# Git 是一个软件/工具/系统，GitHub 是一个网站/平台，GitHub 这个网站使用了 Git 这个工具。

Git是一个分布式**[版本控制系统](https://zhida.zhihu.com/search?content_id=250804842&content_type=Article&match_order=1&q=%E7%89%88%E6%9C%AC%E6%8E%A7%E5%88%B6%E7%B3%BB%E7%BB%9F&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)**。

不仅可以应用于软件源代码的文本文件，而且可以对任何类型的文件进行版本控制。

**协同合作**（多人编辑文件或代码而不出错）

**文件备份**（服务器和本地都有完整的历史版本，如果服务器坏了，本地还有一份完整的历史记录）。

**版本存储**（你改动的每一版本都保存下来，如果改错可以回到之前的版本，如果想加上删除的内容也可以返回去找，也可以对比现在和之前的版本，看改了什么）

# 注册github

打开官网<https://github.com/>

点击右上角 Sign In，登录

lxh90101 /Dir3

# 安装 [Git for Windows](https://git-scm.com/downloads" \t "https://chat.deepseek.com/a/chat/s/_blank)

下载 [Git for Windows](https://git-scm.com/downloads" \t "https://chat.deepseek.com/a/chat/s/_blank)，安装时保持默认选项

Git for windows已保存在F:\server\开发\github

# 开始使用

开始菜单找到git bash, dos的命令窗口，以$开头

git --version 验证是否好使

# 配置 Git

告诉git你的用户名是lxh90101

git config --global user.name "lxh90101"

再告诉git你的邮箱

git config --global user.email "boss66@163.com"

# 创建第一个仓库（Repository）

仓库（Repo）是存放代码的“文件夹”。

登录 GitHub，点击右上角 + → New repository。

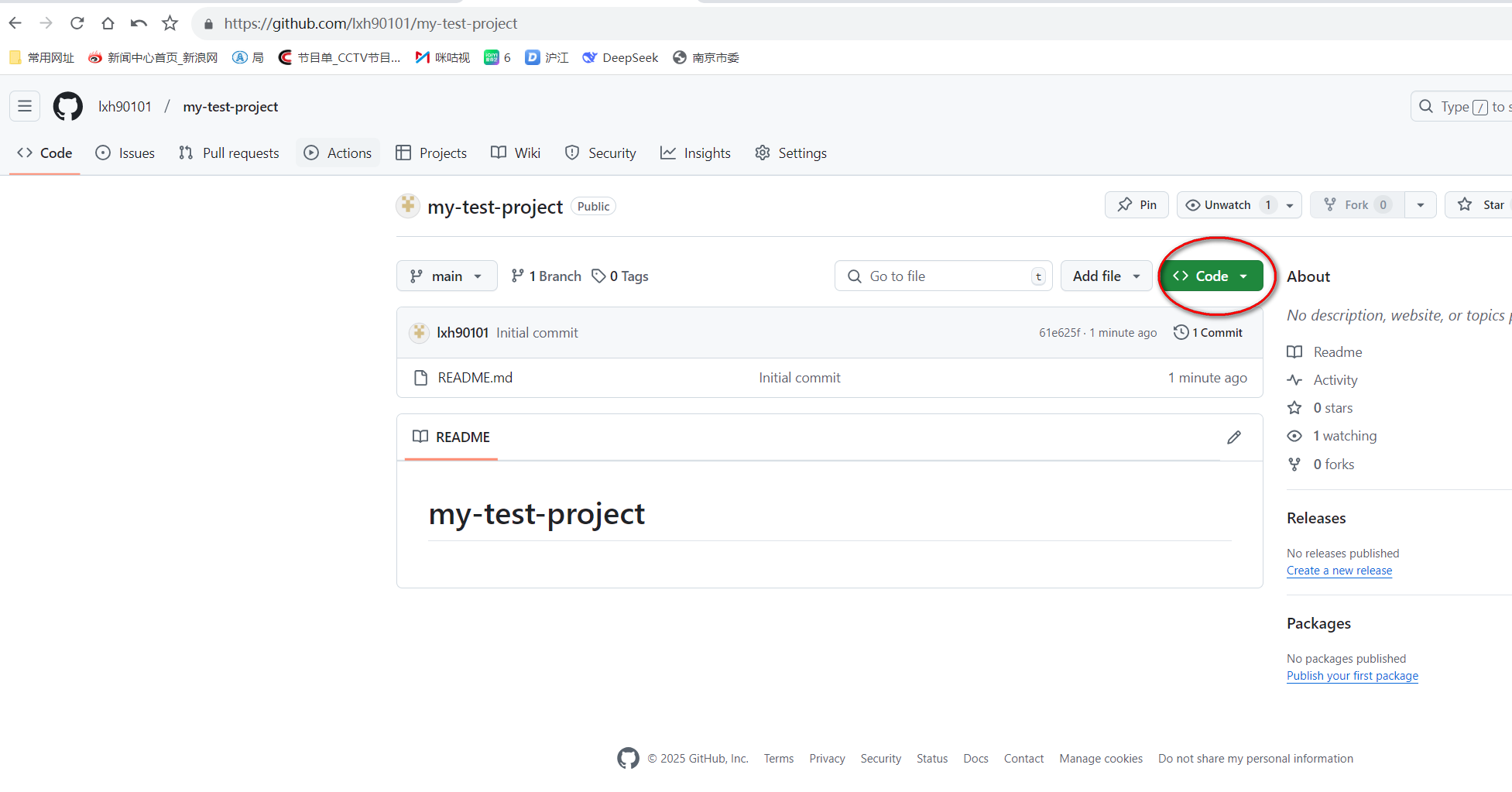
输入仓库名称（如 my-first-project），选择公开（Public）或私有（Private）。

