## 1.git仓库的组成

## 分为本地仓库和远程仓库，本地仓库就是开发者电脑里面的仓库，远程仓库就是GitHub/gitee的仓库

### 开发者可以把本地仓库的内容推送到远程仓库，也可以从远程仓库拉取内容到本地

### 本地仓库由3部分组成（工作区，暂存区，本地仓库），远程仓库只有一个部分组成

|  |
| --- |
|  |

## 提交流程：工作区-》暂存区-》本地仓库-》远程仓库

|  |
| --- |
|  |

### 注意：可以有多个开发者连接到同一个远程仓库

# 2.使用

## 1）基本使用：

## 我们在工作区编写代码，写好后就可以将工作区的代码添加到暂存区，然后再从暂存区添加到本地仓库，这样我们的本地仓库就会有一个保持记录，后续如果代码有问题，就可以从保持记录中进行回退

## 2）git的下载安装

### 网址：[Git - Downloading Package (git-scm.com)](https://git-scm.com/download/win)

# 3.使用命令

## 1）初始化配置

### 设置账号信息

|  |
| --- |
| # 添加全局设置git config --global user.email "kennycai8888@outlook.com"git config --global user.name "kennycaiguo" |

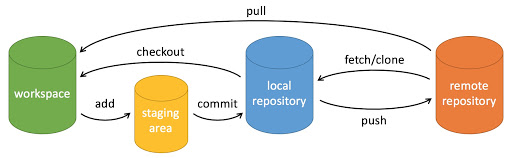
## 2）在GitHub上面新建一个叫做git-test的仓库，此时只有一个readme文件

|  |
| --- |
|  |

Git 的工作就是创建和保存你项目的快照及与之后的快照进行对比。

本章将对有关创建与提交你的项目快照的命令作介绍。

Git 常用的是以下 6 个命令：**git clone**、**git push**、**git add** 、**git commit**、**git checkout**、**git pull**，后面我们会详细介绍。



**说明：**

* workspace：工作区
* staging area：暂存区/缓存区
* local repository：版本库或本地仓库
* remote repository：远程仓库

一个简单的操作步骤：

$ git init

$ git add .

$ git commit

* git init - 初始化仓库。
* git add . - 添加文件到暂存区。
* git commit - 将暂存区内容添加到仓库中。

**创建仓库命令**

下表列出了 git 创建仓库的命令：

|  |  |
| --- | --- |
| **命令** | **说明** |
| [git init](https://www.runoob.com/git/git-init.html) | 初始化仓库 |
| [git clone](https://www.runoob.com/git/git-clone.html) | 拷贝一份远程仓库，也就是下载一个项目。 |

**提交与修改**

Git 的工作就是创建和保存你的项目的快照及与之后的快照进行对比。

下表列出了有关创建与提交你的项目的快照的命令：

|  |  |
| --- | --- |
| **命令** | **说明** |
| [git add](https://www.runoob.com/git/git-add.html) | 添加文件到暂存区 |
| [git status](https://www.runoob.com/git/git-status.html) | 查看仓库当前的状态，显示有变更的文件。 |
| [git diff](https://www.runoob.com/git/git-diff.html) | 比较文件的不同，即暂存区和工作区的差异。 |
| [git commit](https://www.runoob.com/git/git-commit.html) | 提交暂存区到本地仓库。 |
| [git reset](https://www.runoob.com/git/git-reset.html) | 回退版本。 |
| [git rm](https://www.runoob.com/git/git-rm.html) | 将文件从暂存区和工作区中删除。 |
| [git mv](https://www.runoob.com/git/git-mv.html) | 移动或重命名工作区文件。 |

**提交日志**

|  |  |
| --- | --- |
| **命令** | **说明** |
| [git log](https://www.runoob.com/git/git-commit-history.html#git-log) | 查看历史提交记录 |
| [git blame <file>](https://www.runoob.com/git/git-commit-history.html#git-blame) | 以列表形式查看指定文件的历史修改记录 |

**远程操作**

|  |  |
| --- | --- |
| **命令** | **说明** |
| [git remote](https://www.runoob.com/git/git-remote.html) | 远程仓库操作 |
| [git fetch](https://www.runoob.com/git/git-fetch.html) | 从远程获取代码库 |
| [git pull](https://www.runoob.com/git/git-pull.html) | 下载远程代码并合并 |
| [git push](https://www.runoob.com/git/git-push.html) | 上传远程代码并合并 |

# git操作实例-推送到GitHub

## 1.创建一个远程仓库，前提是已经有GitHub/gitee账号

|  |
| --- |
|  |
|  |

## 2.创建一个本地仓库

## 1）在git-repo（我的专门存放git项目的地方）新建一个目录如test2

|  |
| --- |
|  |

## 2）在中国目录里面方2个文件一个是图片一个是文本文件

|  |
| --- |
|  |

## 3）在这个文件夹里面点击右键-》Git Bash Here，会打开一个命令行窗口

|  |
| --- |
|  |

## 4）输入命令：git init，出现下面界面，说明本地仓库初始化成功

|  |
| --- |
|  |

## 5）添加文件到本地仓库暂存区使用命令：git add .

|  |
| --- |
|  |

## 6)将暂存区的内容提交到本地仓库，使用命令：git commit –m “first commit”

|  |
| --- |
|  |

### 至此，本地仓库操作完毕

## 3.设置远程仓库

## 要用git来提交代码到远程，必须设置远程仓库，使用命令：git remote add origin <https://github.com/kennycaiguo/test2>，也就是将远程仓库映射为origin名称，没有报错，就成功了

