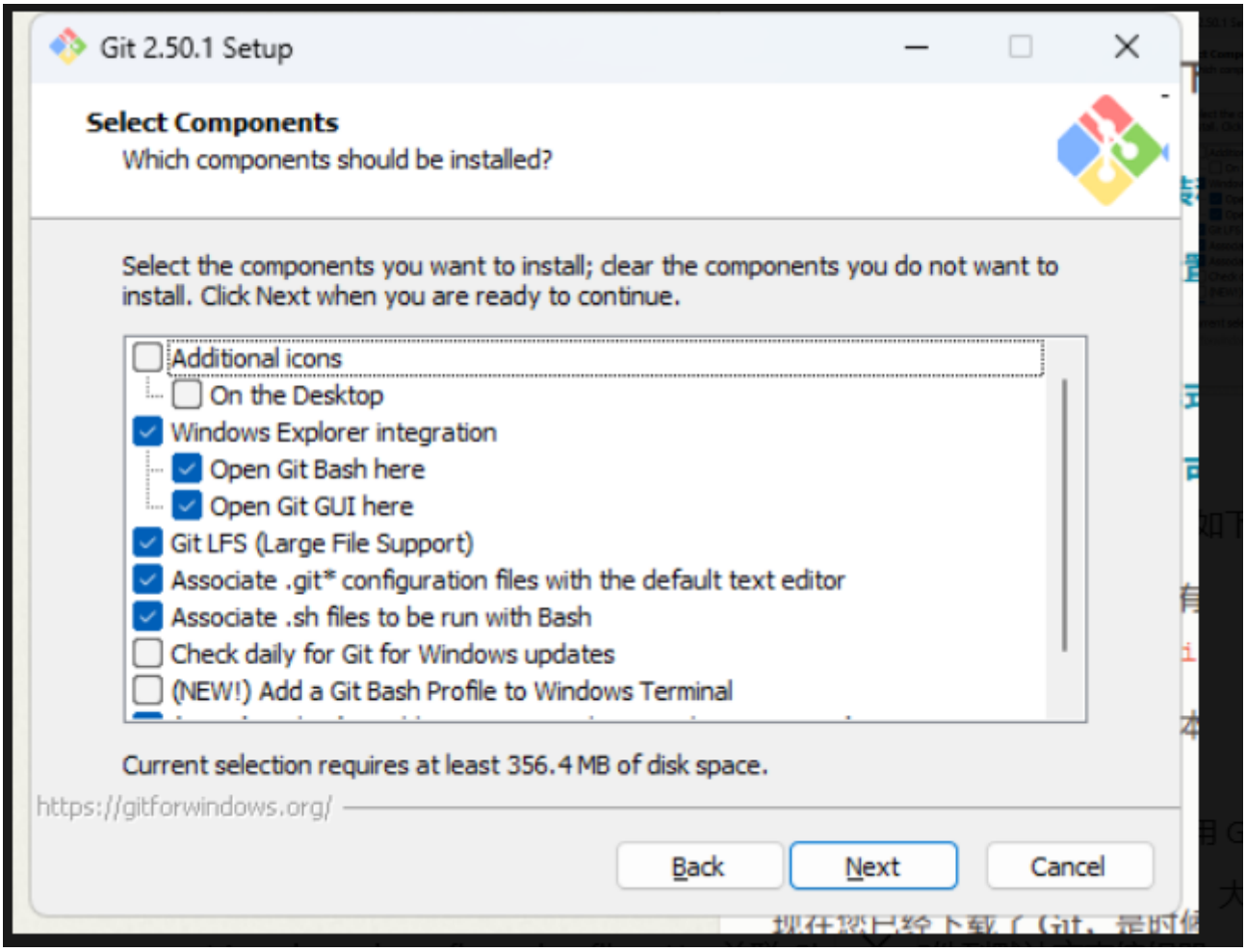
1、下载：

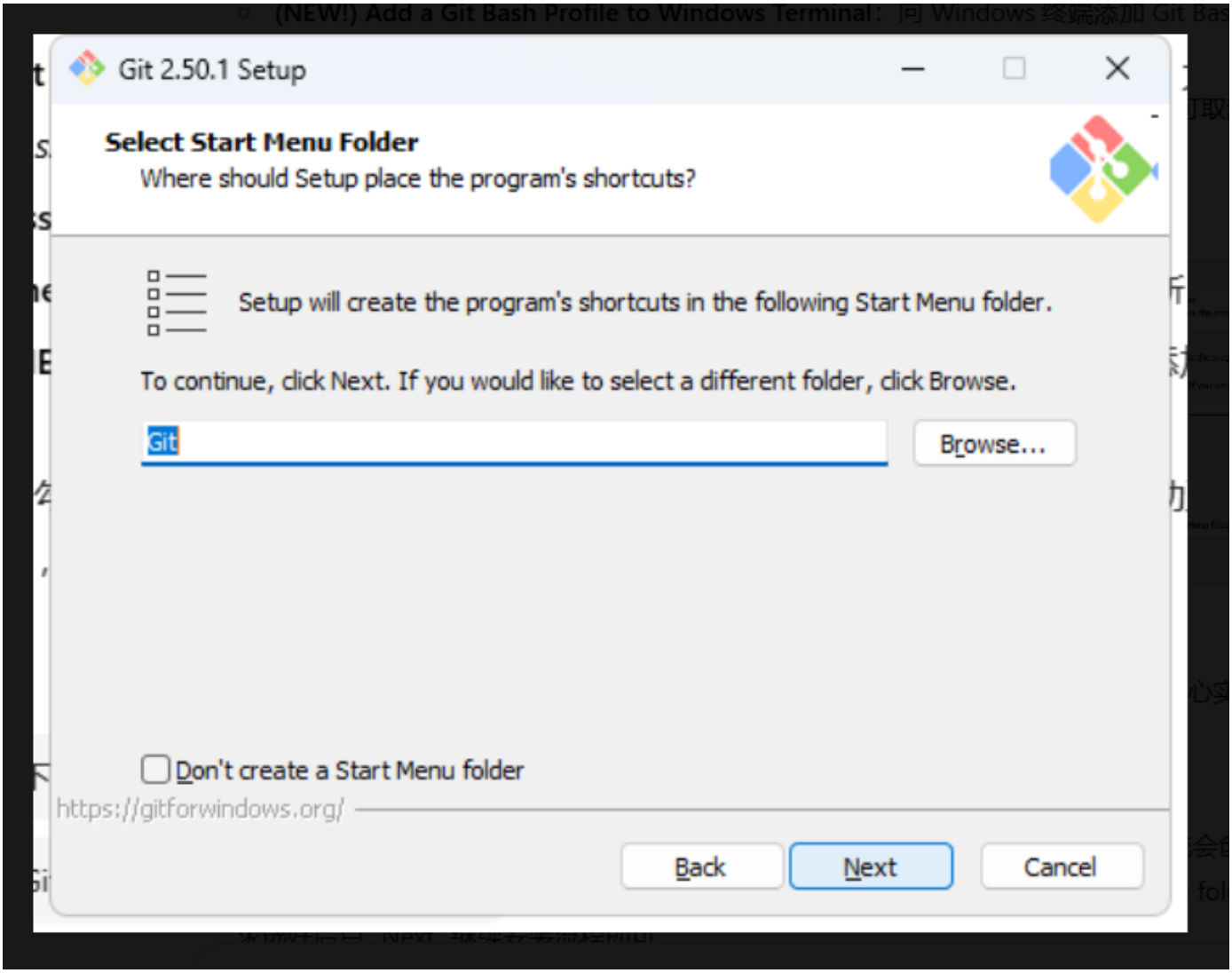
Git - Downloading Package

2、下载设置



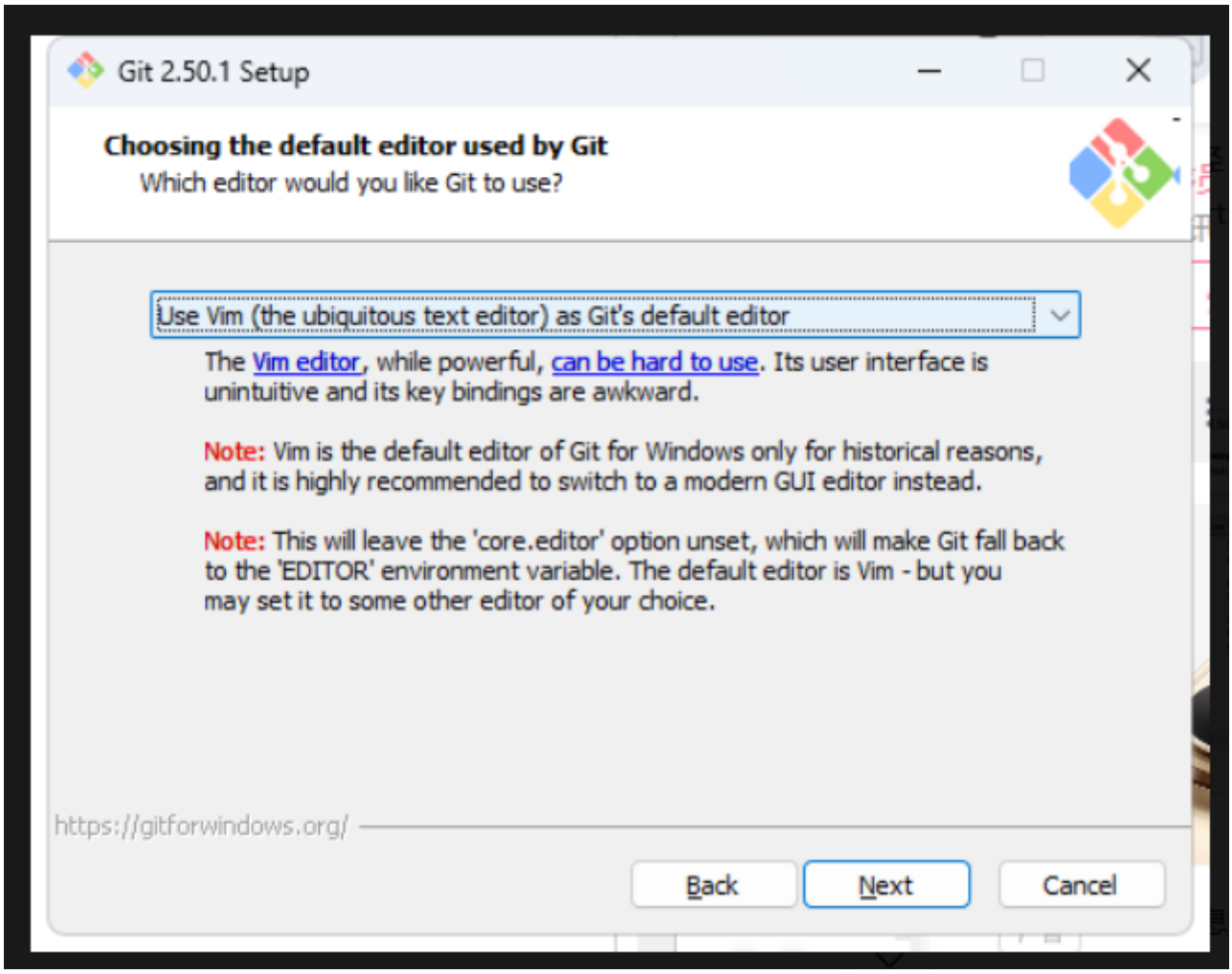
- **软件名称**：Git 2.50.1 Setup（Git 的 Windows 安装程序）
- **组件选项**：
  - **Additional icons**：额外图标（可勾选是否在桌面创建）
  - **Windows Explorer integration**：Windows 资源管理器集成（方便右键调用 Git 功能）
  - **Git LFS (Large File Support)**：Git 大文件支持，如需处理大文件（如视频、大型数据集）可勾选
  - *Associate .git* configuration files...**：关联 Git 配置文件到默认文本编辑器
  - **Associate .sh files...**：关联 .sh 脚本文件，让其能在 Git Bash 中运行
  - **Check daily for Git for Windows updates**：每日检查 Git for Windows 更新
  - **(NEW!) Add a Git Bash Profile to Windows Terminal**：向 Windows 终端添加 Git Bash 配置文件

