1.https://link.zhihu.com/?target=https%3A//github.com/ 在GitHub注册username=昵称 Email=个人邮箱

password=密码 下面是验证 完后就可以使用了 。

GitHub 的仓库分为两种，一种是public repositories公开免费版，一种是private repositories私有付费版。其中，免费版是完全公开的，而私有版一般是由企业或者不愿公开仓库的个人用户购买，在这里，我们选择免费版就可以



按照上图，创建出自己的仓库。



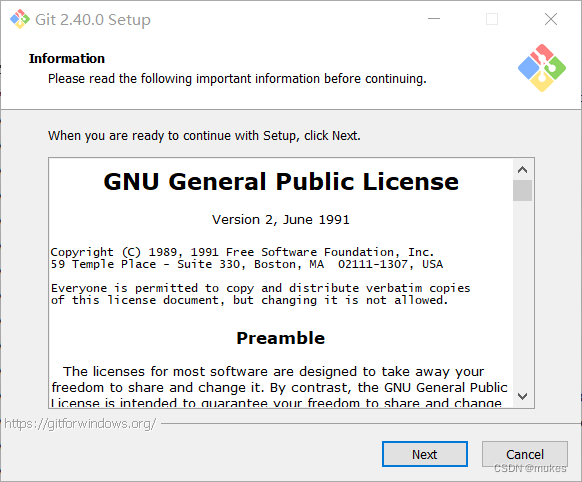
依照上图，点击下载

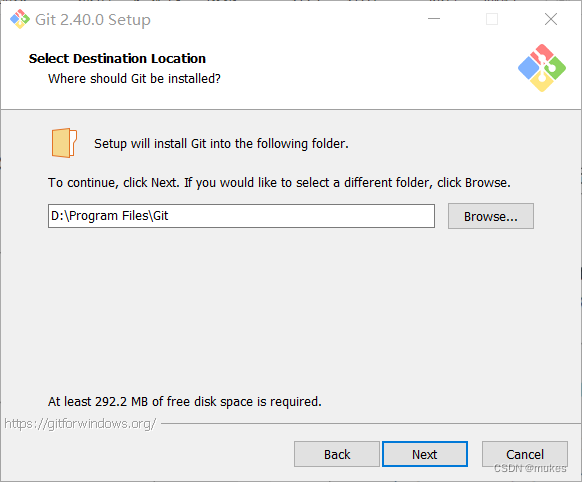


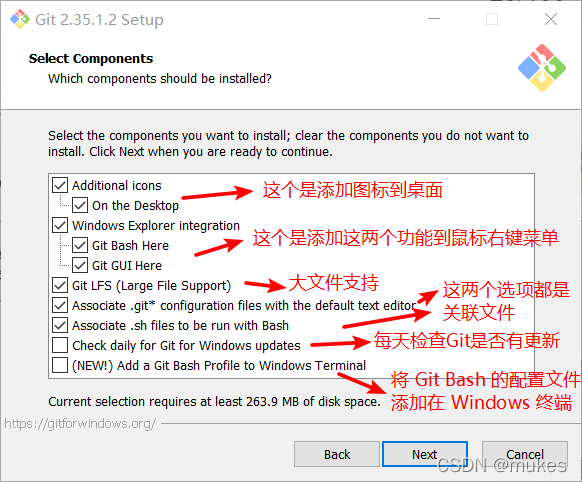
再点击那个窗户（其实是Windows的意思）

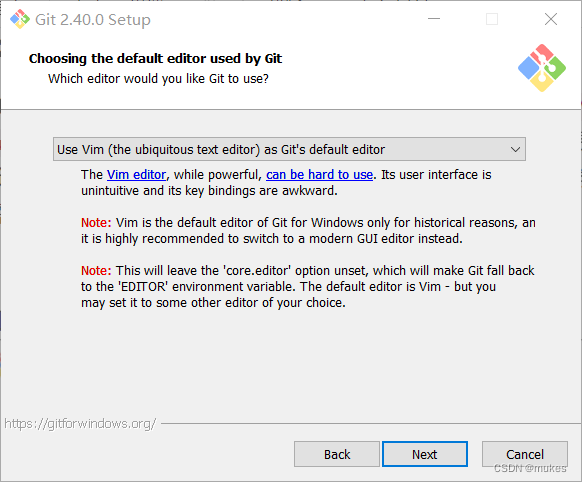


选64bit git for windows setup的（现在电脑应该没有32bit的吧,有当我没说）

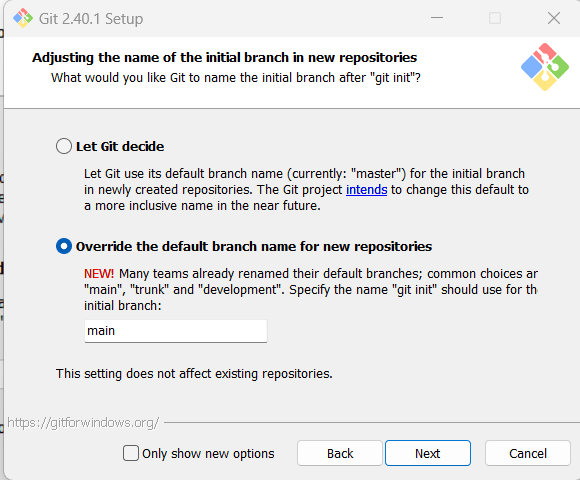






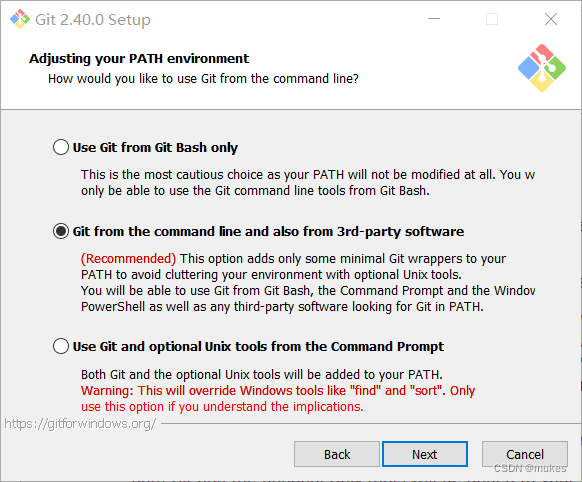


这步为默认编辑器 vim可直接点击 [next] 到第六步



第一种是让 Git 自己选择，名字是 master ，但是未来也有可能会改为其他名字；第二种是我们自行决定，默认是 main，当然，你也可以改为其他的名字。一般默认第一种，点击 [next] 到第七步。

要选第二个，不然就像我一样后面要改，因为GitHub库已经全面改成main了



Use Git from Git Bash only

This is the most cautious choice as your PATH will not be modified at all. You w only be able to use the Git command line tools from Git Bash.

仅从 Git Bash 使用 Git

这是最谨慎的选择，因为您的 PATH 根本不会被修改。您将只能使用 Git Bash 中的 Git 命令行工具。

Git from the command line and also from 3rd-party software

(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools.

You will be able to use Git from Git Bash, the Command Prompt and the Windov PowerShell as well as any third-party software looking for Git in PATH.

