vscode+git本地代码版本管理及远程仓库保存

❗ **广州航海学院ICEBREAKER破冰船机器人实验室培训材料**

如果觉得有帮助，欢迎传阅学习，尊重文章材料出处，二次修改需进行出处说明

内容并不严格严谨，欢迎批评指出

> 花费1.5h来学习vscode+git进行代码版本管理，你就能给你的代码获取存档点，快来学一下吧！

前言

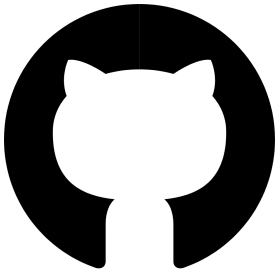
你一定经历过，害怕代码改崩，在自己的电脑上复制粘贴原工程，来实现备份的操作。又或者经历过，代码已经被改崩，但又不知道哪里被改崩的痛苦。

当以上经历发生的时候，打过游戏的你会想着：**《要是写代码也跟游戏一样有存档，能够一键回档》**该有多好。这个时候git就此而生。

💡 **git就像游戏中的存档点，能够对你每一次提交的代码进行版本保存，以便你去任意回溯你的历史版本代码。**

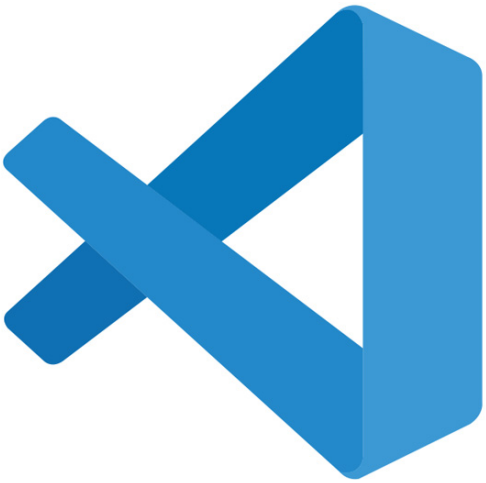
步兵代码的部分历史分支

又或许，你听说过github、gitee等git代码托管平台。想要弄明白个人用户是如何将代码分享到这些平台上进行开源的，或者想使用这些代码托管平台来异地保存自己的项目代码。



因为git的使用，github/gitee的账号注册登录等内容，互联网上已经有了非常多且优秀的教程。因此，针对这一些内容，**不过多赘述**。所以在这里我们更多的是**以vscode的源代码版本管理功能为核心来介绍vscode+git本地代码版本管理和远程仓库保存**。

学习前置准备和目标



在学习vscode+git前，**先确保自己的电脑已经正确安装了vscode和git两个软件（文章最后参考资料附有相相关教程链接），并有一个初步的了解**。这一部分参考互联网教程即可。同时，**在vscode上安装git graph插件**。然后明确我们的几个学习目标：

a. **能够在本地进行简单的代码存档。**

b. **能够在本地进行简单的代码版本回溯。**

c. **能够进行代码的远程仓库的拉取和推送。**

d. **能够进行分支创建和合并。**

> 这几个操作已经能够满足代码管理的入门使用需求。

得益于vscode将git的terminal命令给UI化，我们能够很方便的使用vscode来调用git来维护我们的项目代码。

本地代码版本管理

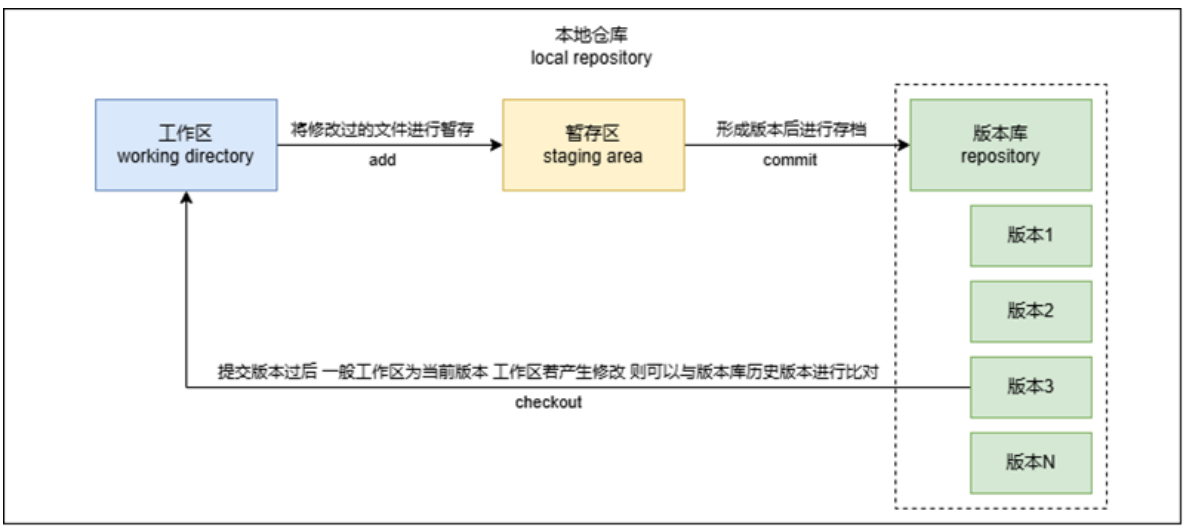
理论

理论和实践的有机辩证统一，我们先来讲理论。

我们根据代码的存储位置，来划分为本地仓库(本地)和远程仓库(远程)的概念。本地仓库就是你代码项目的目录文件夹，而远程仓库就是在代码托管平台所创建的仓库。

首先来看本地仓库。在git的实现逻辑上，我们有这么几个区域划分：**工作区，暂存区，版本库**。

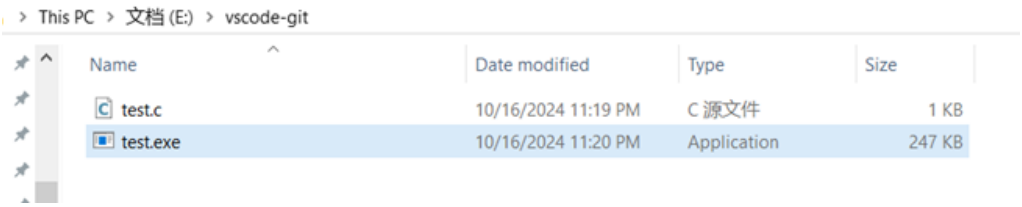
我们进行代码编辑的位置是在工作区当中，当我们在工作区进行代码的编写删除等修改操作后，我们就可以将确认无误的代码提交至暂存区。当暂存区的代码能够形成一个版本的时候，就可以将暂存区的代码提交至版本库当中进行存档。这样我们就可以得到一个版本的代码了。

