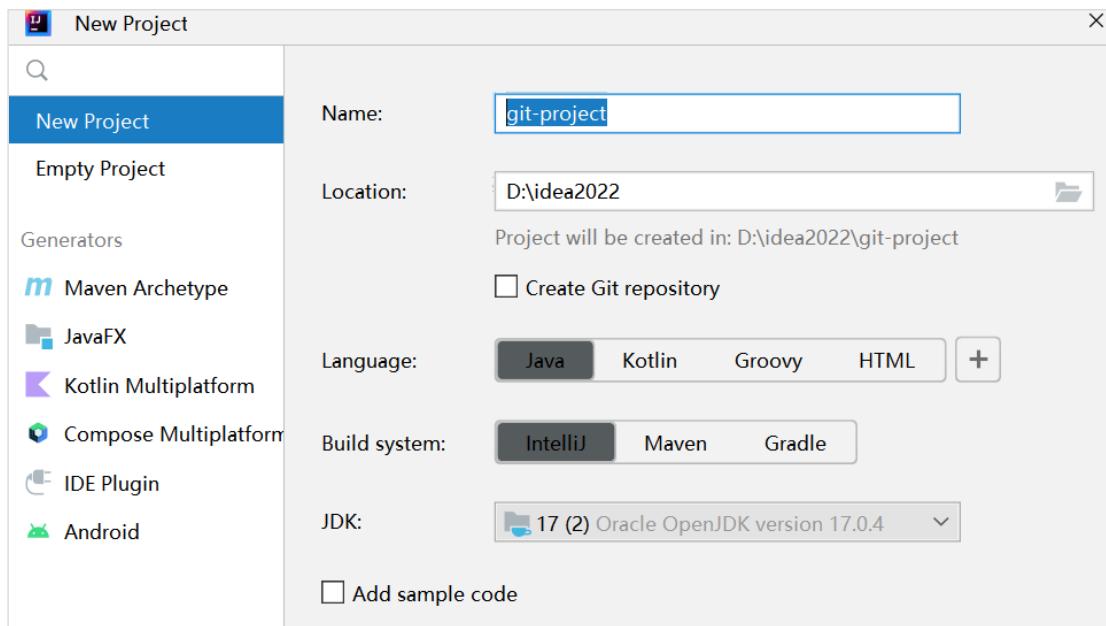
输入 GitHub 账号密码



You have been successfully authorized in GitHub. You can close the page.



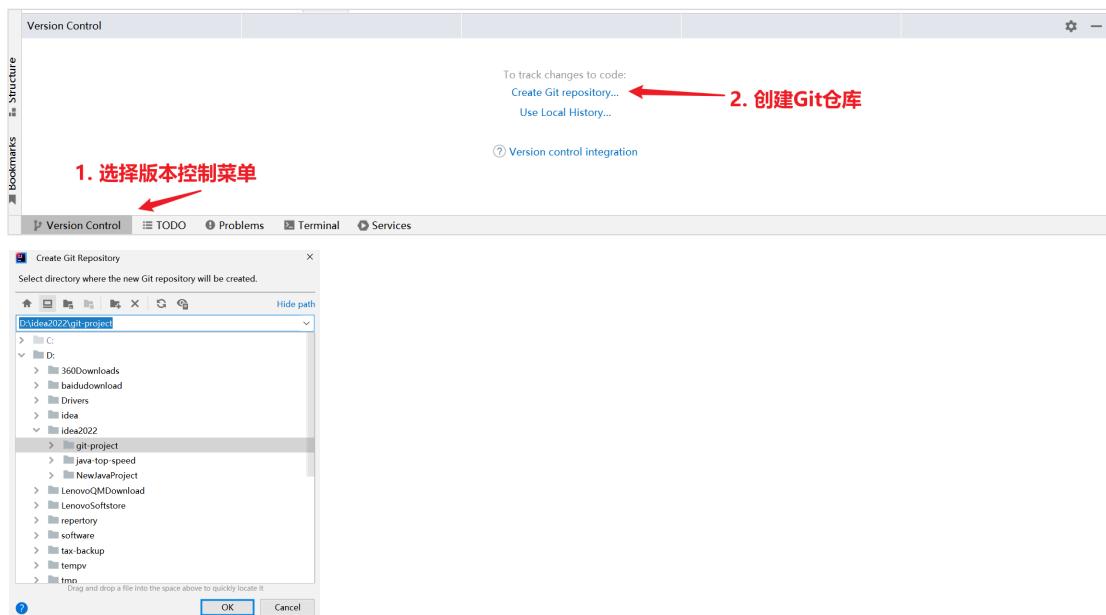
### 4.3.3 创建项目



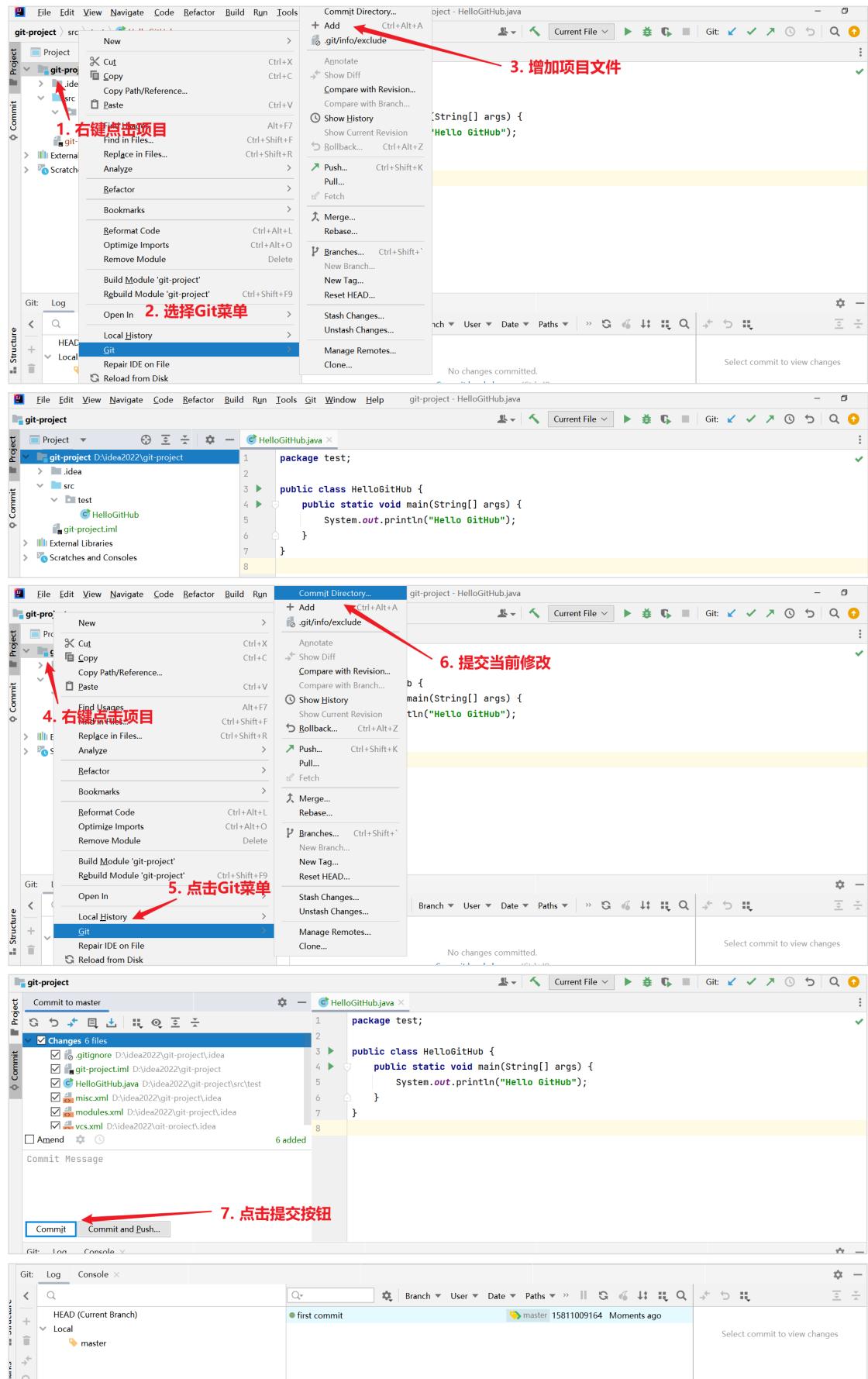
### 4.3.4 添加项目代码



### 4.3.5 创建本地版本库



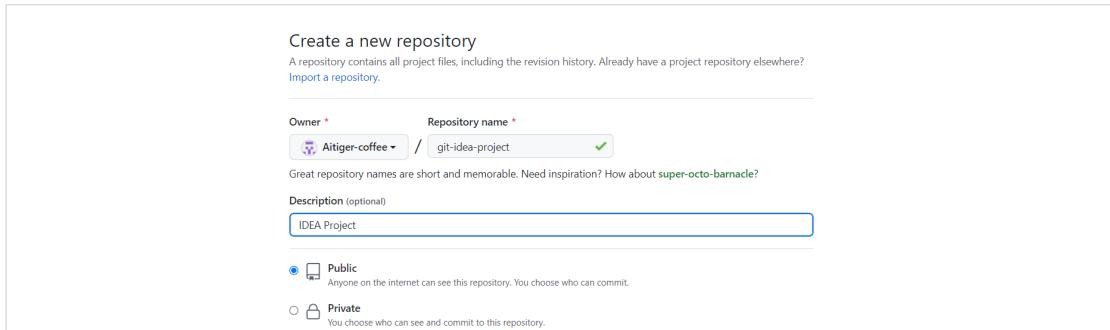
### 4.3.6 提交本地版本库



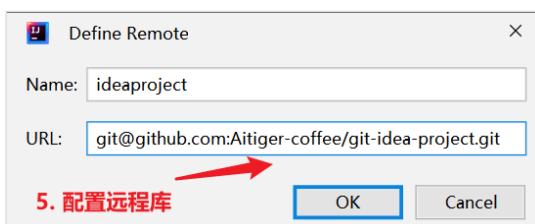
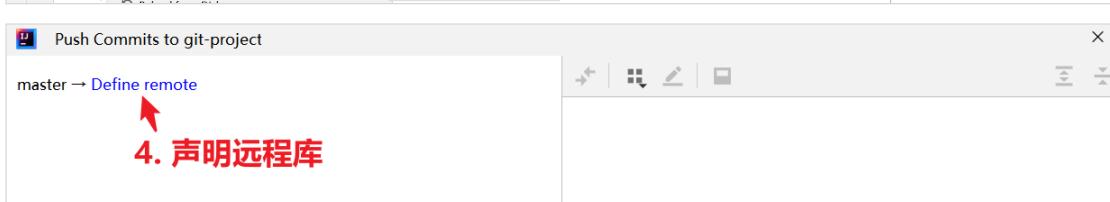
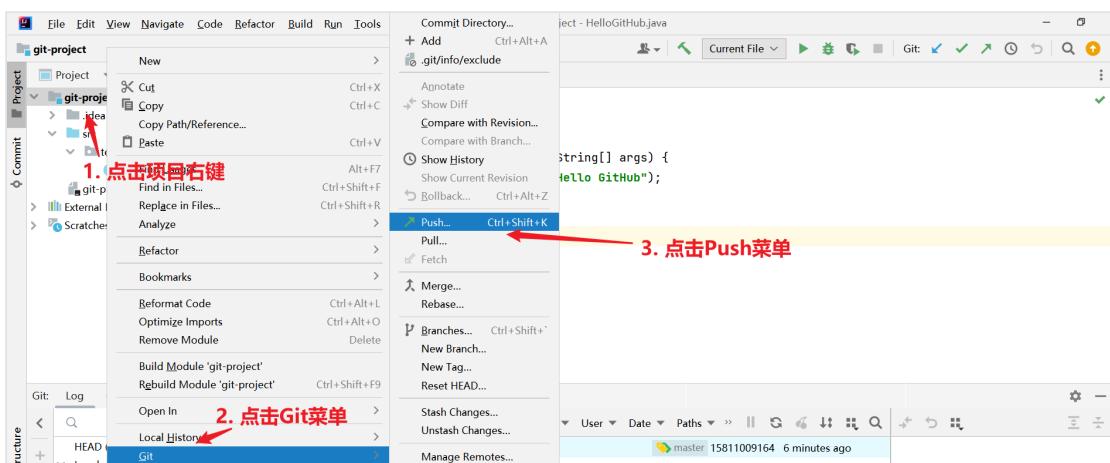
A screenshot of the IntelliJ IDEA interface. The project navigation bar shows a project named 'git-project' containing a 'src' folder with a 'test' subfolder, which contains a file named 'HelloGitHub'. The code editor shows the following Java code:

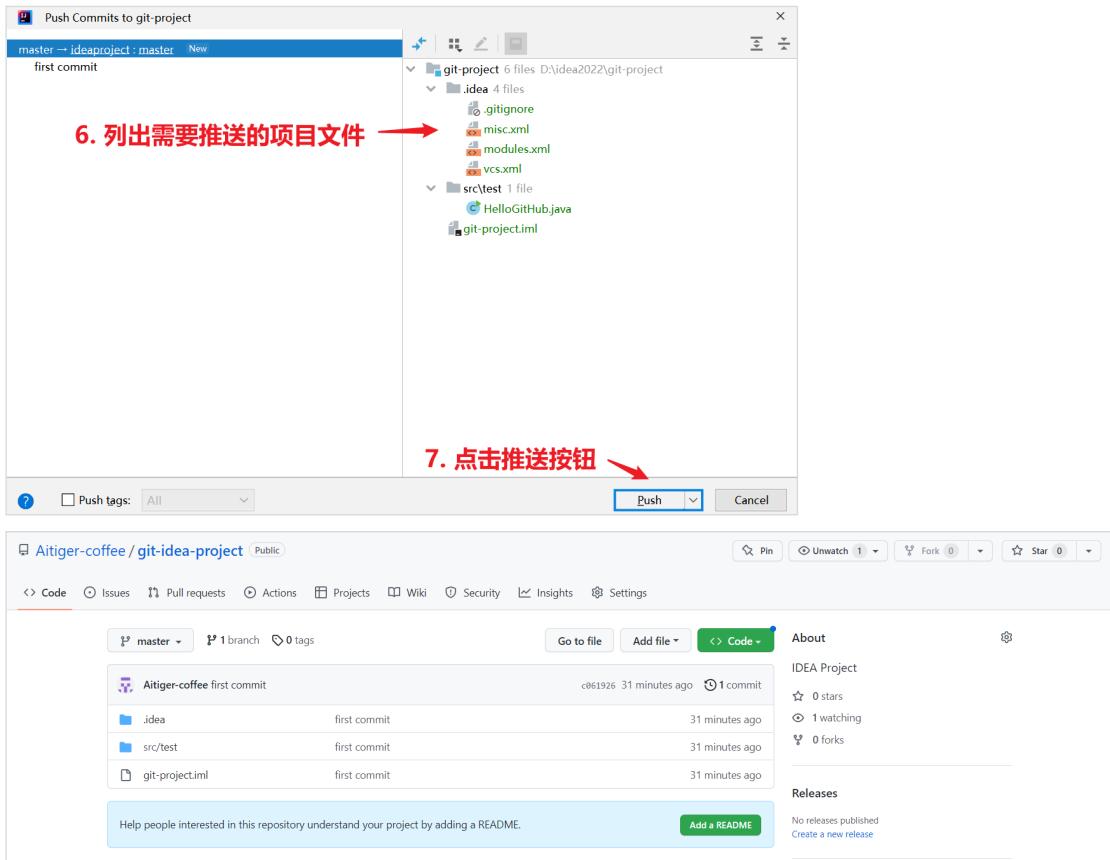
```
package test;
public class HelloGitHub {
    public static void main(String[] args) {
        System.out.println("Hello GitHub");
    }
}
```