从命令行以及第三方软件进行 Git

（推荐）此选项仅将一些最小的 Git 包装器添加到PATH中，以避免使用可选的 Unix 工具使环境混乱。

您将能够使用 Git Bash 中的 Git，命令提示符和 Windov PowerShell 以及在 PATH 中寻找 Git 的任何第三方软件。

Use Git and optional Unix tools from the Command Prompt

Both Git and the optional Unix tools will be added to your PATH.

Warning: This will override Windows tools like "find"and "sort". Only use this option if you understand the implications.

使用命令提示符中的 Git 和可选的 Unix 工具

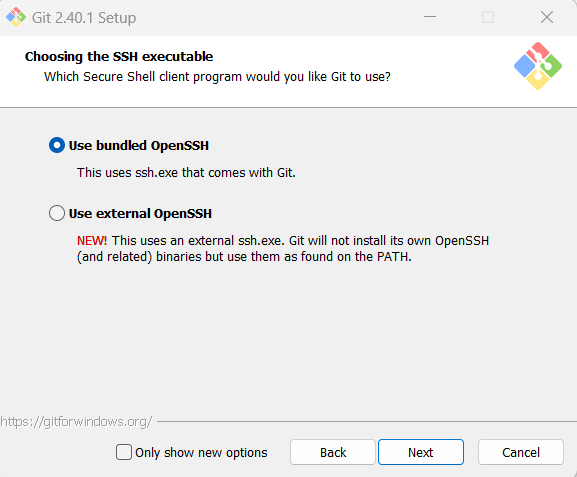
Git 和可选的 Unix 工具都将添加到您的 PATH 中。

警告：这将覆盖 Windows 工具，例如 "find" and "sort". 仅在了解其含义后使用此选项。

第一种是仅从 Git Bash 使用 Git。这个的意思就是你只能通过 Git 安装后的 Git Bash 来使用 Git ，其他的什么命令提示符啊等第三方软件都不行。

第二种是从命令行以及第三方软件进行 Git。这个就是在第一种基础上进行第三方支持，你将能够从 Git Bash，命令提示符(cmd) 和 Windows PowerShell 以及可以从 Windows 系统环境变量中寻找 Git 的任何第三方软件中使用 Git。推荐使用这个。

第三种是从命令提示符使用 Git 和可选的 Unix 工具。选择这种将覆盖 Windows 工具，如 “ find 和 sort ”。只有在了解其含义后才使用此选项。一句话，适合比较懂的人折腾。



Use bundled OpenSSH

This uses ssh. exe that comes with Git.

使用捆绑的 OpenSSH

这使用的 ssh.exe 是 Git 自带的

Use external OpenSSH

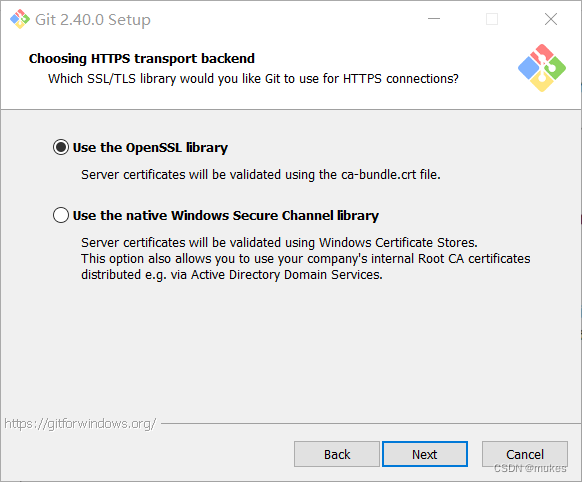
NEW! This uses an external ssh. exe. Git will not install its own OpenSSH

(and related) binaries but use them as found on the PATH.

使用外部 OpenSSH

新！这使用外部 ssh.exe 文件。

Git 不会安装自己的 OpenSSH（和相关）二进制文件，而是使用在环境变量 PATH 中找到的它们。



use the OpenSSL library

Server certificates will be validated using the ca-bundle. crt file.

使用 OpenSSL 库

服务器证书将使用 ca-bundle.crt 文件进行验证。

Use the native Windows Secure Channel library

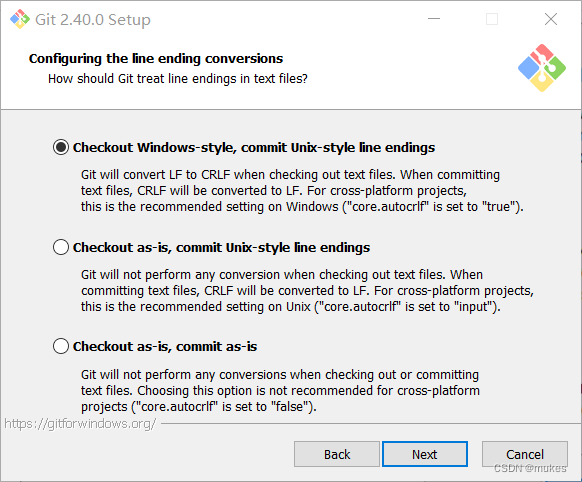
Server certificates will be validated using Windows Certificate Stores.

This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.

使用本机 Windows 安全通道库

服务器证书将使用 Windows 证书存储进行验证。

此选项还允许您使用公司内部分发的内部根 CA 证书，例如通过 Active Directory 域服务。



Checkout Windows-style, commit Unix-style line endings

Git will convert LF to CRLF when checking out text files.

When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows("core. autocrif"is set to "true").

签出 Windows 样式，提交 Unix 样式的行结尾

Git 签出文本文件时，会将 LF 转换为 CRLF。

提交文本文件时，CRLF 将转换为 LF。

对于跨平台项目，这是 Windows 上的建议设置（"core.autocrif" 设置为 "true"）。

Checkout as-is, commit Unix-style line endings

Git will not perform any conversion when checking out text files.

When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix("core.autocrif" is set to "input").

按原样签出，提交 Unix 样式的行结尾

Git 在签出文本文件时不会执行任何转换。提交文本文件时，CRLF 将转换为 LF。

对于跨平台项目，这是在 Unix 上的建议设置（"core.autocrif" 设置为 "input"）。

Checkout as-is, commit as-is

Git will not perform any conversions when checking out or committing text files.

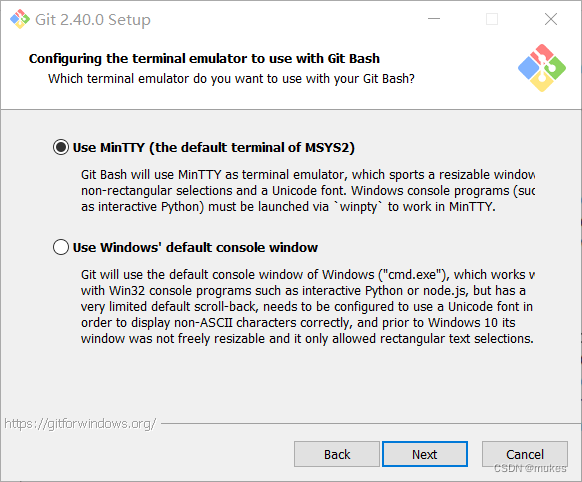
Choosing this option is not recommended for cross-platform projects("core. autocrif"is set to "false").