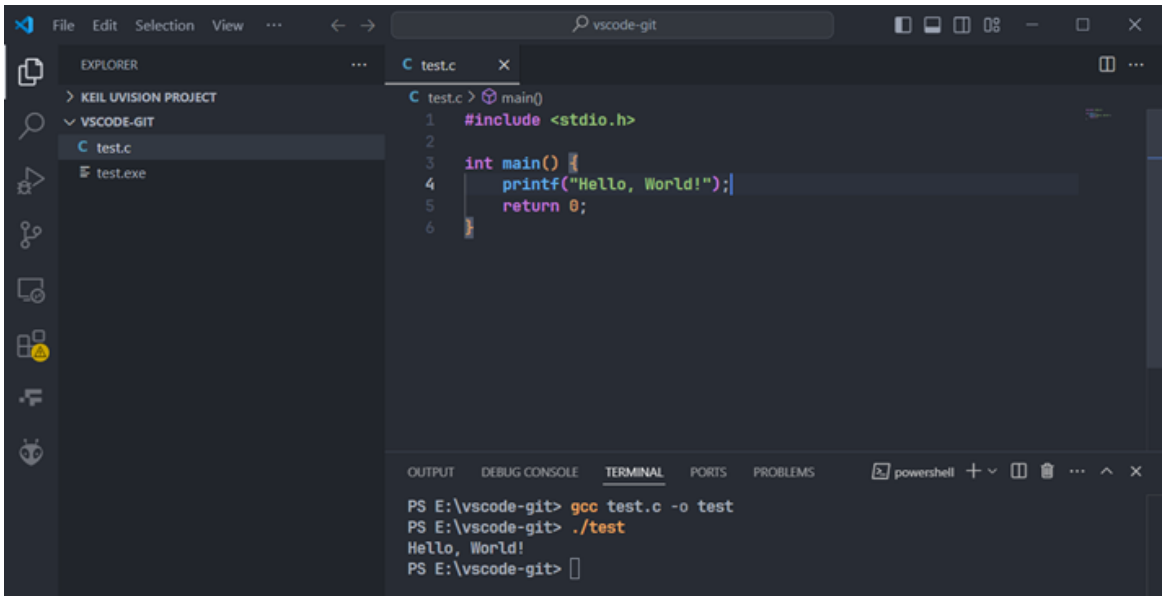


实践

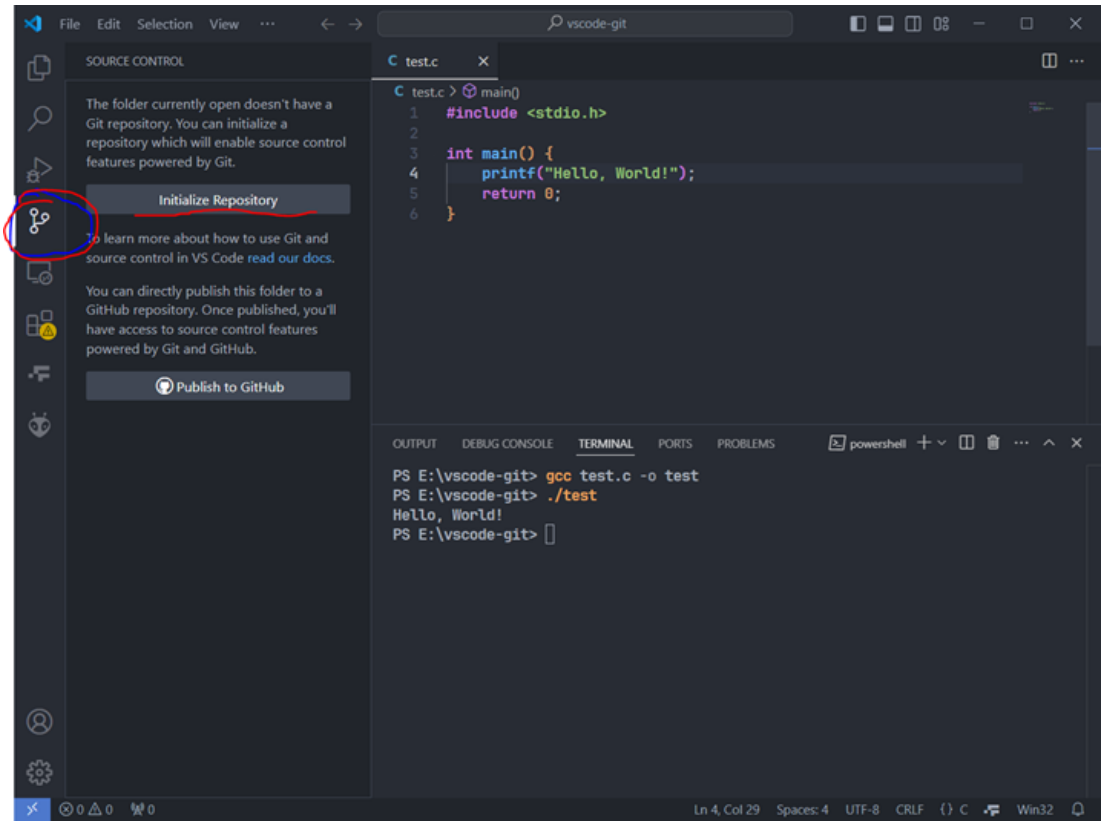
理论存在，开始实践。

A. 这是一个工作区，里面包含一份C的代码和已经编译过的文件。

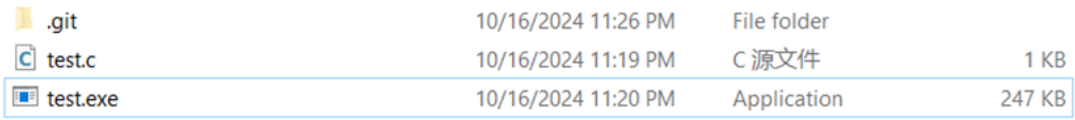


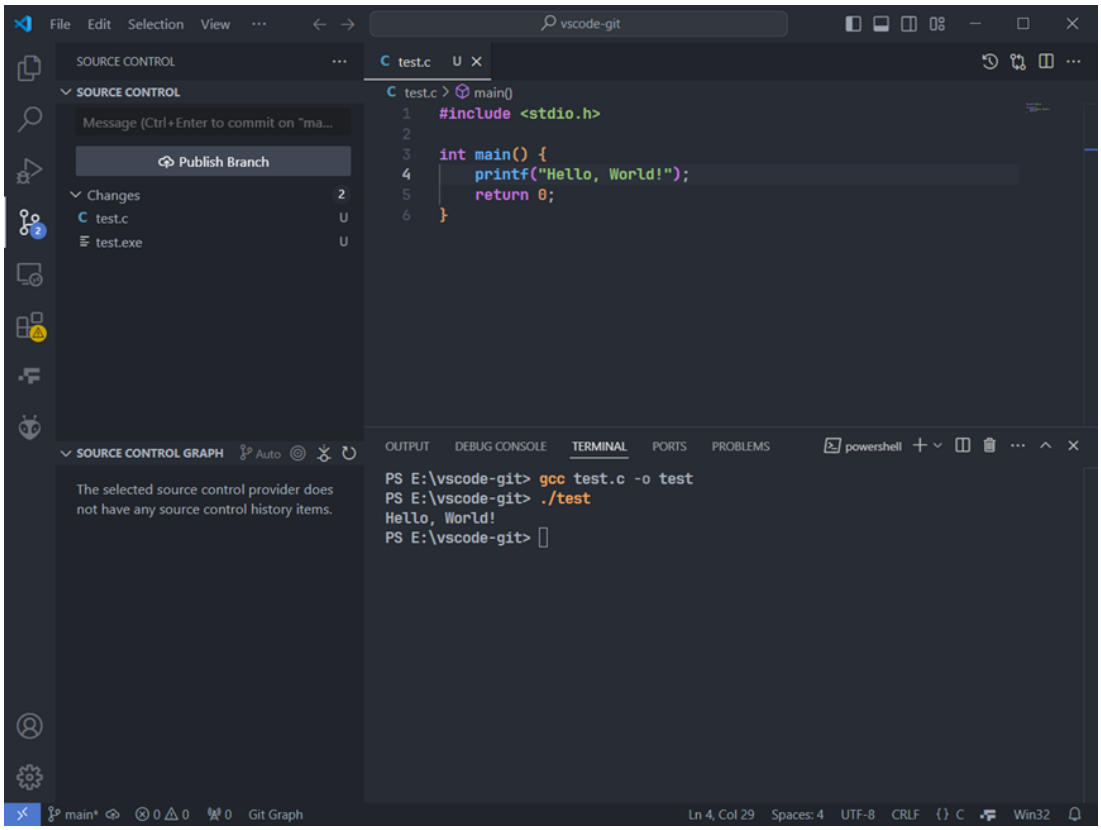


B. 这个工作区还没有仓库，我们需要初始化创建一个仓库。我们选择红蓝圈所示的图标，并进行Initialize repository初始化仓库。

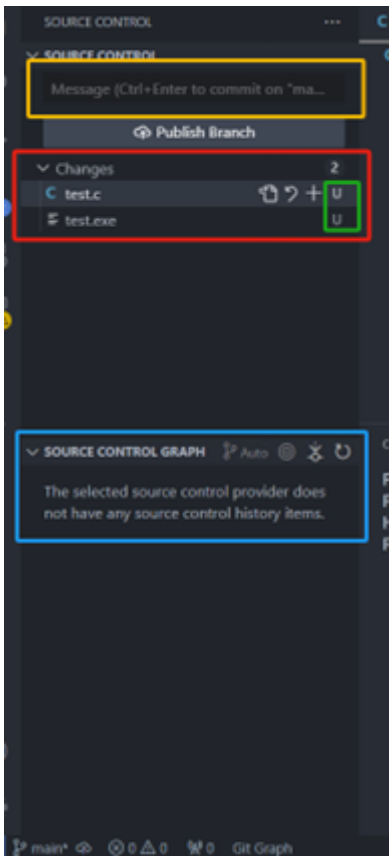


如果初始化成功，并且没有隐藏文件夹的话，可以在目录下看到.git文件夹。并且vscode画面如图所示





C. 简要介绍下这个UI：



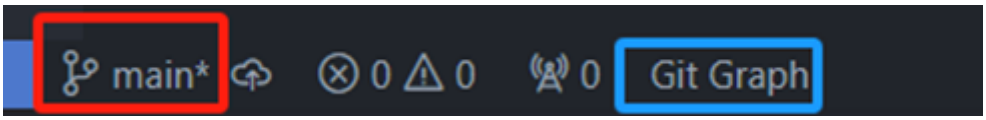
红色区域：是工作区对比版本库发生的修改。

绿色区域：文件的状态，其字母的表达意思为下图所示：

- M `modified`
  你已经在github中添加过该文件，然后你对这个文件进行了修改，就会文件后标记M
- U `untracked`
  你在本地新建了这个文件，还未提交到github上，就会标记U
- D `delete`
  你删除了这个文件，vscode-git会记录下这个状态
- 6,U
  表示有6个错误且untracked

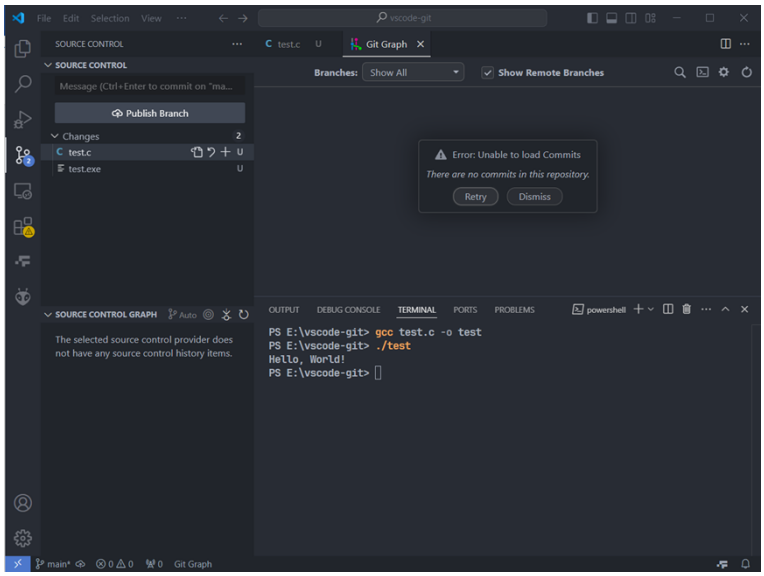
蓝色区域：是自带的一个版本分支图，不是很好看，我们忽略，替换成git graph。

黄色区域：当我们提交版本库的时候，需要打标签所用到的文本框。

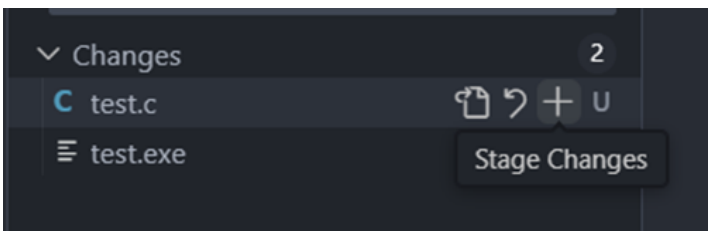


在最下栏，关心两个：红色是代表我们所在分支。蓝色，则是安装gitgraph插件后才会有的功能入口。

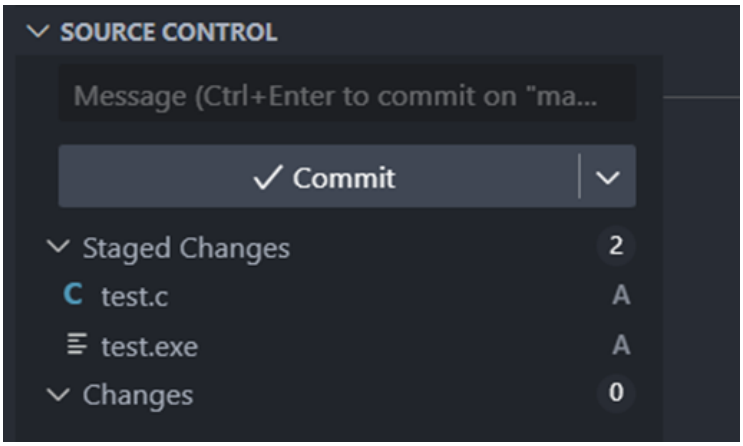
D. 点击打开Git Graph后，是这样一个界面。显示Error:Unable to load Commits。这里说明，我们还没有提交过一次代码，所以无法加载提交记录。



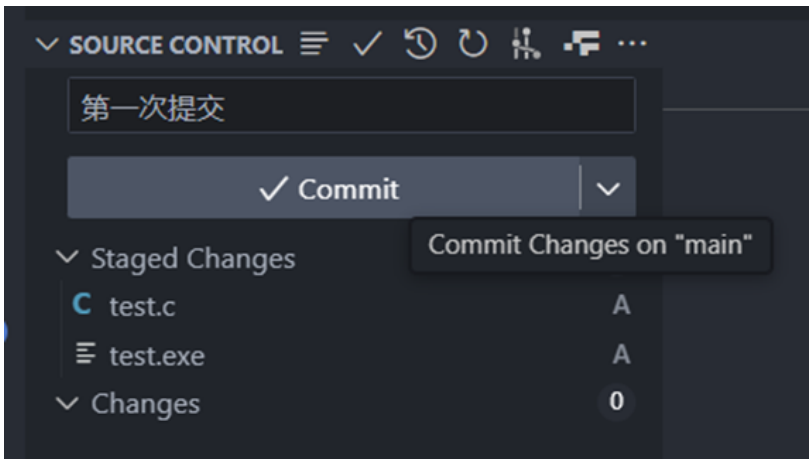
E. 我们来进行第一次提交，点击加号提交到暂存区。



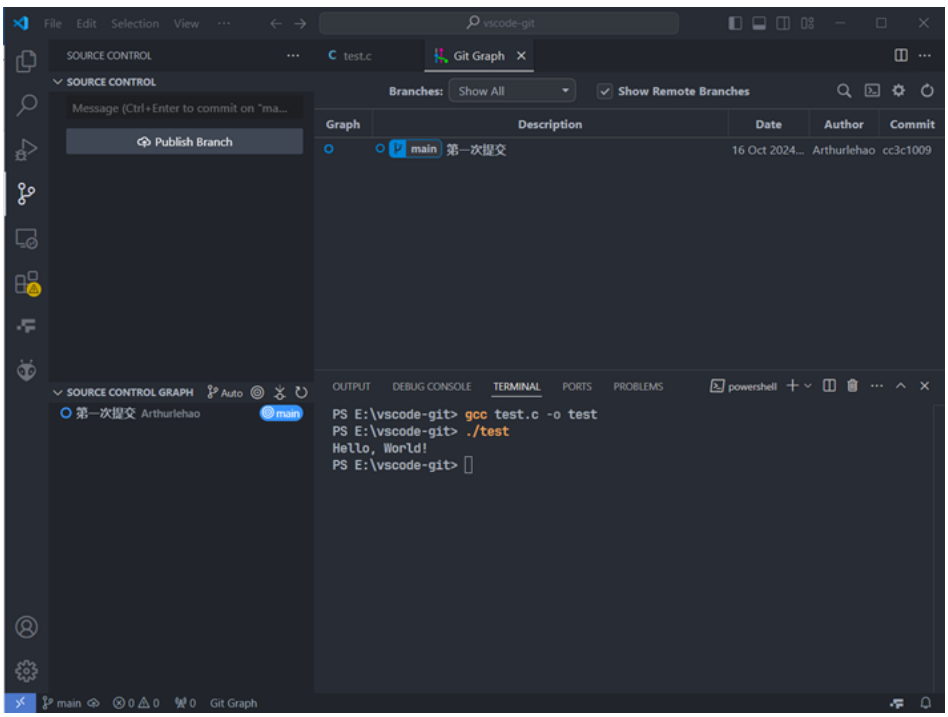
此时，所修改的文件就来到了暂存区。



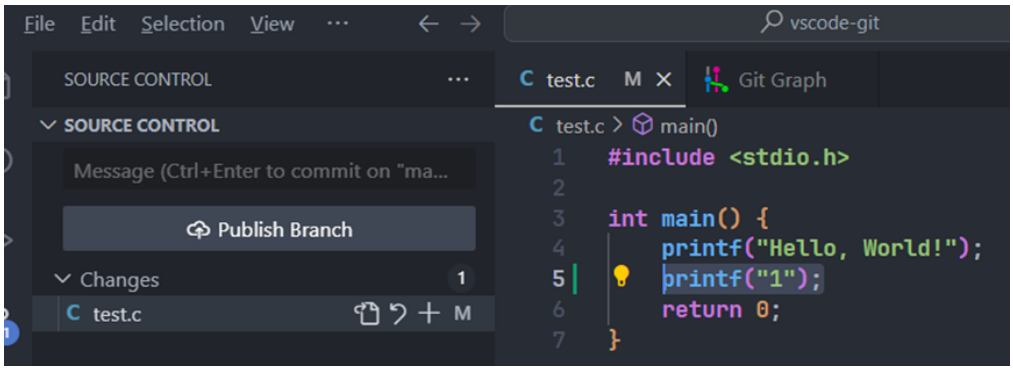
F. 这个时候我们进行，第一次版本库提交。注意一定要在Message写上说明文字。写上说明文字后，点击Commit，完成提交。