一般默认勾选即可满足基础使用，若需精简，可按需取消非必要组件（如不想自动更新可取消“Check daily...”），选好后点 “Next” 继续安装流程 。



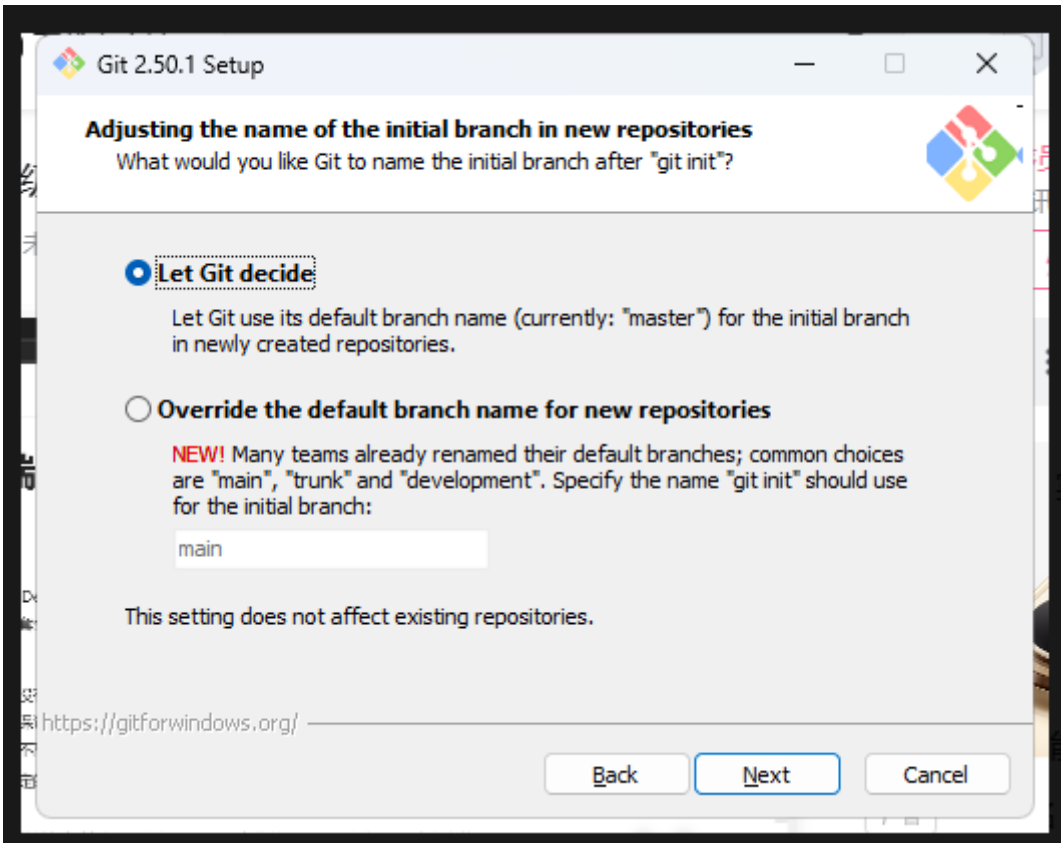
这是 **Git for Windows（Git 2.50.1 版本）安装过程中的“选择开始菜单文件夹”界面**，核心实体信息：

- **软件**：Git 2.50.1（Windows 安装程序）
- **功能**：设置 Git 快捷方式在开始菜单的存放位置，可直接点 “Next” 用默认路径（系统会创建名为 “Git” 的开始菜单文件夹）；也可通过 “Browse...” 自定义路径，或勾选 “Don't create a Start Menu folder” 跳过创建。 按需求选好后点 “Next” 继续安装流程即可。



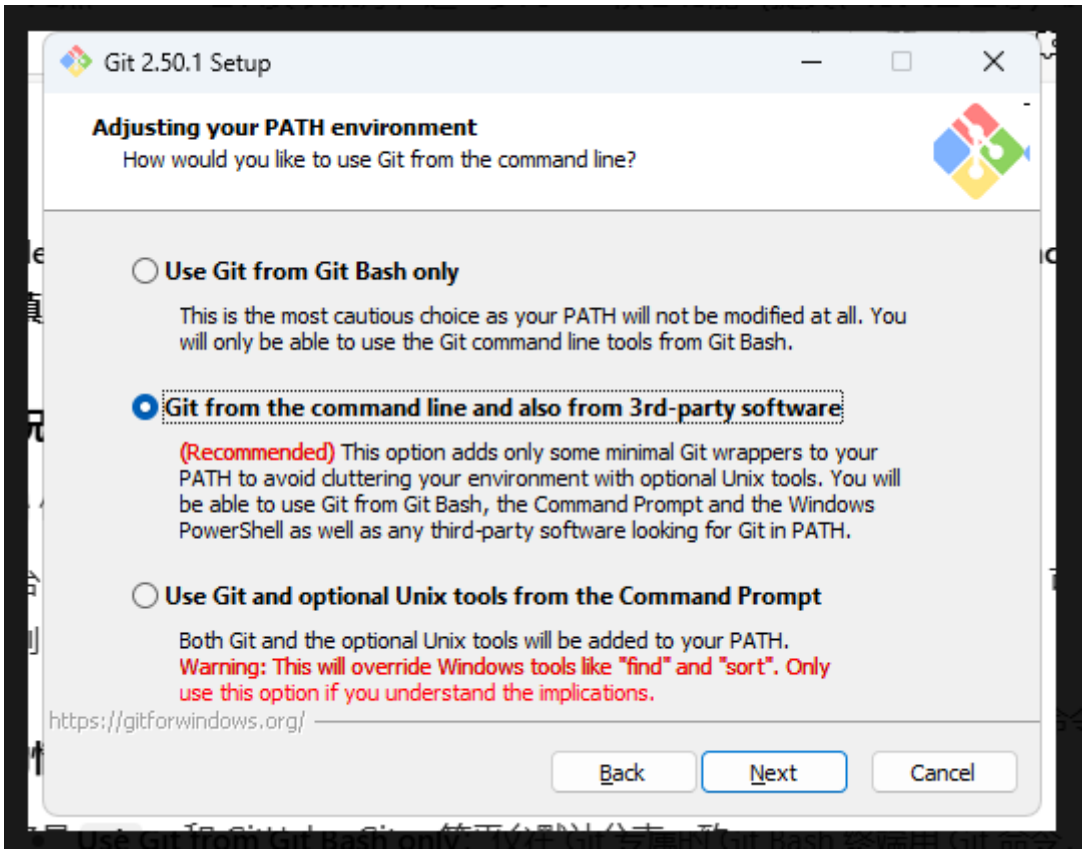
- **操作目标**：为 Git 选择默认文本编辑器，当前默认选项是 “Use Vim (the ubiquitous text editor) as Git’s default editor”（用 Vim 作为默认编辑器 ）
- **说明提示**：Vim 虽强大但学习曲线陡、操作不直观，官方建议换现代 GUI 编辑器（如 VS Code、Notepad++ 等，后续可通过修改 Git 配置更换 ）。