|  |
| --- |
|  |

## 4.推送到远程仓库，使用命令 git push origin master,出现下面信息，说明推送成功

|  |
| --- |
|  |

## 5.查看远程仓库，登录到GitHub查看，注意需要选择master

|  |
| --- |
|  |

## 6创建分支（branch）

### 1）先在本地创建一个分支名称叫做sub，命令：git branch sub，没有报错说明成功

|  |
| --- |
|  |

### 2）查看所有分支，命令git branch

|  |
| --- |
|  |

#### 可以看到有两个分支，当前分支为master

### 3）切换到sub分支，命令：git checkout sub

|  |
| --- |
|  |

#### 可以看到现在切换到了sub分支

### 4）往test2文件夹里面提交一个JQuery3.7文件夹

|  |
| --- |
|  |

## 5）把这个文件夹提交到sub分支

|  |
| --- |
|  |

### 6）把这个分支推送到远程仓库，也叫做sub：git push origin sub

|  |
| --- |
|  |

## 7）可以到远程仓库去查看sub

|  |
| --- |
|  |

#### 注意：推送到的远程仓库分支的名称必须和本地名称一样，否则报错

# 关于合并分支

## 需要先将当前分支切换为master分支，然后选择需要合并的分支进行合并

# git操作实例-推送到Gitee，需要先pull再push

## 1.在gitee新建一个仓库叫做test

|  |
| --- |
|  |

## 2.在本地新建一个目录叫做test

|  |
| --- |
|  |

## 3，复制一个jQuery文件到test文件夹里面

|  |
| --- |
|  |

## 4.在这个文件夹里面打开一个git bash窗口，然后做git init

|  |
| --- |
|  |

## 5.把内容添加到暂存区，git add .

|  |
| --- |
|  |

## 6.将暂存区的内容添加到本地仓库：git commit –m“1”

|  |
| --- |
|  |

## 7.添加远程仓库地址

|  |
| --- |
|  |

## 8.需要先执行命令：git pull orgin master --allow-unrelated-histories

|  |
| --- |
|  |

## 9.成功后，执行推送：git push orgin master

|  |
| --- |
|  |

## 10.在gitee中查看

|  |
| --- |
|  |

# 也可以这么搞：先创建远程仓库，然后使用git把它复制到本地，这样子就可以省去init，设置远程仓库的步骤

## 1.新建远程仓库

|  |
| --- |
|  |

## 2.使用git clone <https://gitee.com/guohuacai/repo1.git>命令把窗口克隆下来

|  |
| --- |
|  |

## 3.进入repo1目录，在这里打开一个git bash窗口

|  |
| --- |
|  |

## 4.把一些文件添加到这里

|  |
| --- |
|  |

## 5.执行git add .

|  |
| --- |
|  |

## 6.执行 git commit -m "a"

|  |
| --- |
|  |

## 7.查看远程仓库名称，执行git remote

|  |
| --- |
|  |

## 8.执行git push origin master

|  |
| --- |
|  |

## 9.在gitee中查看

|  |
| --- |
|  |

### gitee仓库建议使用这种方法

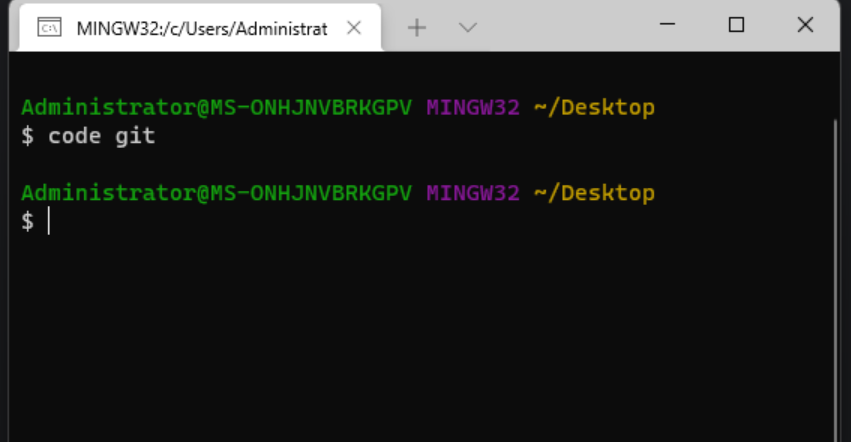
# VS Code 中 GIt 相关操作

（在这里我将使用一个空文件夹 git 作为项目文件进行相关的演示操作。）

本地类操作

### 1. 准备项目文件并初始化仓库

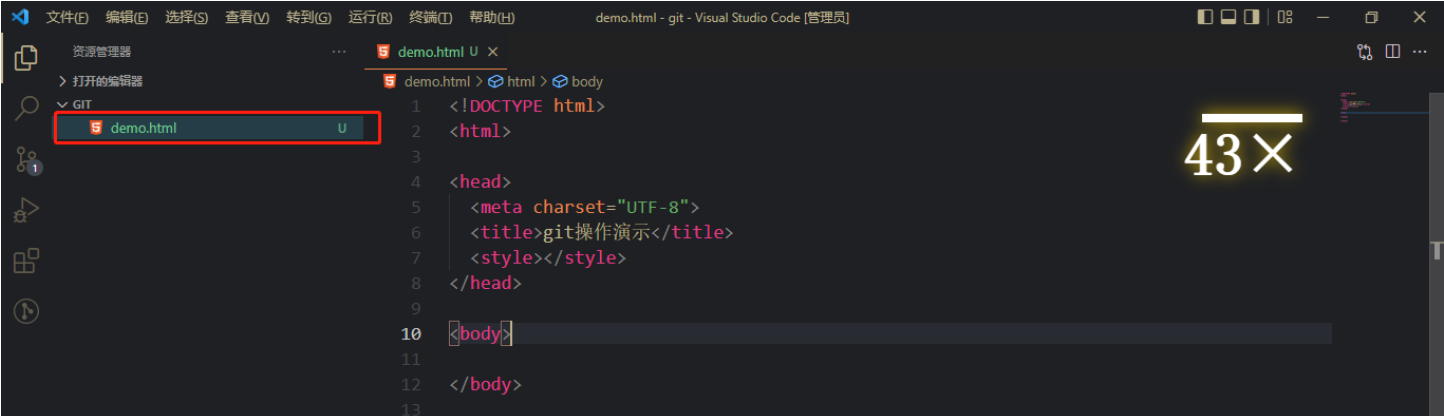
       新建项目文件夹 git 作为项目；桌面进入 Git Bash Here，输入命令 code git 使用 vs code 打开我们的项目文件夹；