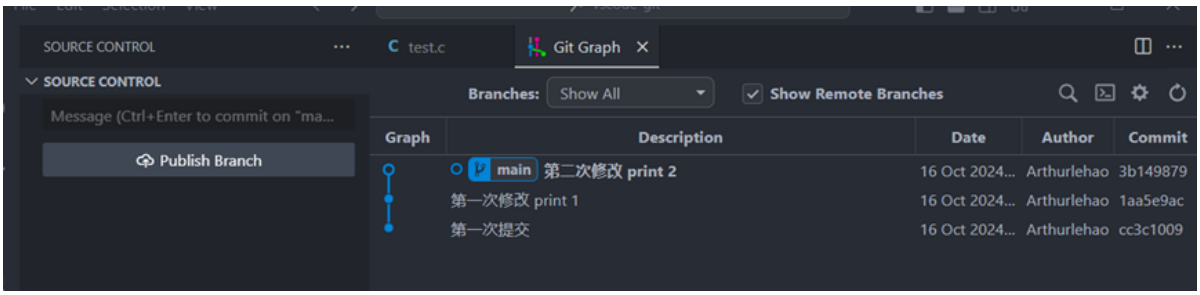
G. 这个时候，你就可以看到，git graph显示了你的提交记录。



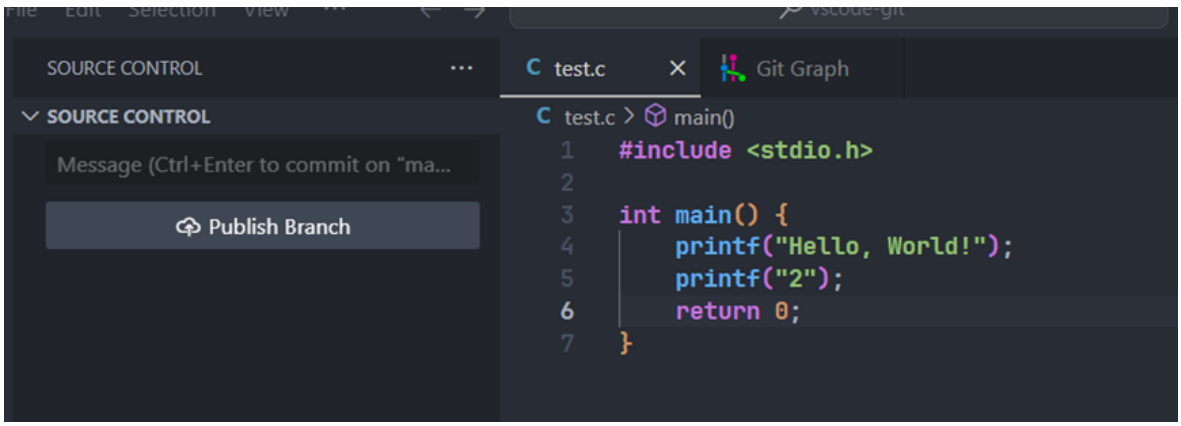
H. 接下来我们对代码进行修改，然后提交。



可以看到，我们修改代码以后，Changes区域也会发生变动。然后我们按刚才的步骤，修改提交两次代码。

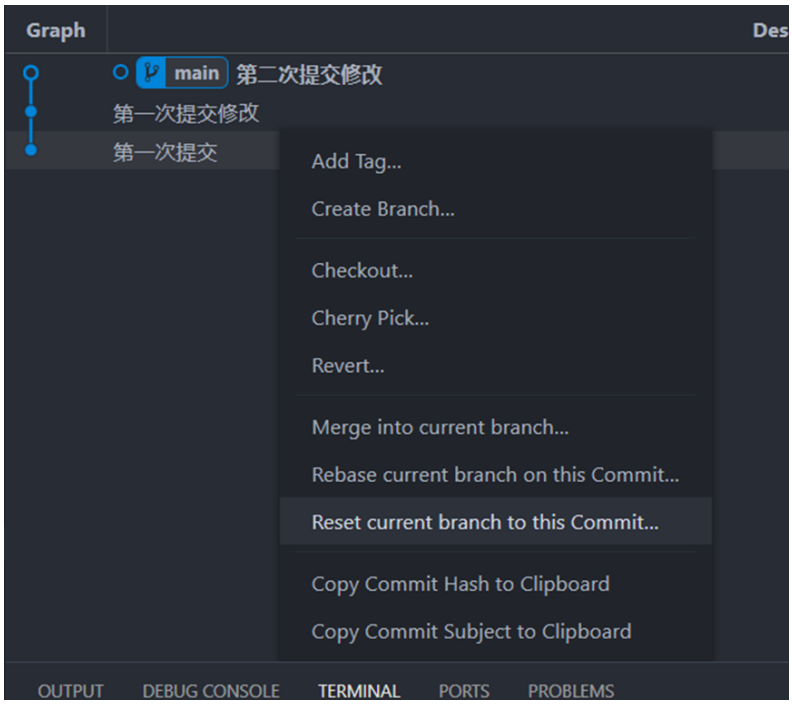


此时这个代码，长这个样子。

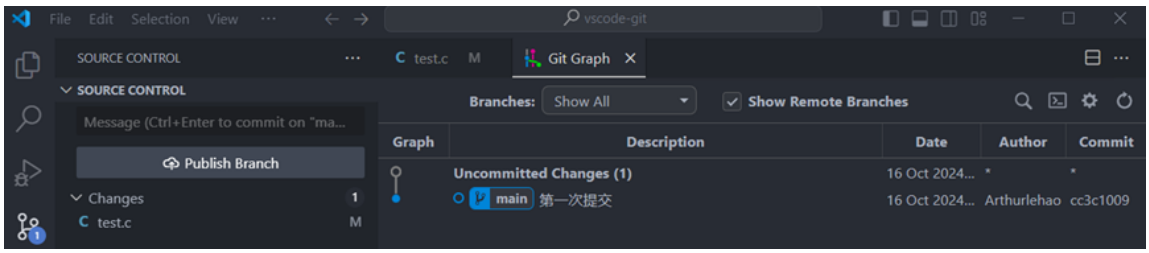


那我们假设，我们想要回到第一次提交的版本的话，那要怎么办呢？

I. 右键，选择 reset cureent branch to this Commit。这时候你就可以回到第一次提交的版本上。

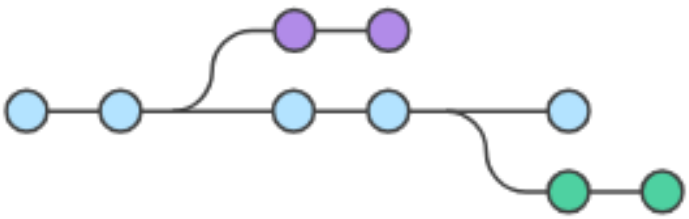


注意：因为这个版本保存是一个历史线性关系，可以看到，当我们回到第一次提交的版本后。第一次提交修改的记录也会随之消失。因此，除非你笃定这几个版本都不想要了，并且真的想要对前多个版本进行修改的话，建议进行版本分支，后续会讲到。



自此，你已经学会了简单的本地代码版本保存和版本回溯，给你的项目多加了一份保险。

分支管理

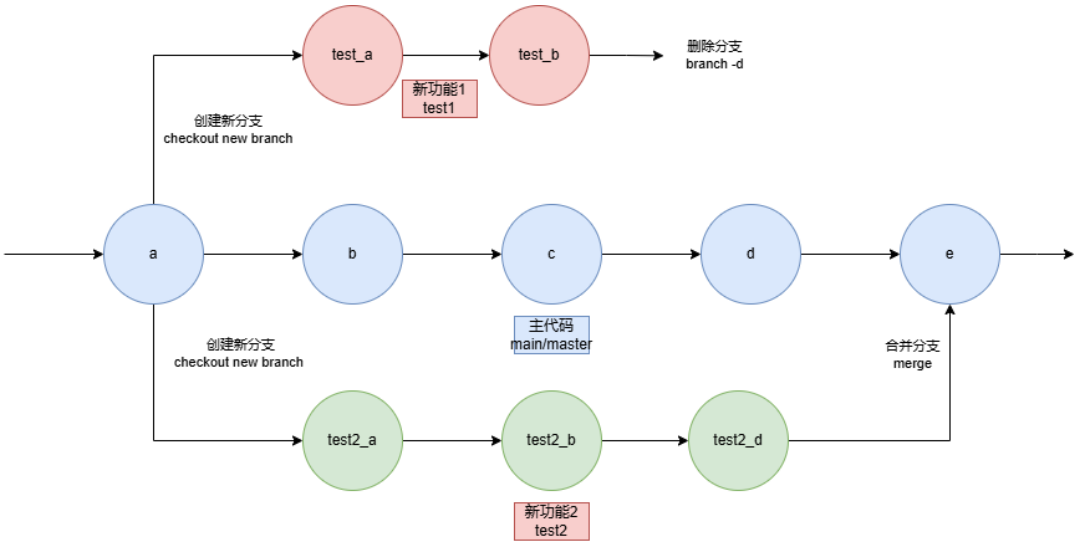


分支管理是git最强大的功能

当我们的项目需要开发新功能时，进行及时的版本推送固然好，但是其会影响我们主代码库。同时，我们发现代码模块化编写的习惯使得项目的各项功能都可以进行解耦。**这时候，我们就希望另起分支处理测试完新功能后，再将分支代码与主代码合并。**分支管理对于小项目而言可能无伤大雅，但是在企业级项目或多人代码维护的需求下，分支管理则是大放异彩。

能够理解git的分支管理模型，就理解了git最为核心的功能。

理论

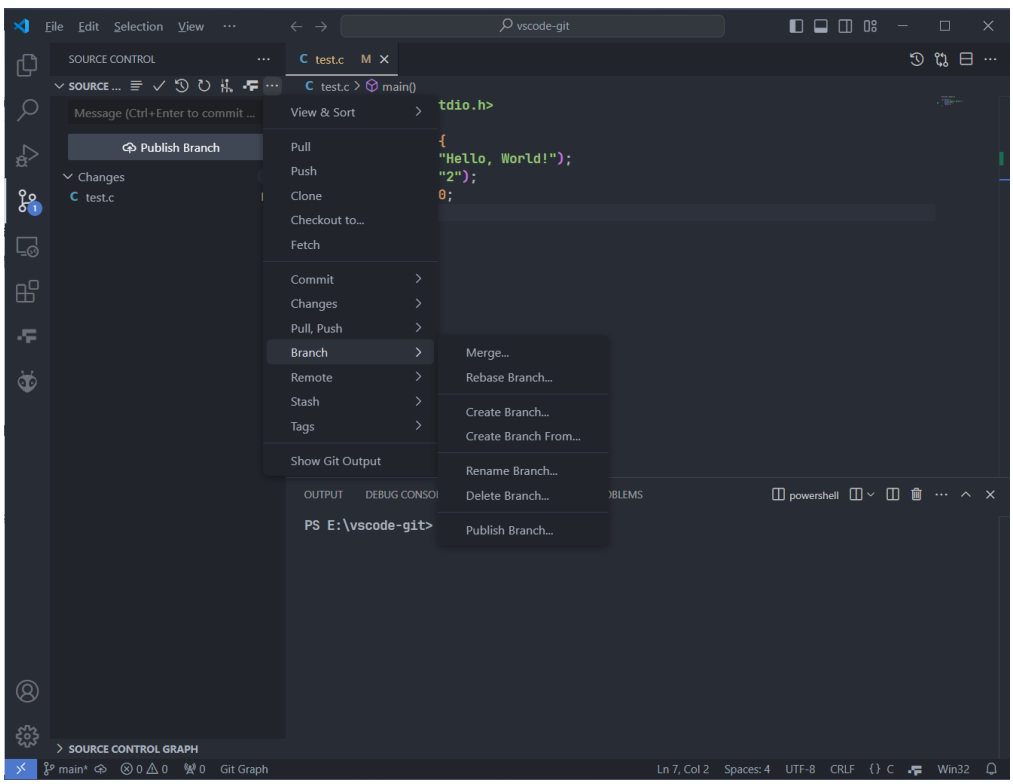


分支管理模型的主要操作有三：创建分支、删除分支、合并分支。这里的分支可以理解为某一版本代码。

需要分支的时候进行创建，不需要某一分支的时候就进行删除。合并分支有两种情况：1. 新的功能代码完成，合并至主代码。2. 远程仓库代码同步合并。远程代码同步合并这部分放到后续的**远程仓库的拉取和推送进行说明。**