若习惯 Vim 可直接点 “Next”；想换其他编辑器，点击下拉框选对应选项（如装了 VS Code 一般能在列表找到）再继续安装流程即可 。



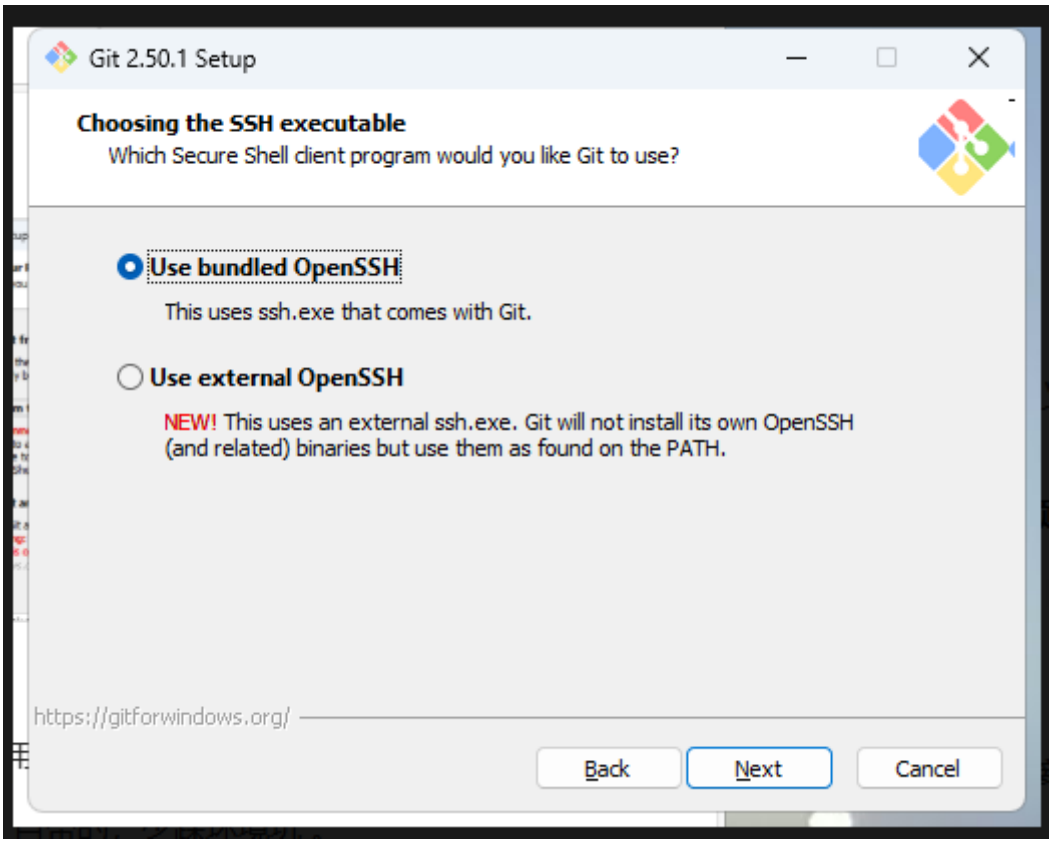
- **功能**：决定执行 `git init` 时，新仓库初始分支的命名规则
- **选项**：
  - “Let Git decide”：让 Git 自行决定（当前默认是 `master` ，但未来可能随社区规范调整）
  - 建议选择这个！！！！！！！！！“Override the default branch name...”：手动指定新仓库初始分支名（如填 `main` ，很多团队为契合社区改名趋势，常用 `main` `trunk` 等）
- **说明**：该设置不影响已存在的仓库，仅对新初始化的仓库生效 。

若想紧跟社区主流（如 GitHub 等平台默认用 `main` ），可选第二个选项填 `main`；习惯传统 `master` 或不想手动改，选第一个选项即可，选完点 “Next” 继续安装流程。



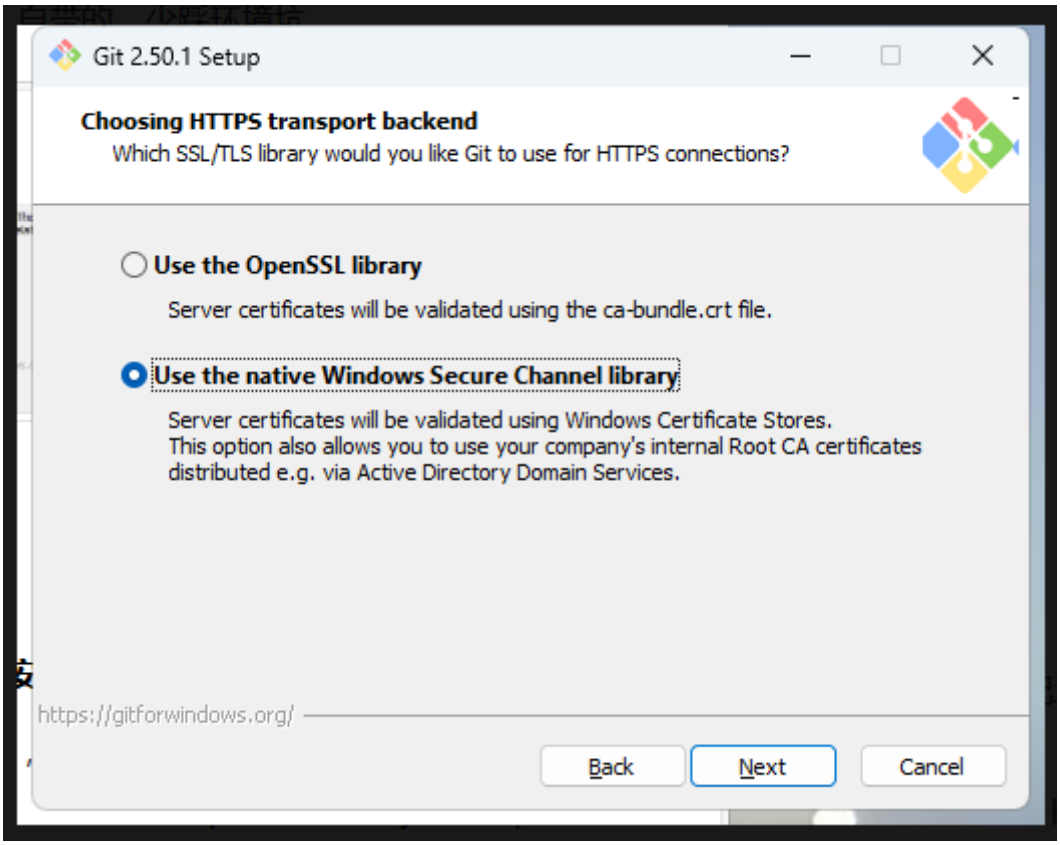
- **Use Git from Git Bash only**：仅在 Git 专属的 Git Bash 终端用 Git 命令，系统 PATH 不添加 Git，CMD、PowerShell 等用不了 Git。
- **Git from the command line and also from 3rd-party software（推荐）**：把 Git 加到系统 PATH，CMD、PowerShell、IDE（如 VS Code）等都能直接调用 Git 命令，兼容性最好，**新手无脑选这个**。
- **Use Git and optional Unix tools from the Command Prompt**：除加 Git 到 PATH，还装一些类 Unix 工具（如 `find` `sort`），但可能覆盖 Windows 系统自带工具，**新手别选，容易出奇怪问题**。