vs code 中进入源代码管理，点击存储库的初始化。这步也就完成了 git 仓库的初始化，此时可以看到文件夹已有 .git 文件夹(默认隐藏，需要设置显示隐藏文件即可看到)；



随后在项目文件 git 中新建文件 demo.html；

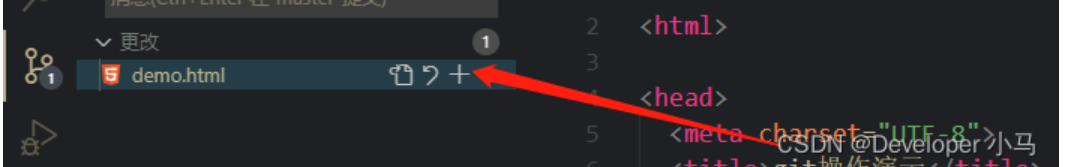


此时我们可以看到新建的文件为绿色，且后面有 U 字提示；U 意为 Untracked 未追踪，也就是说此文件目前仅在我们本地，未被 git 进行跟踪。

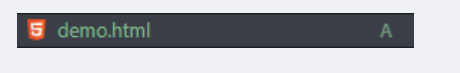
image.png

### 2. 添加文件至暂存区

       进入源代码管理，点击所要操作文件后面的 “+”，即可将文件添加至暂存区，相当于执行命令 git add；



此时文件名仍然为绿色，但是后面的 U 变成了 A，A 则意为此文件已在暂存区。



### 3. 提交暂存区文件至 git 仓库

       将暂存区的文件提交至 git，进入源代码管理，在消息框中输入相关备注，然后点击 “√” 即可进行一次提交，也可使用 Ctrl+Enter 快捷键直接提交；