按原样签出，按原样提交

Git 在签出或提交文本文件时不会执行任何转换。

不建议跨平台项目选择此选项（"core.autocrif" 设置为 "false"）。

这三种选择分别是：  
签出 Windows 样式，提交 Unix 样式的行结尾。  
按原样签出，提交Unix样式的行结尾。  
按原样签出，按原样提交。



Use MinTTY(the default terminal of MSYS2)

Git Bash will use MinTTY as terminal emulator, which sports a resizable window

non-rectangular selections and a Unicode font.Windows console programs(such

as interactive Python) must be launched via 'winpty' to work in MinTTY.

使用 MinTTY（MSYS2的默认终端）

Git Bash 将使用 MinTTY 作为终端仿真器，该仿真器具有可调整大小的窗口非矩形选择和 Unicode 字体。

Windows 控制台程序（例如交互式 Python）必须通过 "winpty" 启动才能在 MinTTY 中运行。

Use Windows' default console

window Git will use the default console window of Windows("cmd.exe"), which works v

with Win32 console programs such as interactive Python or node. js, but has a

very limited default scroll-back,needs to be configured to use a Unicode font in

order to display non-ASCII characters correctly, and prior to Windows 10 its

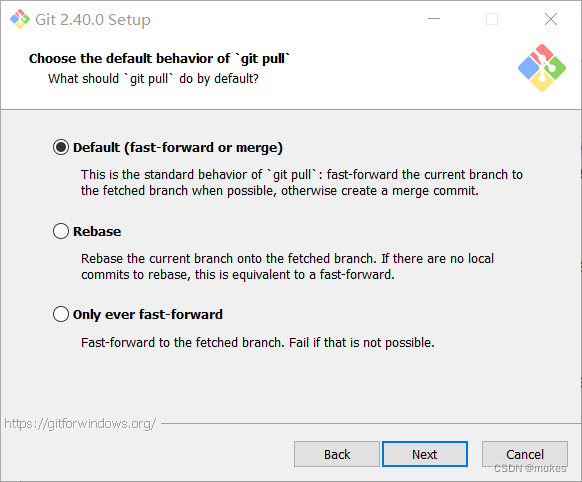
window was not freely resizable and it only allowed rectangular text selections.<br>

使用 Windows 的默认控制台窗口

Git 将使用 Windows 的默认控制台窗口（"cmd.exe"），该窗口可与 Win32 控制台程序（例如交互式Python 或

node.js）一起使用，但默认回滚非常有限，需要将其配置为使用 Unicode 字体才能正确显示非 ASCII 字符，并且在

Windows 10 之前，其窗口不可随意调整大小，并且仅允许选择矩形文本。



ODefault(fast-forward or merge)

This is the standard behavior ofgit pull": fast-forward the current branch to

the fetched branch when possible, otherwise create a merge commit.

默认（快进或合并）

这是 "git pull" 的标准行为：在可能的情况下将 当前分支 快进到 获取的分支，否则创建合并提交。

ORebase Rebase the current branch onto the fetched branch. If there are no local

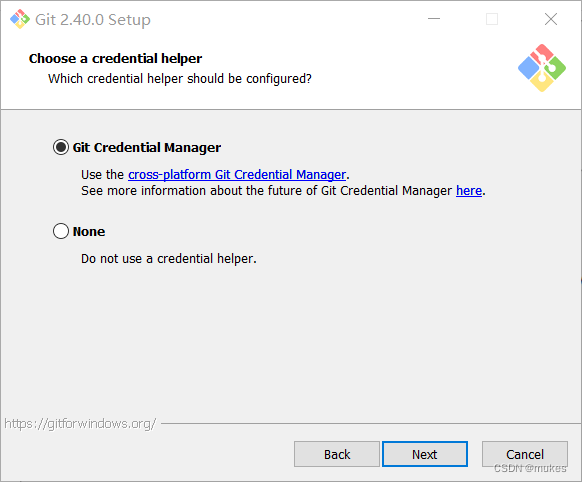
commits to rebase, this is equivalent to a fast-forward.

变基将当前分支变基到获取的分支上。如果没有本地提交要变基，则等同于快进。

Oonly ever fast-forward

Fast-forward to the fetched branch. Fail if that is not possible.

只能快进快进到获取的分支。如果不可能，则失败。



Git Credential Manager

Use the cross-platform Git Credential Manager.

See more information about the future of Git Credential Manager here.

Git 凭证管理

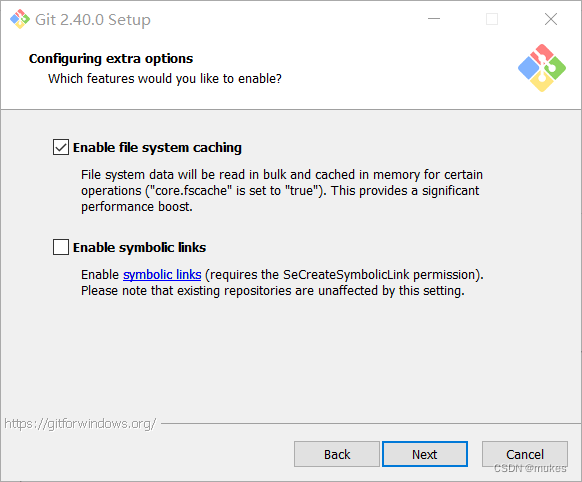
使用跨平台的 Git 凭证管理。

在此处查看有关 Git 凭证管理未来的更多信息。

None

Do not use a credential helper.

不使用凭证助手。



Enable file system caching

File system data will be read in bulk and cached in memory for certain operations("core.fscache" is set to "true").

This provides a significant performance boost.

启用文件系统缓存

将批量读取文件系统数据并将其缓存在内存中以进行某些操作（"core.fscache” 设置为 "true"）。

这可以显着提高性能。

Enable symbolic links

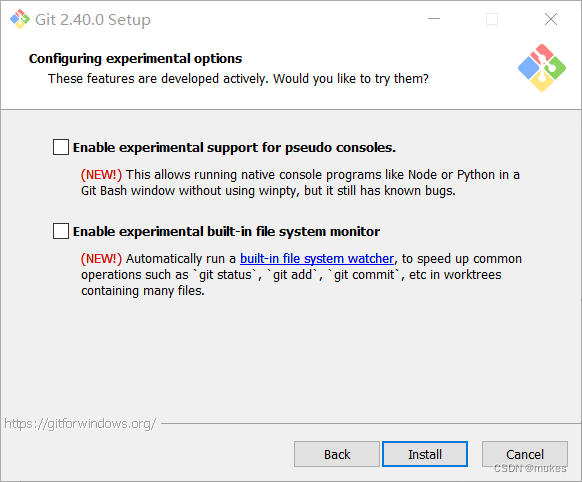
Enable symbolic links(requires the SeCreateSymbolicLink permission).

Please note that existing repositories are unaffected by this setting.

启用符号链接

启用符号链接（需要SeCreateSymbolicLink权限）。

请注意，现有存储库不受此设置的影响。



Enable experimental support for pseudo consoles.

(NEW!) This allows running native console programs like Node or Python in a Git Bash window without using winpty,

but it still has known bugs.