勾选 Initialize this repository with a README（可选，用于创建初始文件）。

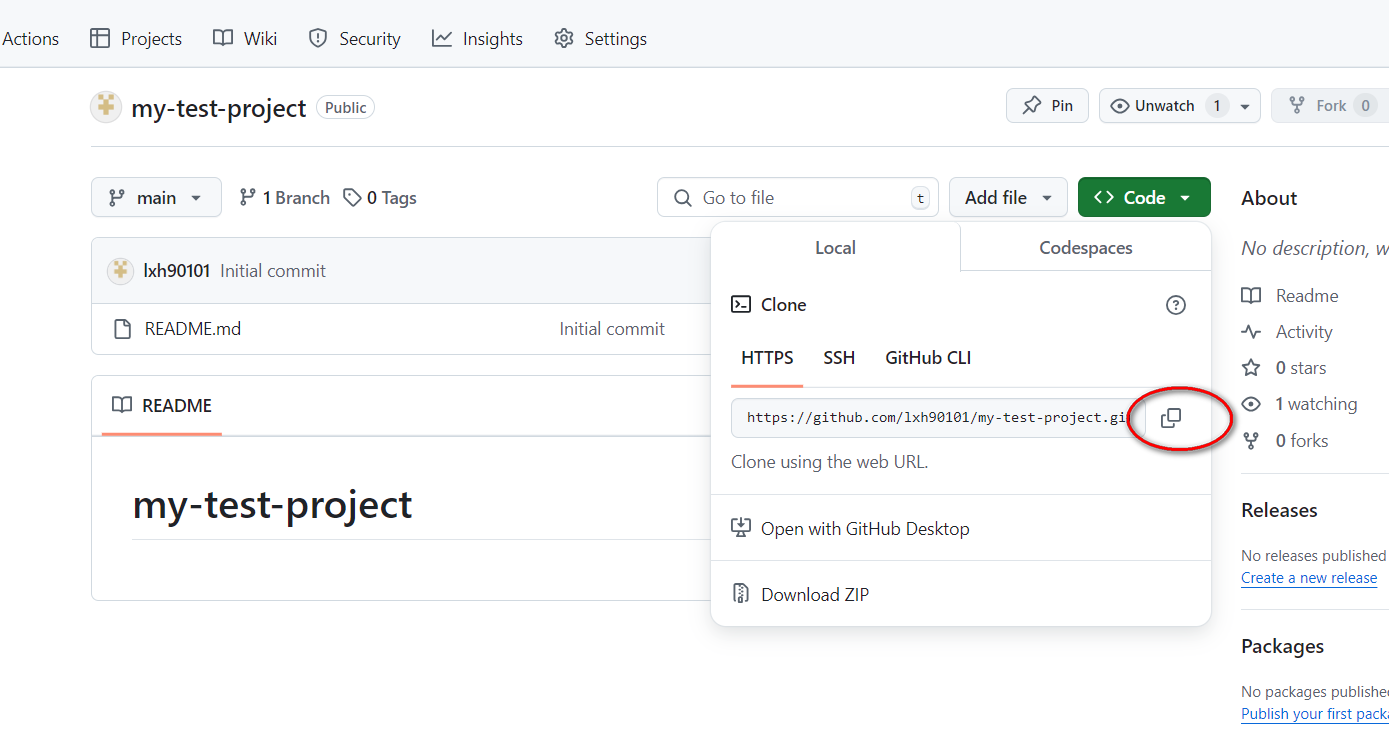
点击 Create repository。

# 克隆仓库到本地

将远程仓库下载到本地电脑：

在github.com网站上，在仓库页面点击绿色按钮 Code，复制仓库地址（HTTPS 或 SSH）。  


在code中选择https中，地址右侧的“复制”按钮，点击复制。



在终端中运行：

git clone <https://github.com/lxh90101/my-test-project.git> //地址是复制的

# 添加文件并提交更改

进入克隆到本地的仓库文件夹：

cd my-test-project

新建或修改文件（例如 hello.txt）。

使用以下命令提交更改：

git add . # 添加所有修改到暂存区

git add hello.txt # 或单独添加某个文件

git commit -m "添加了hello.txt文件" # 提交并写说明

# 推送代码到 GitHub

将本地的提交推送到远程仓库：

git push origin main # 如果分支是 main（旧版本可能是 master）

# 分支管理

分支（Branch）用于在不影响主代码的情况下开发新功能：

git branch new-feature # 创建分支

git checkout new-feature # 切换到新分支

# 修改代码后提交

git add .

git commit -m "在新分支上开发功能"

git push origin new-feature # 推送分支到远程

# 发起 Pull Request（PR）

将分支合并到主分支：

在 GitHub 仓库页面，点击 Pull requests → New pull request。

选择要合并的分支（如 new-feature → main）。

填写说明，点击 Create pull request，等待审核合并。

# 同步远程更新

如果其他人修改了代码，本地需要同步：

git pull origin main //拉取远程最新代码到本地