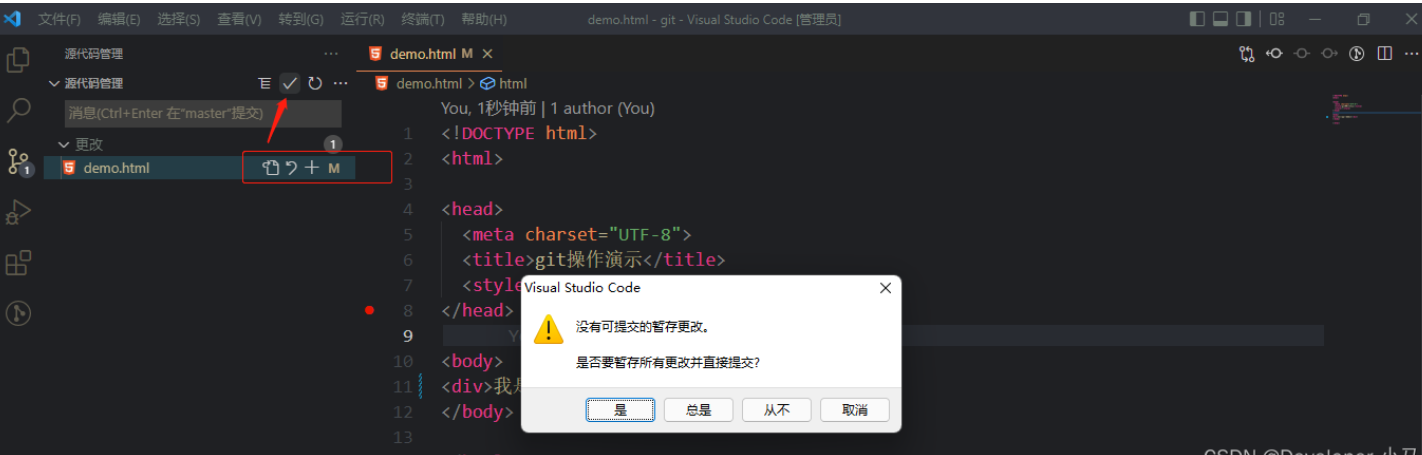


此时文件颜色变为正常，且无字母提示，说明文件已提交至 git。

image.png

       提交也可使用快速提交方式，所谓快速提交就是将新文件或者修改后的文件直接提交，而不用先保存至暂存区。省略了添加至暂存区操作；

如下，如果我们将修改后的文件未保存至暂存区就进行了提交，vs code 会给出警告，这时我们选择“总是”，即可在未来的操作中自动省略添加至暂存区操作。



### 4. 修改提交后的文件

       将提交后的文件内容进行修改后，文件名称会有如下提示；



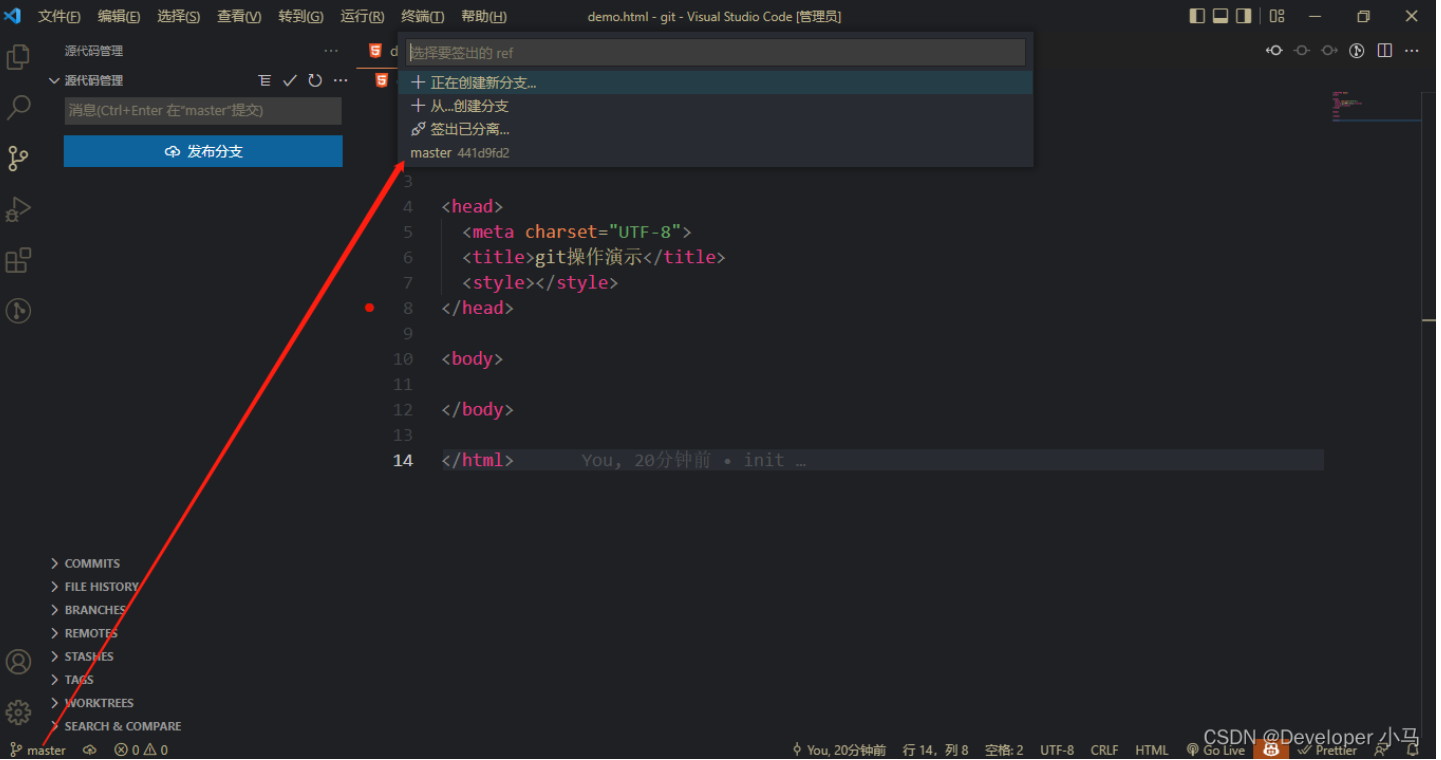
此时文件名变为黄色，且有提示字母 M；M 意为 modify，即已修改。



想要撤销修改，进入源代码管理，点击返回箭头放弃更改即可；

### 5. 查看/新建/切换分支

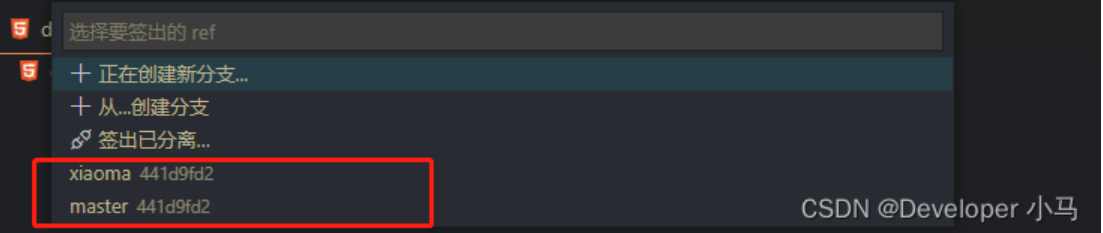
       点击 vs code 底部状态栏的 “master” 即可显示项目中的分支，此处的 master 意为当前所在分支为 master；如下图所示，项目中只有一个 master 分支；