#### 4.3.7 创建新的远程版本库

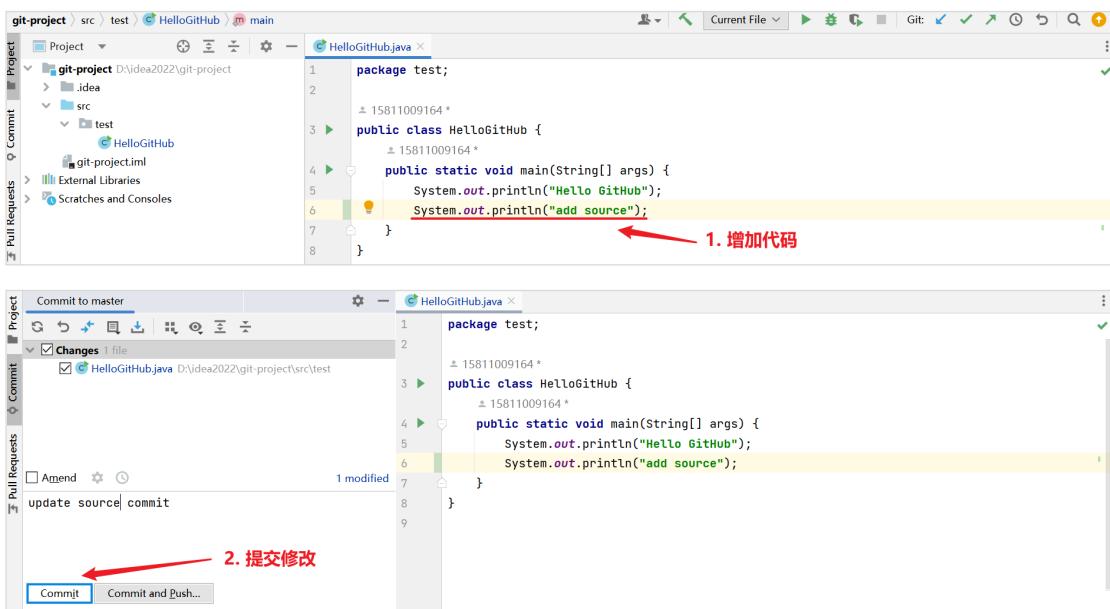


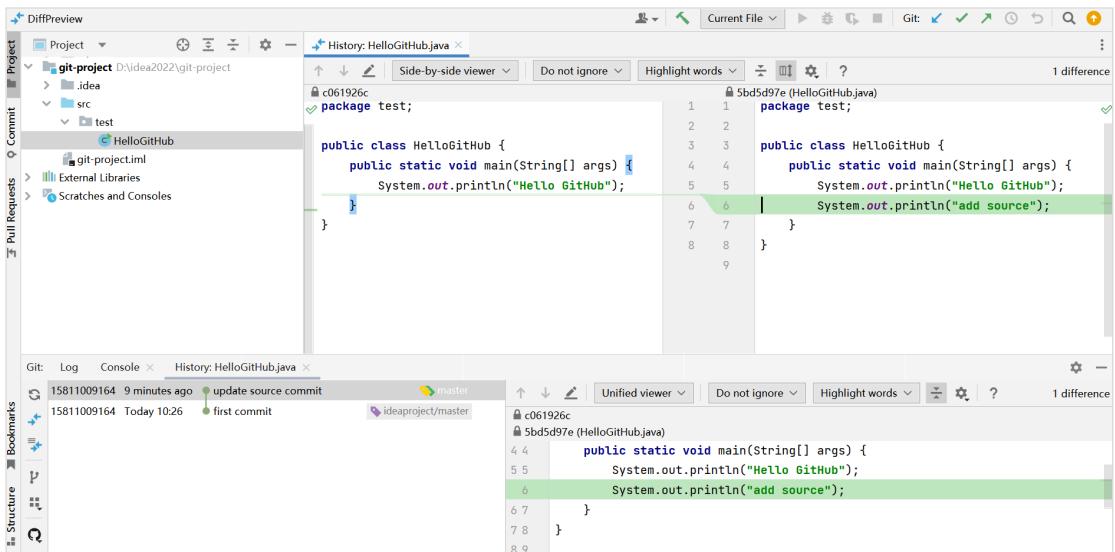
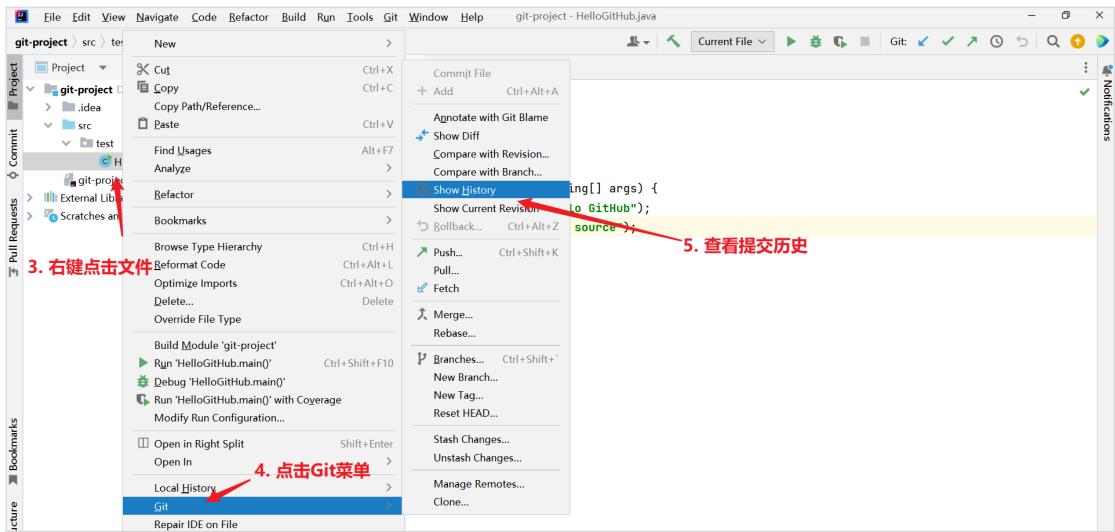
#### 4.3.7 推送到远程版本库





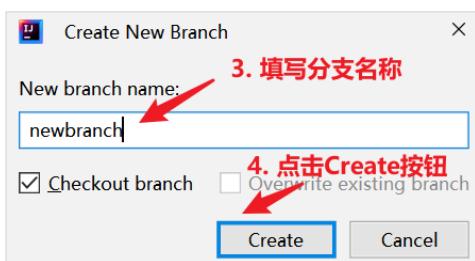
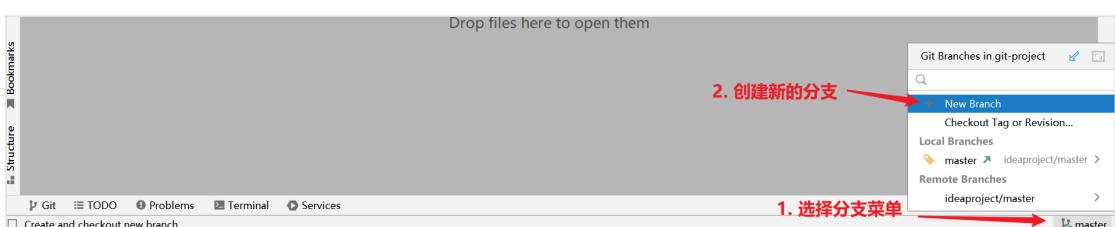
#### 4.3.8 查看历史版本

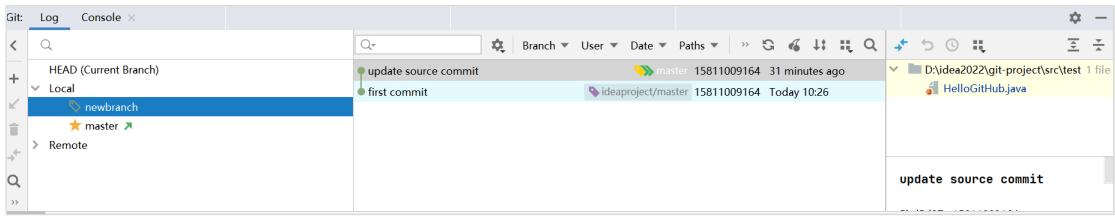




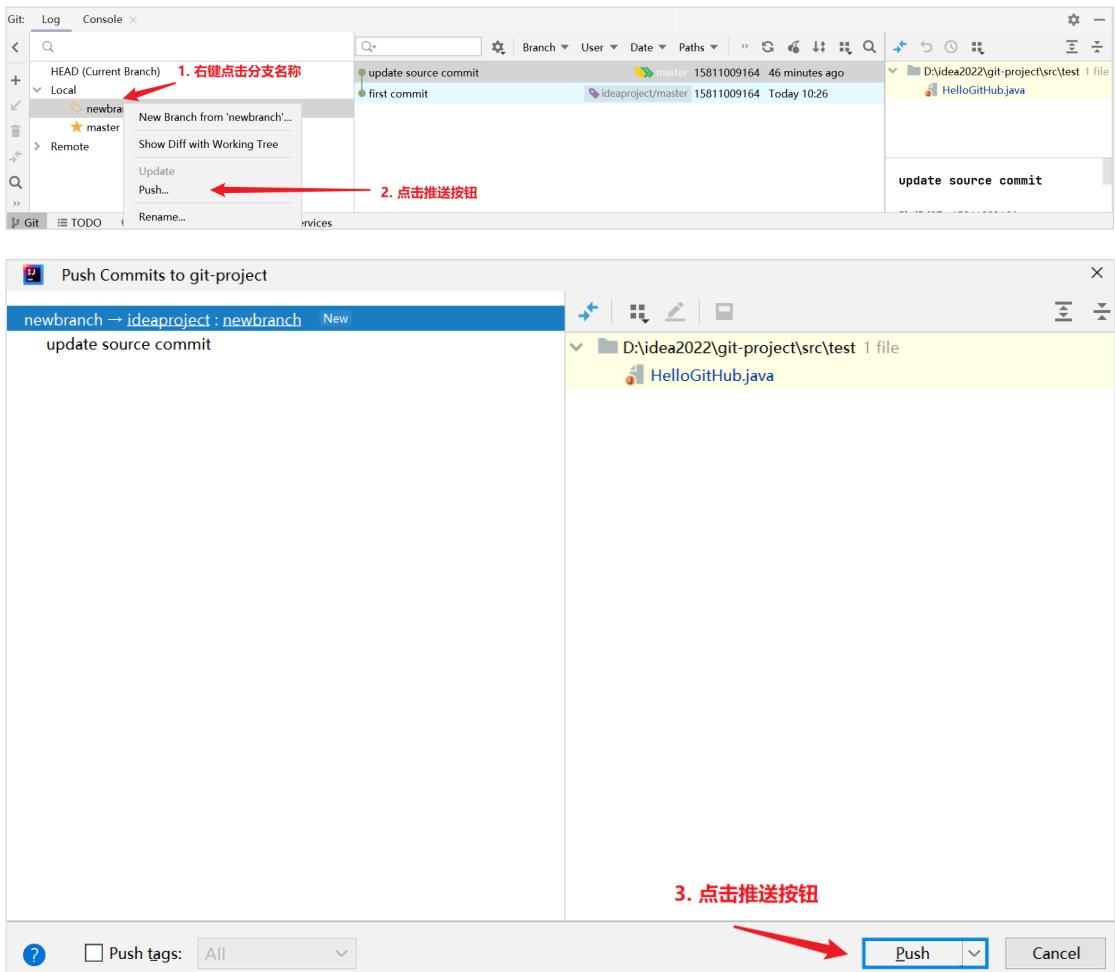
### 4.3.9 分支操作

#### 4.3.9.1 创建分支





#### 4.3.9.2 将分支推送到远程仓库



#### 4.4 Gitee 集成

相对于 GitHub 来讲，由于网络的原因，我们在连接时不是很稳定，所以我们在采用第三方远程仓库时，也可以选择国内的 Gitee 平台。



#### 4.4.1 注册网站会员



#### 4.4.2 用户中心

#### 4.4.3 创建远程仓库

填写仓库名称

填写仓库访问路径

填写仓库介绍

设置仓库权限

点击按钮创建仓库



#### 4.4.4 远程仓库简易操作指令

```
# Git 全局设置，修改成自己的信息
git config --global user.name "Aitiger"
git config --global user.email "12252591+aitiger@user.noreply.gitee.com"
# 创建 git 仓库，基本操作指令和其他远程仓库一致。
mkdir git-study
cd git-study
git init
touch README.md
git add README.md
git commit -m "first commit"
git remote add origin git@gitee.com:aitiger/git-study.git
git push -u origin "master"
# 已有仓库
cd existing_git_repo
git remote add origin git@gitee.com:aitiger/git-study.git
git push -u origin "master"
```

#### 4.4.5 配置 SSH 免密登录

##### 4.4.5.1 本地生成 SSH 密钥

```
# ssh-keygen -t rsa -C Gitee 账号
ssh-keygen -t rsa -C 12252591+aitiger@user.noreply.gitee.com
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/remote-rep
$ ssh-keygen -t rsa -C 12252591+aitiger@user.noreply.gitee.com
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/18801/.ssh/id_rsa):
/c/Users/18801/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/18801/.ssh/id_rsa
Your public key has been saved in /c/Users/18801/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:sBFTRWlKKq0sso7HC7G1L/aBeJz3yMNWAmH0tLWEdIY 12252591+aitiger@user.noreply
.gitee.com
The key's randomart image is:
+--[RSA 3072]----+
| ..oo o...o. |
| Eoo o. o   |
| =ooo o     |
| =o o+.    |
| .o o. S   |
| o*o+.    |
| ==++o.   |
| +=oo=o   |
| +++++o.. |
+---[SHA256]-----+
18801@LAPTOP-J9IRK5BM MINGW64 /e/project/git/remote-rep
$
```

#### 4.4.5.2 集成用户公钥

执行命令完成后,在 window 本地用户.ssh 目录 C:\Users\用户名\.ssh 下面生成如下名称的公钥和私钥:

Windows-SSD (C:) > 用户 > 18801 > .ssh			
名称	修改日期	类型	大小
id_rsa	2022/12/22 17:11	文件	3 KB
id_rsa.pub	2022/12/22 17:11	PUB 文件	1 KB
known_hosts	2022/12/21 23:21	文件	1 KB
known_hosts.old	2022/12/21 23:21	OLD 文件	1 KB