# 仓库级别的权限管理

## 仓库可见性设置

创建仓库时需选择访问级别：

公开（Public）：所有人可查看、克隆，但不能直接修改（需通过 Pull Request）25。

私有（Private）：仅授权用户或团队可访问，适用于敏感项目24。

内部（Internal，仅限组织仓库）：组织内成员可见2

## 协作者权限分配

在仓库的 Settings → Manage Access 中可邀请用户或团队，并分配以下权限级别：

Admin：完全控制（包括删除仓库）。

Write：可推送代码，但无法修改仓库设置。

Maintain：管理 Issues 和 PR，无删除权限。

Triage：处理 Issues/PR，但不可修改代码。

Read：仅查看代码。

# 组织级别的权限设计

## 团队与角色分层

创建组织后，可划分多个团队（如前端组、后端组），并为每个团队分配不同仓库的权限（如读写或只读）。

团队权限继承：成员加入团队后自动获得团队对仓库的权限，简化管理6。

## 外部协作者管理

可邀请非组织成员作为外部协作者，限制其对特定仓库的访问。

# 分支保护规则

## 强制代码审查

启用 Require pull request reviews，要求 PR 必须通过指定人数的审核才能合并25。

## 状态检查与同步

Require status checks：确保 CI/CD 测试通过2。

Require branch up to date：合并前需同步主分支最新代码2。

# 访问令牌（Token）的权限控制

## 生成精细化的 Token

在 Settings → Developer settings → Personal access tokens 中创建 Token，按需勾选权限（如 repo 全权限或仅 read:org）。

## Token 的安全使用

替换本地 Git 凭据中的密码为 Token，提升安全性（通过系统凭据管理器操作）。

# 高级权限策略

## 代码所有者（CODEOWNERS）

在仓库根目录创建 .github/CODEOWNERS 文件，指定特定文件/目录的负责人，其必须审核相关 PR。

## GitHub Actions 的权限隔离

通过 Workflow 配置文件限制 Actions 的访问范围（如仅允许读写特定路径）。