按推荐选第二个（中间带 Recommended 标识的），点 Next 继续装就行，装完所有终端、开发工具都能直接用 Git 命令，超方便。



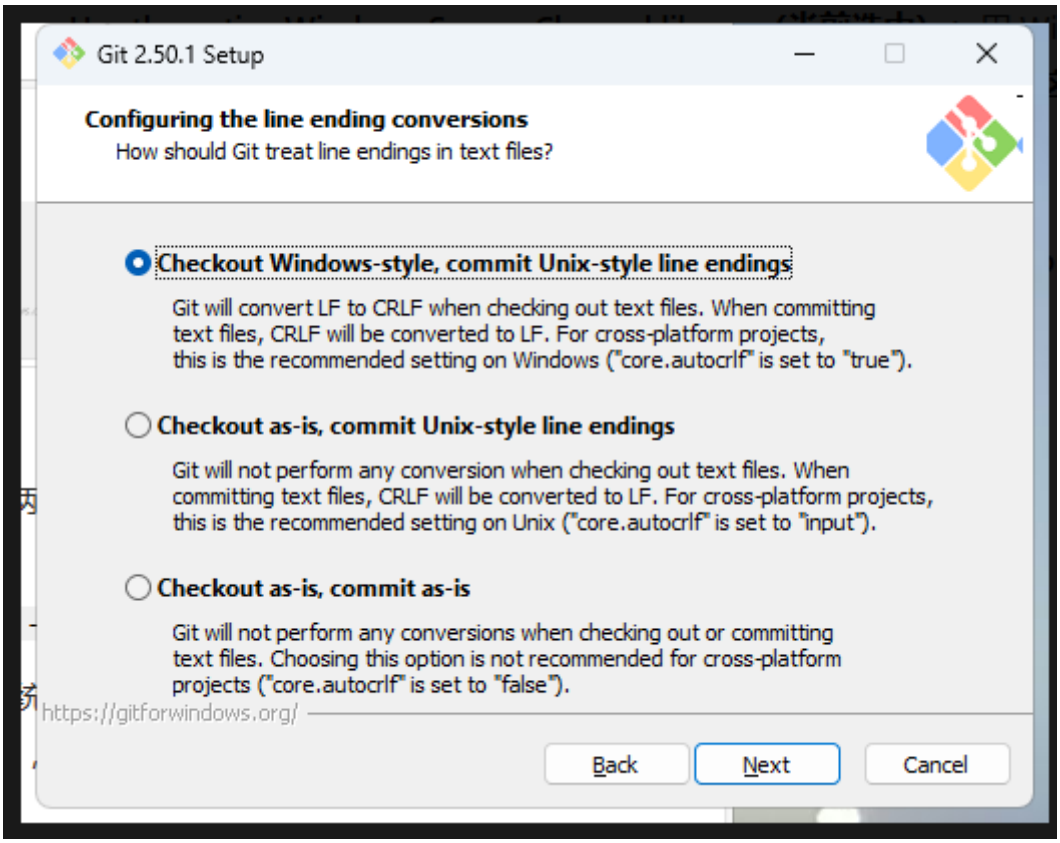
- **Use bundled OpenSSH**：用 Git 自带的 OpenSSH（`ssh.exe`），**无需额外配置，安装即用**，适合新手。
- **Use external OpenSSH**：用系统已有的 OpenSSH（需系统 PATH 里有 `ssh.exe`），Git 不装自己的，**新手别选，容易因系统没配好无法用 SSH** 。

新手直接选 “Use bundled OpenSSH”，点 Next 继续装，后续用 SSH 连接 GitHub/Gitee 等仓库时，直接能用 Git 自带的，少踩环境坑 。



- **Use the OpenSSL library**：用 OpenSSL 库验证 HTTPS 连接，依赖 Git 自带的 `ca-bundle.crt` 证书文件。
- **Use the native Windows Secure Channel library（当前选中）**：用 Windows 系统自带的安全通道库，依托 Windows 证书存储验证，**企业内网用户（需用公司内部 CA 证书）选这个更方便**，普通用户选它也能自动适配系统证书，少操心。

新手 / 普通用户直接默认选 “Use the native Windows Secure Channel library” 就行，点 Next 继续安装，适配性更好，不用额外管证书配置～



1. 选项 1（默认推荐，新手选这个）

- **Checkout Windows-style, commit Unix-style line endings**
  - 拉取（checkout）代码时：把仓库里的 `LF` 换行符转成 Windows 常用的 `CRLF`。
  - 提交（commit）代码时：把文件里的 `CRLF` 转回 `LF` 存到仓库。
  - **优点**：在 Windows 本地编辑文件是 `CRLF`（符合系统习惯），提交到仓库统一用 `LF`（符合 Linux/Mac 规范），**跨平台协作（多人用不同系统开发）时，不会因换行符乱码 / 冲突吵架**，Git 官方也推荐 Windows 系统选这个。

2. 选项 2

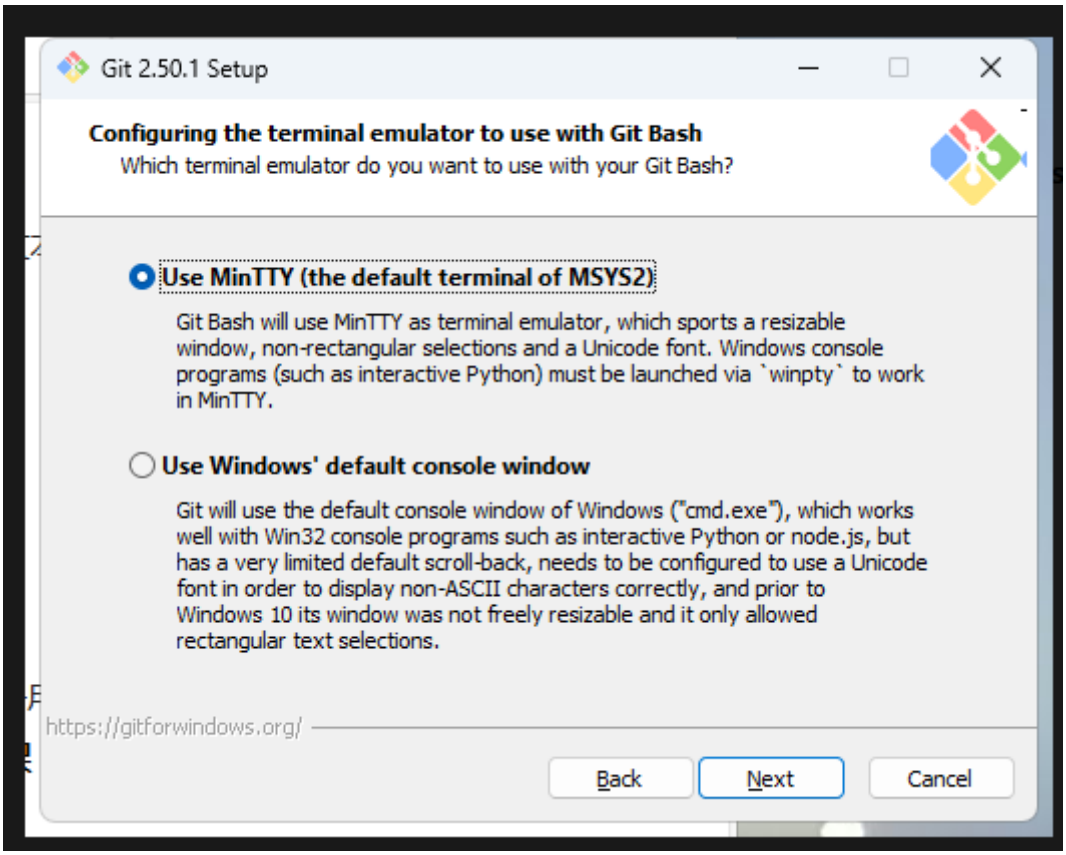
- **Checkout as-is, commit Unix-style line endings**
  - 拉取时：不转换换行符，保持仓库里的原样。
  - 提交时：把 `CRLF` 转成 `LF`。
  - **适合**：Windows 上想严格对齐 Linux/Mac 规范，且自己能处理换行符问题的场景（新手别碰，容易本地编辑报错）。

3. 选项 3

- **Checkout as-is, commit as-is**
  - 拉取和提交都不转换换行符，完全保持文件原本的换行符。
  - **缺点**：跨平台协作时，Windows 本地 `CRLF` 提交到仓库，Linux/Mac 同事拉取可能报错，**除非你明确知道自己需要“完全不处理换行符”，否则别选**。

总结

新手 / 普通 Windows 用户，直接选第一个默认选项（Checkout Windows-style...），点 Next 继续安装，完美适配跨平台开发，不用操心换行符冲突～



1. 选项 1（默认推荐，新手选这个）

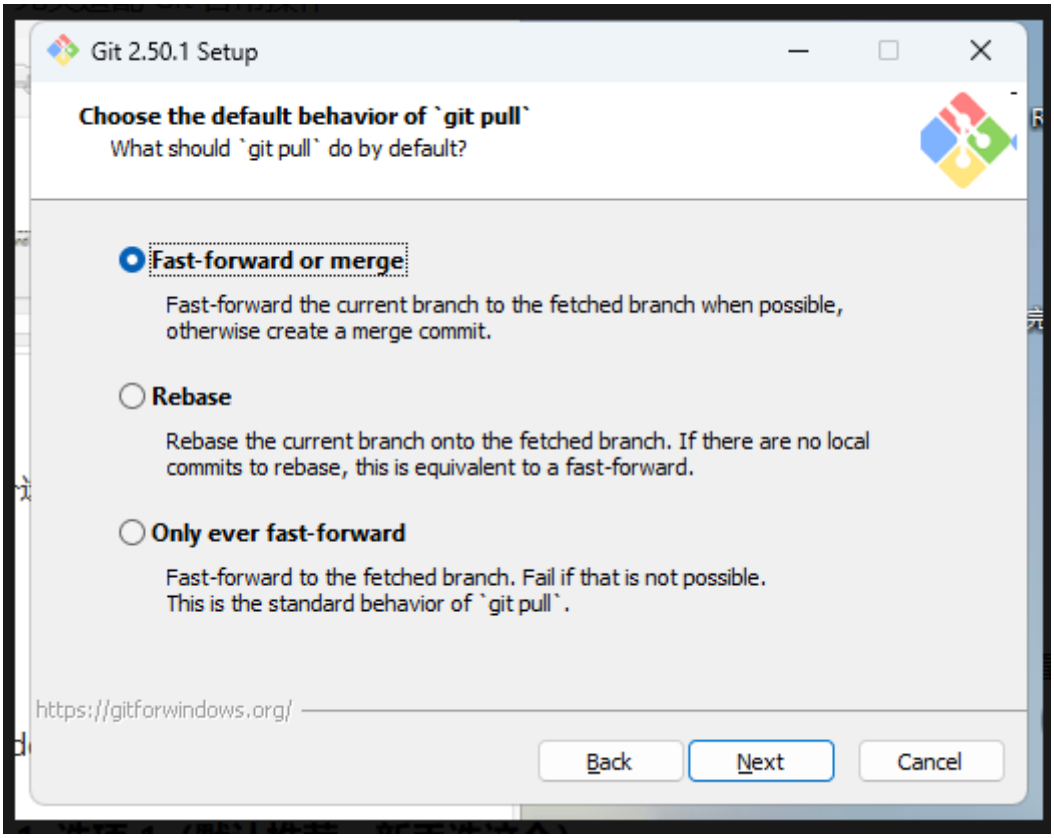
- **Use MinTTY (the default terminal of MSYS2)**
  - 是 Git Bash 默认的终端模拟器，支持调整窗口大小、选中文本更灵活，还能很好显示 Unicode 字符（比如中文、特殊符号）。
  - **小缺点**：如果要在 Git Bash 里运行需要交互的程序（比如某些 Python 脚本），得用 `winpty` 命令启动（不过日常用 Git 命令基本遇不到，别担心）。