创建分支点击 “+ 正在创建新分支...”；



此时项目中已有两个分支；



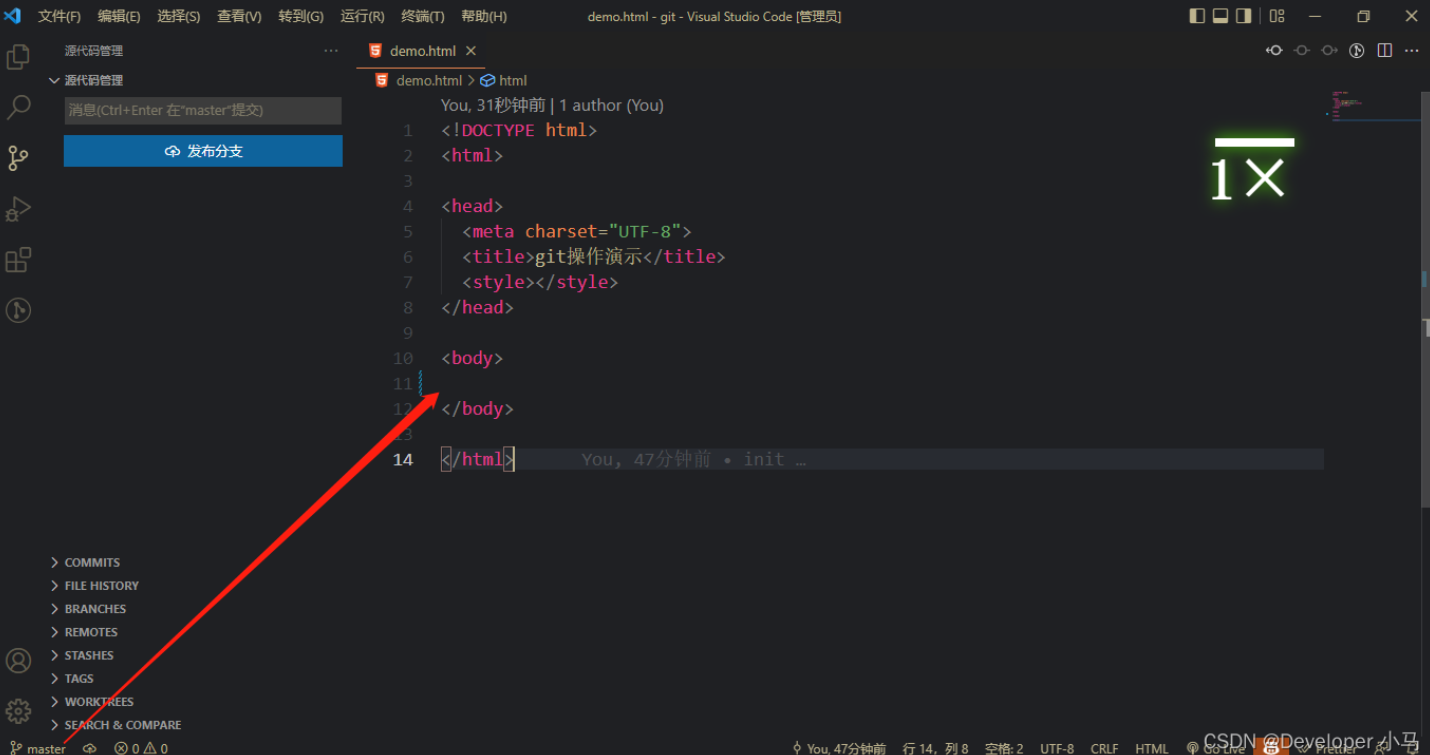
切换分支只需点击对应分支名称即可；

### 6. 合并分支

       我们先在新建的 xiaoma 分支中修改文件内容并提交；



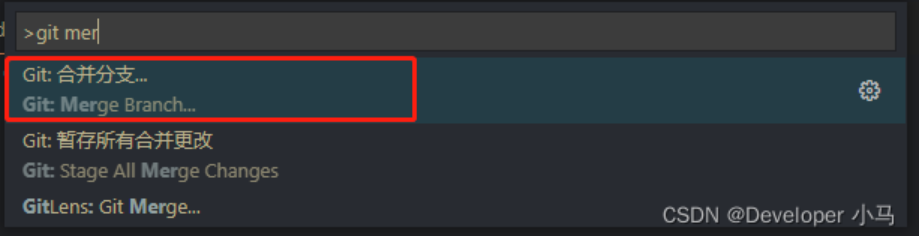
随后切换到 master 分支，此时 master 分支是没有修改后的内容的；



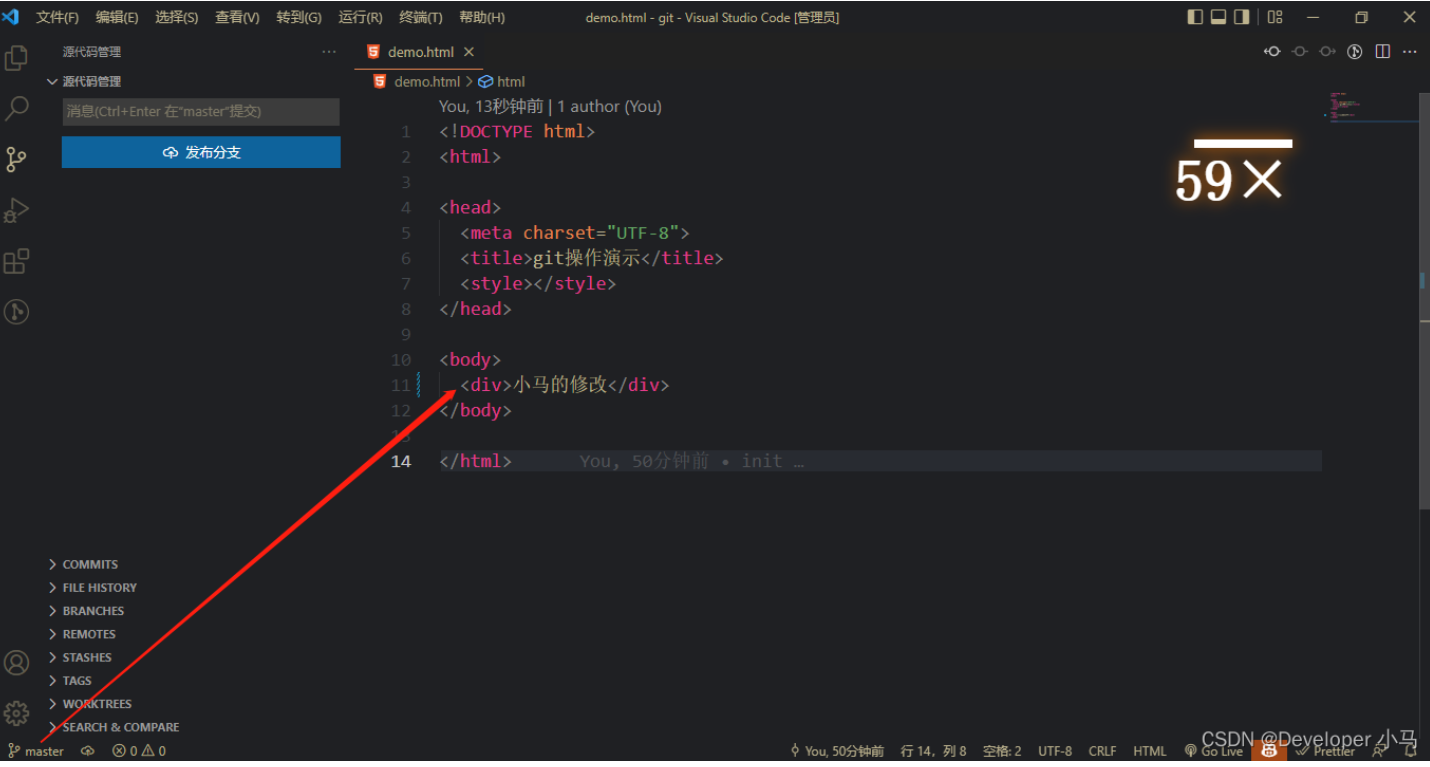
接下来进行合并，首先点击设置图标 “⚙” 打开命令面板；



在命令面板中搜索 git merge，找到合并分支操作，点击合并分支；



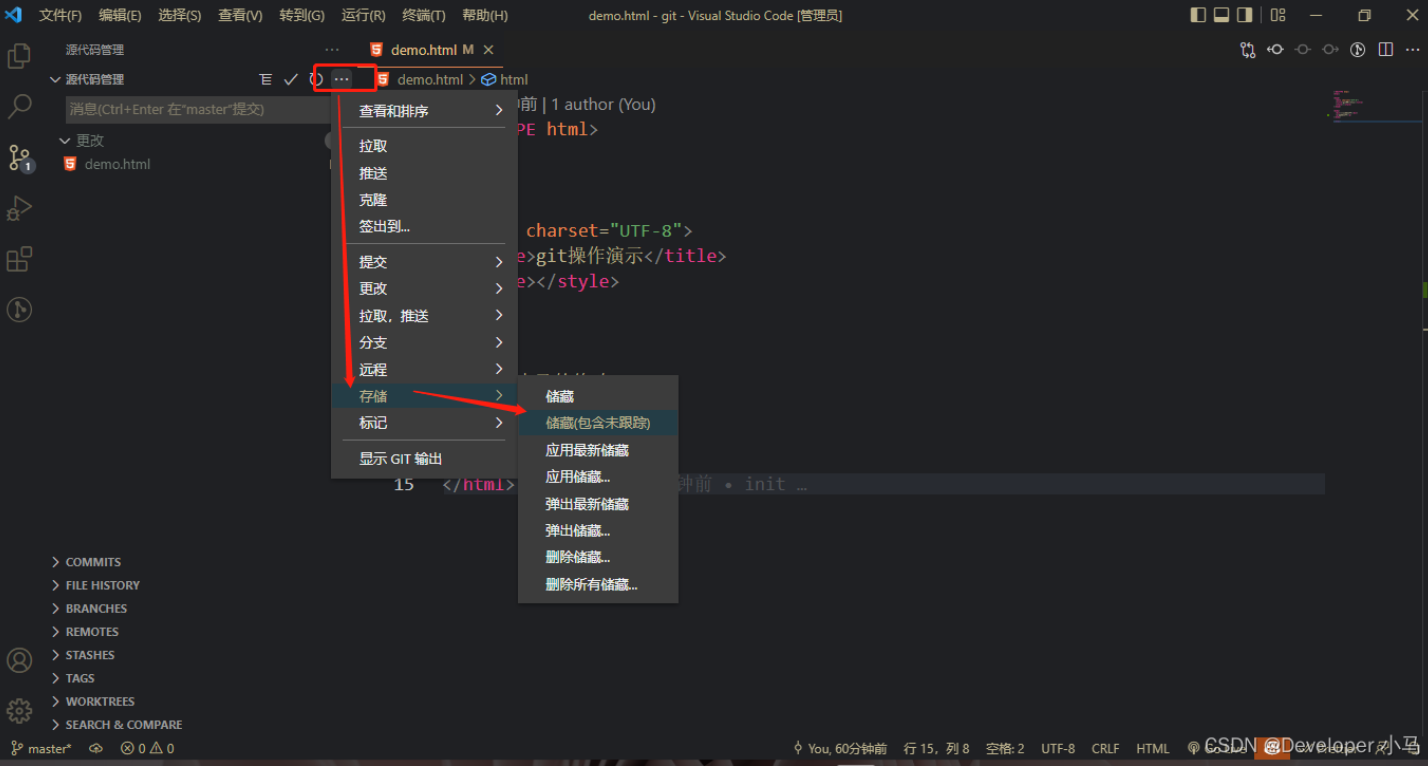
此时 master 分支已有 xiaoma 的相关操作；



### 7. 临时保存当前分支状态

       当我们在工作过程中代码写到一半，并没有完成某个模块的开发，而这时又不得不离开当前分支切换到另一个分支时，就需要将当前分支的状态进行临时保存(储藏、藏匿)，相当于 git 命令 git stash；

当在当前分支做了一定的修改后，点击更多 “···”，找到存储-->储藏；



### 8. 查看当前文件做了哪些修改

       当我们正在修改已经提交到 git 的文件内容时，vs code 会即使提示我们做了哪些操作，相当于 git 命令 git diff，如下：

蓝色代表此处已经被修改或者删除

绿色代表此处代码为新增内容

# 注意：如果有一些文件不需要上传到远程仓库，可以新建一个.gitignore文件，把不需要上传的文件名称写在里面，上传时git就会忽略这些文件

# git撤销修改

#取消对文件的修改。还原到最近的版本，废弃本地做的修改。

git checkout -- <file>

<file>是指文件名

#取消已经暂存的文件。即，撤销先前"git add"的操作

git reset HEAD <file>...