实践

A. 点开vscode的soure control的more action。可以看到关于分支branch的全部操作。



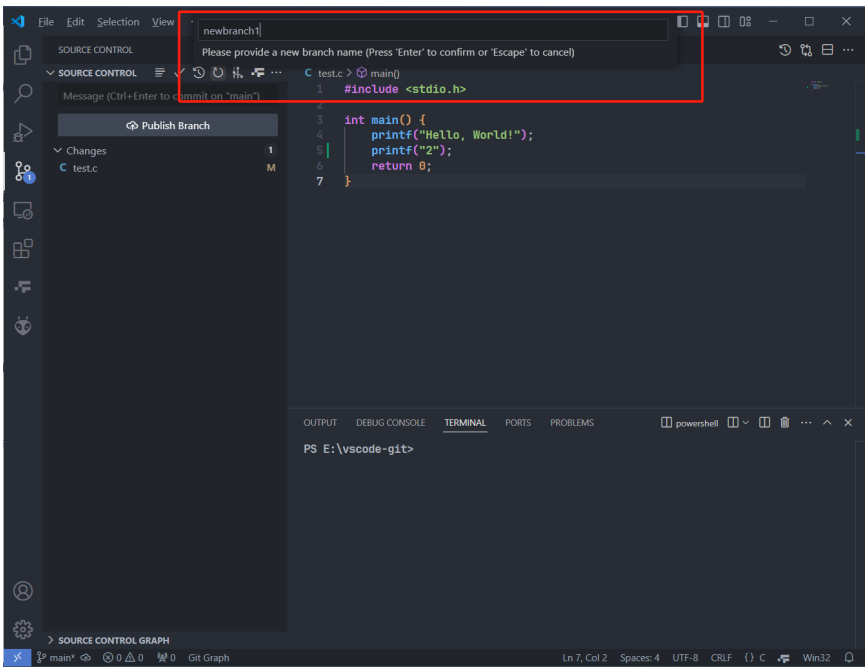
> Megre：合并分支
>
> Rebase Branch：跳转至某分支

> Create Branch：创建新分支
>
> Create Branch from Branch：创建一个来自某一分支的新分支
>
> Rename Branch：重命名这个分支
>
> Delete Branch：删除这个分支
>
> Publish Branch：发布这个分支到远程仓库

B. 我们现在来进行创建一个新的分支。

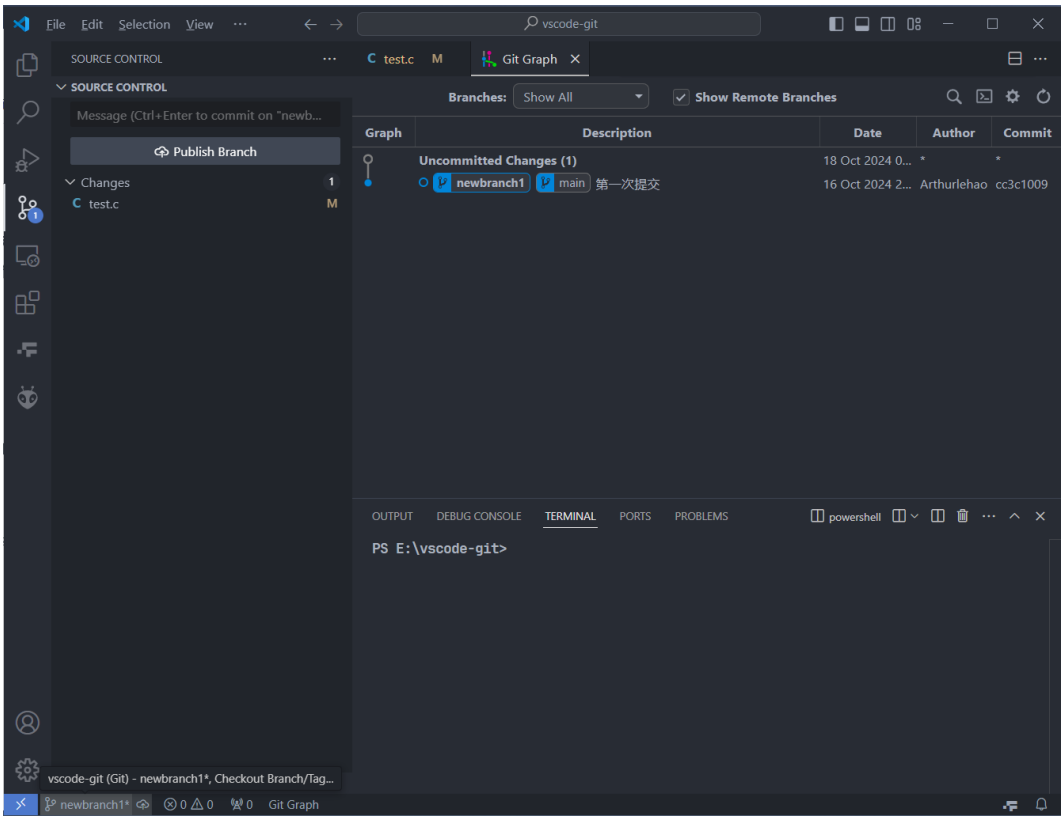
可以看到，有Create Branch 和Create Branch From两个选项。这两个选项的差别就是，Create Branch From会多一个让你选择分支的操作，而Create Branch则是直接在当前分支下进行创建新的分支。在创建新分支时会要求用户给新分支进行命名。

选择Create Branch进行创建新分支，并进行命名。



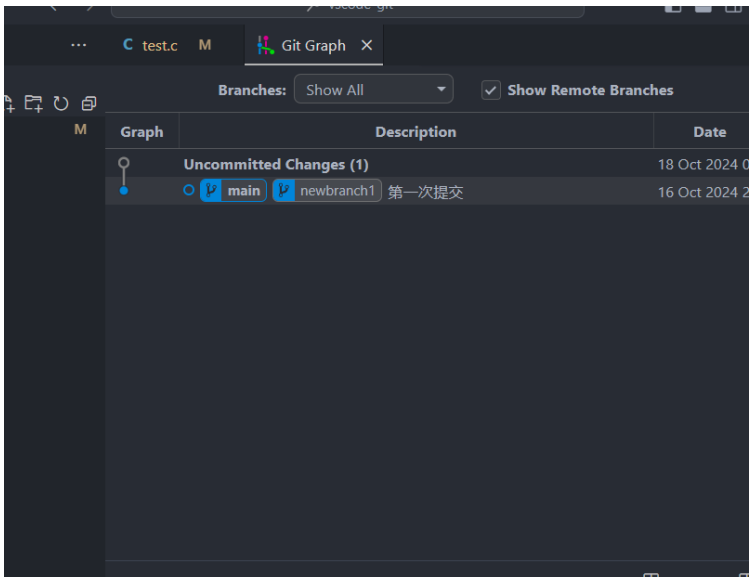
创建新分支并命名

通过git graph 可以看到我们已经创建了新的分支。



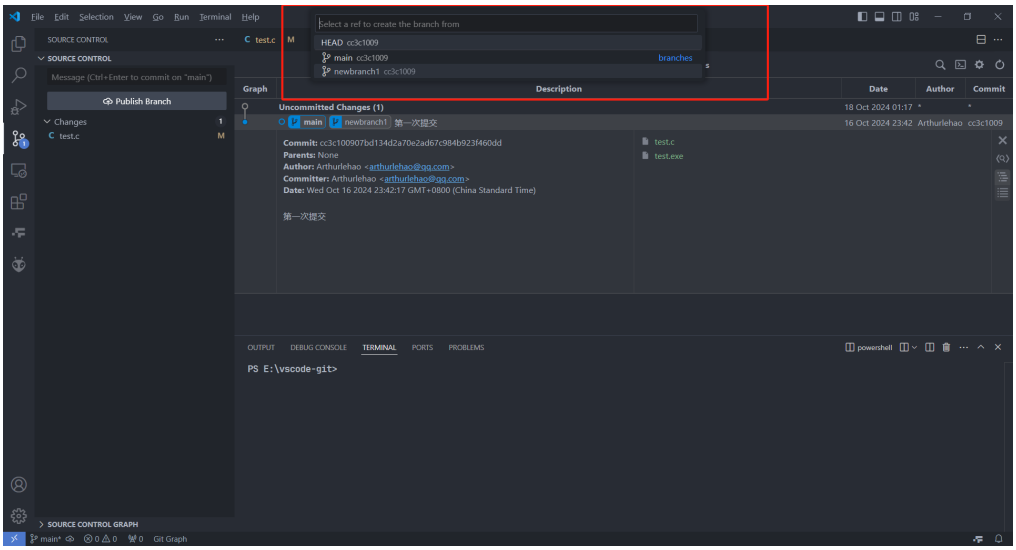
注意看git graph两个分支的字体粗细。字体粗的表示的是，我们所在的分支。而上面的Uncommited Changes则说明，工作区的文件与此分支的提交有区别。一种即在又不完全在的意思。

此时双击main，我们就可以跳转到main分支上了。

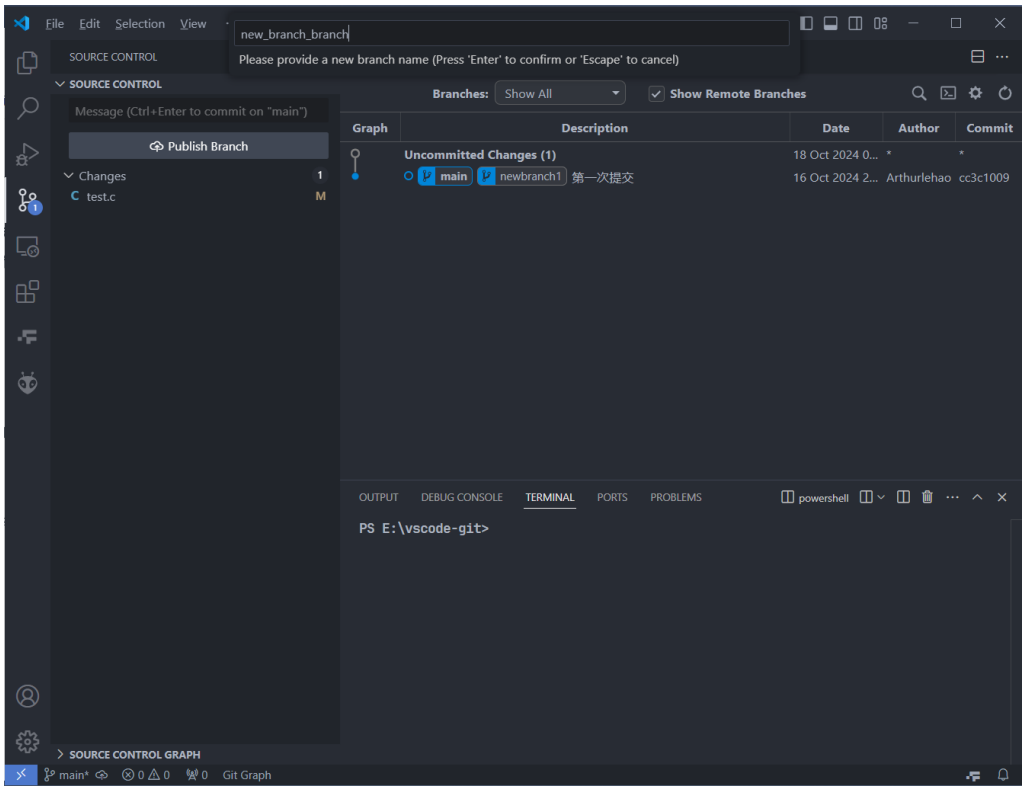


双击跳转到main分支

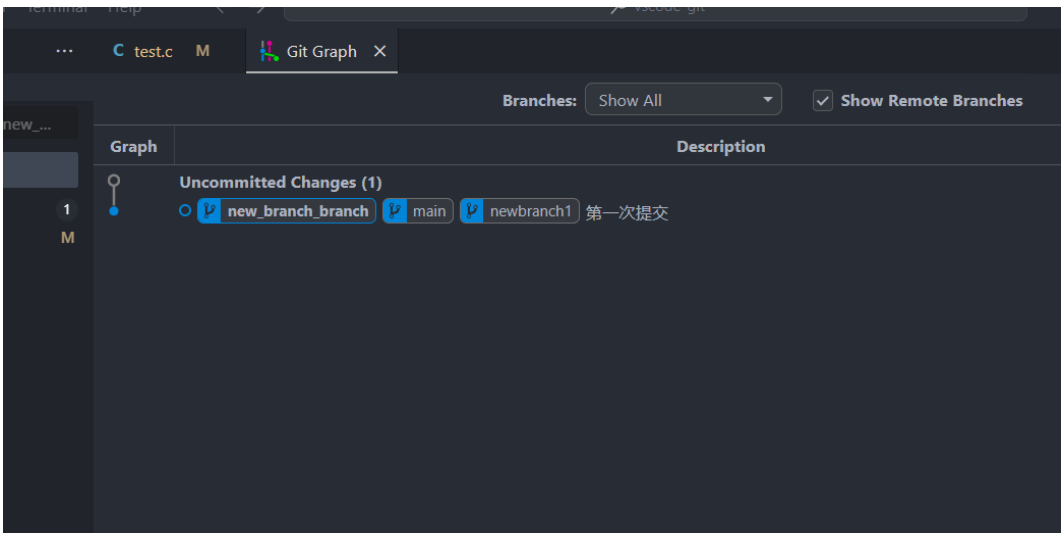
然后我们使用create branch from branch，在newbranch1的基础上再创建new_branch_branch。这个时候就会要求你选择分支，我们选择newbranch1后，同样的，进行命名为new_branch_branch。