2. 选项 2

- **Use Windows' default console window**
  - 用 Windows 系统自带的命令提示符（cmd.exe 那种黑框）当终端。
  - **缺点**：显示中文 / 特殊字符容易乱码，调整窗口大小也麻烦，**除非你特别需要“和系统 cmd 完全一致的终端”，否则别选**。

总结

新手 / 普通用户，直接选第一个默认选项（Use MinTTY...），点 Next 继续安装，用起来更顺手、显示更清晰，完美适配 Git 日常操作～



1. 选项 1（默认推荐，新手选这个）

- **Fast-forward or merge**
  - 逻辑：优先尝试“快进合并”（`fast-forward`），如果不行（比如本地分支有额外提交），就创建一个合并提交（`merge commit`）。
  - **优点**：兼容大多数场景，既保留简洁的分支历史（能快进就快进），又能处理复杂分支合并（本地有新提交时自动合并），**新手 / 普通用户选这个最稳，不用操心分支历史乱掉**。

2. 选项 2

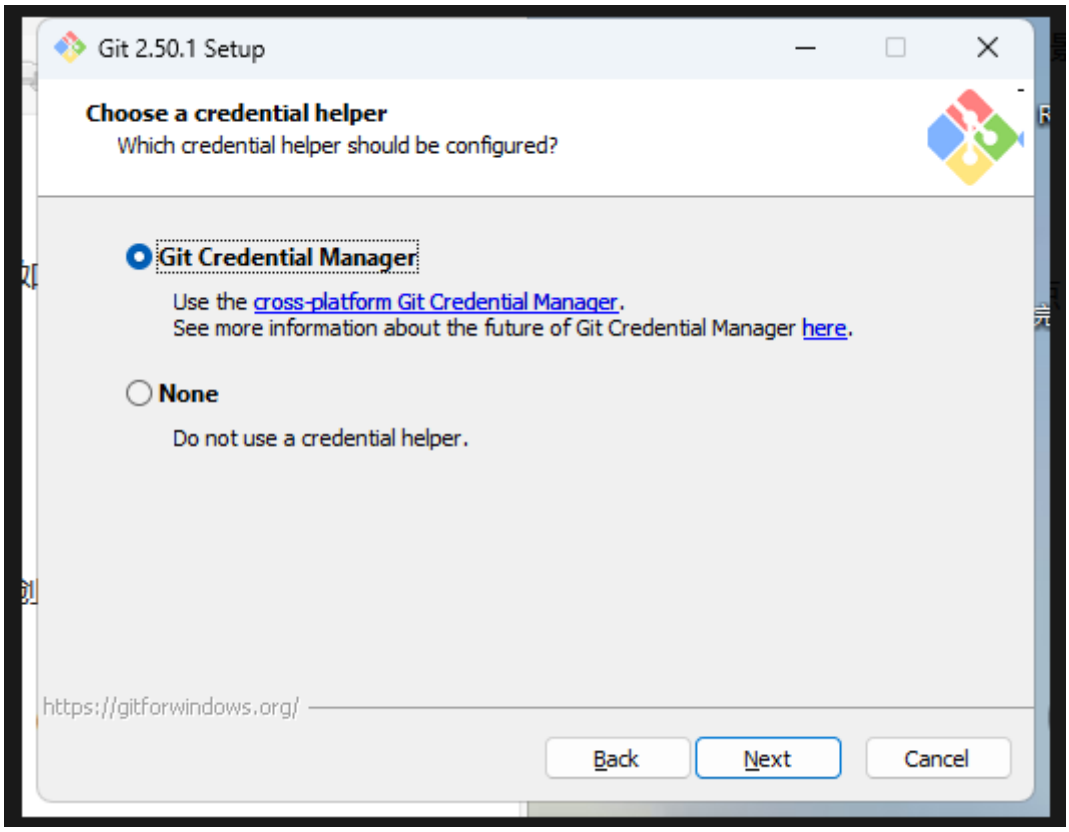
- **Rebase**
  - 逻辑：把本地分支的提交，“变基”到远程分支最新提交之后（相当于让本地提交看起来是基于远程最新代码写的）。
  - **适合**：想让分支历史更“线性”、干净的场景（比如多人协作严格规范分支的团队）。
  - **缺点**：变基会改写提交历史，如果没玩明白，**容易把团队分支搞乱、冲突难解决**，新手别碰。

3. 选项 3

- **Only ever fast-forward**
  - 逻辑：只允许“快进合并”，如果本地有新提交（无法快进），`git pull` 直接失败。
  - **适合**：严格要求分支历史“绝对干净”，且自己能手动处理合并的场景（新手选这个，遇到复杂情况会直接报错，反而麻烦）。

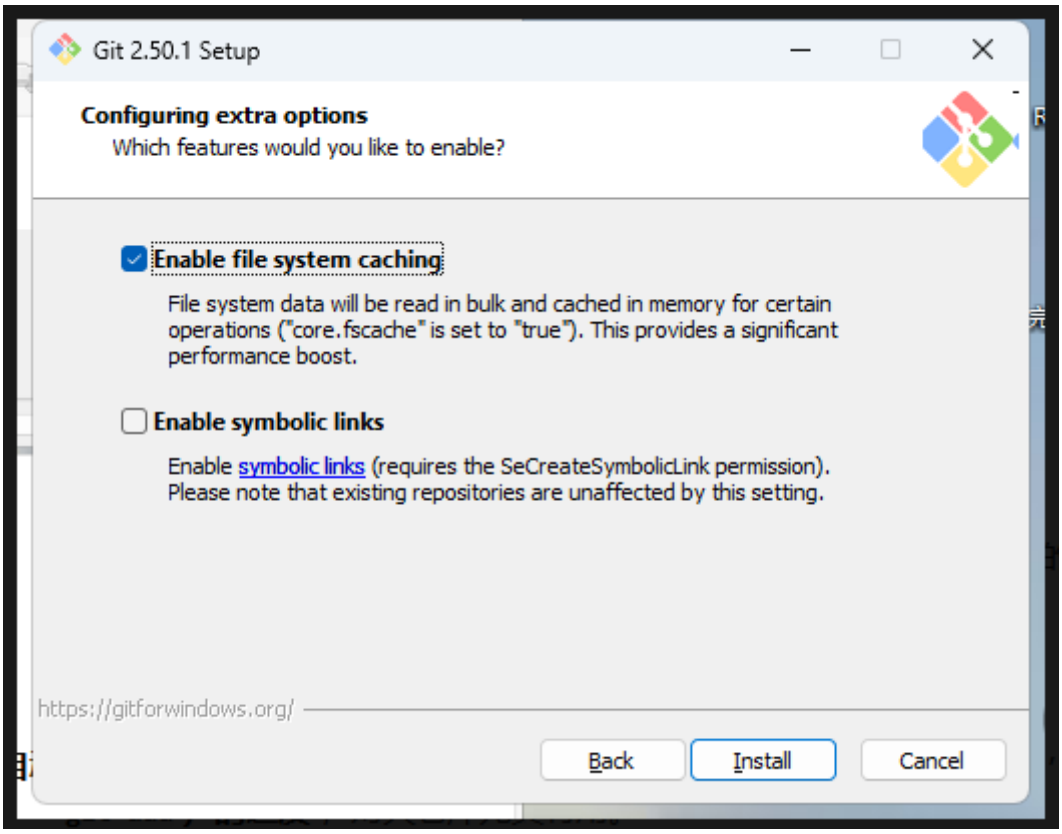
总结

新手 / 普通用户，直接选第一个默认选项（Fast-forward or merge），点 Next 继续安装，`git pull` 行为最兼容、最省心，不用纠结复杂的分支历史问题～



- **Git Credential Manager**（默认推荐，选这个）： 自动帮你管理 Git 连接远程仓库（如 GitHub、Gitee）的账号密码 / Token，**拉取、推送代码时自动弹框保存凭证，不用每次输密码**，超方便，尤其适配 GitHub 等平台的 OAuth 登录。
- **None**：不启用凭证管理器，每次连接远程仓库都得手动输账号密码 / Token，**麻烦又容易忘，新手别选**。