启用对伪控制台的实验性支持。

(新功能!) 这允许在不使用 winpty 的情况下在 Git Bash 窗口中运行诸如 Node 或 Python 之类的本机控制台程序，

但是它仍然存在已知的 bug。

Enable experimental built-in file system monitor

(NEW!) Automatically run a built-in file system watcher, to speed up common operations such as ' git status', ' git add', ' git commit', etc in worktrees containing many files.

启用实验性内置文件系统监视器

（新！）自动运行内置文件系统监视器，以加快包含许多文件的工作树中的常见操作，例如 'git status'、'git add'、'git commit' 等.

这是实验性功能，可能会有一些小错误之类的，建议不用开启。

以上就是git的安装步骤

给大家讲一下新装的git，他有git bash,git cmd,git gui,git faqs,git release note,

Git bash:是基于CMD的，在CMD的基础上增添一些新的命令与功能，平时主要用这个，功能很丰富

Git cmd:不能说和 cmd 完全一样，只能说一模一样，功能少得可怜

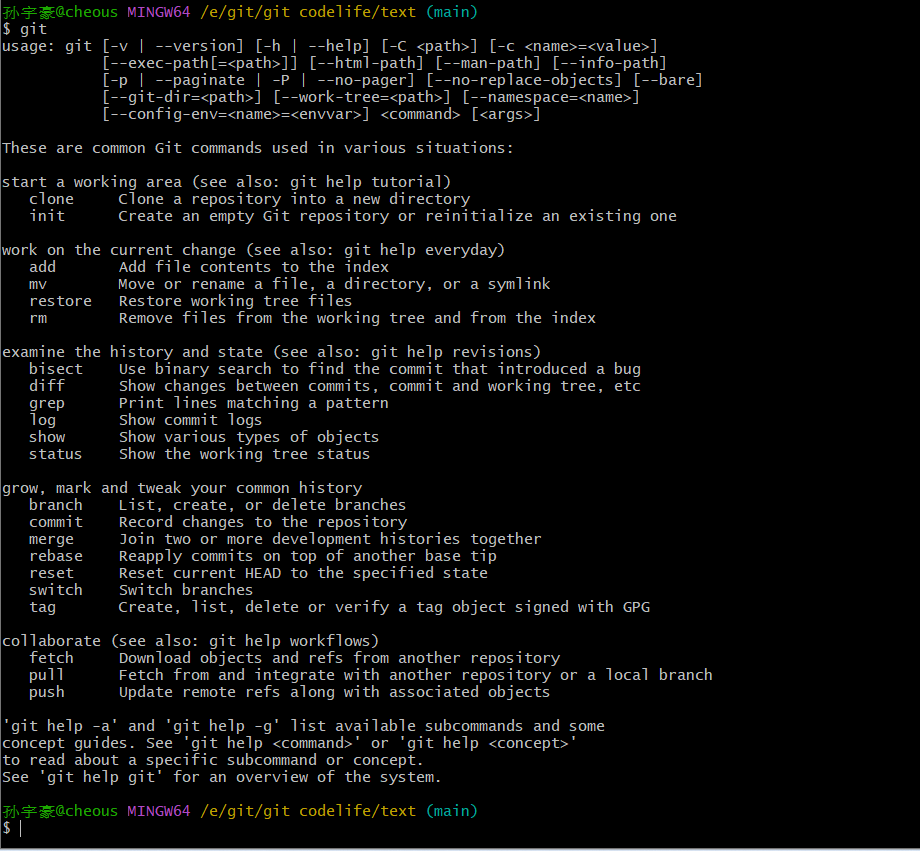
Git faqs:就是 Git Frequently Asked Questions（常问问题）

Git gui:就是 Git 的图形化界面

7.



点击鼠标右键，打开git bash here

8.

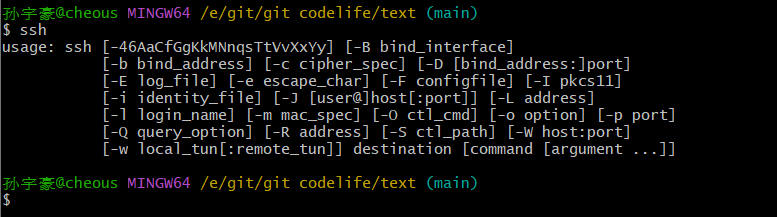
打开后输入git,回车得到上图，你的git就安装完成了

9.在之前我已经完成了GitHub的注册还有git的安装，现在就是将git与GitHub绑定了。为什么要绑定GitHub呢？就是为了方便我们上传文件。

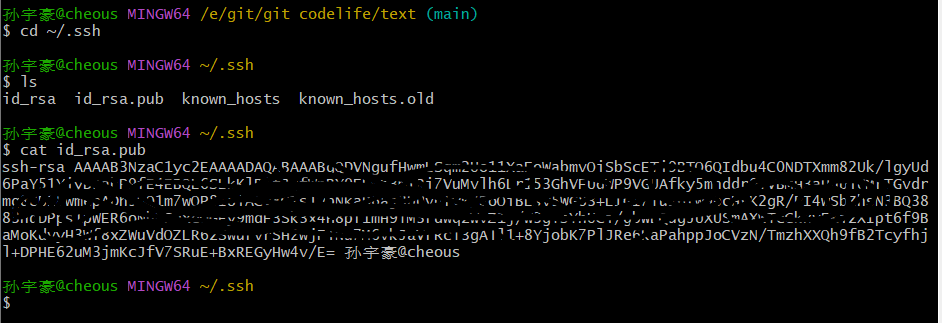
**10.绑定GitHub**



我们要用git上传文件到GitHub首先得利用SSH登录远程主机，而登录方式有两种：一种是口令登录；另一种是公钥登录。口令登录每次都要输入密码十分麻烦，而公钥登录就省去了输入密码的步骤，所以我们选择公钥授权。首先我们得在 GitHub 上添加 SSH key 配置，要想生成SSH key，就要先安装 SSH，不过我们安装了 Git Bash，其应该自带了 SSH。检验一下是否安装 SSH，我们在新建的文件夹中右键打开 Git Bash（这是我自己新建的）