按照操作步骤, 将 id\_rsa.pub 文件内容复制到 Gitee 仓库中

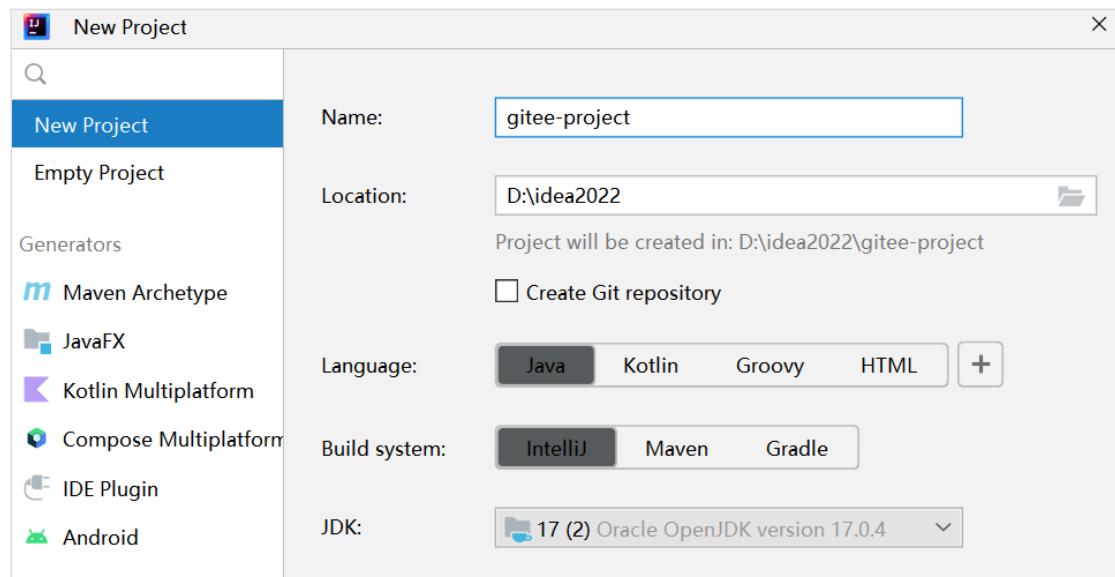
The screenshot shows the Gitee user interface for adding an SSH public key. It includes five numbered steps:

1. 点击设置按钮 → (A) 在侧边栏点击“设置”按钮。
2. 点击SSH公钥设置 → (B) 在侧边栏点击“SSH公钥”。
3. 输入公钥标题 → 在“添加公钥”输入框中输入“idea公钥”。
4. 输入生成的公钥信息 → 将 id\_rsa.pub 文件的内容粘贴到“公钥”输入框中。
5. 点击确定按钮 → 点击“确定”按钮完成操作。

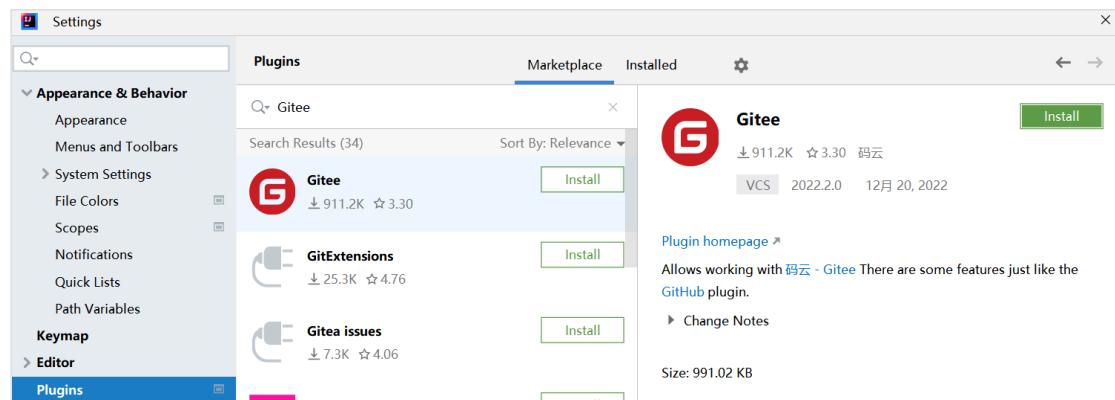
下方显示了“帐号安全验证”对话框，提示“你已成功添加 SSH 公钥”，并有一个绿色的勾形图标。

## 4.4.6 集成 IDEA

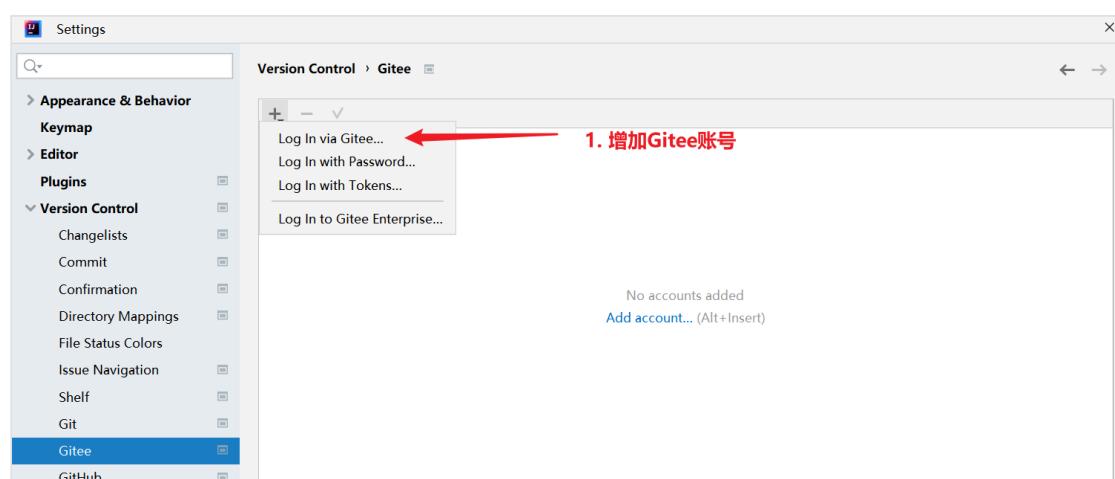
### 4.4.6.1 创建新项目



### 4.4.6.2 安装 Gitee 插件



### 4.4.6.3 配置 Gitee 账户授权



The image shows the Gitee homepage and a login page side-by-side. The homepage features a dark blue header with the Gitee logo and the text '企业级 DevOps 研发管理平台'. Below the header are four large statistics: '800 万+' (Developers), '2000 万+' (Code Repositories), '20 万+' (Enterprise Customers), and '3500+' (Universities). The login page has a '登录' (Login) button and a '短信验证登录' (SMS Verification Login) link. There are also links for '已有帐号, 忘记密码?' (Have account, forgot password?) and '使用 OSChina 帐号登录' (Log in with OSChina account).

The image shows two screenshots of an OAuth authorization request. The first screenshot shows the 'IntelliJ-Gitee 客户端' (IntelliJ-Gitee Client) logo and a list of permissions being requested, including '访问你的个人信息、最新动态等' (Access your personal information, latest news, etc.) and '查看、创建、更新你的项目' (View, create, update your projects). The second screenshot shows a success message: '您已成功授权使用 Gitee, 现在可以关闭此页面.' (You have successfully authorized Gitee, you can now close this page).

The image shows the IntelliJ IDEA settings interface. In the 'Version Control' section, under 'Gitee', there is a list of remotes, including one for 'Aitiger' with the URL 'git@gitee.com:aitiger/git-study.git'.

#### 4.4.6.4 增加远程地址

# 增加远程地址

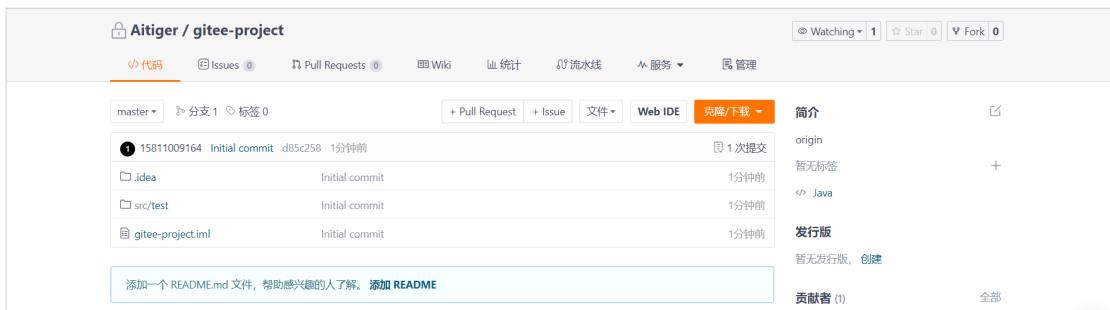
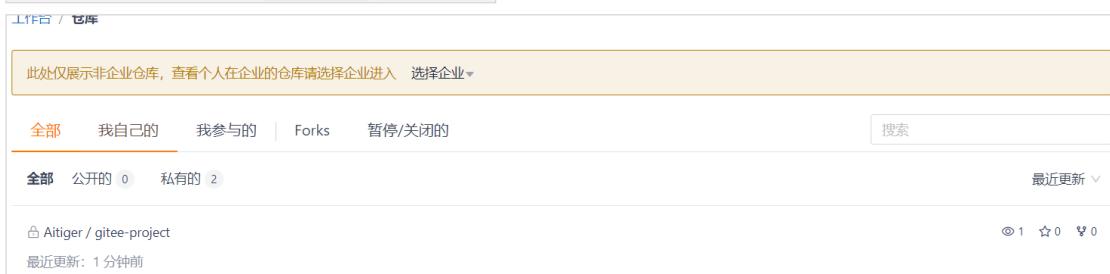
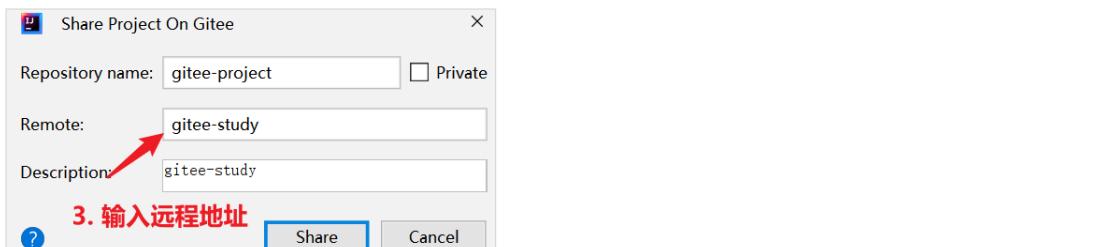
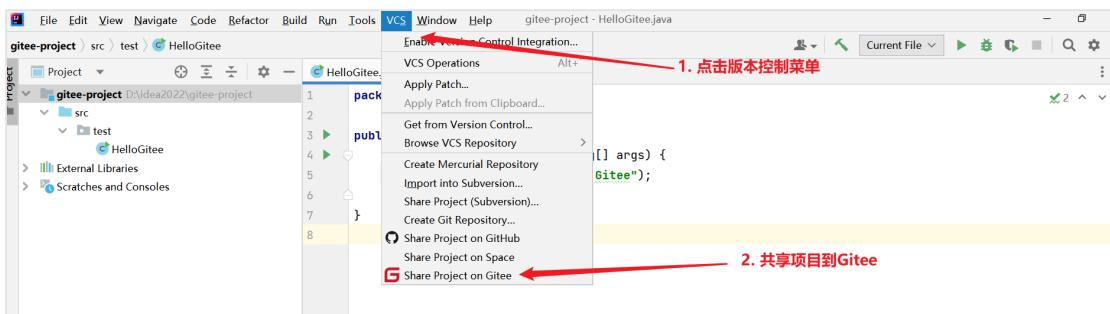
```
git remote add gitee-study git@gitee.com:aitiger/git-study.git
```

2. 输入指令增加远程仓库地址

```
PS D:\idea2022\gitee-project> git remote add gitee-study git@gitee.com:aitiger/git-study.git
```

1. 打开终端面板

#### 4.4.6.5 提交本地代码



## 4.4.7 多用户协作

**1. 点击仓库项目管理**

**2. 点击所有**

**3. 点击添加**

**4. 邀请用户**

**5. 选择直接添加**

**6. 直接输入账号**

**7. 点击添加按钮**

**邀请详情**

序号	用户邮箱	用户名称	个人空间地址	仓库权限	用户状态	邀请结果
1	18801282948@163.com	Evaima	evaima	管理员	Gitee 用户	✓ 已成为仓库成员

**关闭**

## 4.5 GitLab 集成

前面给大家讲解的都是如何使用第三方代码托管平台来管理咱们的代码库。那么我们自己搭建一个这样的平台行不行呢？其实咱们之前已经用 Git 软件搭建了一个远程版本库，但是功能相对来讲，比较单一，而且操作起来也不像 GitHub, Gitee 平台那样更加人性化，所以我们这里介绍一个 GitLab 软件，用于搭建自己的代码托管平台。

### 4.5.1 GitLab 介绍

GitLab 是由 GitLabInc 开发，使用 MIT 许可证的基于网络的 Git 仓库管理工具，且具有 wiki 和 issue 跟踪功能。使用 Git 作为代码管理工具，并在此基础上搭建起来的 Web 服务。GitLab 由乌克兰程序员 DmitriyZaporozhets 和 ValerySizov 开发，它使用 Ruby 语言写成。后来，一些部分用 Go 语言重写。GitLab 被 IBM, Sony, JulichResearchCenter, NASA, Alibab, Invincea, O'ReillyMedia, Leibniz-Rechenzentrum(LRZ), CERN, SpaceX 等组织使用。

### 4.5.2 GitLab 软件下载

官网地址：<https://about.gitlab.com/>

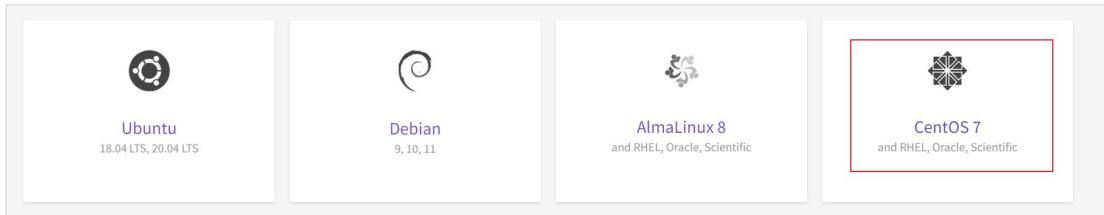
这里我们可以根据个人情况，选择下载不同版本的软件：



我们这里主要是教学，所以下载使用社区版(CE)即可



这里我们选择下载适用 CentOS 7 系统的版本



下载地址: <https://packages.gitlab.com/gitlab/gitlab-ce>

The screenshot shows the GitLab package manager interface. On the left is a sidebar with icons for Cloud, Docs, Help, gitlab (highlighted in blue), Packages (selected), Installation, Search, Mirroring, GPG, and Stats. The main area has a search bar at the top with the placeholder "Search 10054 packages...". Below it is a section titled "Quick install instructions for:" with radio buttons for "Debian" and "RPM". A table titled "Packages" lists several RPM packages for CentOS 7, all pushed by "gitlab" about 2 hours ago. The table columns are "Name", "Distro/Version", and "Uploaded on".