直接选默认的“Git Credential Manager”，点 Next 继续装，后续连接远程仓库能自动记住凭证，少折腾～



1. Enable file system caching（已勾选，推荐保留）

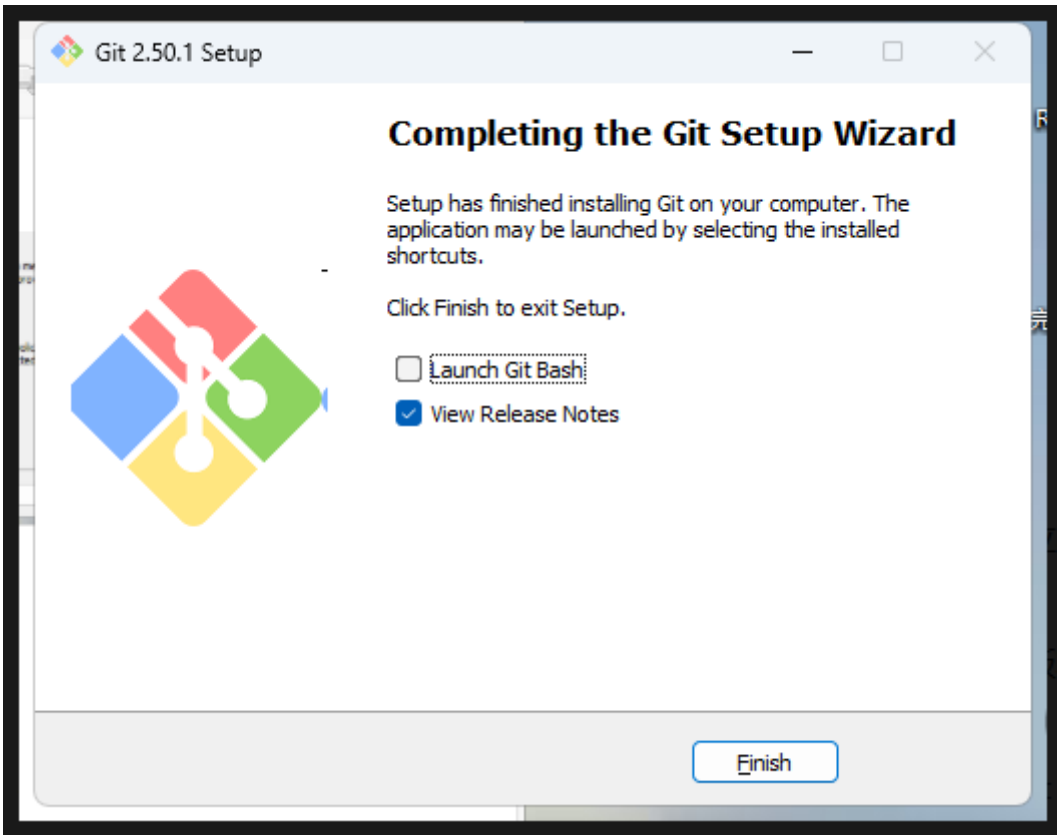
- **作用**：开启文件系统缓存，Git 会批量读取文件系统数据并缓存到内存，能**显著提升 Git 操作（如 `git status` `git add` ）的速度**，对大仓库尤其有用。
- **建议**：默认勾选就好，日常开发能让 Git 跑得更快，没副作用。

2. Enable symbolic links（未勾选，谨慎选择）

- **作用**：启用符号链接支持（类似 Windows 快捷方式，但更底层），需要系统权限（`SeCreateSymbolicLink` ）。
- **适合**：开发中明确需要用符号链接管理文件（比如某些跨平台项目）的场景。
- **缺点**：普通开发很少用到，启用后如果权限没配好，可能导致文件访问异常，**新手 / 普通用户别勾选** 。

总结

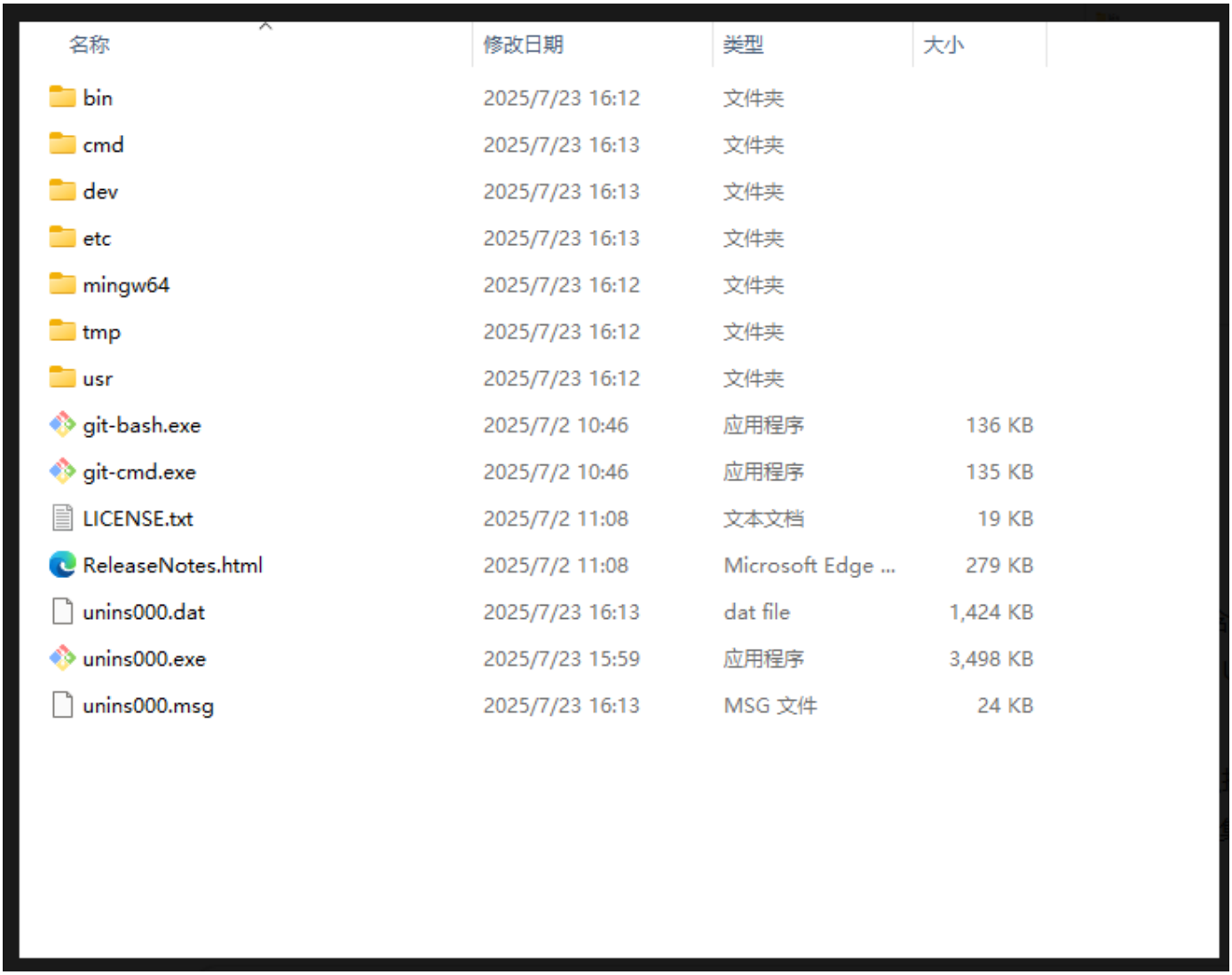
保持默认配置（只勾选 Enable file system caching ），直接点 `Install` 开始安装，既能享受性能加速，又不会因额外功能踩坑，安装完成后 Git 就能正常用啦 ~



- Git 已成功安装到电脑，点 `Finish` 可结束安装。
- 可选操作：
  - `Launch Git Bash`：勾选则安装完成后自动打开 Git Bash 终端（想立刻试试 Git 命令就勾，不想弹窗口就不勾）。
  - `View Release Notes`：勾选则打开 Git 2.50.1 的更新说明（看看新版本加了啥功能，新手不看也没事）。

操作建议：直接点 `Finish` 结束安装（想玩 Git 再自己找快捷方式打开 Git Bash），安装流程就完美收尾啦～

下载完成后的目录



- `git - bash.exe`：这是 Git Bash 程序，它提供了一个类 Unix 风格的命令行环境，包含了常用的 Unix 命令工具，如 `ls`、`cd`、`grep` 等，同时也能执行 Git 命令。在 Windows 系统上，如果你习惯类 Unix 的操作方式， 或者项目需要在类 Unix 环境下运行一些脚本，那么推荐使用这个程序。
- `git - cmd.exe`：这是基于 Windows 命令提示符（CMD）的程序，在这个环境中可以执行 Git 命令 。如果你更习惯 Windows 的命令行操作方式，或者需要在 CMD 环境下与其他 Windows 工具集成使用，那么可以选择使用 `git - cmd.exe` 。

3、初始配置

3.1、bash - 命令行执行

$ 设置用户名

git config --global user.name = 'git 账号的名字'

$ 设置邮箱 （建议和 GitHub / Gitee 相同）

git config --global user.email '邮箱号'