输入ssh，要是和上图一样就ok



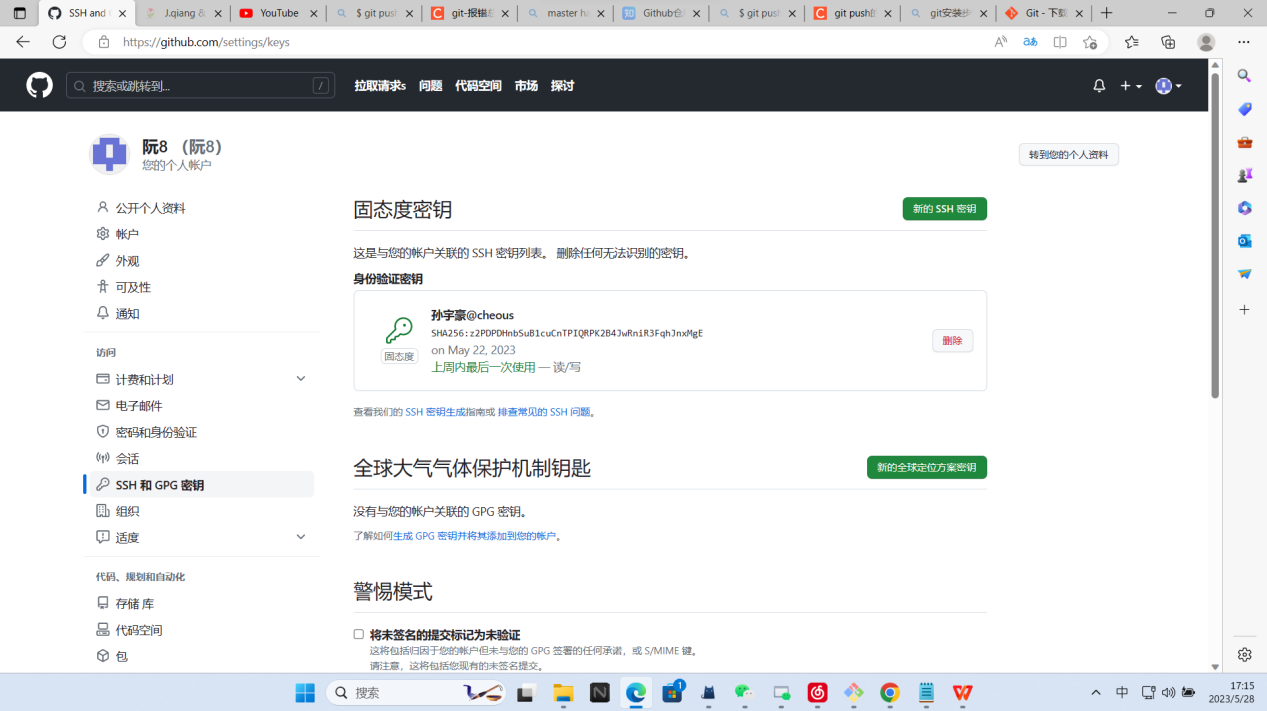
$ cd **~/**.ssh

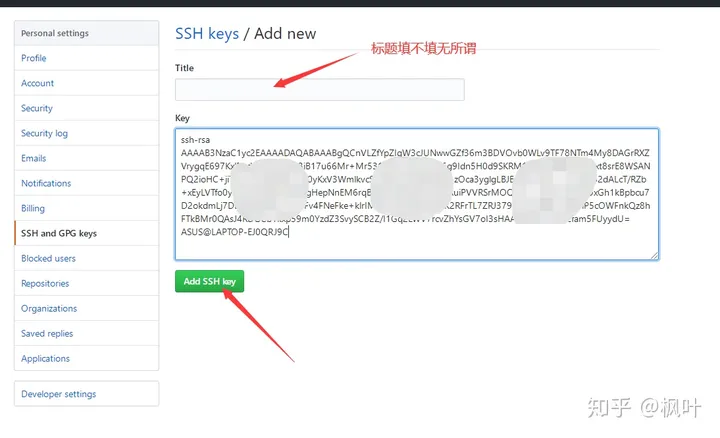
$ ls

$ cat id\_rsa.pub

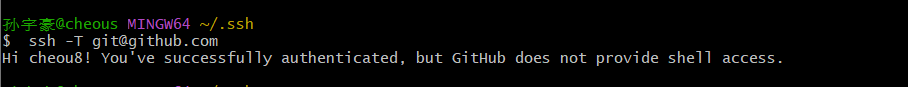
复制那段公钥，（git中的复制粘贴不是 Ctrl+C 和 Ctrl+V，而是 Ctrl+insert 和Shift+insert）







前两张是我的，但我已经做了，在网上找了一个把之前ssh公钥照这样粘过去就行



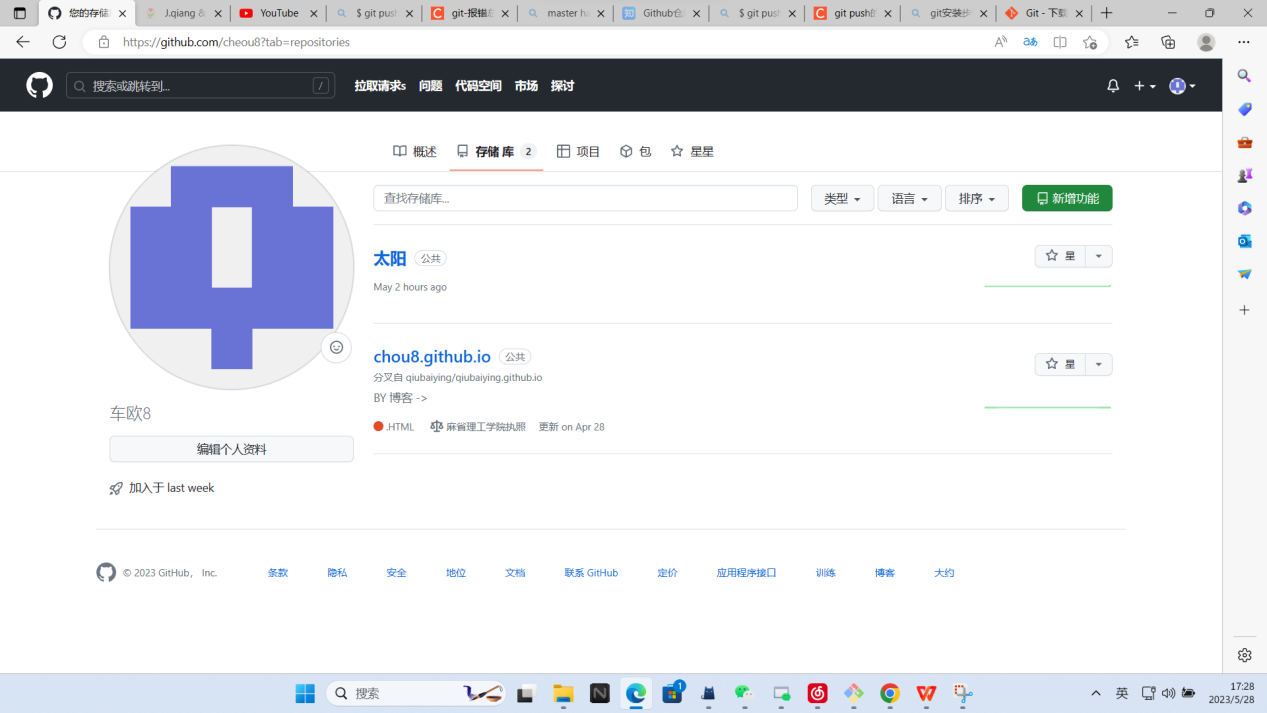
验证是否成功，我们可以通过在 Git Bash 中输入 ssh -T [git@github.com](mailto:git@github.com) 进行检验，我这样就是成功了，第一次验证时会询问你，填yes就行