Name	Distro/Version	Uploaded on
gitlab-ce-15.7.0-ce.0.el7.x86_64.rpm	el7	pushed by gitlab about 2 hours ago
gitlab-ce-15.7.0-ce.0.el7.x86_64.rpm	ol7	pushed by gitlab about 2 hours ago
gitlab-ce-15.7.0-ce.0.el8.aarch64.rpm	el8	pushed by gitlab about 2 hours ago
gitlab-ce-15.7.0-ce.0.amazon2.aarch64.rpm	amazon/2	pushed by gitlab about 2 hours ago
gitlab-ce-15.7.0-ce.0.sles15.aarch64.rpm	opensuse/15.3	pushed by gitlab about 2 hours ago
gitlab-ce-15.7.0-ce.0.amazon2.x86_64.rpm	amazon/2	pushed by gitlab about 2 hours ago
gitlab-ce-15.7.0-ce.0.sles15.x86_64.rpm	opensuse/15.3	pushed by gitlab about 2 hours ago

如果下载不了，或下载比较慢，可以根据提示在在 linux 系统中直接采用 wget 指令下载



### 4.5.3 GitLab 安装

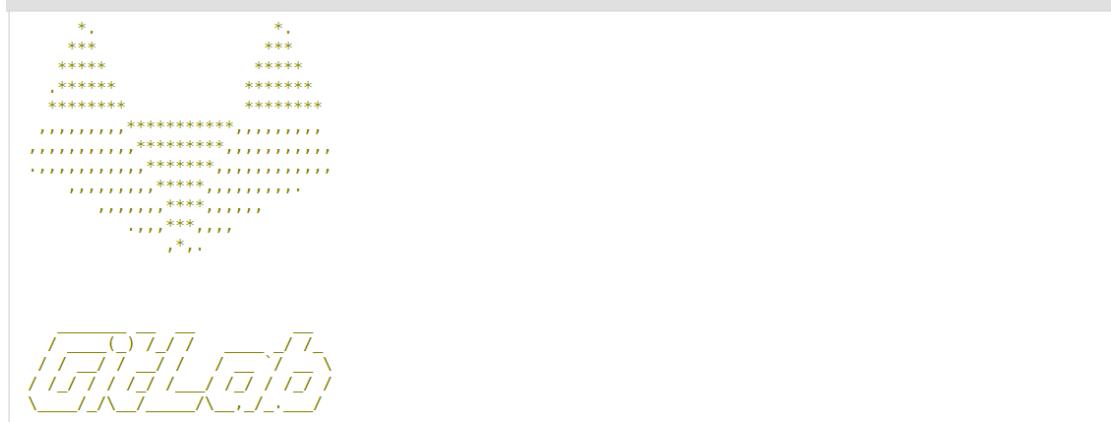
#### 4.5.3.1 安装 linux 系统

Linux 系统的安装不在本课程学习范畴中, 请同学们自行安装 CentOS 7 即可。

#### 4.5.3.2 安装 GitLab

直接采用下载的 RPM 软件包安装即可

```
sudo rpm -ivh /opt/module/software/gitlab-ce-15.7.0-ce.0.el7.x86_64.rpm
```



```
Thank you for installing GitLab!
GitLab was unable to detect a valid hostname for your instance.
Please configure a URL for your GitLab instance by setting `external_url`
configuration in /etc/gitlab/gitlab.rb file.
Then, you can start your GitLab instance by running the following command:
  sudo gitlab-ctl reconfigure
```

#### 4.5.3.3 安装配置依赖项

在 CentOS 7 上，下面的命令也会在系统防火墙中打开 HTTP、HTTPS 和 SSH 访问。这是一个可选步骤，如果您打算仅从本地网络访问极狐 GitLab，则可以跳过它

```
sudo yum install -y curl policycoreutils-python openssh-server perl  
sudo systemctl enable sshd  
sudo systemctl start sshd  
sudo firewall-cmd --permanent --add-service=http  
sudo firewall-cmd --permanent --add-service=https
```

```
sudo systemctl reload firewalld  
# 为了演示方便，我们也可以直接关闭防火墙  
sudo systemctl stop firewalld
```

```
更新完毕:  
  openssh-server.x86_64 0:7.4p1-22.el7_9          perl.x86_64 4:5.16.3-299.el7_9  
  policycoreutils-python.x86_64 0:2.5-34.el7  
  
作为依赖被升级:  
  libsemanage.x86_64 0:2.5-14.el7                libsemanage-python.x86_64 0:2.5-14.el7  
  openssh.x86_64 0:7.4p1-22.el7_9              openssh-clients.x86_64 0:7.4p1-22.el7_9  
  perl-libs.x86_64 4:5.16.3-299.el7_9          policycoreutils.x86_64 0:2.5-34.el7  
  setools-libs.x86_64 0:3.3.8-4.el7
```

#### 4.5.3.4 初始化 GitLab

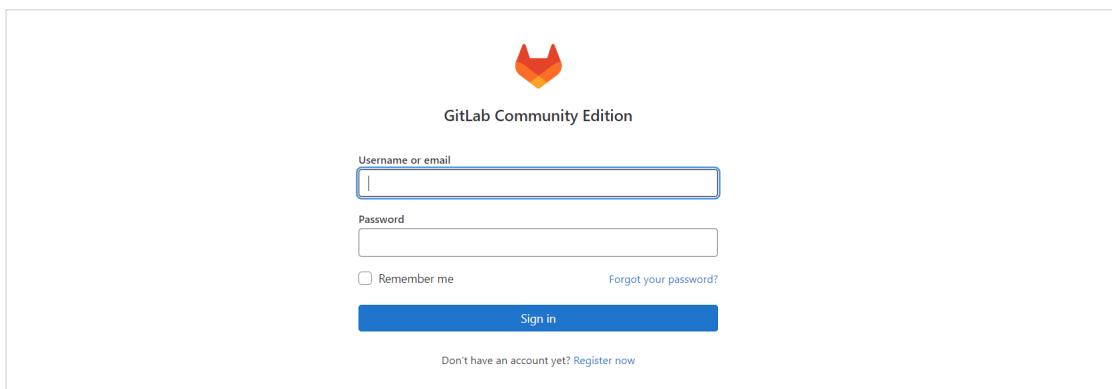
```
# 配置软件镜像  
curl -fsSL https://packages.gitlab.cn/repository/raw/scripts/setup.sh |  
/bin/bash  
  
# 安装  
sudo EXTERNAL_URL="https://linux1" yum install -y gitlab-ce  
  
# 初始化  
sudo gitlab-ctl reconfigure
```

#### 4.5.3.5 启动 GitLab

```
# 启动  
gitlab-ctl start  
  
# 停止  
gitlab-ctl stop
```

#### 4.5.3.6 访问 GitLab

使用浏览器访问 GitLab，输入网址：[http://linux1/users/sign\\_in](http://linux1/users/sign_in)



初始化时，软件会提供默认管理员账户：root，但是密码是随机生成的。

```
Default admin account has been configured with following details:  
Username: root  
Password: You didn't opt-in to print initial root password to STDOUT.  
Password stored to /etc/gitlab/initial_root_password. This file will be cleaned up in first reconfigure run after 24 hours.
```

根据提示，在/etc/gitlab/initial\_root\_password 文件中查找密码

```

initial_root_password
1 # WARNING: This value is valid only in the following conditions
2 #   1. If provided manually (either via `GITLAB_ROOT_PASSWORD` environment variable or via `gitlab_rails['initial_root_pass
3 #   2. Password hasn't been changed manually, either via UI or via command line.
4 #
5 #   If the password shown here doesn't work, you must reset the admin password following https://docs.gitlab.com/ee/securit
6 #
7 Password: iFYe3xqWJdz6wrYSAlefPT9NdBRQ/22aojy27AMX+Ao=
8 #
9 # NOTE: This file will be automatically deleted in the first reconfigure run after 24 hours.
10

```

输入账号，密码，进入系统

The screenshot shows the terminal output of the initial root password, followed by the GitLab web interface. The interface includes a warning about sign-up restrictions, a project named 'GitLab Instance / Monitoring' (Owner), and a navigation bar with 'Projects'.

#### 4.5.3.7 修改密码

默认的密码是随机的，且不容易记忆，还会在系统初始化后 24 小时被删除，所以需要先修改一下密码

The guide consists of three screenshots illustrating the password change process:

- 1. 点击菜单**: Shows the user menu with 'Edit profile' highlighted.
- 2. 选择Password配置面板**: Shows the 'User Settings' sidebar with 'Password' selected.
- 3. 输入当前登录密码**: Shows the 'Edit Password' page with the 'Current password' field highlighted.
- 4. 输入新密码**: Shows the 'New password' field highlighted.
- 5. 输入确认密码**: Shows the 'Password confirmation' field highlighted.
- 6. 保存密码**: Shows the 'Save password' button highlighted.

After saving, a success message appears: 'Password was successfully updated. Please sign in again.'

#### 4.5.3.8 创建项目

1. 点击按钮 → **New project**

2. 选择创建空白项目

3. 填写项目信息

4. 创建项目

Project name: Test

Project URL: http://linux1/root / test

Visibility Level: Public

Project Configuration:

- Initialize repository with a README
- Enable Static Application Security Testing (SAST)

Administrator > Test

Project 'Test' was successfully created.

Test Project ID: 2

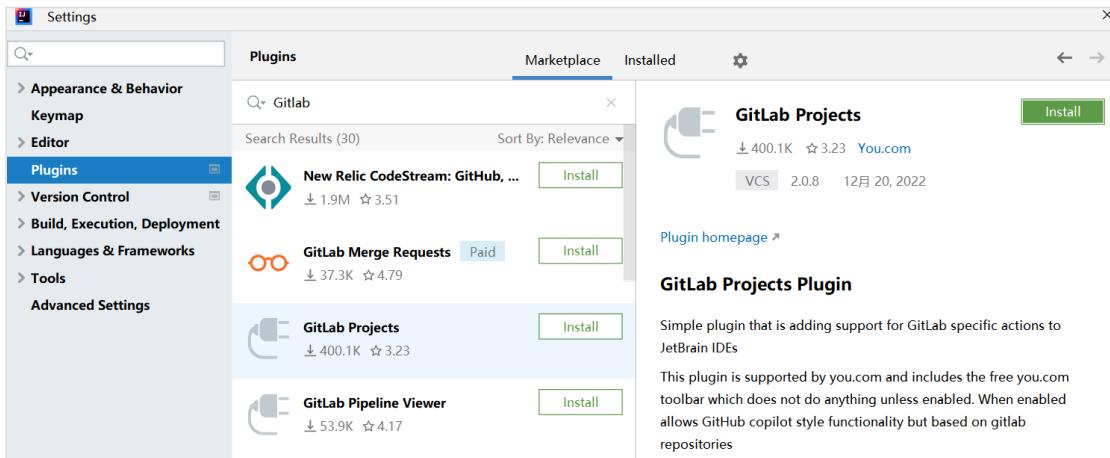
1 Commit 1 Branch 0 Tags 31 KB Project Storage

Initial commit  
Administrator authored just now

Find file Web IDE Clone

## 4.5.4 集成 IDEA

### 4.5.4.1 安装 GitLab 插件

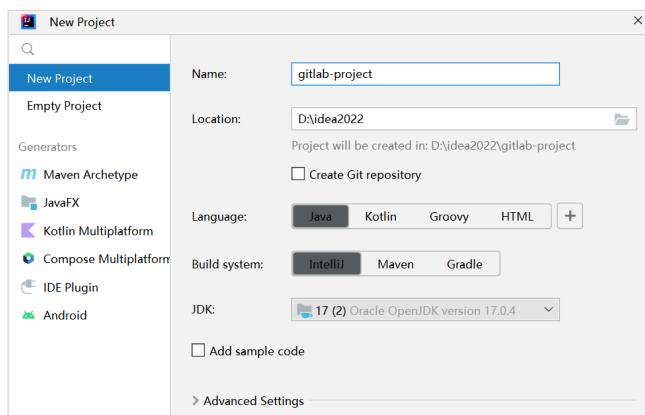


### 4.5.4.2 配置 GitLab

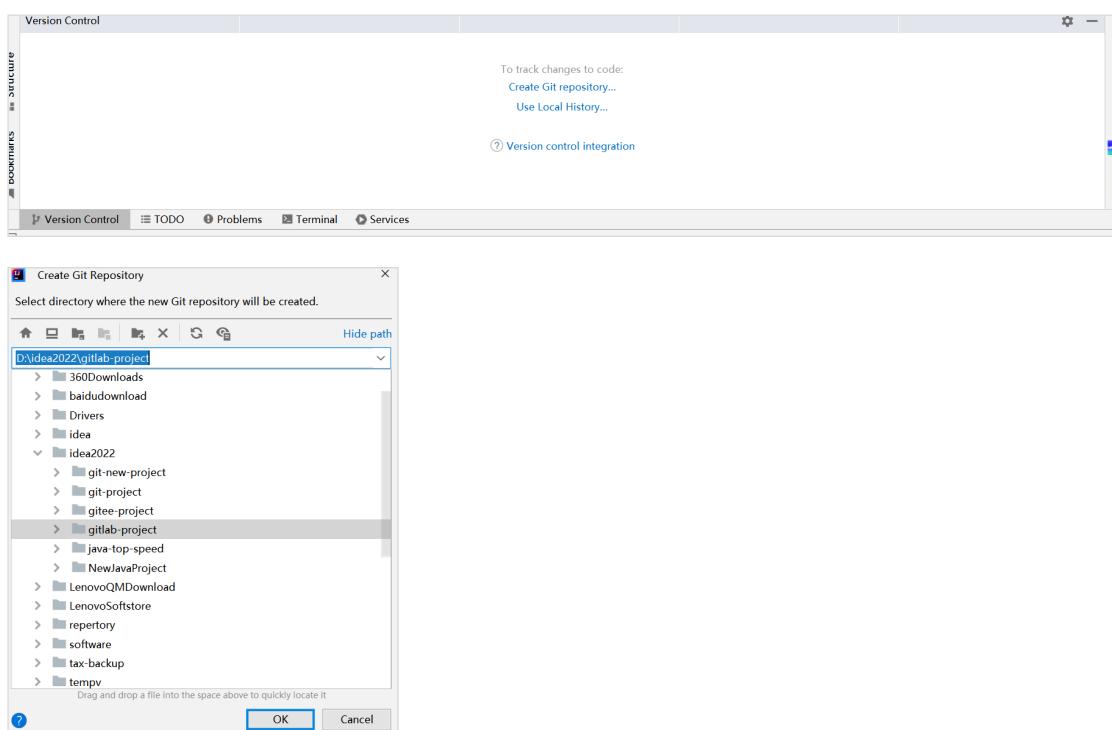
The first part of the image shows the 'Version Control > GitLab' settings page. The 'GitLab' option in the sidebar is highlighted with a red arrow and labeled '1. 选择GitLab'. A red arrow also points to the 'Add New GitLab Server' button at the top right, labeled '2. 点击添加按钮'.

The second part shows the 'GitLab Server Details' dialog. It contains fields for 'GitLab UI Server Url' (with 'http://linux1' entered) and 'Preferred checkout method' (set to 'HTTPS'). Red arrows and labels provide instructions: '1. 输入服务地址' points to the URL field, and '2. 选择HTTPS协议' points to the 'Preferred checkout method' dropdown.