Select a ref to create the branch from
HEAD cc3c1009
main cc3c1009
newbranch1 cc3c1009
branches
Graph
Description
Date
Author
Commit
Uncommitted Changes (1)
main
newbranch1
第一次提交
Commit: cc3c100907bd134d2a70e2ad67c984b923f460dd
Parents: None
Author: Arthurlehao <arthurlehao@qq.com>
Committer: Arthurlehao <arthurlehao@qq.com>
Date: Wed Oct 16 2024 23:42:17 GMT+0800 (China Standard Time)
第一次提交
test.c
test.exe
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PROBLEMS
powershell
PS E:\vscode-git>
SOURCE CONTROL GRAPH

选择分支

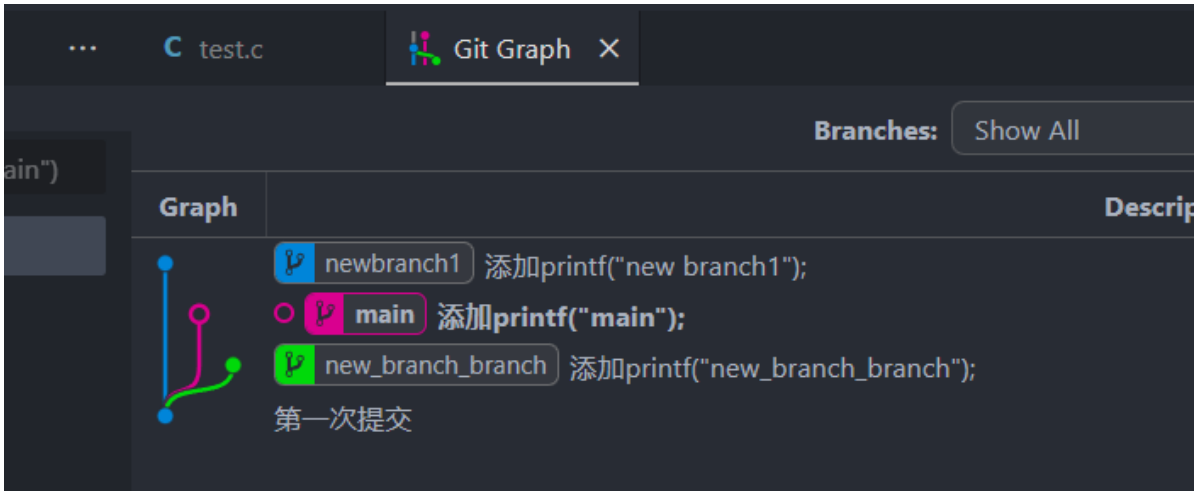

new_branch_branch
Please provide a new branch name (Press 'Enter' to confirm or 'Escape' to cancel)
SOURCE CONTROL
Message (Ctrl+Enter to commit on "main")
Publish Branch
Changes
test.c
Branches:
Show All
Show Remote Branches
Graph
Description
Date
Author
Commit
Uncommitted Changes (1)
main
newbranch1
第一次提交
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PROBLEMS
powershell
PS E:\vscode-git>
SOURCE CONTROL GRAPH
Git Graph

命名


test.c
M
Git Graph
Branches:
Show All
Show Remote Branches
Graph
Description
Uncommitted Changes (1)
new_branch_branch
main
newbranch1
第一次提交