#修改最后一次提交。用于修改上一次的提交信息，或漏提交文件等情况。

git commit --amend

#回退所有内容到上一个版本

git reset HEAD^

#回退a.py这个文件的版本到上一个版本

git reset HEAD^ a.py

#向前回退到第3个版本

git reset –soft HEAD~3

#将本地的状态回退到和远程的一样

git reset –hard origin/master

#回退到某个版本

git reset 057d

#回退到上一次提交的状态，按照某一次的commit完全反向的进行一次commit.(代码回滚到上个版本，并提交git)

git revert HEAD

# git回退详细讲解

自然，你是不会犯错的。不过现在是凌晨两点，你正在赶一份工作报告，你在readme.txt中添加了一行：

$ cat readme.txt

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes of files.

My stupid boss still prefers SVN.

在你准备提交前，一杯咖啡起了作用，你猛然发现了stupid boss可能会让你丢掉这个月的奖金！

既然错误发现得很及时，就可以很容易地纠正它。你可以删掉最后一行，手动把文件恢复到上一个版本的状态。如果用git status查看一下：

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to **update** what will be committed)

(use "git checkout -- <file>..." **to** discard changes **in** working directory)

modified: readme.txt

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

你可以发现，Git会告诉你，git checkout -- file可以丢弃工作区的修改：

$ git checkout -- readme.txt

命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两种情况：

一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

总之，就是让这个文件回到最近一次git commit或git add时的状态。

现在，看看readme.txt的文件内容：

$ cat readme.txt

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes of files.

文件内容果然复原了。

git checkout -- file命令中的--很重要，没有--，就变成了“切换到另一个分支”的命令，我们在后面的分支管理中会再次遇到git checkout命令。

现在假定是凌晨3点，你不但写了一些胡话，还git add到暂存区了：

$ cat readme.txt

Git is a distributed version control system.

Git is free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes of files.

My stupid boss still prefers SVN.

$ git add readme.txt

庆幸的是，在commit之前，你发现了这个问题。用git status查看一下，修改只是添加到了暂存区，还没有提交：

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: readme.txt

Git同样告诉我们，用命令git reset HEAD <file>可以把暂存区的修改撤销掉（unstage），重新放回工作区：

$ git re**set** HEAD readme.txt

Unstaged changes after reset:

M readme.txt

git reset命令既可以回退版本，也可以把暂存区的修改回退到工作区。当我们用HEAD时，表示最新的版本。

再用git status查看一下，现在暂存区是干净的，工作区有修改：

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: readme.txt

还记得如何丢弃工作区的修改吗？

$ git checkout *-- readme.txt*

$ git status

On branch master

nothing to **commit**, working tree clean

整个世界终于清静了！

现在，你已经学会了修改文件，然后把修改提交到Git版本库，现在，再练习一次，修改readme.txt文件如下：

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

然后尝试提交：

$ git add readme.txt

$ git commit -m "append GPL"

[master 1094adb] append GPL

1 file changed, 1 insertion(+), 1 deletion(-)

像这样，你不断对文件进行修改，然后不断提交修改到版本库里，就好比玩RPG游戏时，每通过一关就会自动把游戏状态存盘，如果某一关没过去，你还可以选择读取前一关的状态。有些时候，在打Boss之前，你会手动存盘，以便万一打Boss失败了，可以从最近的地方重新开始。Git也是一样，每当你觉得文件修改到一定程度的时候，就可以“保存一个快照”，这个快照在Git中被称为commit。一旦你把文件改乱了，或者误删了文件，还可以从最近的一个commit恢复，然后继续工作，而不是把几个月的工作成果全部丢失。

现在，我们回顾一下readme.txt文件一共有几个版本被提交到Git仓库里了：

版本1：wrote a readme file

Git **is** a version control system.

Git **is** free software.

版本2：add distributed

Git **is** a distributed version control system.

Git **is** free software.

版本3：append GPL

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

当然了，在实际工作中，我们脑子里怎么可能记得一个几千行的文件每次都改了什么内容，不然要版本控制系统干什么。版本控制系统肯定有某个命令可以告诉我们历史记录，在Git中，我们用git log命令查看：

$ git log

commit 1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master)

Author: Michael Liao <askxuefeng@gmail.com>

Date: Fri May 18 21:06:15 2018 +0800

append GPL

commit e475afc93c209a690c39c13a46716e8fa000c366

Author: Michael Liao <askxuefeng@gmail.com>

Date: Fri May 18 21:03:36 2018 +0800

add distributed

commit eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0

Author: Michael Liao <askxuefeng@gmail.com>

Date: Fri May 18 20:59:18 2018 +0800

wrote a readme file

git log命令显示从最近到最远的提交日志，我们可以看到3次提交，最近的一次是append GPL，上一次是add distributed，最早的一次是wrote a readme file。

如果嫌输出信息太多，看得眼花缭乱的，可以试试加上--pretty=oneline参数：

$ git log --pretty=oneline

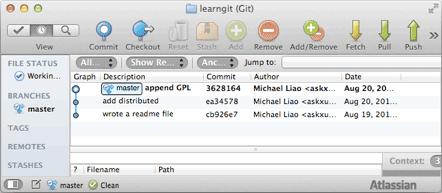
1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master) append GPL

e475afc93c209a690c39c13a46716e8fa000c366 add distributed

eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0 wrote a readme file

需要友情提示的是，你看到的一大串类似1094adb...的是commit id（版本号），和SVN不一样，Git的commit id不是1，2，3……递增的数字，而是一个SHA1计算出来的一个非常大的数字，用十六进制表示，而且你看到的commit id和我的肯定不一样，以你自己的为准。为什么commit id需要用这么一大串数字表示呢？因为Git是分布式的版本控制系统，后面我们还要研究多人在同一个版本库里工作，如果大家都用1，2，3……作为版本号，那肯定就冲突了。