## **提交文件**

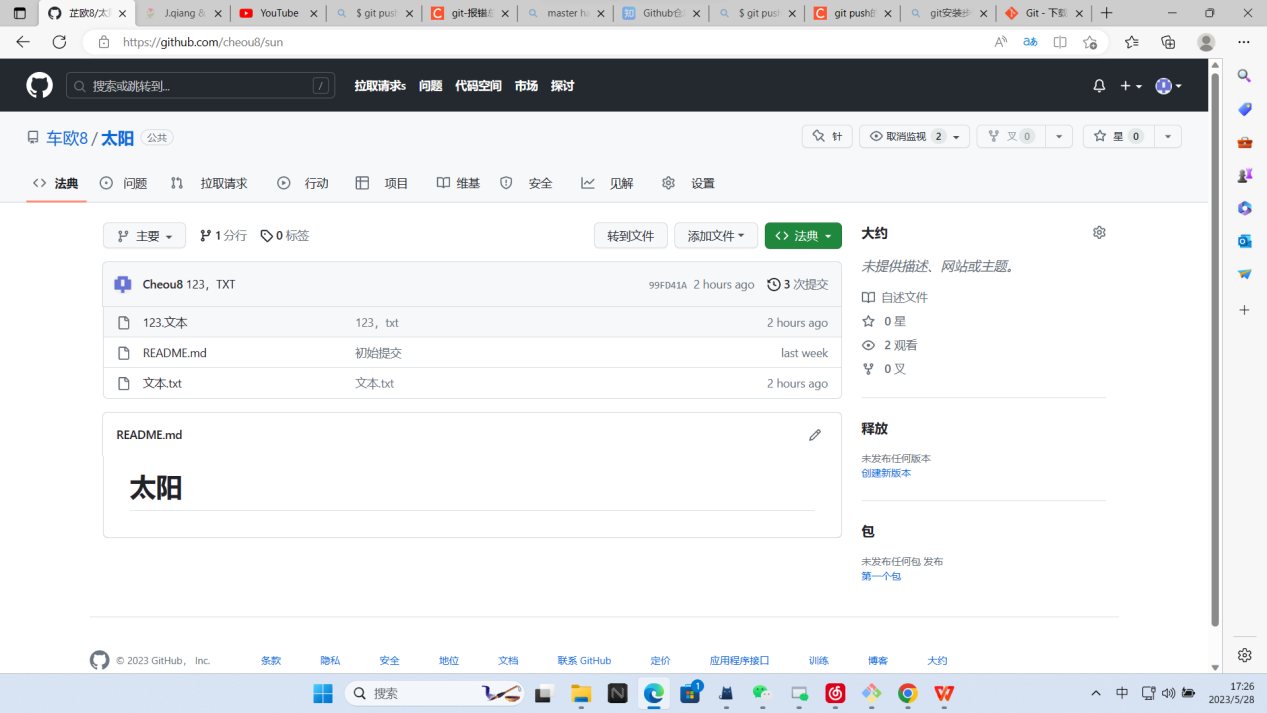
1. 本地没有 git 仓库
2. 直接将远程仓库 clone 到本地；
3. 将文件添加并 commit 到本地仓库；
4. 将本地仓库的内容push到远程仓库。