完成创建

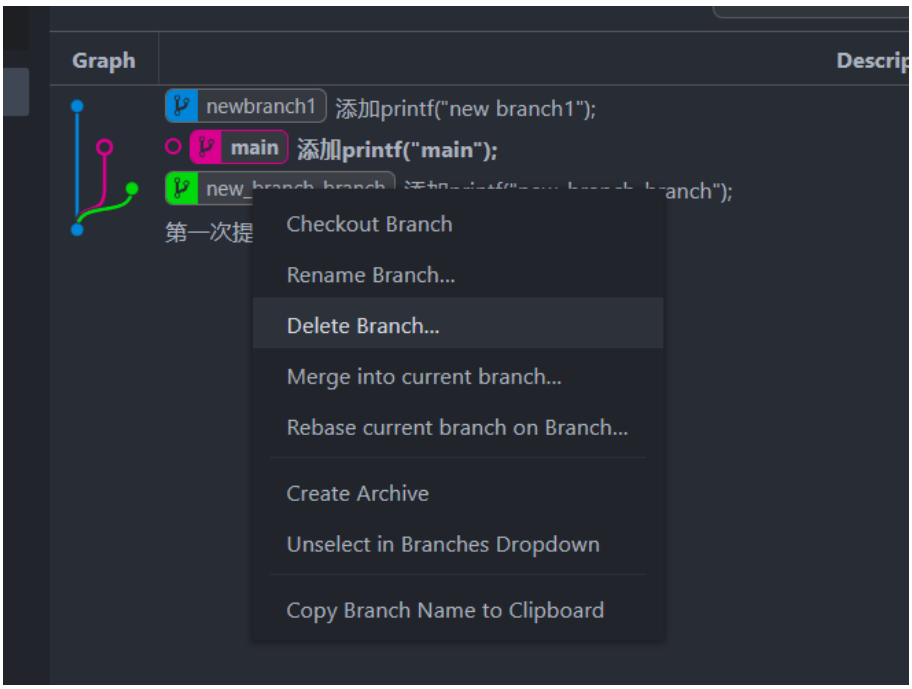
C. 我们对新分支进行差异化修改，并进行代码提交。



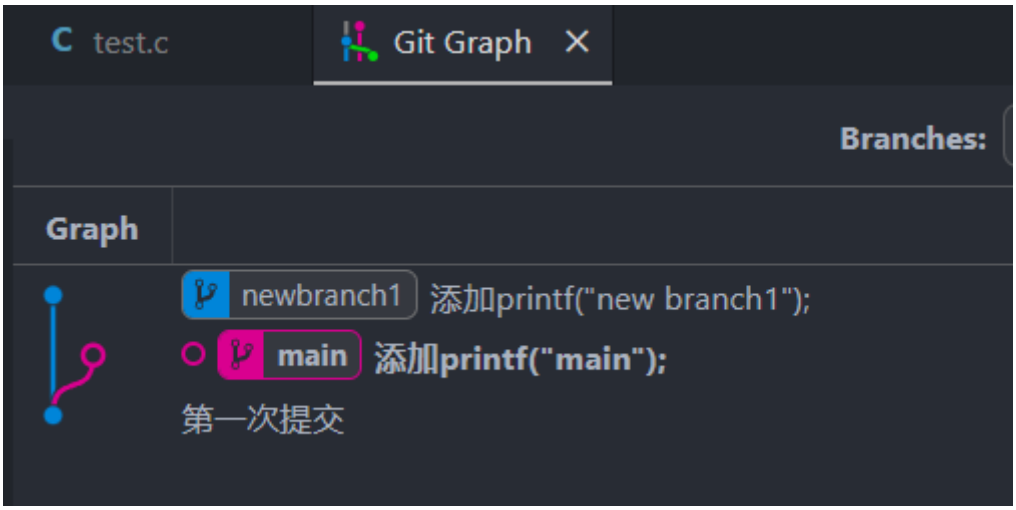
差异化修改后的git graph变化

这个时候可以看到git graph所示，我们看字体粗细可以看出我们目前所在main分支上。并且图形化的分支能清楚的表达出分支模型。

D. 这时候我们来删除分支。例如觉得new_branch_branch没有用，我们直接右键选择delete，就能删除。

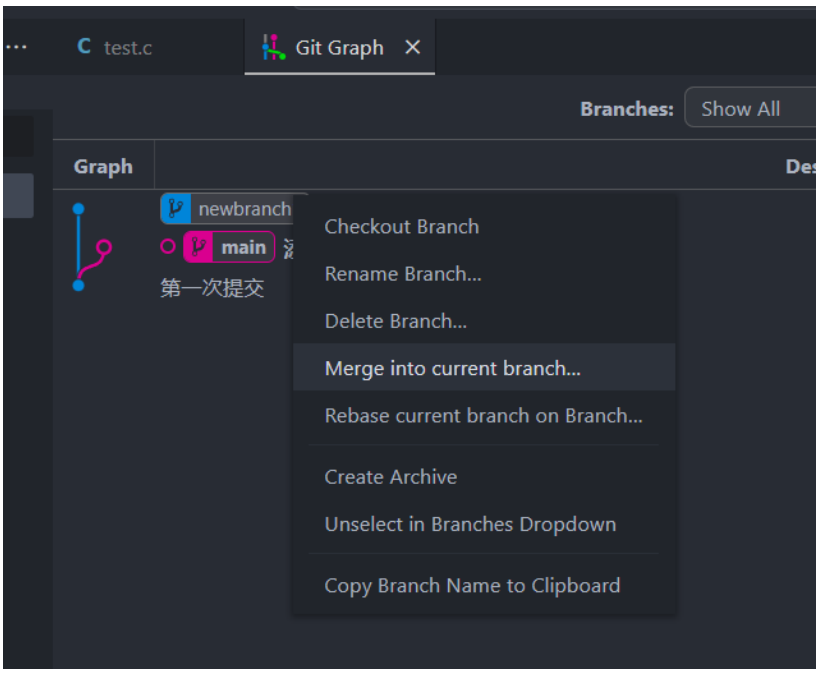


选择Delete Branch进行删除

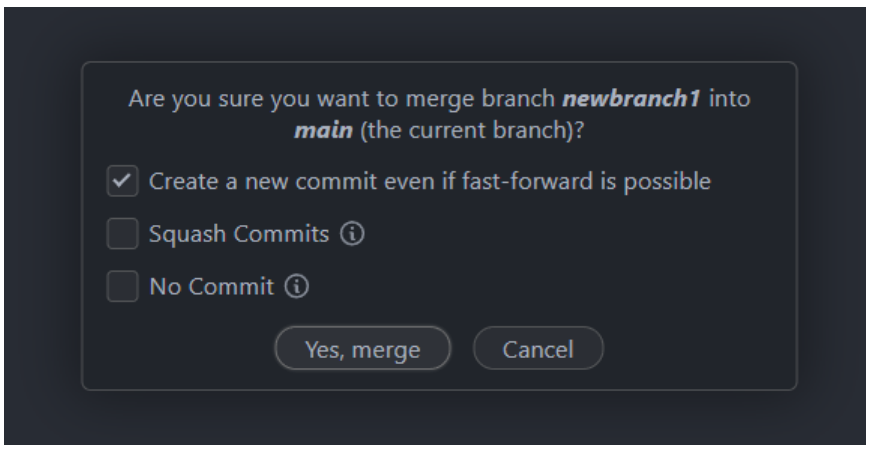


删除过后的分支样子

E. 然后我们来将分支合并。将分支newbranch1合并到main当中去。

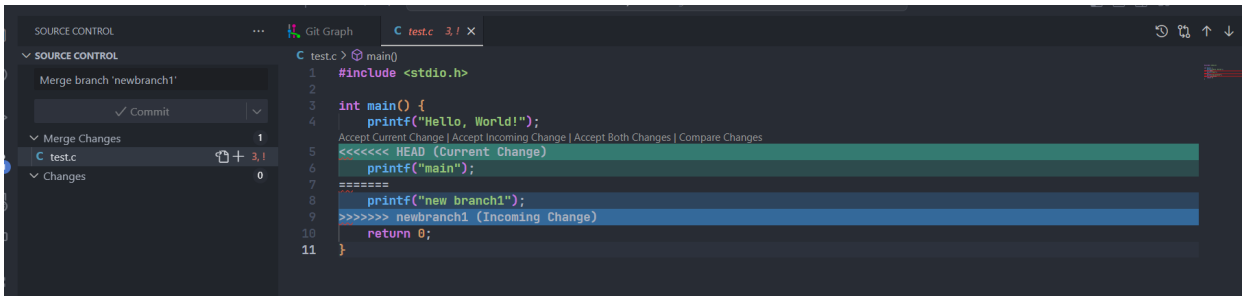


选择Merge into curent branch

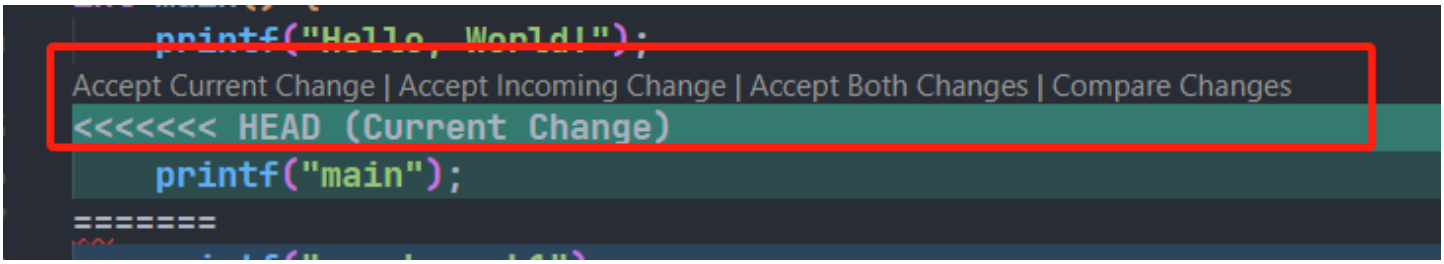


勾选yes，merge

此时，部分文件会出现！,并出现Merge Changes。这时候，说我们需要进行手动的**合并冲突处理**。

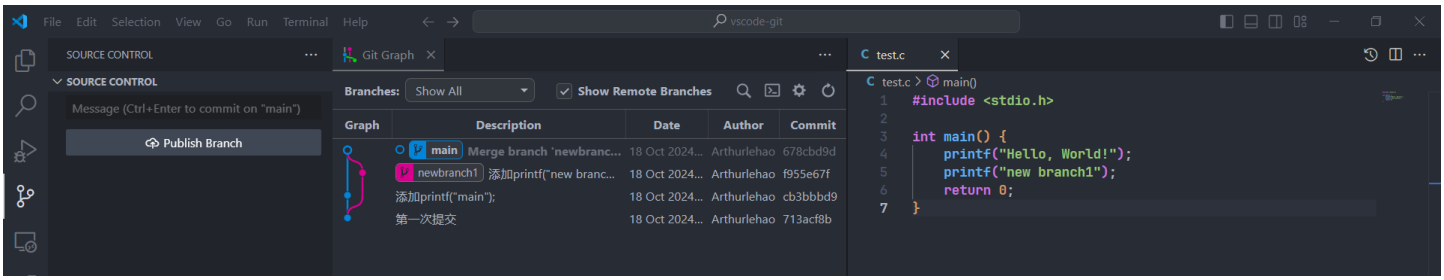


合并冲突处理，是分支合并、远程仓库拉取推送，必不可少的操作。意在让用户来选择，合并中哪些是需要保留，哪些是需要替换的内容，还是我全都要。



上面会出现小字选择:保留本分支的内容，替换成被合并分支的内容，我全都要，最后是比对全部变化。

在这里我们选择，Accept Incoming Change，就可以看到原本main中的print("main");被替换成了print("new branch1");。然后进行提交。



这个时候，我们可以看到newbranch1分支已经被合并到了main里去，并根据我们的合并冲突处理方式，将原本main中的内容替换成newbranch1的内容。newbranch1并没有消失，我们依旧可以选择进入到newbranch1中进行修改，这里就不再进行拓展了。

自此，你已经学会了使用vscode来进行git的分支管理功能。

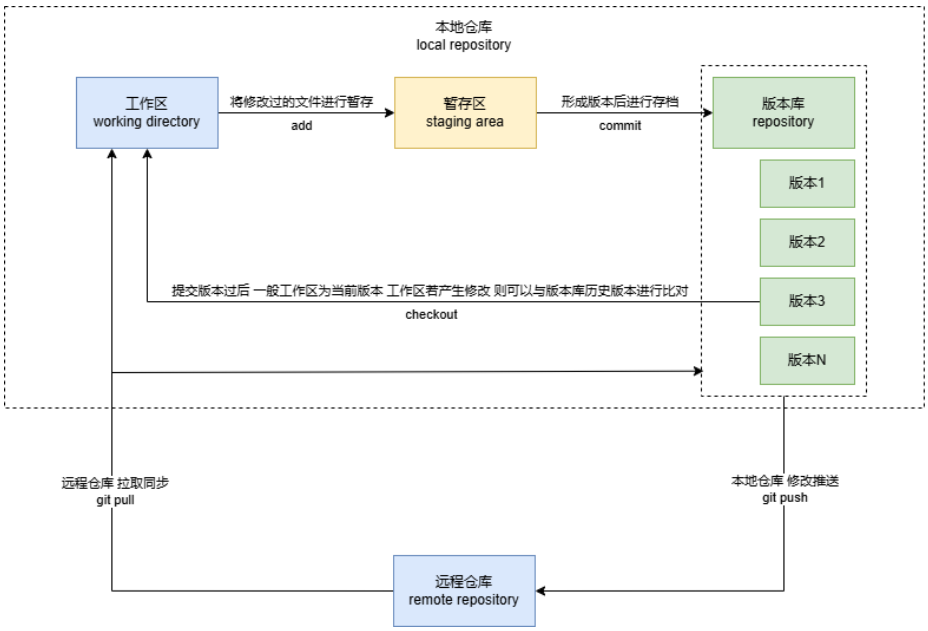
远程仓库的拉取和推送

现在，你已经非常满意你的项目工程并且想要进行开源，又或者害怕项目因为本地电脑损坏发生代码丢失，又或者你想要**成为祖传的祖**。那么一个git托管平台就可以满足你的需求。接下来将会讲解如何使用vscode+git进行远程仓库的拉取和推送。

为了代码的安全，git设计了相对复杂的系统，导致这一部分的逻辑会较为抽象，初学入门时会时常出现远程仓库拉取或推送发生失败的情况。请不要慌张，理清楚代码版本之间的关系即可。**只要不进行代码回退的操作，一般情况下，远程仓库的代码也会与本地仓库一样，有着可以版本回溯的能力，所以请大胆一些。**

理论

对于远程仓库，我们只做两件事：**拉取和推送。拉取远程仓库进行同步，修改代码提交完后进行推送**。这个远程仓库可以是github gitee这样的git代码托管平台，也可以是自建的git托管服务器。



本地与远程仓库的交互

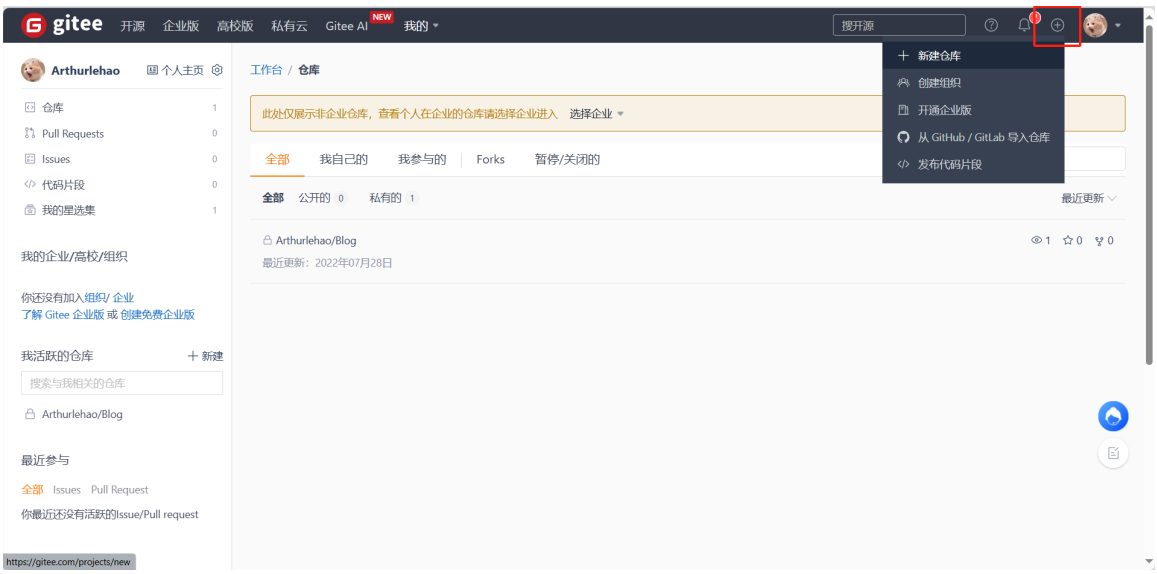
实践

创建远程仓库，拉取同步，修改代码，提交至本地仓库，进行推送。

> 为了初学者在使用github时不遇到因科学上网导致的问题，我们这里选用gitee平台为例子。值得一提的是，相对于gitee，github在**平台监管层面**上会弱许多。

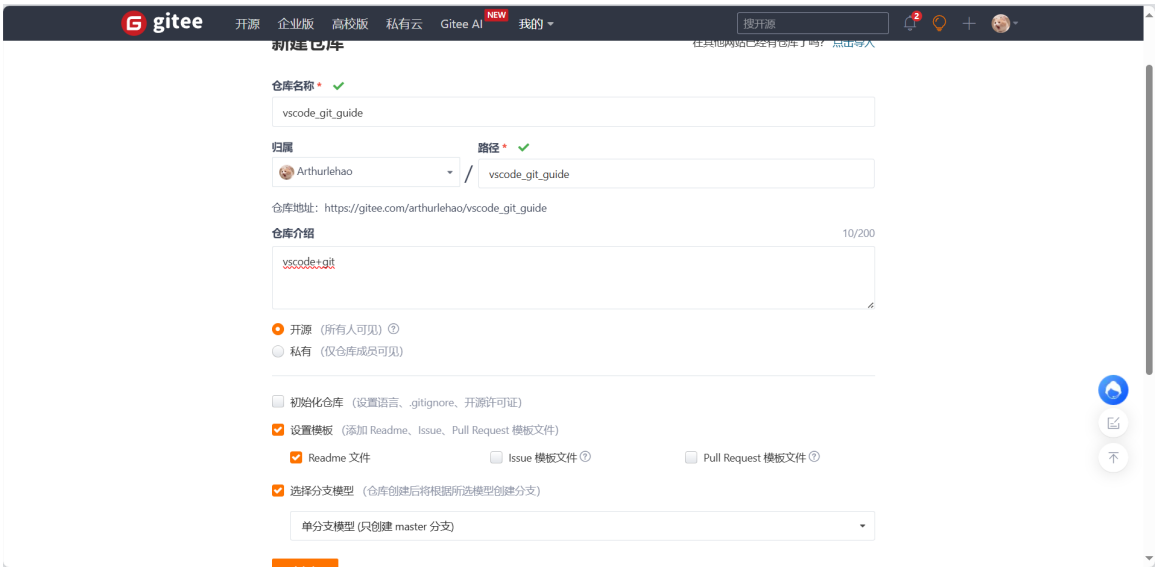
以gitee平台为例

A. 我们先来创建一个远程仓库，注册登入gitee，然后新建个远程仓库。



找到新建仓库的按钮

我们为这个远程仓库进行一个命名，添加介绍，并选择是开源还是私有。然后设置模板添加readme.md 文件，选择分支模型（这里我们选择单分支模型），然后点击新建仓库。