4、生成 SSH 密钥 - 建立 SSH连接

4.1、生成 SSH - bash

$ 执行完成命令后 会让你添加密码 连续回车（默认路径、空密码），生成 `id_ed25519`（私钥）和 `id_ed25519.pub`（公钥）。

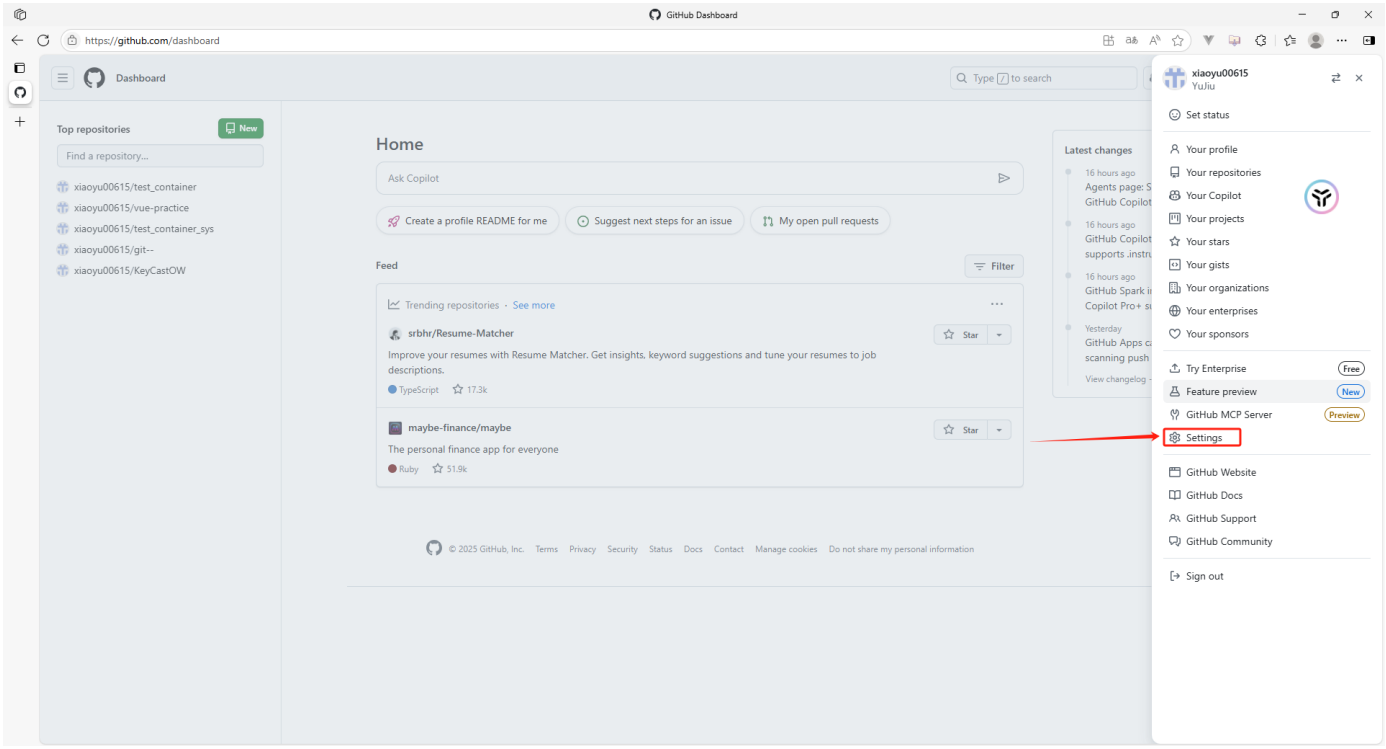
ssh-keygen -t ed25519 -C "填写邮箱"

4.2、将 SSH 密钥添加到 GitHub 账号

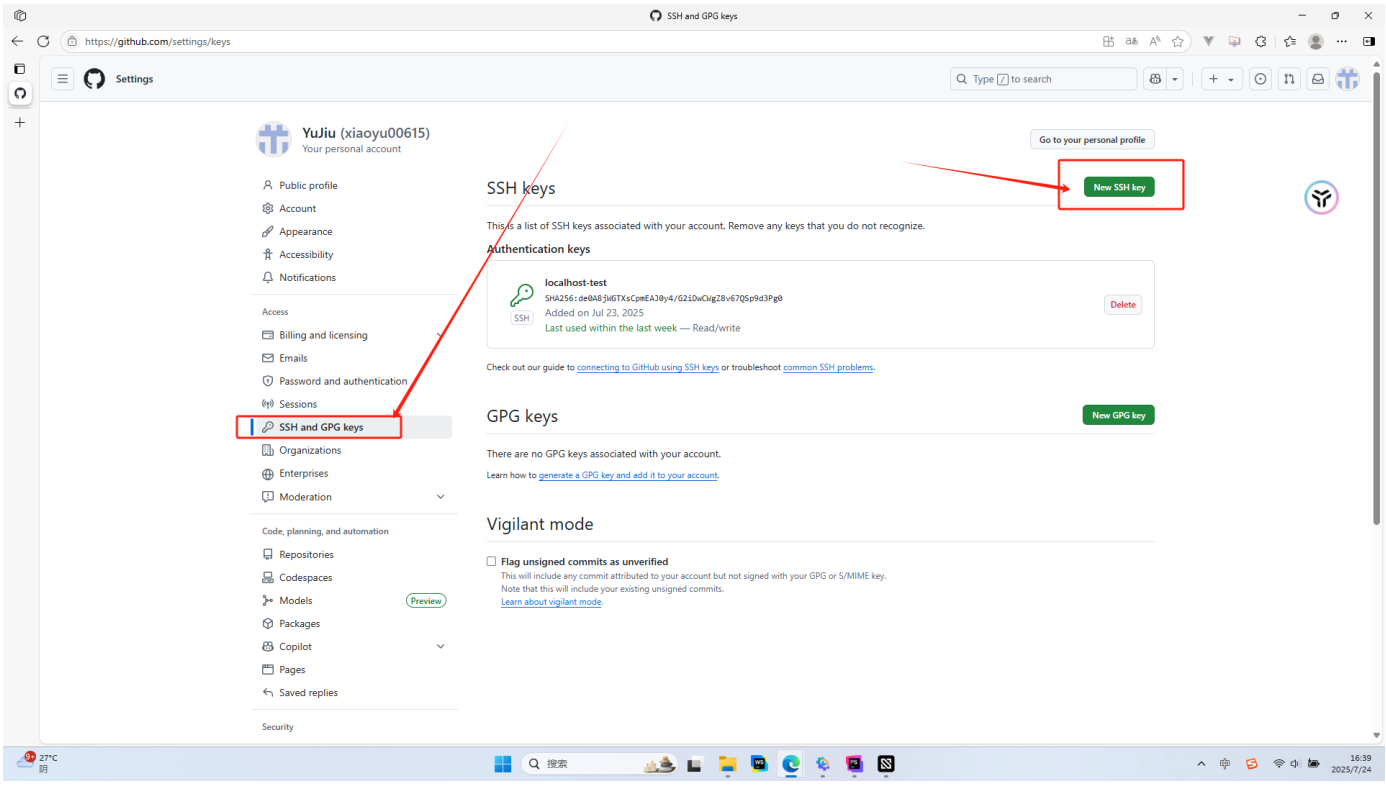
1. 生成密钥后，找到本地的 SSH 公钥文件。在 Windows 系统中，默认路径是 `C:\Users\你的用户名\.ssh\id_ed25519.pub`；在 Mac/Linux 系统中，默认路径是 `~/.ssh/id_ed25519.pub`。
2. 用文本编辑器打开公钥文件，全选并复制里面的内容。
3. 登录你的 GitHub 账号，点击右上角头像，选择 `Settings`。
4. 在左侧菜单中选择 `SSH and GPG keys`，然后点击 `New SSH key`。
5. 在 `Title` 栏随便填一个便于识别的名称，在 `Key` 栏粘贴刚才复制的公钥内容，最后点击 `Add SSH key`。
6. 使用 - bash 打印出密钥 你的用户名要进行修改 但是在默认情况下是 `Administrator`