#### 4.5.4.3 创建新项目



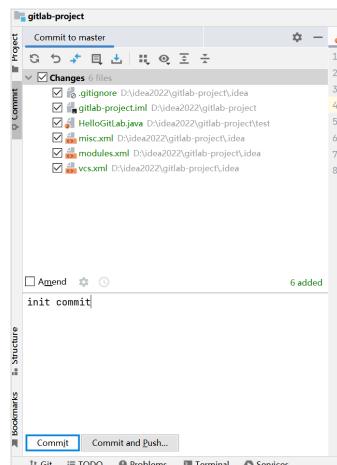
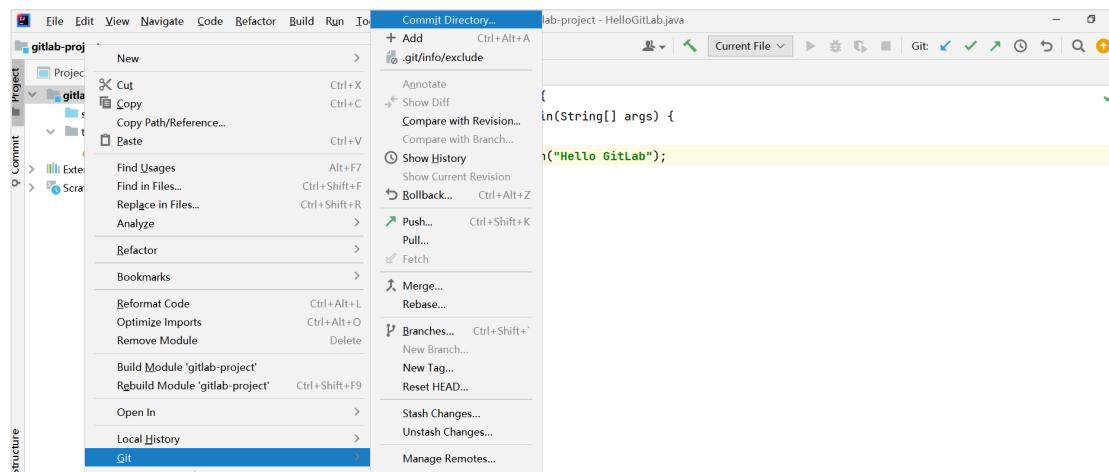
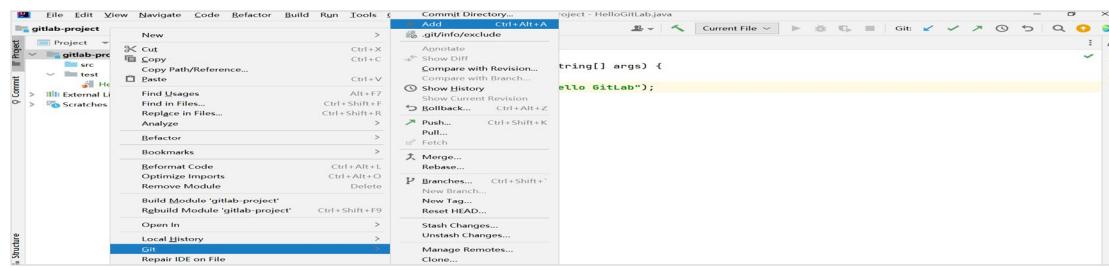
#### 4.5.4.4 创建本地仓库



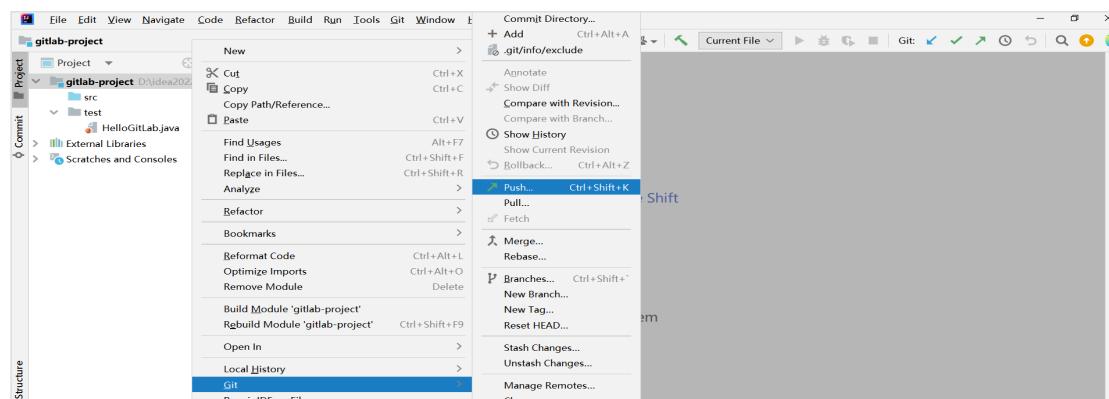
#### 4.5.4.5 创建新代码



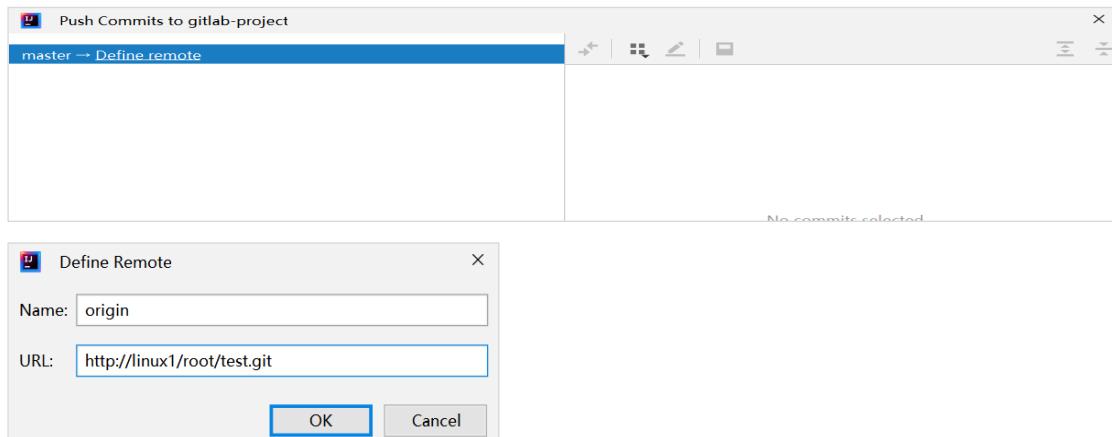
#### 4.5.4.6 提交文件



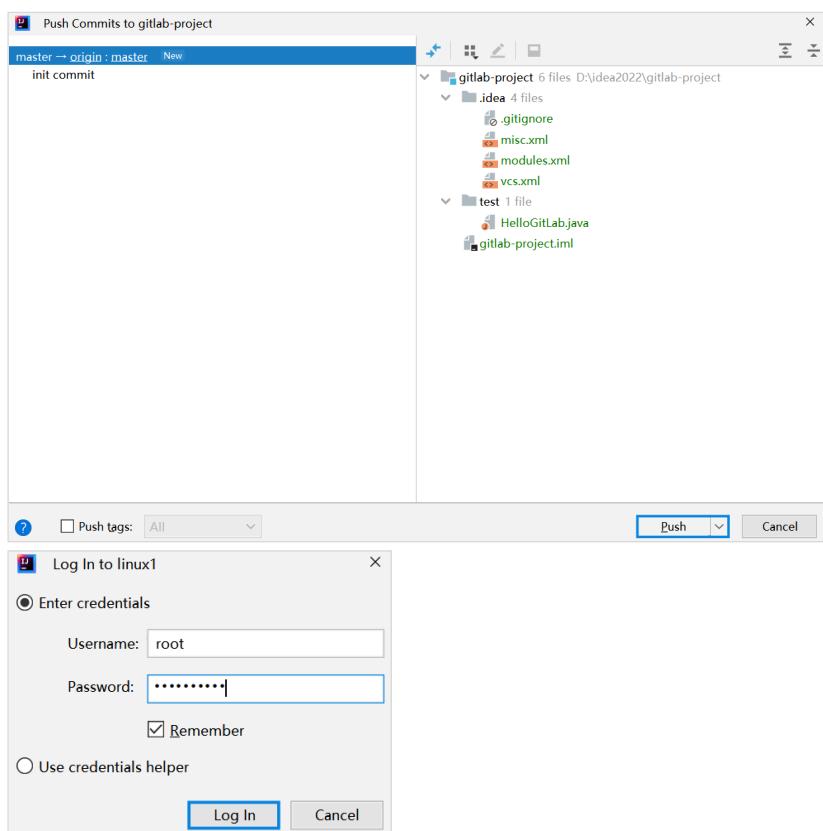
#### 4.5.4.7 推送远程库



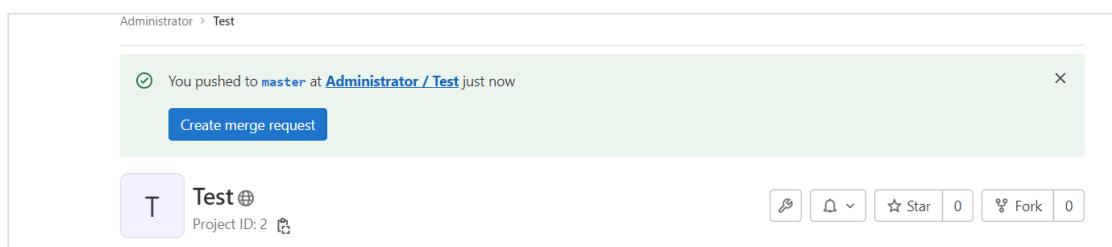
#### 4.5.4.8 配置远程库

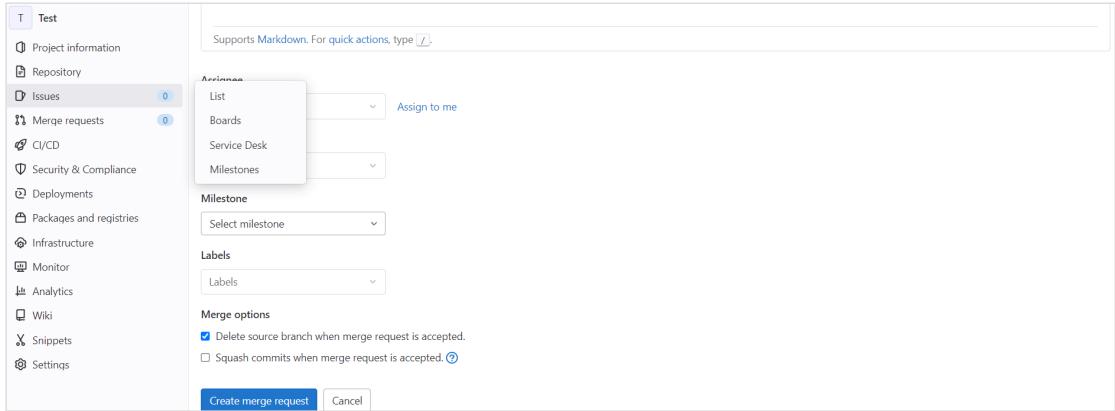


#### 4.5.4.9 推送文件



#### 4.5.4.10 合并提交请求





## 合并

## 确认文件提交

## 4.6 GitHub Desktop 客户端

### 4.6.1 下载 & 安装

#### 4.6.1.1 下载

Git 官网提供对应得下载链接页面:

 **git** --local-branching-on-the-cheap

Search entire site...

**About**

**Documentation**

**Downloads**

- [GUI Clients](#)
- [Logos](#)

**Community**

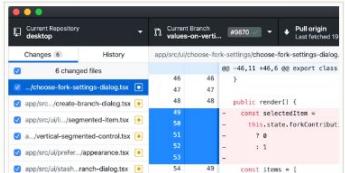
The entire [Pro Git book](#) written by Scott Chacon and Ben Straub is available to read [online for free](#). Dead tree versions are available on [Amazon.com](#).

## GUI Clients

Git comes with built-in GUI tools for committing ([git-gui](#)) and browsing ([gitk](#)), but there are several third-party tools for users looking for platform-specific experience.

If you want to add another GUI tool to this list, just [follow the instructions](#).