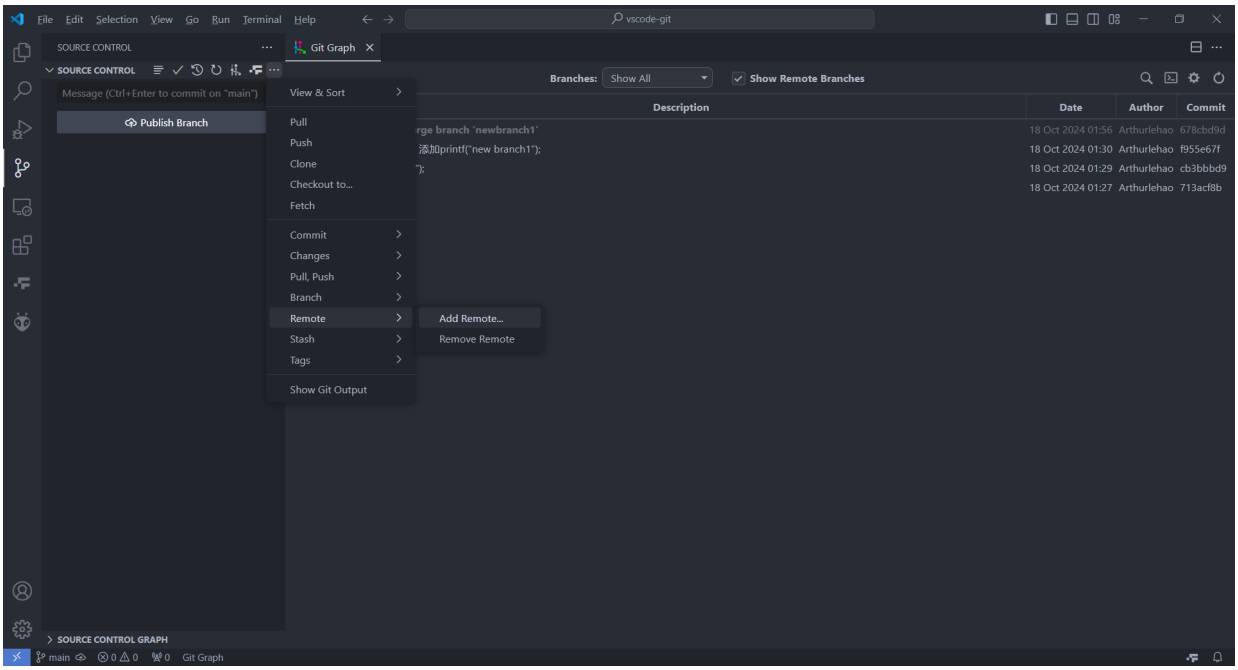
新建仓库配置

这样，我们一个新的远程仓库就建立好了。

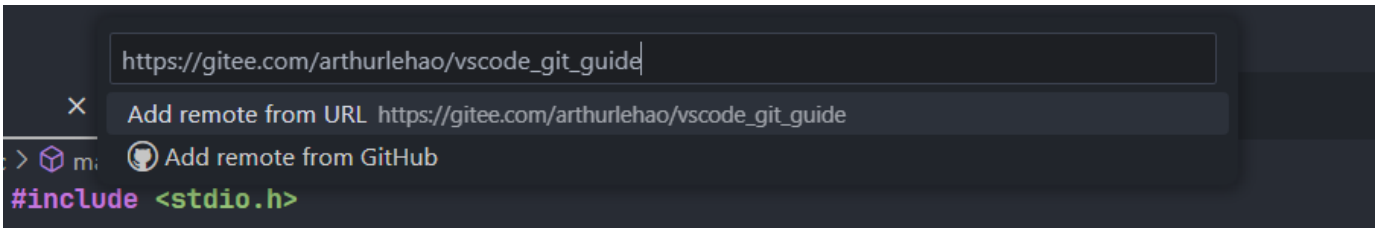
这里没截图，所以用了推送过后的仓库界面，但只要你有这个界面说明就创建好了

B. 完成创建以后，我们来到VScode，添加远程仓库，并进行拉取同步和推送。

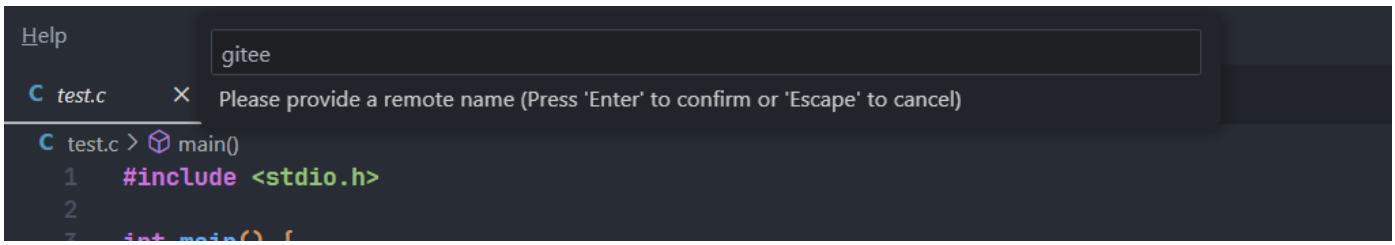
在Remote选项中，选择Add Remote Repository。然后复制黏贴刚才创建的仓库网址，并进行备注命名。



在Remote选项中，选择Add Remote Repository



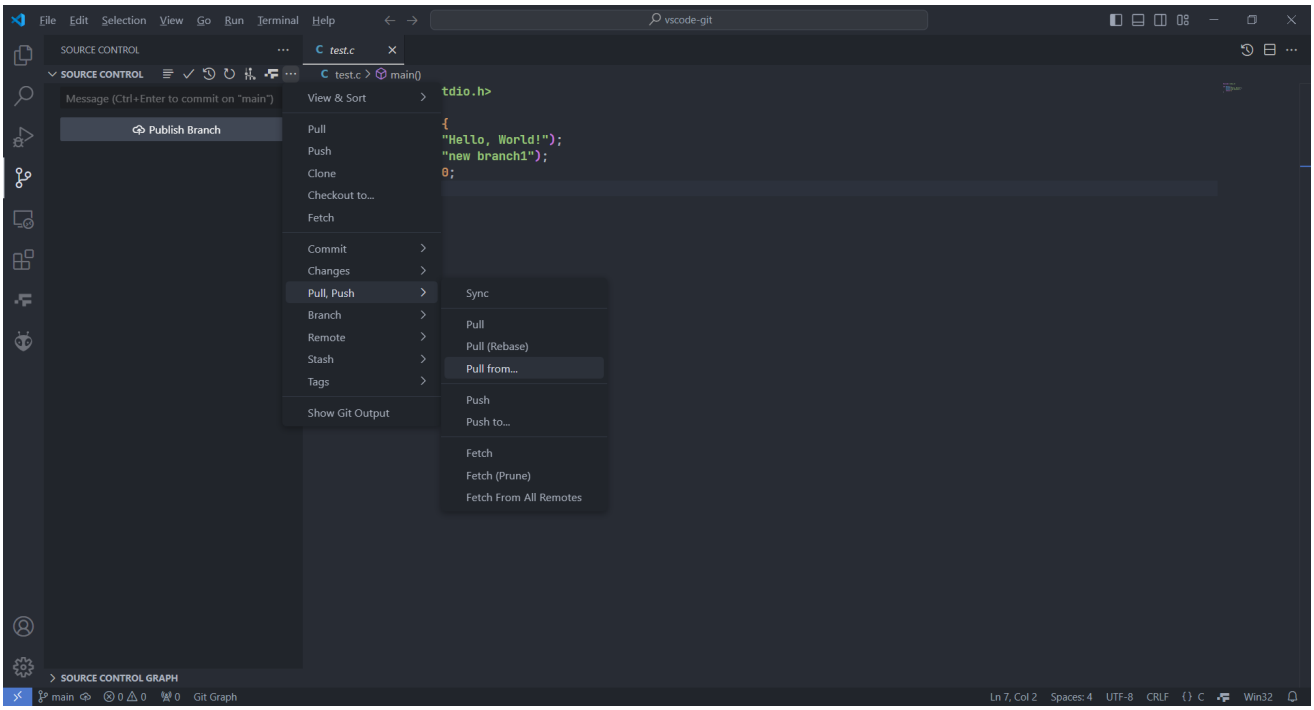
复制黏贴刚才创建的仓库网址



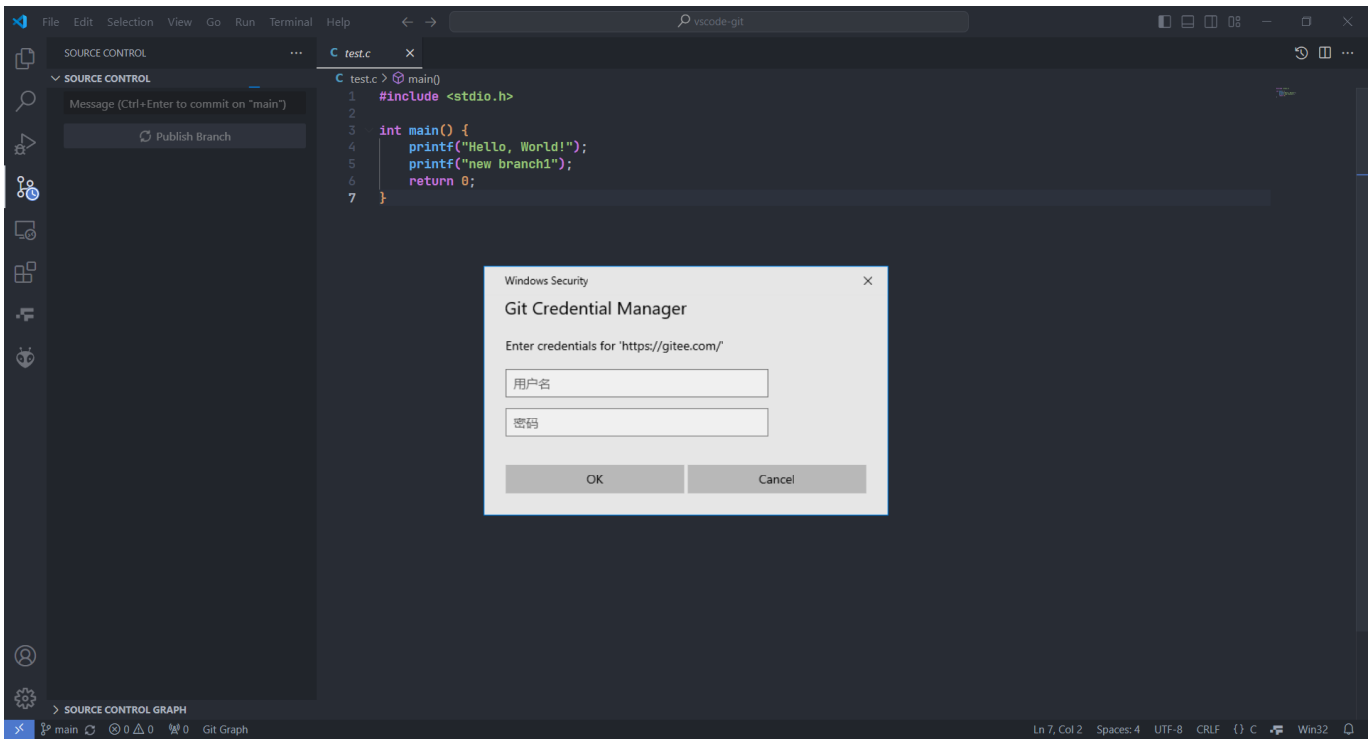
进行备注命名

添加完成以后，我们对远程仓库进行拉取同步。选择Pull,Push中的Pull from。然后选择我们刚刚添加的远程仓库。这时候会出现一个登录页面，我们使用gitee平台账号登陆即可。

此时，你可能会显示报错，不用慌张。这是因为远程仓库里并没有你本地仓库的main分支，所以会产生同步失败的报错。如果远程仓库有相同的分支，则需要进行合并冲突处理的操作进行同步。