   type C:\Users\你的用户名.ssh\id_ed25519.pub

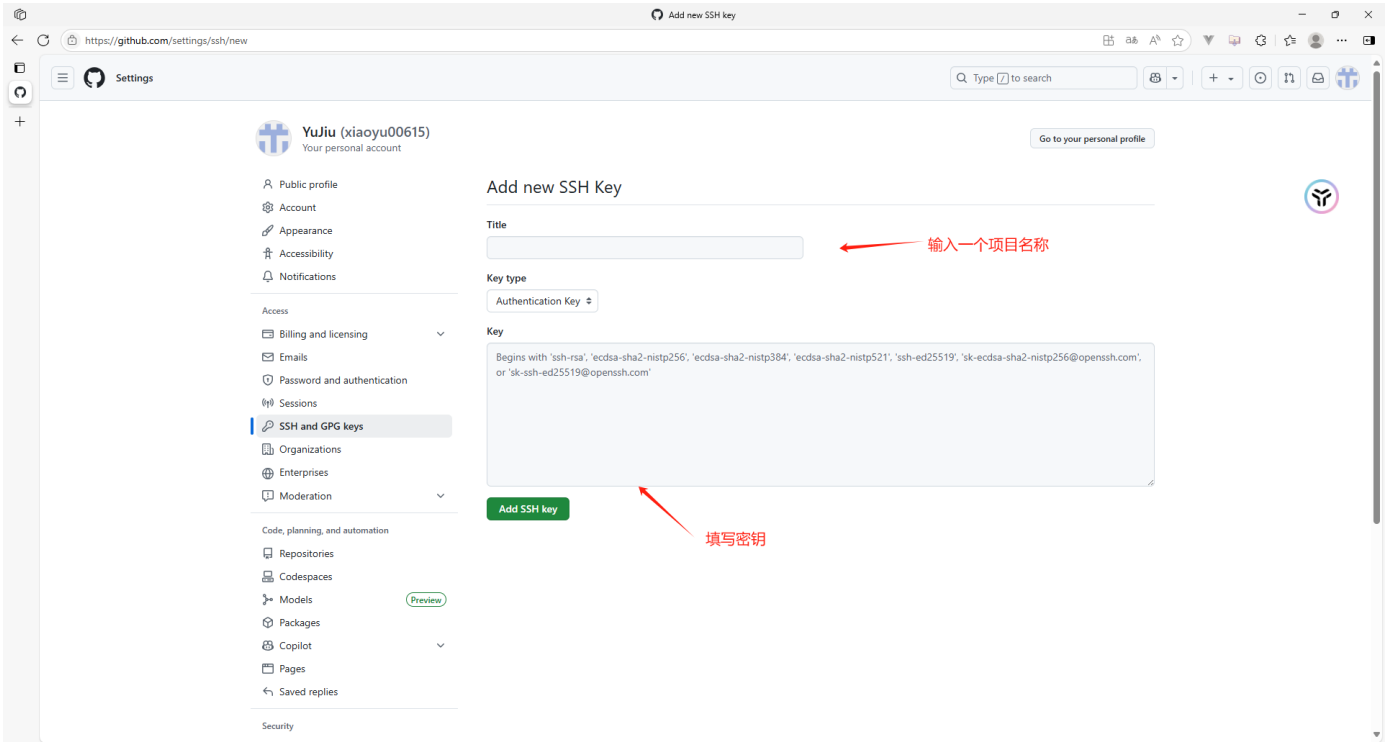
4.3.1


GitHub Dashboard
https://github.com/dashboard
Dashboard
Type / to search
Top repositories
New
Find a repository...
xiaoyu00615/test_container
xiaoyu00615/vue-practice
xiaoyu00615/test_container_sys
xiaoyu00615/git--
xiaoyu00615/KeyCastOW
Home
Ask Copilot
Create a profile README for me
Suggest next steps for an issue
My open pull requests
Feed
Filter
Trending repositories · See more
srbhr/Resume-Matcher
Improve your resumes with Resume Matcher. Get insights, keyword suggestions and tune your resumes to job descriptions.
TypeScript 17.3k
Star
maybe-finance/maybe
The personal finance app for everyone
Ruby 51.9k
Latest changes
16 hours ago
Agents page:
GitHub Copilot
16 hours ago
GitHub Copilot
supports .instru
16 hours ago
GitHub Spark i
Copilot Pro+ s
Yesterday
GitHub Apps c
scanning push
View changelog
© 2025 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information
xiaoyu00615
YuJiu
Set status
Your profile
Your repositories
Your Copilot
Your projects
Your stars
Your gists
Your organizations
Your enterprises
Your sponsors
Try Enterprise
Free
Feature preview
New
GitHub MCP Server
Preview
Settings
GitHub Website
GitHub Docs
GitHub Support
GitHub Community
Sign out

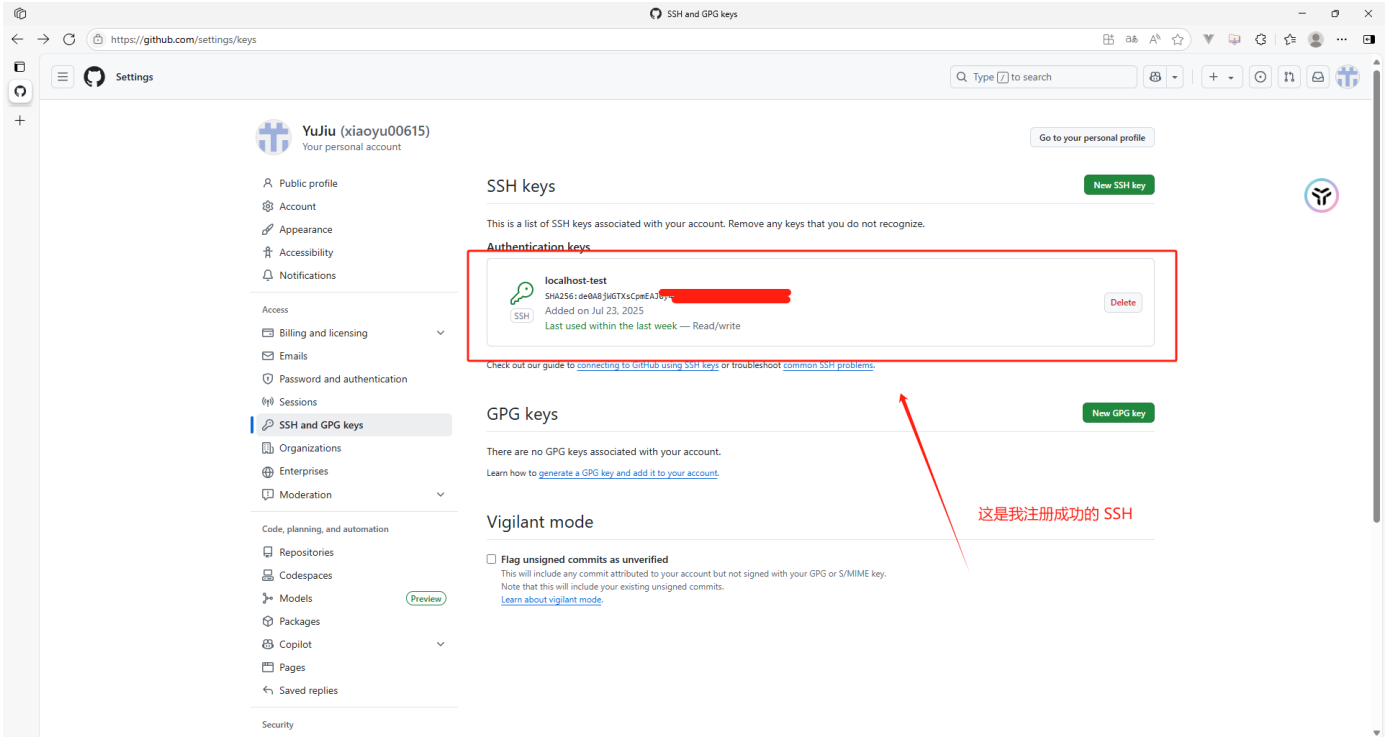
4.3.2


SSH and GPG keys
https://github.com/settings/keys
Settings
Type / to search
YuJiu (xiaoyu00615)
Your personal account
Go to your personal profile
Public profile
Account
Appearance
Accessibility
Notifications
Access
Billing and licensing
Emails
Password and authentication
Sessions
SSH and GPG keys
Organizations
Enterprises
Moderation
Code, planning, and automation
Repositories
Codespaces
Models
Preview
Packages
Copilot
Pages
Saved replies
Security
SSH keys
New SSH key
This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.
Authentication keys
localhost-test
SHA256:de0A8jWGTXsCpmEAJ0y4/G2iDwCWgZ8v67QSp9d3Pg0
Added on Jul 23, 2025
Last used within the last week — Read/write
SSH
Delete
Check out our guide to connecting to GitHub using SSH keys or troubleshoot common SSH problems.
GPG keys
New GPG key
There are no GPG keys associated with your account.
Learn how to generate a GPG key and add it to your account.
Vigilant mode
Flag unsigned commits as unverified
This will include any commit attributed to your account but not signed with your GPG or S/MIME key.
Note that this will include your existing unsigned commits.
Learn about vigilant mode.
27°C
阴
搜索
16:39
2025/7/24

4.3.3



4.3.4



4.4、验证是否连接成功

验证 SSH 连接是否成功：
执行以下命令：

```
bash
ssh -T git@github.com
```

首次连接会提示 Are you sure you want to continue connecting (yes/no)?，输入 yes 并回车。

```
成功会显示：Hi xiaoyu00615! You've successfully authenticated, but GitHub does not
provide shell access.
如果失败，检查公钥是否复制正确，或重新生成密钥重试。
```

5、工作流程

Git 的核心是「工作区→暂存区→本地仓库→远程仓库」的流程，常用命令如下：

5.1、创建 / 初始化仓库 - bash

$ 新建一个文件夹

mkdir my-project

cd my-project

$ 初始化后 会在当前目录生成 .git 隐藏文件夹 （仓库核心文件） - 本地仓库

git init

$ 在当前文件夹中添加文件 - 工作区

index.html

style.css

script.js

$ 添加文件夹所有修改 工作区的代码 工作区 -> 暂存区

git add .

$ 暂存区 -> 本地仓库

git commit

$ 连接远程仓库 进行关联 后面的远程仓库是我的记得修改 - 建议使用 SSH

git remote add origin 填写远程仓库

如：git@github.com:xiaoyu00615/vue-practice.git

$ 本地仓库推送到远程仓库 本地仓库 -> 远程仓库

git push -u origin main