All Windows Mac Linux Android iOS



GitHub Desktop

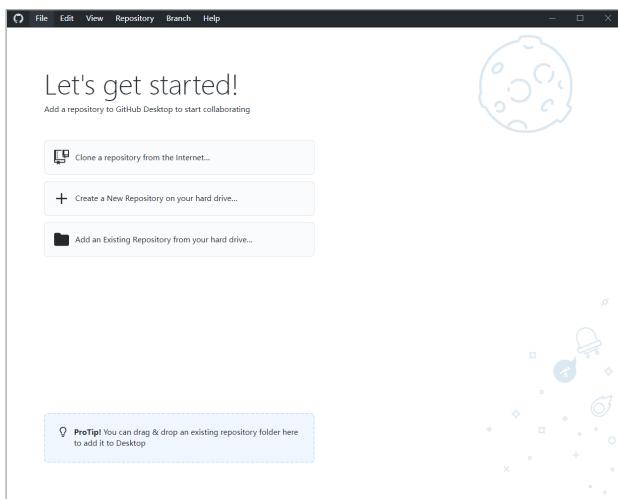


SourceTree

下载地址：<https://central.github.com/deployments/desktop/desktop/latest/win32s>

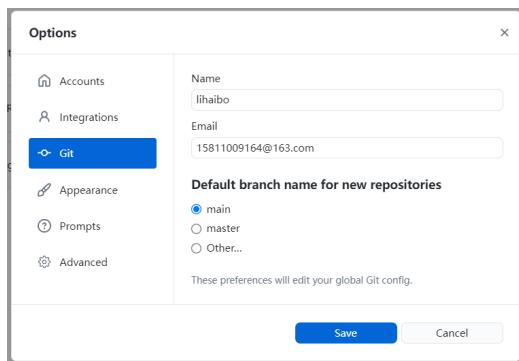
### 4.6.1.2 安装

无安装过程，安装完成后，弹出应用界面



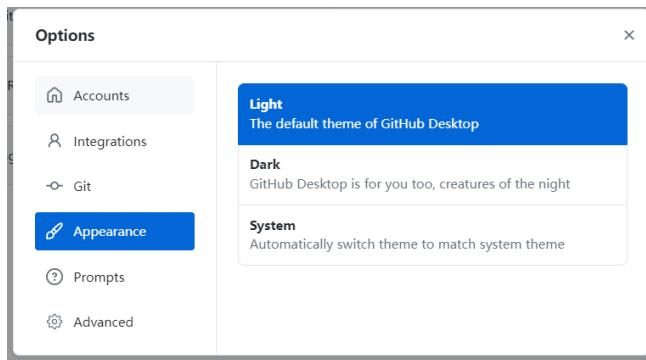
### 4.6.1.3 配置

点击软件得 File 菜单后，选择 Options，设定软件得操作用户名称及对应得邮箱地址。



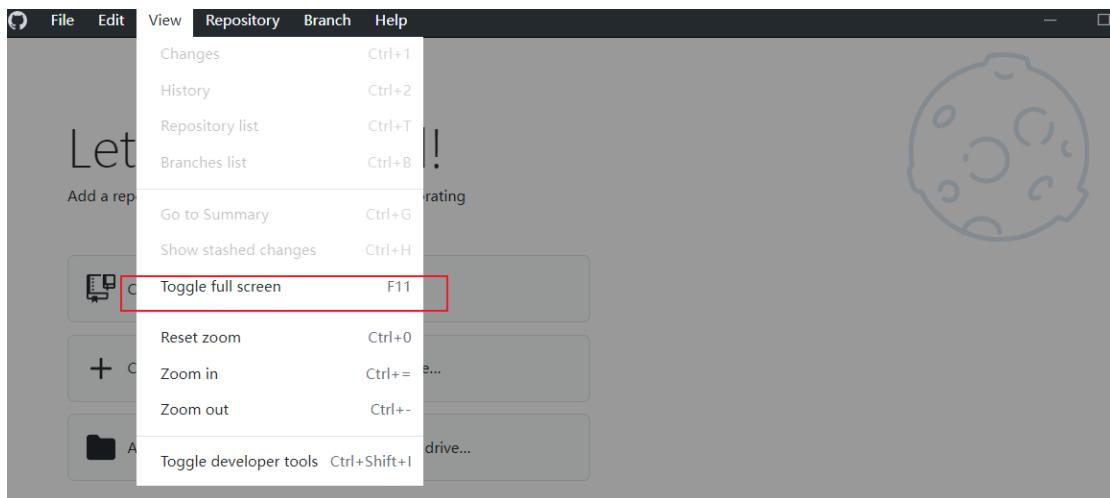
#### 4.6.1.4 主题样式

可以根据自己得偏好设定软件主题样式。



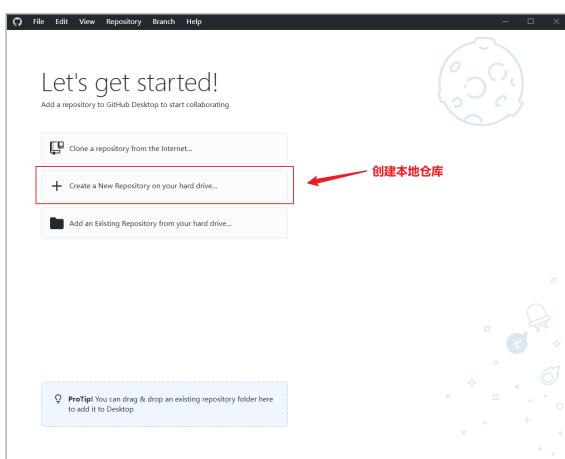
#### 4.6.1.5 全屏

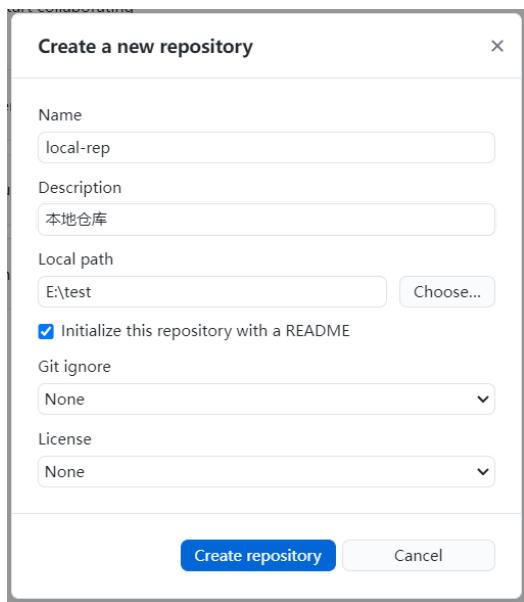
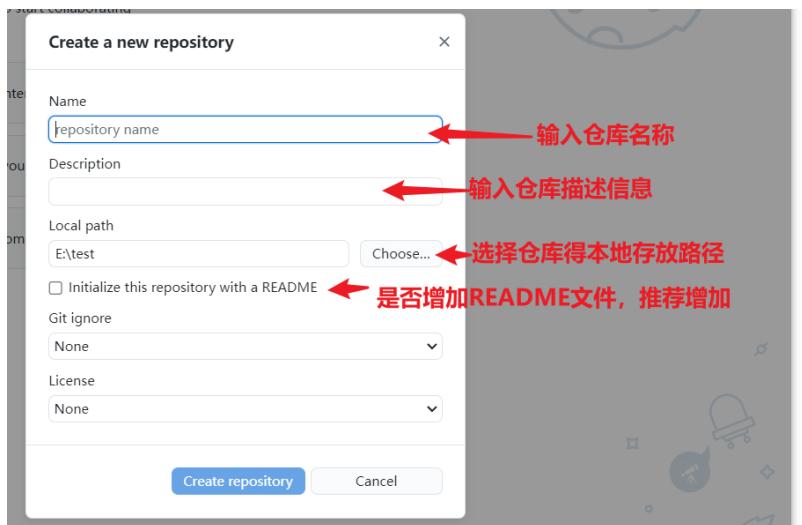
如果觉得软件界面比较小，可以适当进行调整或全屏

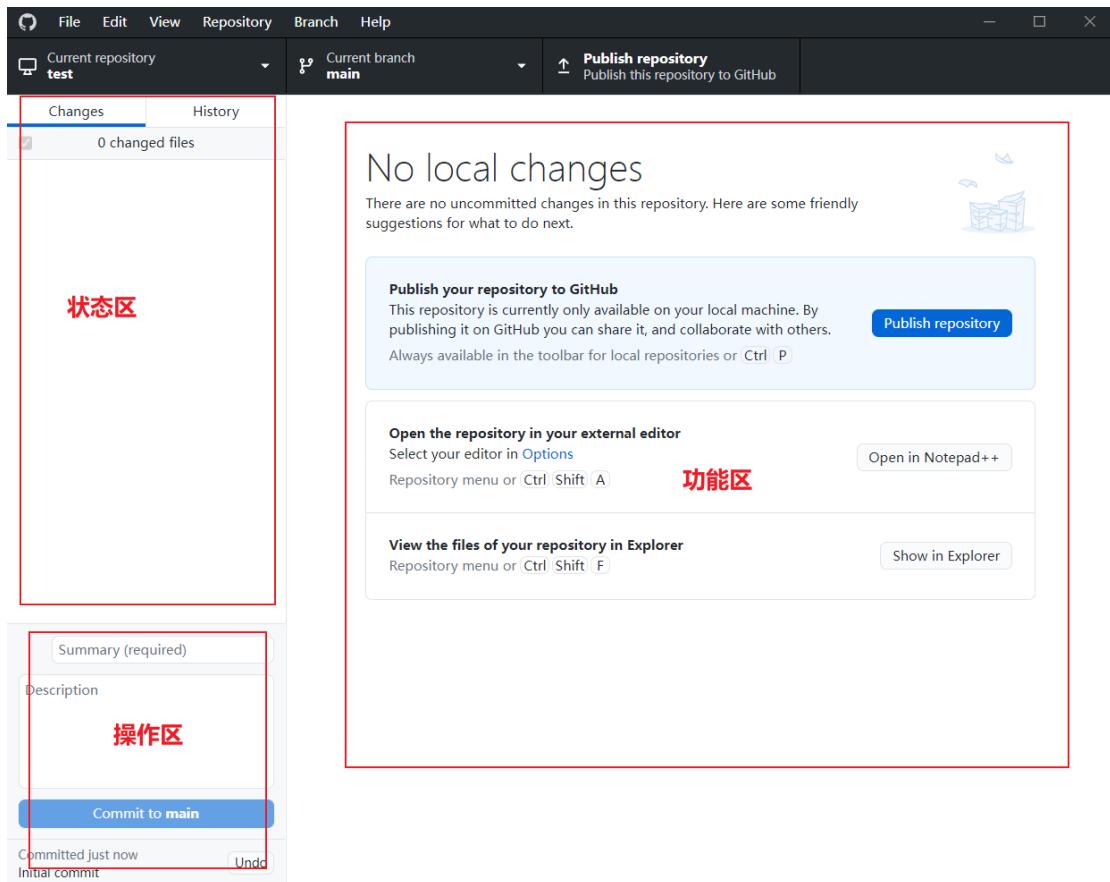


### 4.6.2 创建本地仓库

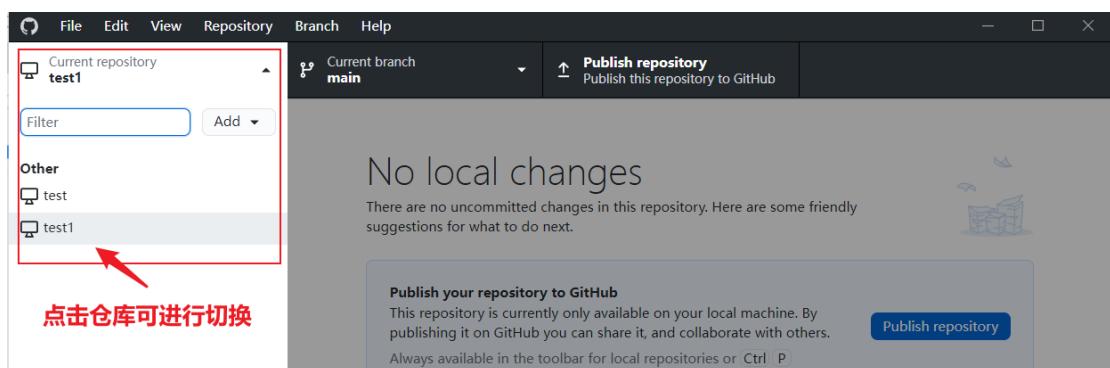
#### 4.6.2.1 创建仓库



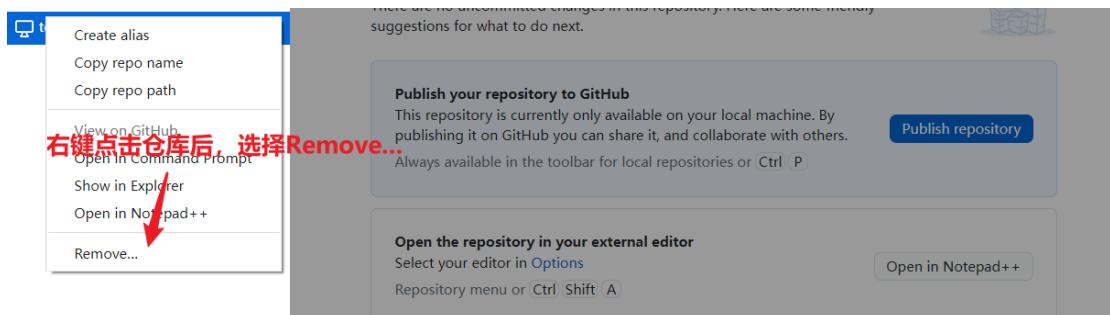


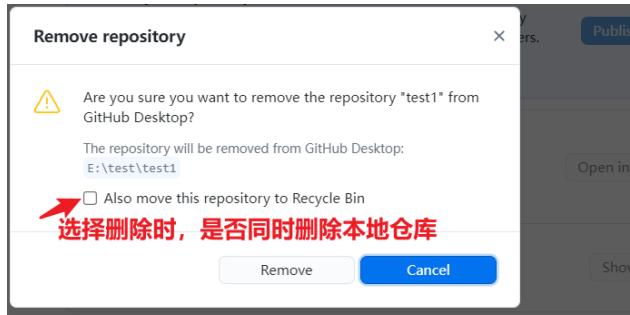


#### 4.6.2.2 切换仓库



#### 4.6.2.3 删除仓库

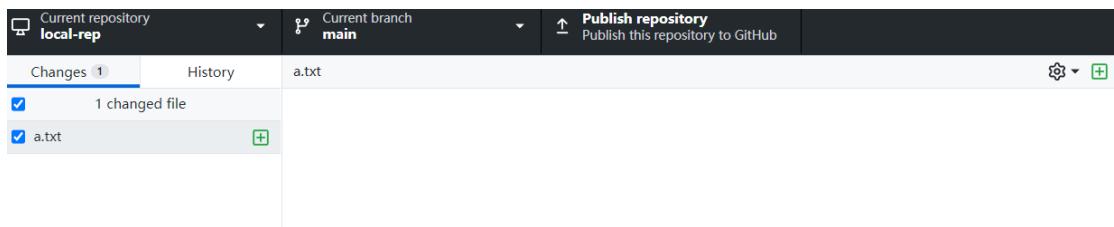




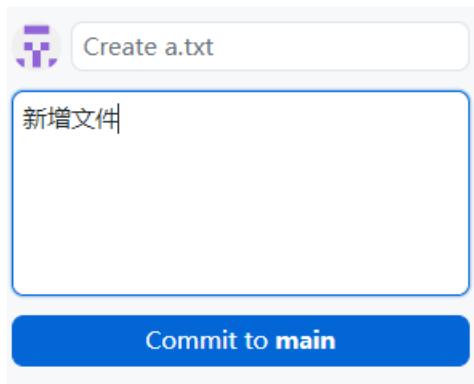
### 4.6.3 文件操作

#### 4.6.3.1 新增文件

当工作区域创建了一份新文件，工具可以自动识别并进行对应得显示

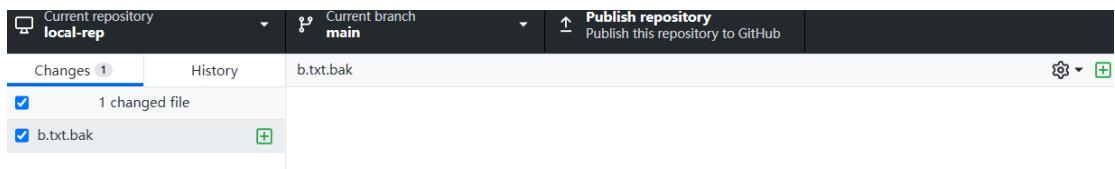


此时 Git 仓库中并没有这份文件，所以需要执行 commit 操作，将文件保存到 Git 仓库中。



#### 4.6.3.2 忽略文件

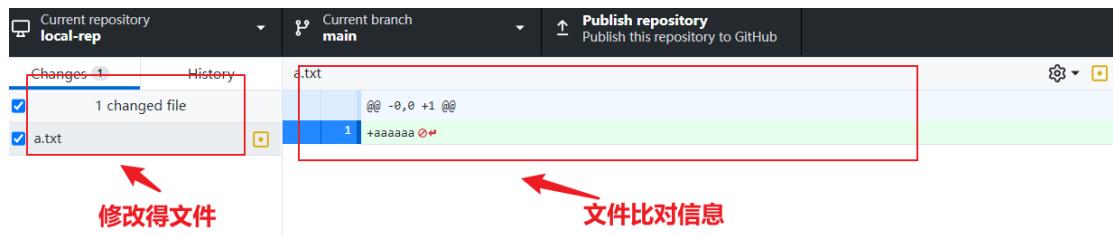
如果某一个文件或某一类得文件，不想被 Git 软件进行管理。可以在忽略文件中进行设定