每提交一个新版本，实际上Git就会把它们自动串成一条时间线。如果使用可视化工具查看Git历史，就可以更清楚地看到提交历史的时间线：



好了，现在我们启动时光穿梭机，准备把readme.txt回退到上一个版本，也就是add distributed的那个版本，怎么做呢？

首先，Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，也就是最新的提交1094adb...（注意我的提交ID和你的肯定不一样），上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。

现在，我们要把当前版本append GPL回退到上一个版本add distributed，就可以使用git reset命令：

$ git re**set** --hard HEAD^

HEAD **is** now **at** e475afc **add** distributed

--hard参数有啥意义？这个后面再讲，现在你先放心使用。

看看readme.txt的内容是不是版本add distributed：

$ cat readme.txt

Git **is** a distributed version control system.

Git **is** free software.

果然被还原了。

还可以继续回退到上一个版本wrote a readme file，不过且慢，让我们用git log再看看现在版本库的状态：

$ git log

commit e475afc93c209a690c39c13a46716e8fa000c366 (HEAD -> master)

Author: Michael Liao <askxuefeng@gmail.com>

Date: Fri May 18 21:03:36 2018 +0800

add distributed

commit eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0

Author: Michael Liao <askxuefeng@gmail.com>

Date: Fri May 18 20:59:18 2018 +0800

wrote a readme file

最新的那个版本append GPL已经看不到了！好比你从21世纪坐时光穿梭机来到了19世纪，想再回去已经回不去了，肿么办？

办法其实还是有的，只要上面的命令行窗口还没有被关掉，你就可以顺着往上找啊找啊，找到那个append GPL的commit id是1094adb...，于是就可以指定回到未来的某个版本：

$ git re**set** --hard 1094a

HEAD **is** now **at** 83b0afe append GPL

版本号没必要写全，前几位就可以了，Git会自动去找。当然也不能只写前一两位，因为Git可能会找到多个版本号，就无法确定是哪一个了。

再小心翼翼地看看readme.txt的内容：

$ cat readme.txt

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

果然，我胡汉三又回来了。

Git的版本回退速度非常快，因为Git在内部有个指向当前版本的HEAD指针，当你回退版本的时候，Git仅仅是把HEAD从指向append GPL：

┌────┐

│HEAD│

└────┘

│

└──▶ ○ append GPL

│

○ add distributed

│

○ wrote a readme file

改为指向add distributed：

┌────┐

│HEAD│

└────┘

│

│ ○ append GPL

│ │

└──▶ ○ add distributed

│

○ wrote a readme file

然后顺便把工作区的文件更新了。所以你让HEAD指向哪个版本号，你就把当前版本定位在哪。

现在，你回退到了某个版本，关掉了电脑，第二天早上就后悔了，想恢复到新版本怎么办？找不到新版本的commit id怎么办？

在Git中，总是有后悔药可以吃的。当你用$ git reset --hard HEAD^回退到add distributed版本时，再想恢复到append GPL，就必须找到append GPL的commit id。Git提供了一个命令git reflog用来记录你的每一次命令：

$ git reflog

e475afc HEAD@{1}: reset: moving to HEAD^

1094adb (HEAD -> master) HEAD@{2}: commit: append GPL

e475afc HEAD@{3}: commit: add distributed

eaadf4e HEAD@{4}: commit (initial): wrote a readme file

终于舒了口气，从输出可知，append GPL的commit id是1094adb，现在，你又可以乘坐时光机回到未来了。

# vscode gitee插件

## 在Vscode中点击插件按钮，输入gitee，会出现一些gitee插件，我们使用白羊的那个

|  |
| --- |
|  |

## 安装后如图

|  |
| --- |
|  |

## 安装完成后点击vscode的是设置按钮，进入设置界面

|  |
| --- |
|  |

## 在搜索框里面输入gitee，然后填写相关的配置，其中有一项是“您的gitee私人令牌“，那么怎么获取这个私人令牌？回到您的gitee仓库，点击你的用户标记，在弹出菜单中点击设置

|  |
| --- |
|  |

## 进入如下界面

|  |
| --- |
|  |

## 点击私人令牌按钮，会出现如下界面

|  |
| --- |
|  |

## 点击生成新令牌，出现如下界面，然后点击提交

|  |
| --- |
|  |

## 需要输入私人令牌描述，再点击提交

|  |
| --- |
|  |

## 需要输入你的密码，点击验证

|  |
| --- |
|  |

## 验证通过后会弹出一个有私人令牌的窗口需要把令牌复制下来粘贴到vscode的插件私人令牌一栏，并且另外保存，然后勾选同意xx复选框，点击确认并且关闭按钮

## 输入保存目录和输入令牌后，其他两项保持不变，关闭设置

## 这个插件可以获取你在gitee的所有仓库形成一个仓库列表，方便你编辑管理

|  |
| --- |
|  |

### 注意：设置好后需要重启vscode，这些仓库是不能直接编辑的，但是可以下载到本地编辑后可以上传同步

### 可以右键点击需要的项目，在弹出菜单中选择下载，或者打开网页或者打开在vscode（先下载再用vscode打开）等等

|  |
| --- |
|  |

### 注意：这个插件并没有回退功能，想回退的话还是在本地仓库目录中打开git bash窗口，然后使用git 相关命令

# 合并冲突的解决

## 插件有解决冲突的功能，你需要点击在冲突编辑器中打开，然后他会列出两个文件的不一样的地方，你可以根据需要勾选要保留的代码，然后点击接受合并

|  |
| --- |
|  |
|  |