————————————————————————————————————

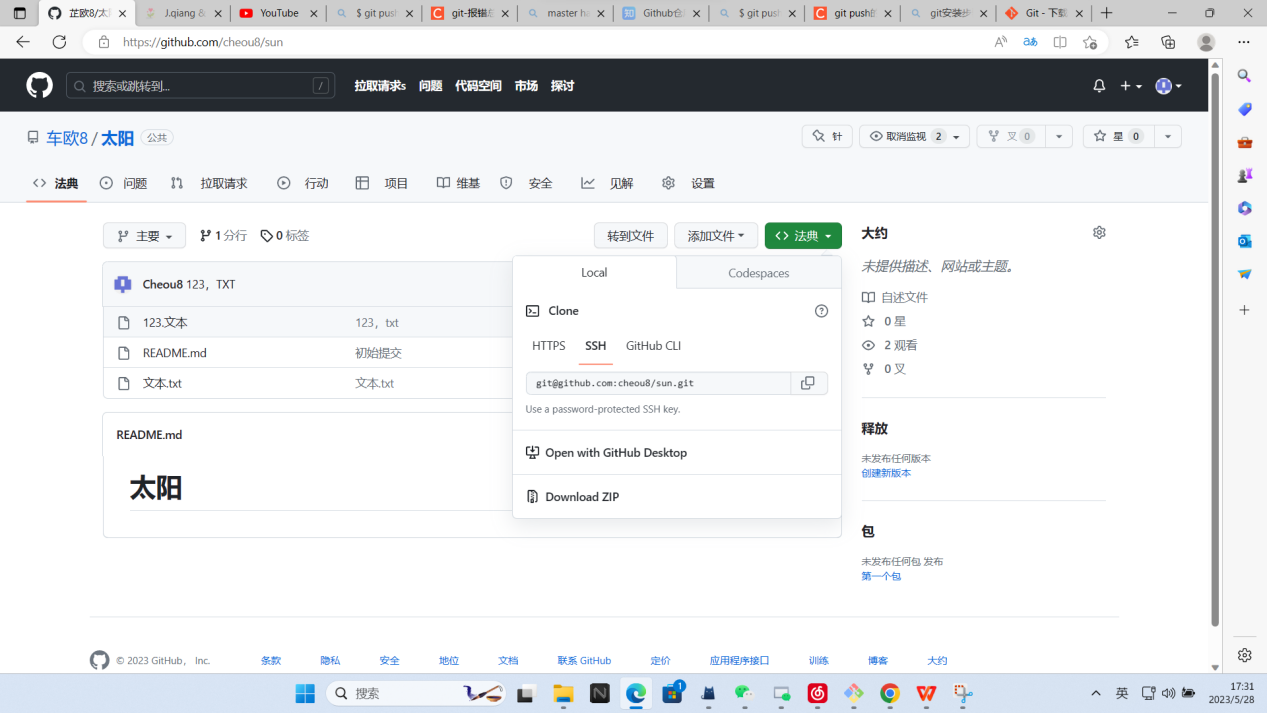
上面是原理，下面是演示



打开GitHub个人主页



打开我创建的sun（就是太阳，我用了翻译）项目

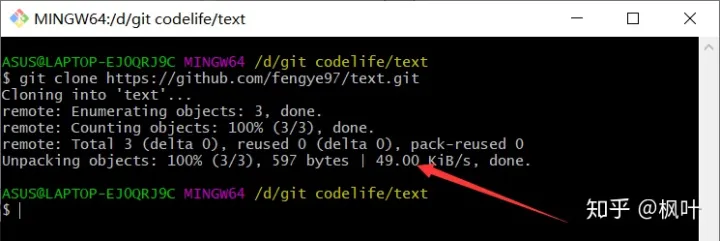


我用的是ssh,因为翻了资料，大佬是这么说的：（简单来说，你要安全就ssh,如没这要求就https）

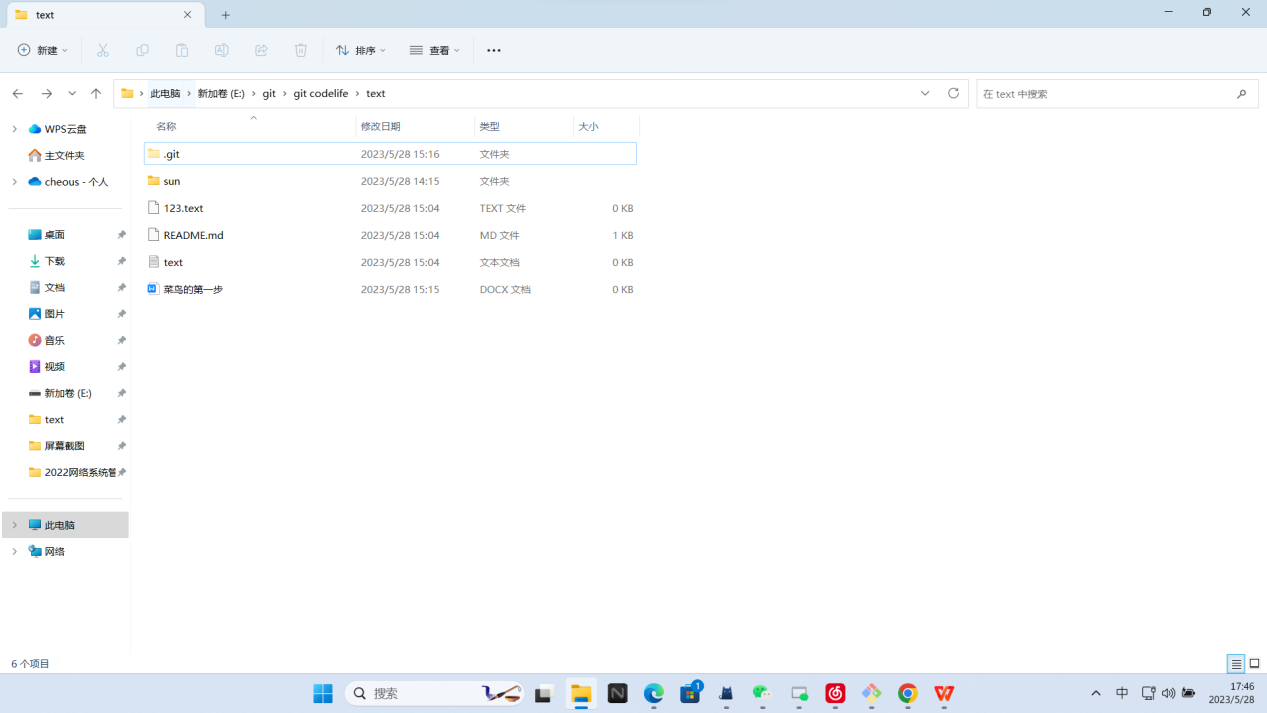
HTTPS：不管是谁，拿到url随便clone，但是在push的时候需要验证用户名和密码；

SSH：clone的项目你必须是拥有者或者管理员，而且需要在clone前添加SSH Key。SSH 在push的时候，是不需要输入用户名的，如果配置SSH key的时候设置了密码，则需要输入密码的，否则直接是不需要输入密码的。

言归正传，复制这段地址

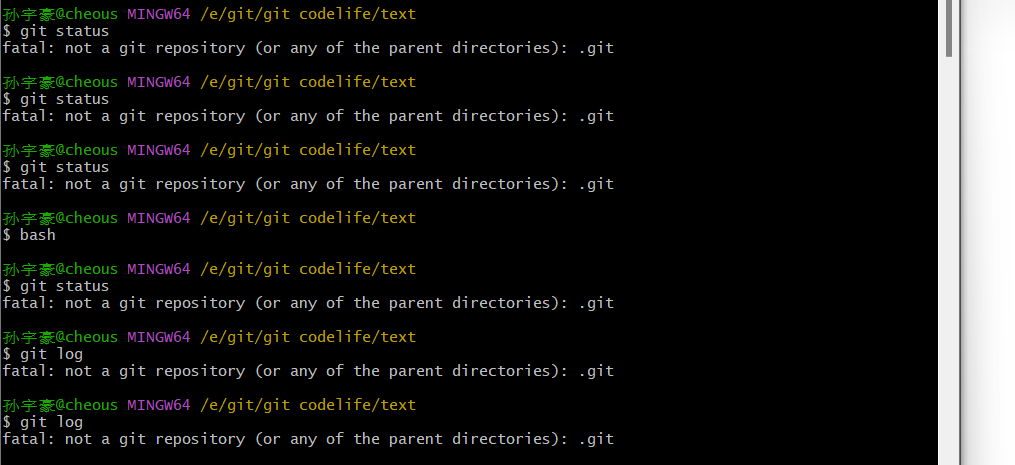


接下来，就输入git clone 你复制的地址

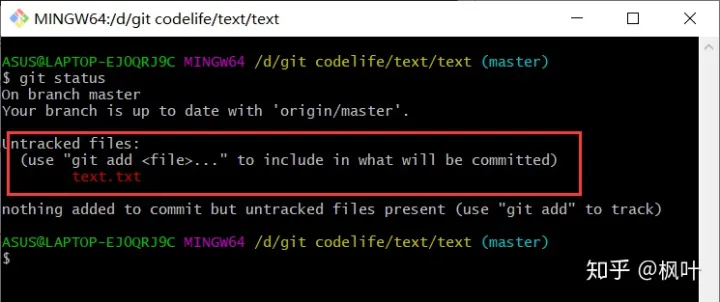


接下来看看你的本地仓库是否有了你远程仓库的文件（第一个.git是隐藏文件）

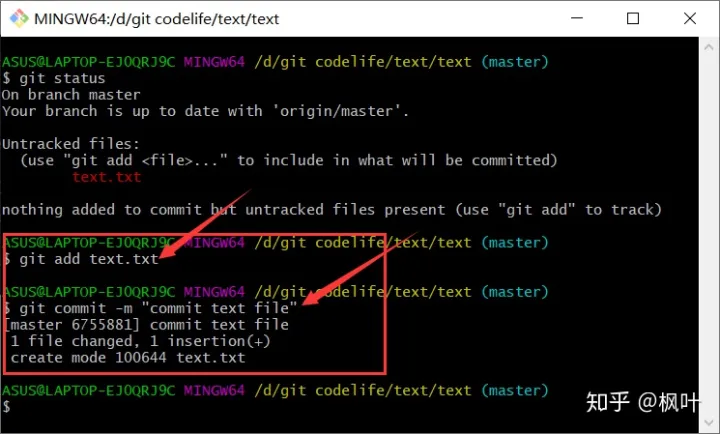
那我们创建一个菜鸟的第一步试一试吧（其实第一个是text.txt,你们第一个可以先创一个text.txt）



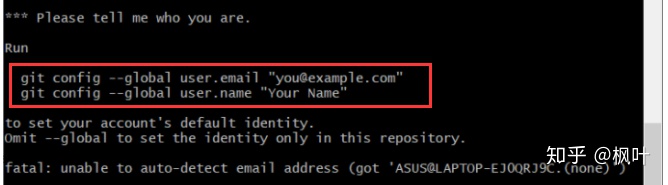
当你输入**git status** 时，会发现emmm,没用，错了。为什么呢？提示说没有.git这样一个目录，是由于git没有初始化，解决办法如下：**git init**就可以了！



从此目录进入 Git Bash，输入 git status 命令查看仓库状态



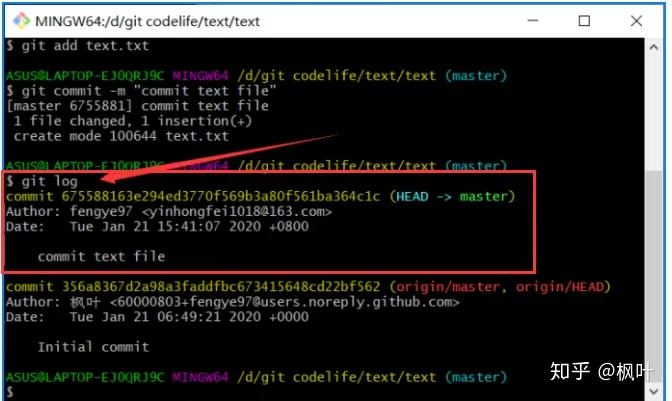
现在我们使用 **git add** 命令将文件添加到了「临时缓冲区」，再用 **git commit -m 提交信息** 将其提交到本地仓库