#### 4.6.3.3 修改文件

修改文件只是将工作区域得文件进行修改，但是对于 Git 软件来讲，其实本质上还是提交，因为底层会生成新得文件



#### 4.6.3.4 删除文件

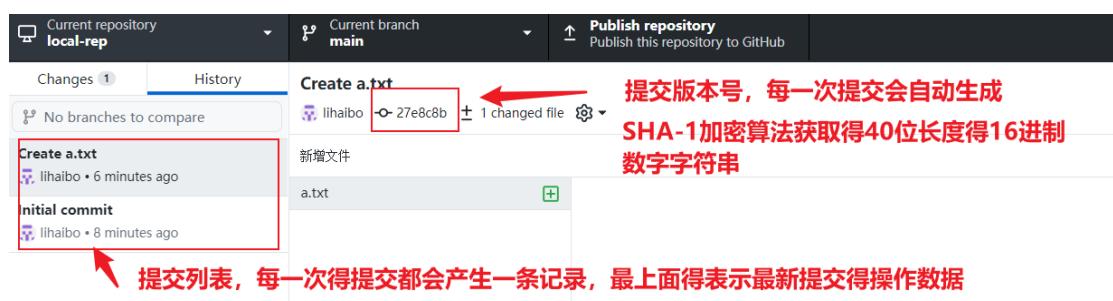
删除文件对于 Git 软件来讲，依然是一个提交



提交后，最新版本得文件也会被“删除”

#### 4.6.3.5 历史记录

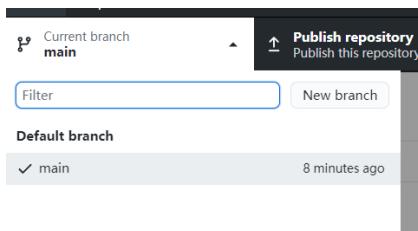
如果存在多次得提交操作得话，可以查看提交得历史记录



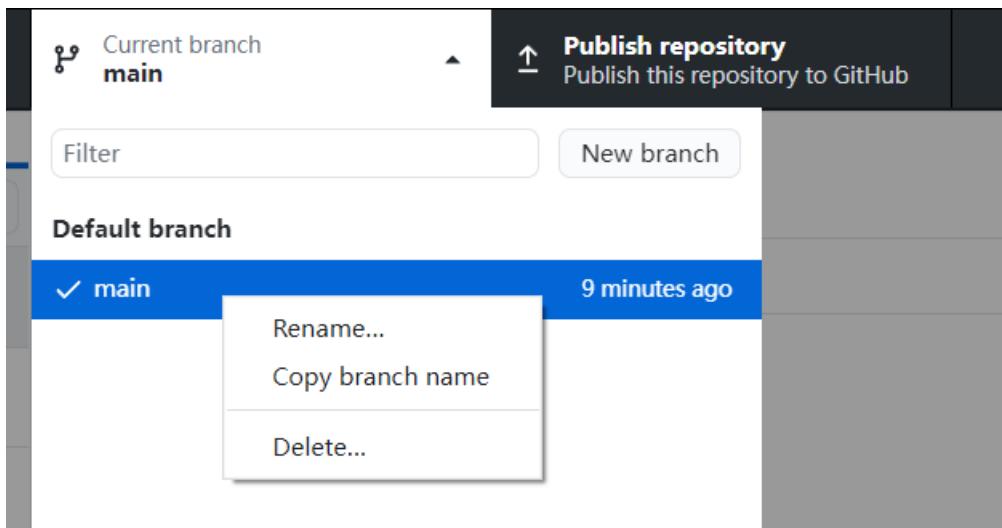
#### 4.6.4 分支操作

##### 4.6.4.1 默认分支

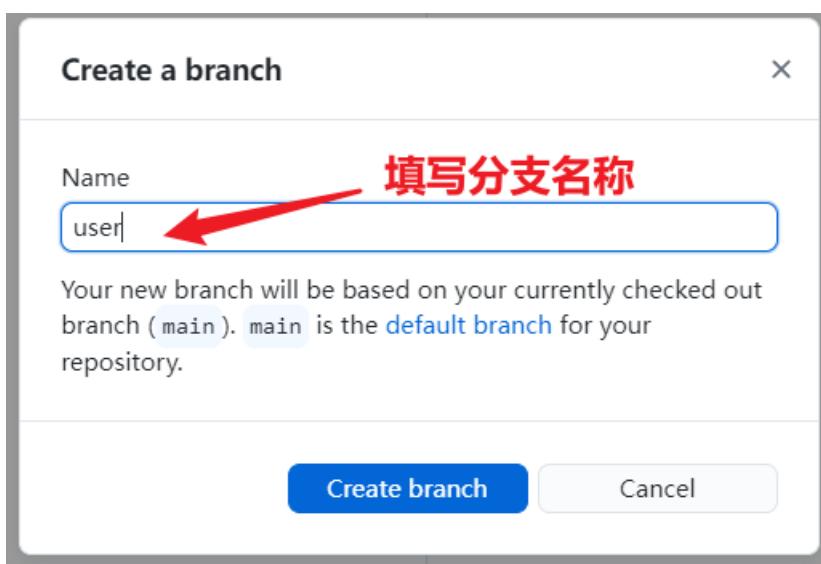
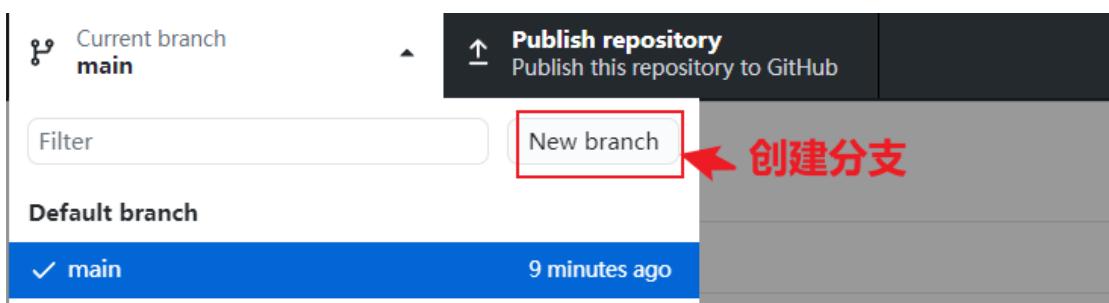
软件创建仓库时，默认创建得分支为 main



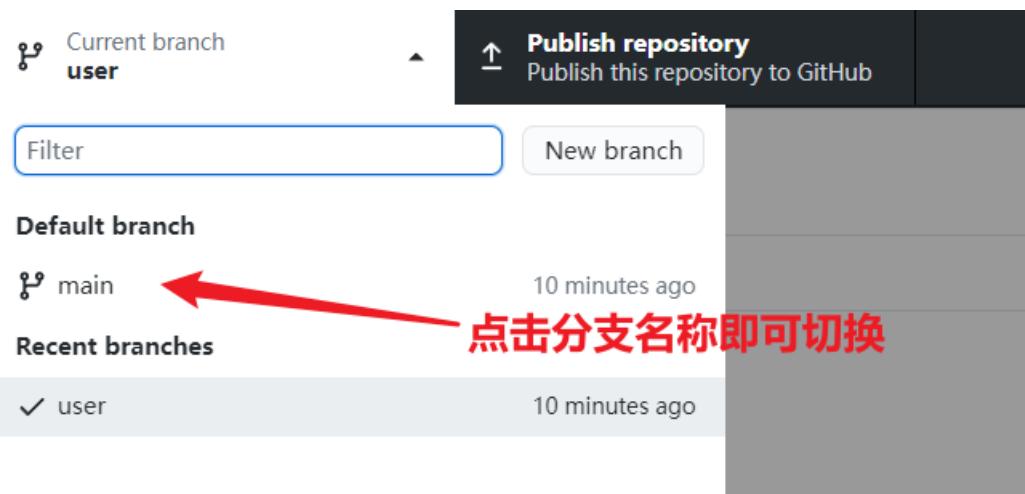
点击右键可以改名



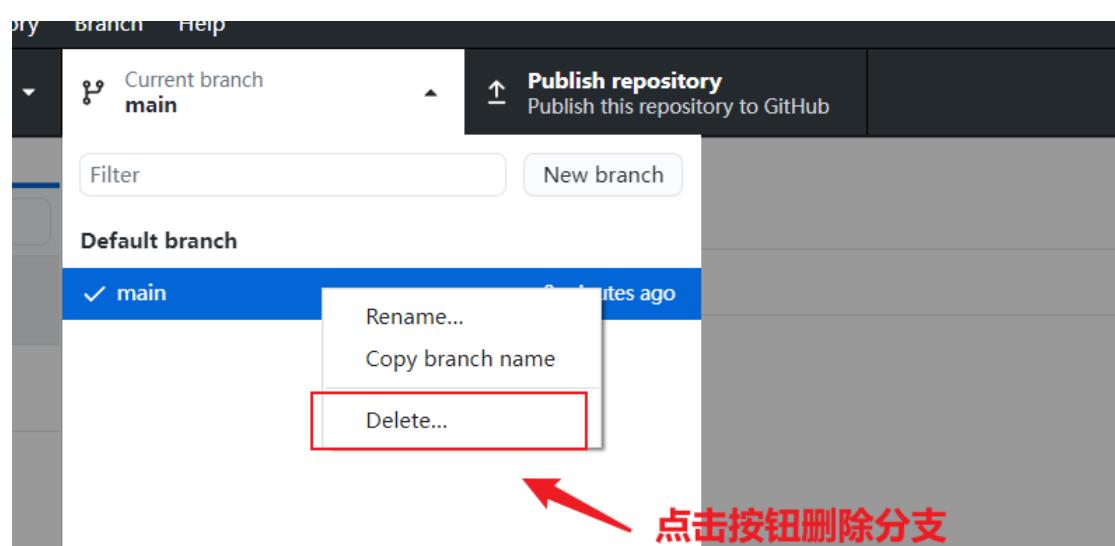
#### 4.6.4.2 创建分支



#### 4.6.4.3 切换分支



#### 4.6.4.4 删除分支



#### 4.6.4.5 合并分支

The screenshot shows a GitHub repository interface. At the top, it displays the current branch as "main". There is a prominent "Publish repository" button. Below the header, there is a search bar labeled "Filter" and a "New branch" button. The main area is divided into sections: "Default branch" and "Recent branches". The "Default branch" section shows a single entry: "main" with a checkmark, updated "11 minutes ago". The "Recent branches" section shows another entry: "user" with a gear icon, updated "11 minutes ago". At the bottom, there is a red callout with the text "点击后可合并分支" (Click here to merge branches) pointing to a dropdown menu. The dropdown menu contains the instruction "Choose a branch to merge into main".