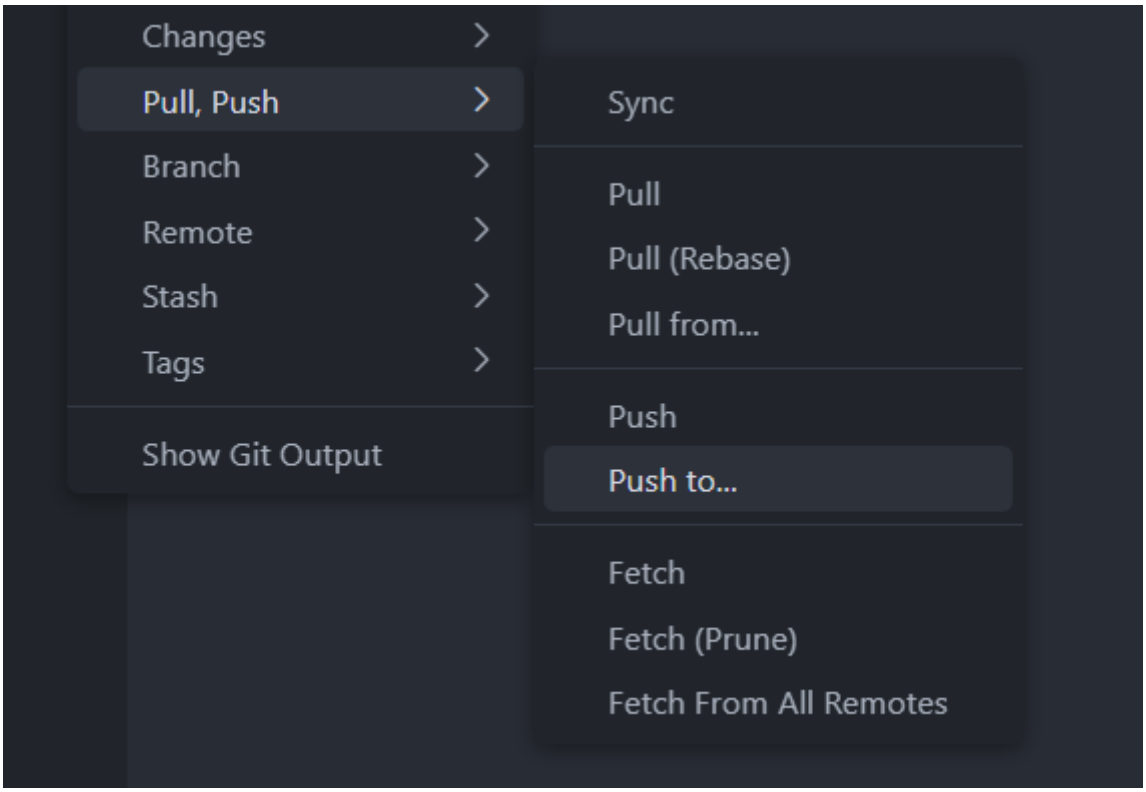
选择Pull,Push中的Pull from



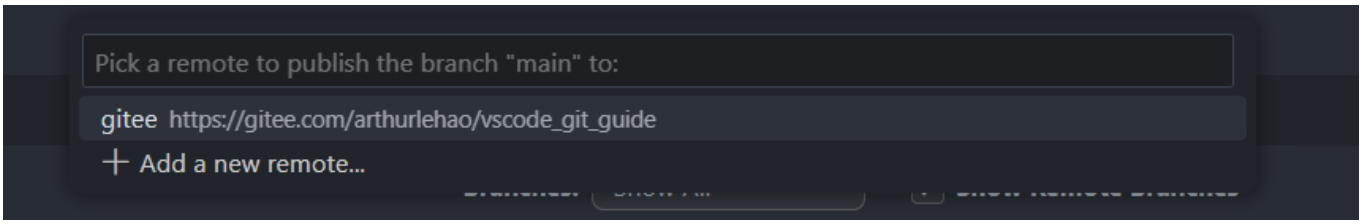
gitee平台登入

C．同步完成以后，我们将当前的仓库进行推送。

接下来我们进行仓库推送，选择Pull,Push中的push to remote repository 将main分支推送至gitee远程仓库去。

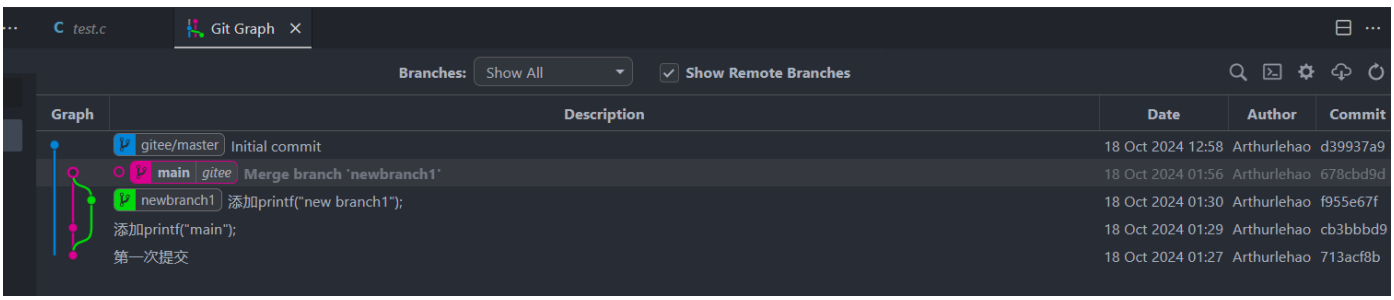


选择Pull,Push中的push to remote repository



选择gitee远程仓库

打开git graph，这个时候你就可以看到远程仓库的master分支还有main的命名旁出现了远程仓库的备注名，就说明拉取同步成功，推送成功。

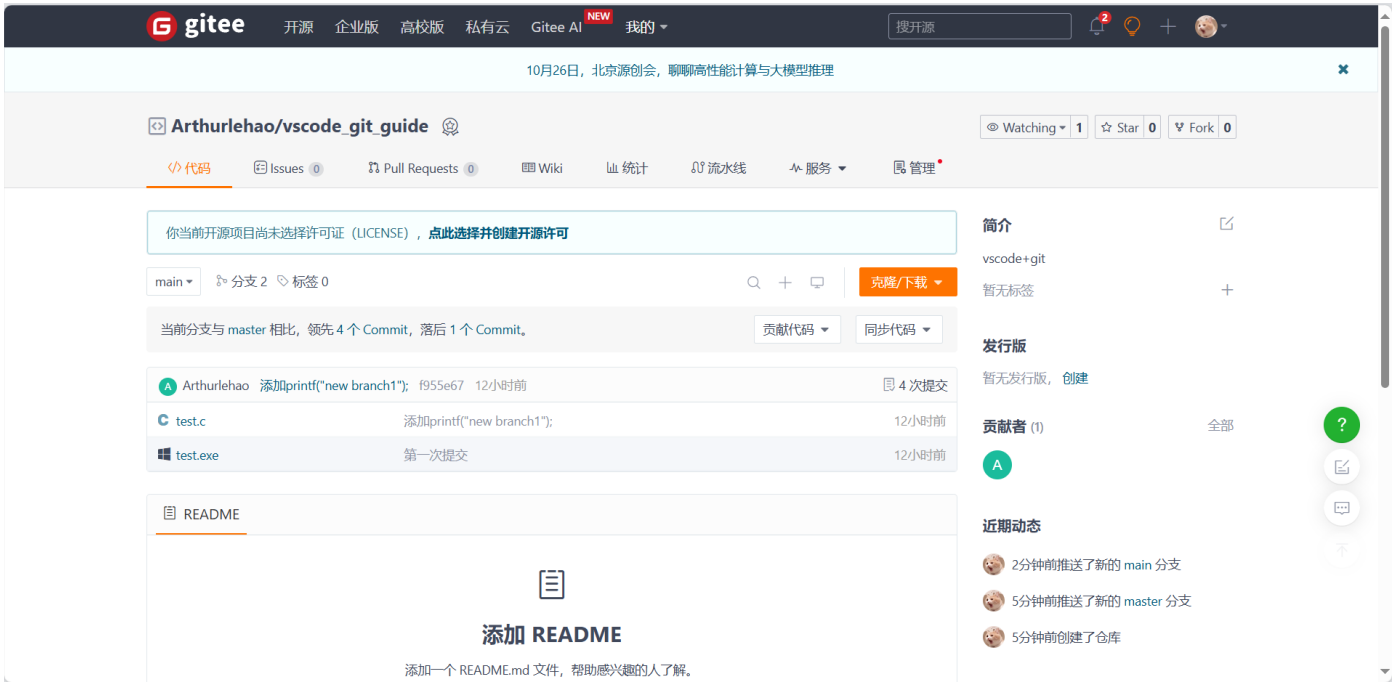


拉取推送远程仓库后的git graph界面

此时打开gitee远程仓库的网页，你可能会发现并没有什么变化，甚至没有看见本地仓库的代码。这是因为你正在查看master分支。我们下拉分支菜单，就可以看到本地的main分支了。这个时候点进去，就可以看到main分支上我们所推送的全部内容。

可能会发现并没有什么变化，甚至没有看见本地仓库的代码

下拉分支菜单，就可以看到本地的main分支



main分支上我们所推送的全部内容

自此，你已经学会了使用vscode来进行git远程仓库的拉取和推送。

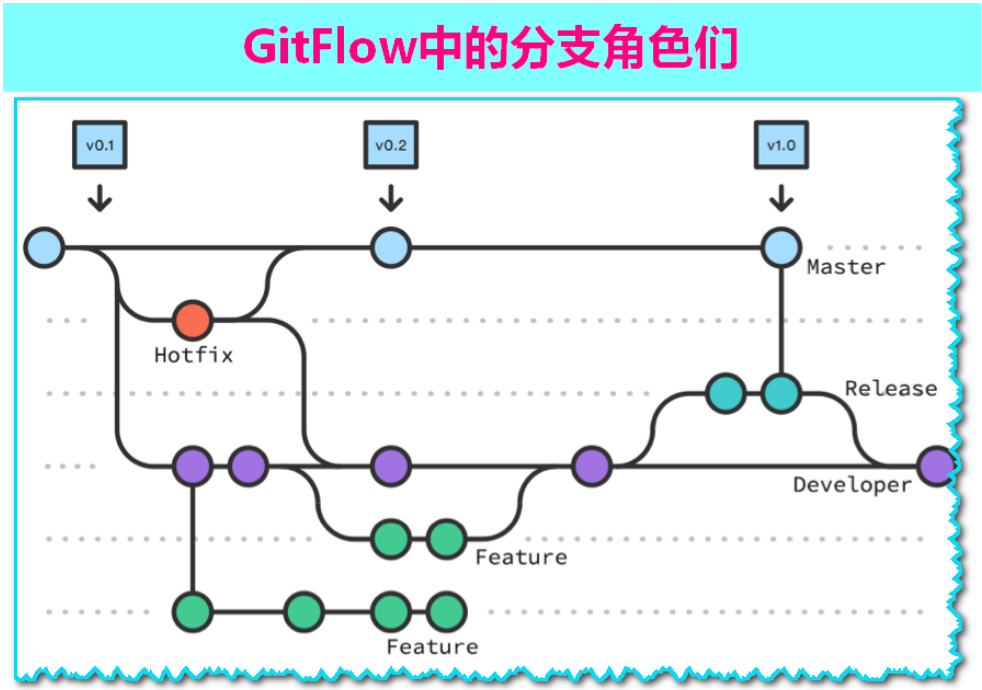
相信聪明的你，学会了向gitee推送拉取远程仓库后，你也一定会知识迁移，进行github的推送拉取流程。

恭喜你已经学会了vscode+git的操作，勤加练习使用，熟能定生巧。

vscode+git的操作并不止于此，在之后或之前你们就可能会了更加快捷方便的操作方式。包括使用vscode直接进行远程仓库的新建和代码上传，clone开源仓库，修改文件tag，commit信息等。让项目代码版本管理变得更加有序和优雅吧。

题外话：多人协作开发

使用git进行多人协作开发，是个非常不错的选择。git的**分支管理模型+远程仓库**，为多人协作开发提供了十分高效的git flow。每个开发者只需要开发好自己的代码部分，包括测试等内容后，再将健全的代码合并至主代码当中，形成新的版本。



参考：如何使用 Git 进行多人协作开发(全流程图解)_git多人协作开发流程-CSDN博客

题外话：仓库精简

在工程的编译当中，会在工作区当中出现编译过程中的链接文件等内容。在实际中，我们一般只对源码进行版本管理，以达到精简仓库的效果，这时候就需要git去忽略掉这些文件，不对这些内容进行跟踪。这时候，我们就需要在仓库所在的目录下**添加.gitignore文件**，并在其中编写我们所需忽略的文件后缀或文件目录即可。此时git，会自动根据.gitignore文件的内容进行文件忽略，以达成精简仓库的目的。

参考：.gitignore 文件——如何在 Git 中忽略文件和文件夹详细教程-CSDN博客

参考资料

vscode官方源代码管理doc：Introduction to Git in Visual Studio Code

git教程：Git 教程 | 菜鸟教程

git安装：Git 详细安装教程(详解 Git 安装过程的每一个步骤)_git安装-CSDN博客

VSCode安装：VSCode安装配置使用教程(最新版超详细保姆级含插件)一文就够了_vscode使用教程-CSDN博客

由Arthurlehao@qq.com进行编写并在飞书自动生成

2024年11月2日