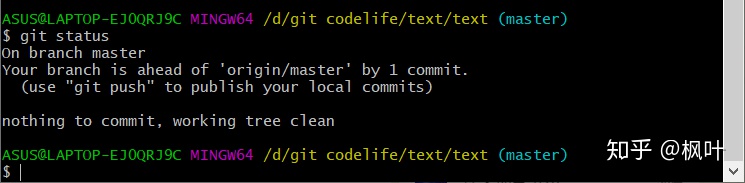
如果你是第一次提交的话，就会出现下图的情况，会让你输入用户名和邮箱：

$ git config **--**global user.name"fengye97"

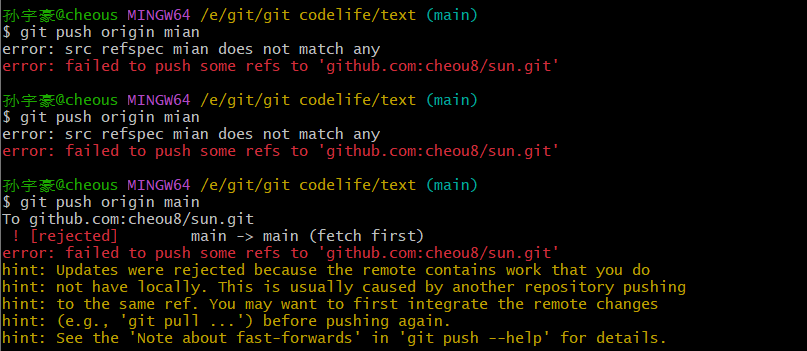
$ git config **--**global user.email"\*\*\*@\*\*\*.com"



完成后输入 git log 命令查看仓库提交日志



再输入 git status 查看一下仓库状态



到最后要把它上传到远程仓库时，我发现怎么也弄不好，后来一搜发现我没有origin这个东西。

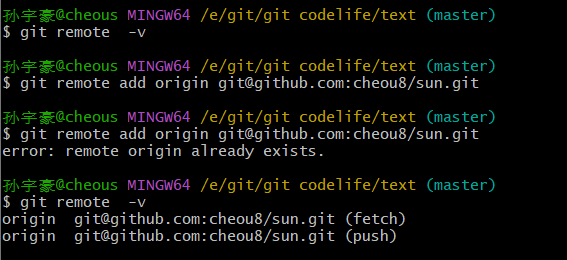
git remote -v： 查看远程仓库详细信息，可以看到仓库名称，关联地址

git remote remove orign：删除orign仓库（比如名称错误）

git remote add origin 仓库地址：重新添加远程仓库地址

gti push -u origin master： 提交到远程仓库的master主干

我们要怎么用呢？

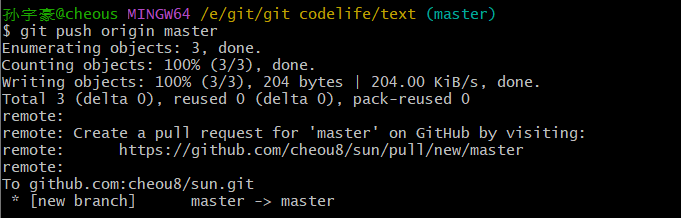


一定要记得远程仓库的地址也要用ssh的（原因如下）

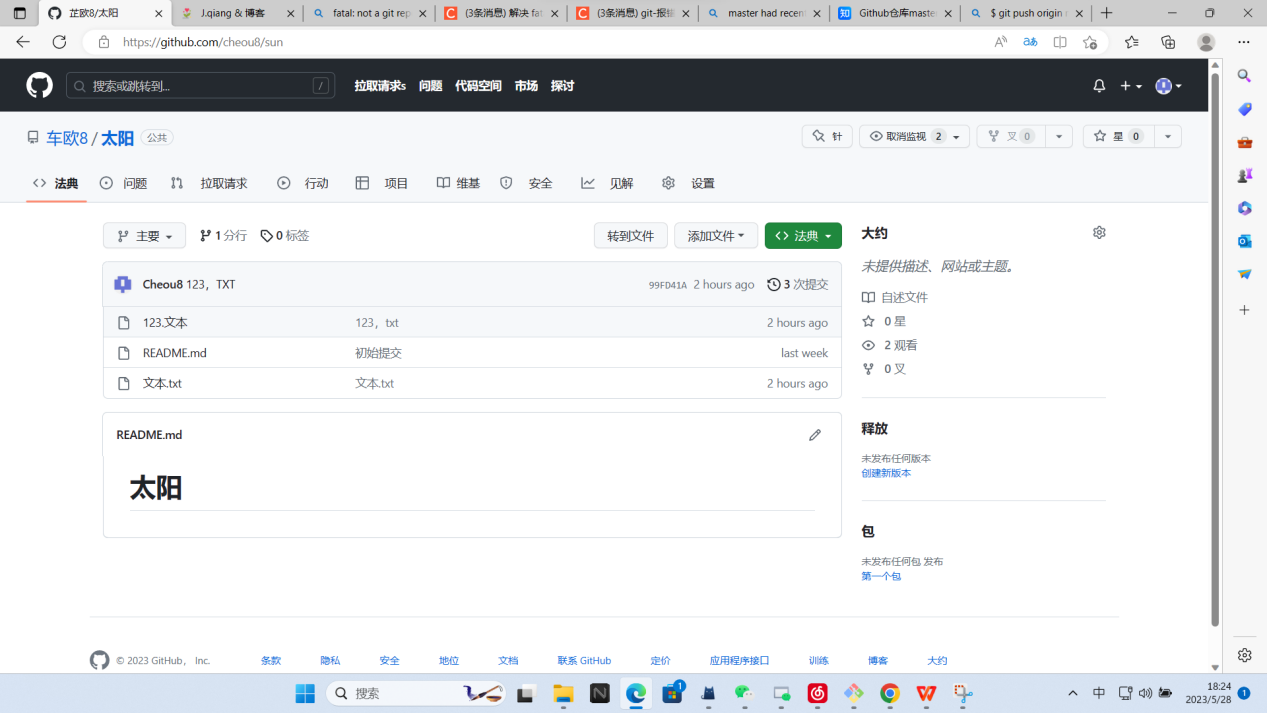
本地Git与远端服务器GitLab之间连接方式主要分为：SSH方式和HTTP方式。

 HTTP方式：这种方式要求project在创建的时候只能选择“Public”公开状态，Private和Internal私有模式下不能使用http方式进行连接，如果考虑到安全性必须进行安全设置。

 SSH方式：这是一种相对安全的方式，本地git仓库和远端gitlab仓库之间的传输是通过SSH加密，SSH方式在三种project模式下都可以使用。



这样就成功了！！！（这是之前踩坑做的所以括号里是master）



查看自己的博客有没有。