Current branch  
main

Publish repository  
Publish this repository

Filter New branch

**Default branch**

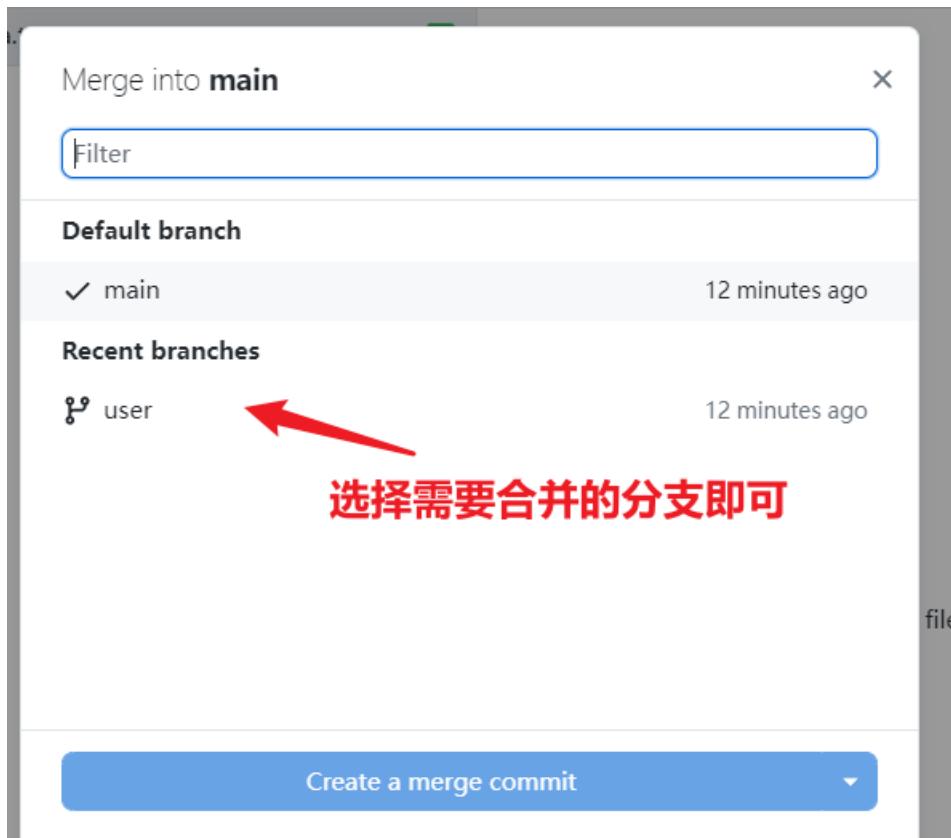
✓ main 11 minutes ago

**Recent branches**

gear user 11 minutes ago

点击后可合并分支

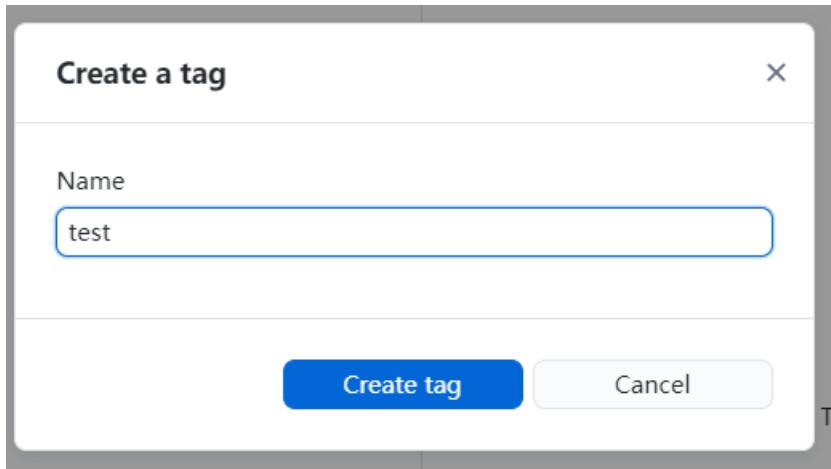
Choose a branch to merge into main



## 4.6.5 标签操作

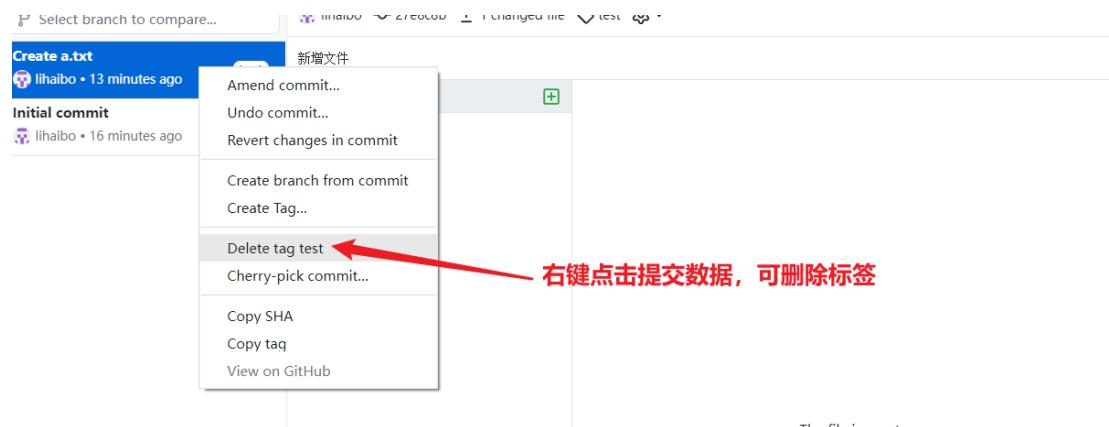
### 4.6.5.1 创建标签





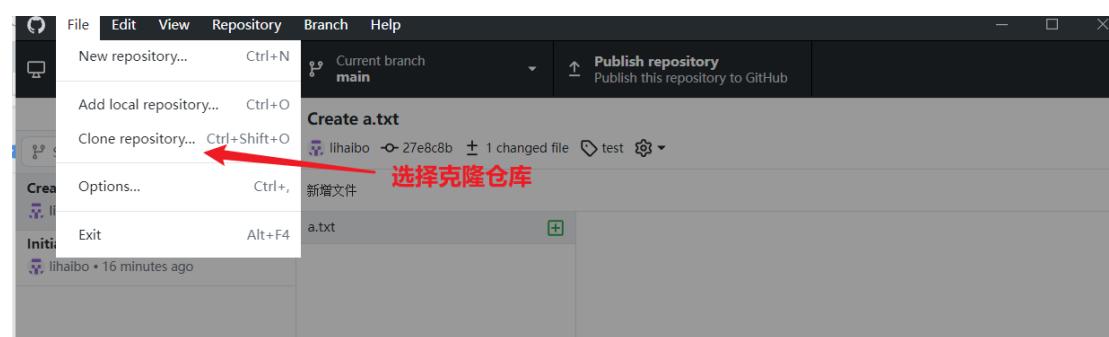
Select branch to compare...  
**Create a.txt**  
lihaibo • 13 minutes ago  
**Initial commit**  
lihaibo • 15 minutes ago  
这就是标签

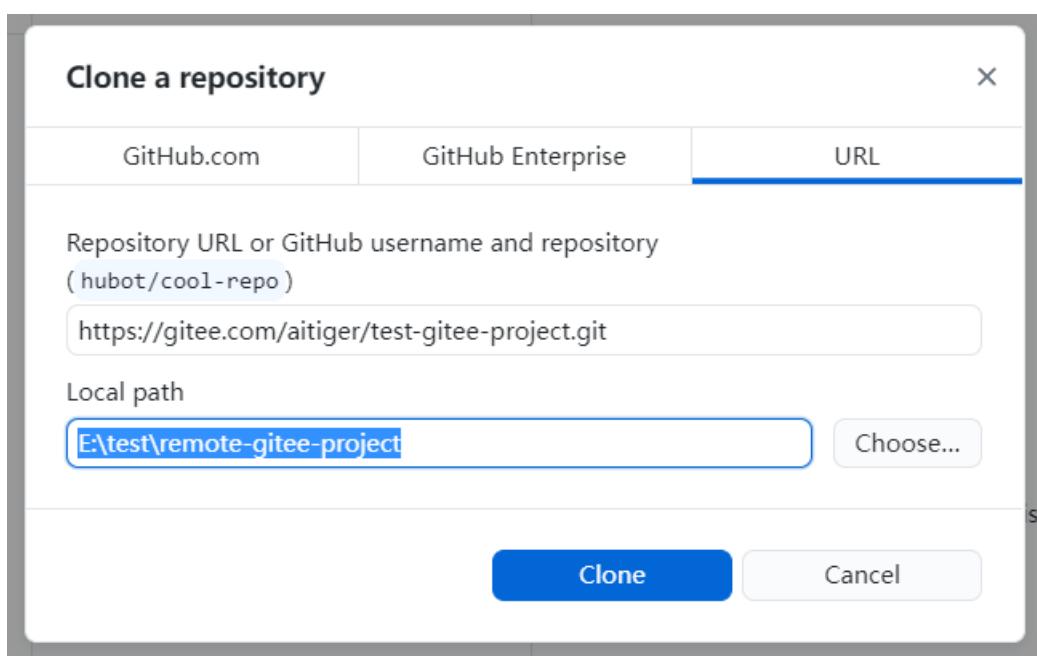
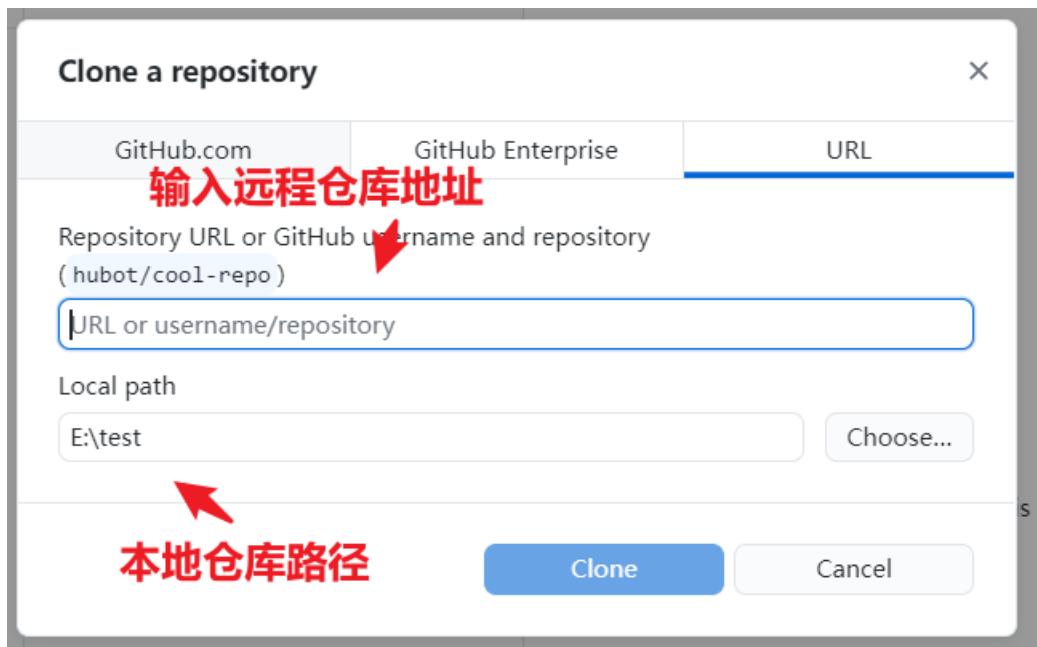
#### 4.6.5.2 删 除 标 签



#### 4.6.6 远程仓库

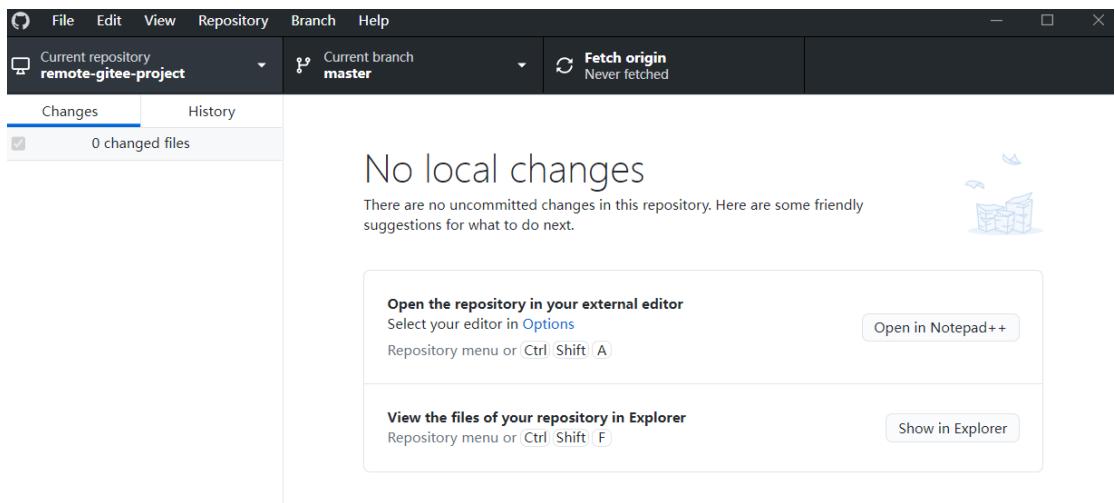
##### 4.6.6.1 克隆仓库





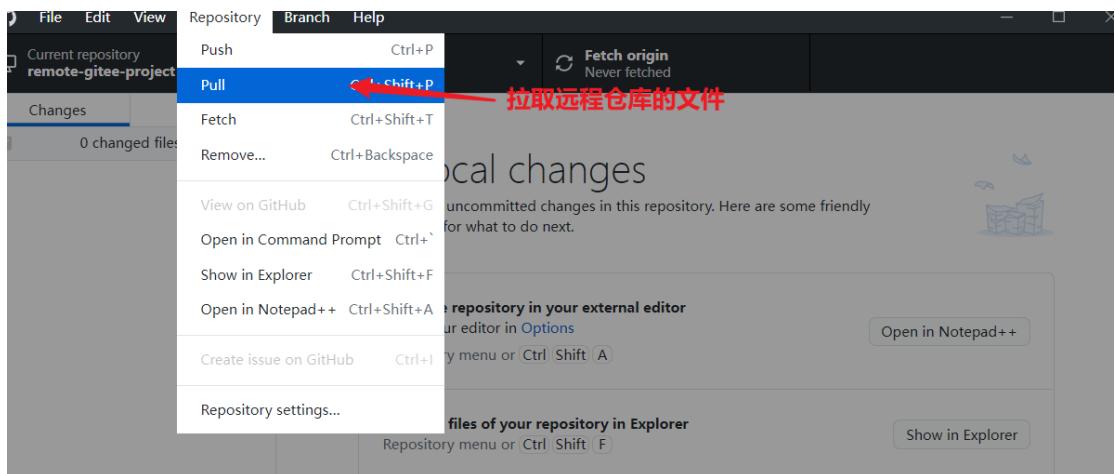
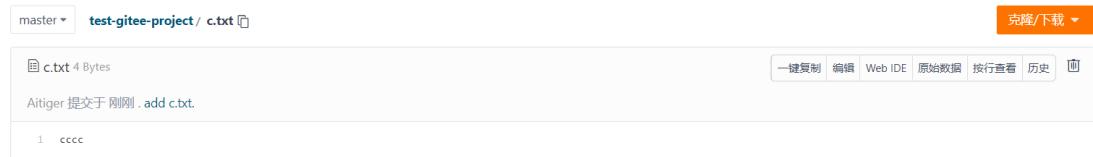
Cloning test-gitee-project

Cloning into 'E:\test\remote-gitee-project'...

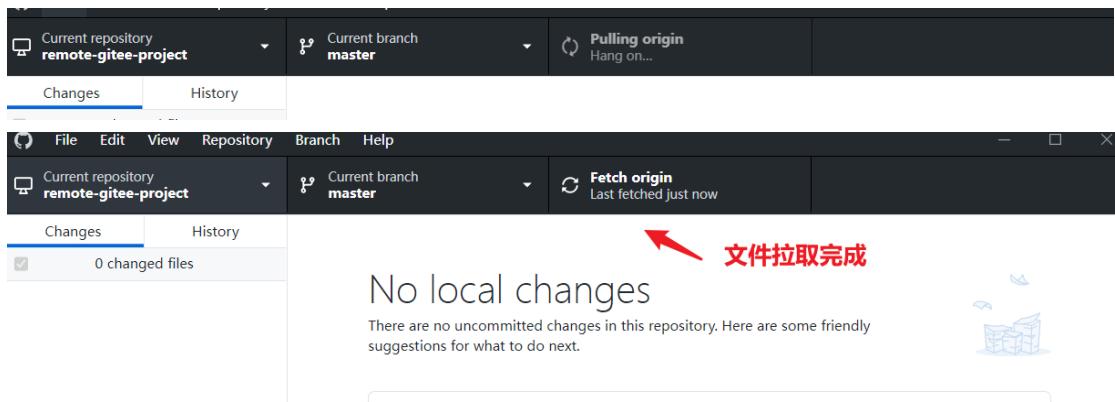


#### 4.6.6.2 拉取文件

远程仓库更新了文件



拉取文件中



ture (E:) > test > remote-gitee-project

名称	修改日期	类型	大小
.git	2023/3/30 1:20	文件夹	
a.txt	2023/3/30 1:18	文本文档	1 KB
b.txt	2023/3/30 1:18	文本文档	1 KB
c.txt	2023/3/30 1:20	文本文档	1 KB

文件拉取成功

#### 4.6.6.3 推送文件

##### 本地创建新文件

Changes 1 History d.txt

1 changed file

d.txt

The file is empty

Create d.txt

Description

Commit to master

提交新文件到本地仓库

0 changed files

## No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

**向远程仓库推送文件**

**Push commits to the origin remote**  
You have 1 local commit waiting to be pushed to the remote.  
Always available in the toolbar when there are local commits waiting to be pushed or **Ctrl + P**

**Push origin**

**Open the repository in your external editor**

Changes History 0 changed files

Current repository: **remote-gitee-project** Current branch: **master** Pushing to origin: Hang on...

## No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

**文件推送中**

**Push commits to the origin remote**  
You have 1 local commit waiting to be pushed to the remote.  
Always available in the toolbar when there are local commits waiting to be pushed or **Ctrl + P**

**Push origin**

**Open the repository in your external editor**  
Select your editor in **Options**  
Repository menu or **Ctrl + Shift + A**

**Open in Notepad++**

Atiger / test-gitee-project

代码 Issues Pull Requests Wiki 统计 流水线 服务 管理

分支 1 标签 0

laihaibo Create d.txt 215af25 1分钟前

a.txt update a.txt. 1小时前

b.txt update b.txt. 14小时前

c.txt add c.txt. 4分钟前

d.txt Create d.txt 1分钟前

简介 暂无描述  
暂无标签

发行版 暂无发行版, 创建

贡献者 (2)

远程仓库的文件发生了变化

# 第5章 Git 问题

## 5.1 SSH 公钥错误

```
fatal: Could not read from remote repository.  
Please make sure you have the correct access rights  
and the repository exists.  
17:16:30.554: [git-project] git -c ... push --progress --porcelain ideaproject refs/heads/newbranch:refs/heads/newbranch --set-upstream  
git@github.com: Permission denied (publickey).  
fatal: Could not read from remote repository.  
Please make sure you have the correct access rights  
and the repository exists.
```

一般出现如上错误，就是 Git 远程仓库的 SSH 免密公钥和推送用户提供的公钥不一致导致的。

## 5.2 IDEA 集成 Gitee 失败

如果 IDEA 集成 Gitee 时，向远程仓库 push 代码失败，且没有弹出账号窗口，可以尝试修改 IDEA 得相关配置。

