

SOPtoWorkWithGit

冷知识：SOP是什么？

作业标准书，英文全称“**Standard Operating Procedure**”，简称SOP，英文直译是“标准操作流程”，国内习惯称之为作业标准书，又称作业指导书。作业标准书是把现场所有的工作制定出一套流程每个人按部就班的按照流程来执行。在实际的生产中在不断的对SOP进行修改和完善，以提高操作的规范性和高效性。

为了防止大家在本次数据库课程设计中在使用git进行版本控制时出现各种各样的问题，本文会对git——这个对大家以后的一生工作都会打交道的工具进行一般标准工作流程的梳理。

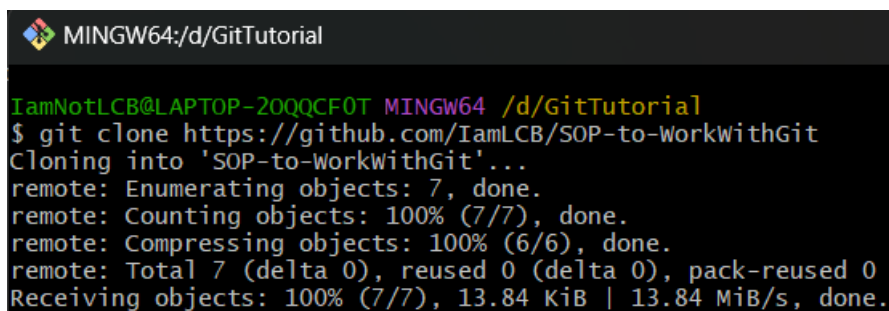
本教程的github链接：<https://github.com/IamLCB/SOP-to-WorkWithGit>，本文也会在该仓库中进行操作示范。

0x00.获得远程仓库-gitgit clone

很显然，工作开始时，你需要得到初始的仓库内容（本文不涉及创建仓库的部分，默认从co-worker的视角进行）。

在你的任一目录下（一般为了不出bug，建议使用全英文目录）运行 **git clone** 命令：

```
git clone <your-repo-url>
```

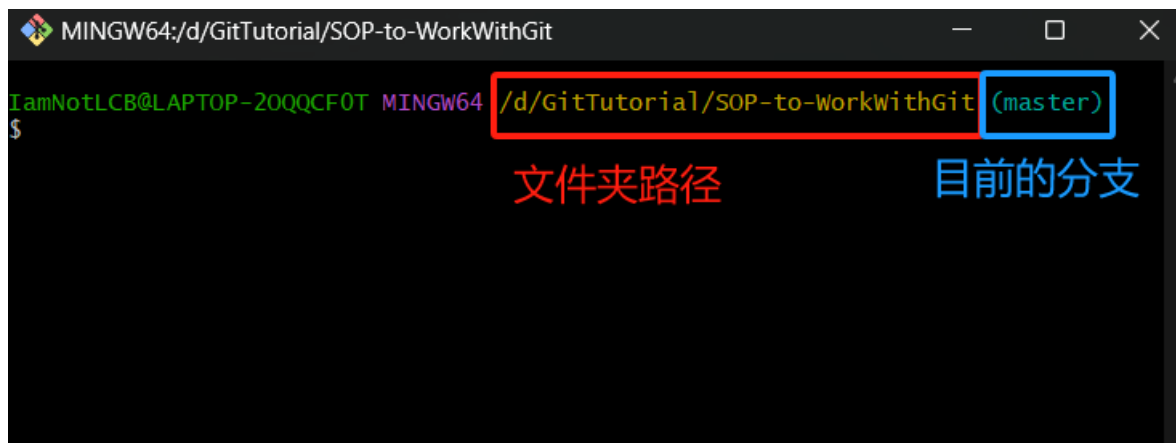


```
MINGW64:/d/GitTutorial
IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial
$ git clone https://github.com/IamLCB/SOP-to-WorkWithGit
Cloning into 'SOP-to-WorkWithGit'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), 13.84 KiB | 13.84 MiB/s, done.
```

这样，你就能在你的目录下得到一个新的文件夹，在本文中，为 `SOP-to-WorkWithGit`

0x01.正式开始工作前的准备-git checkout与git pull

打开上一步获得的 `SOP-to-WorkWithGit`，在空白处 右键-显示更多选项 (for win11) -Open Git Bash Here 即能打开一个新的git窗口，此时会显示如下信息：

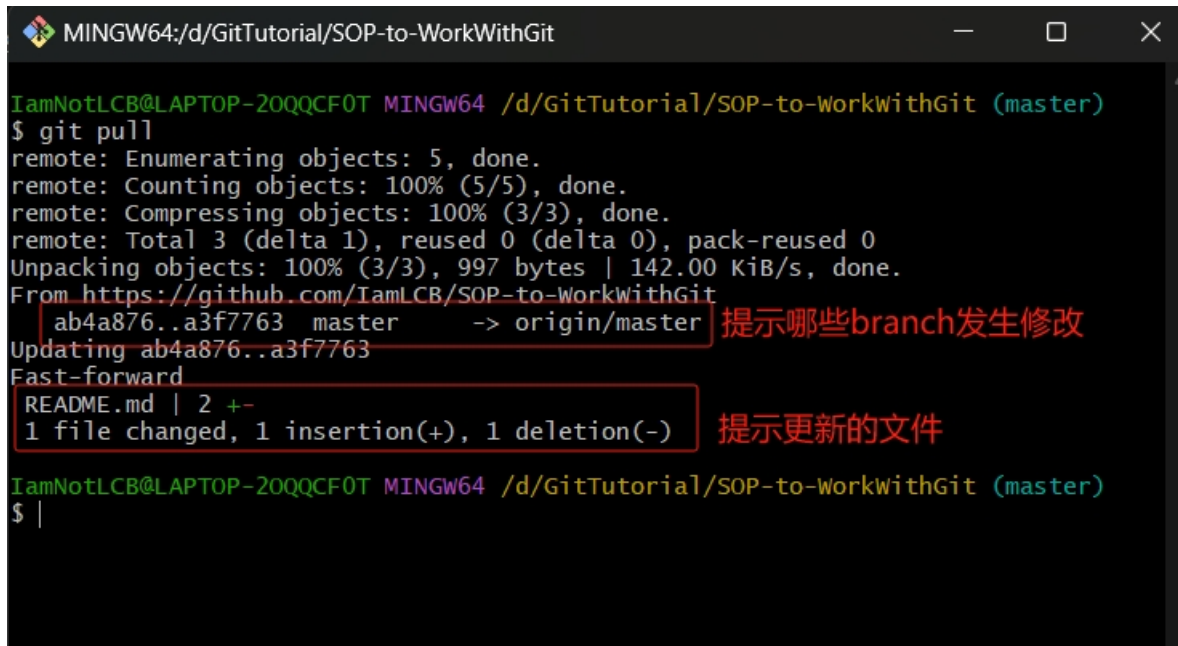


```
MINGW64:/d/GitTutorial/SOP-to-WorkWithGit
IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (master)
$
```

文件夹路径 目前的分支

如果你不想被复杂的 `merge conflict` 所苦恼，我们先要对有可能被别的用户所更改的远端仓库内容做更新，此时，使用 `git pull` 命令，只需要在命令行中输入：

```
git pull
```



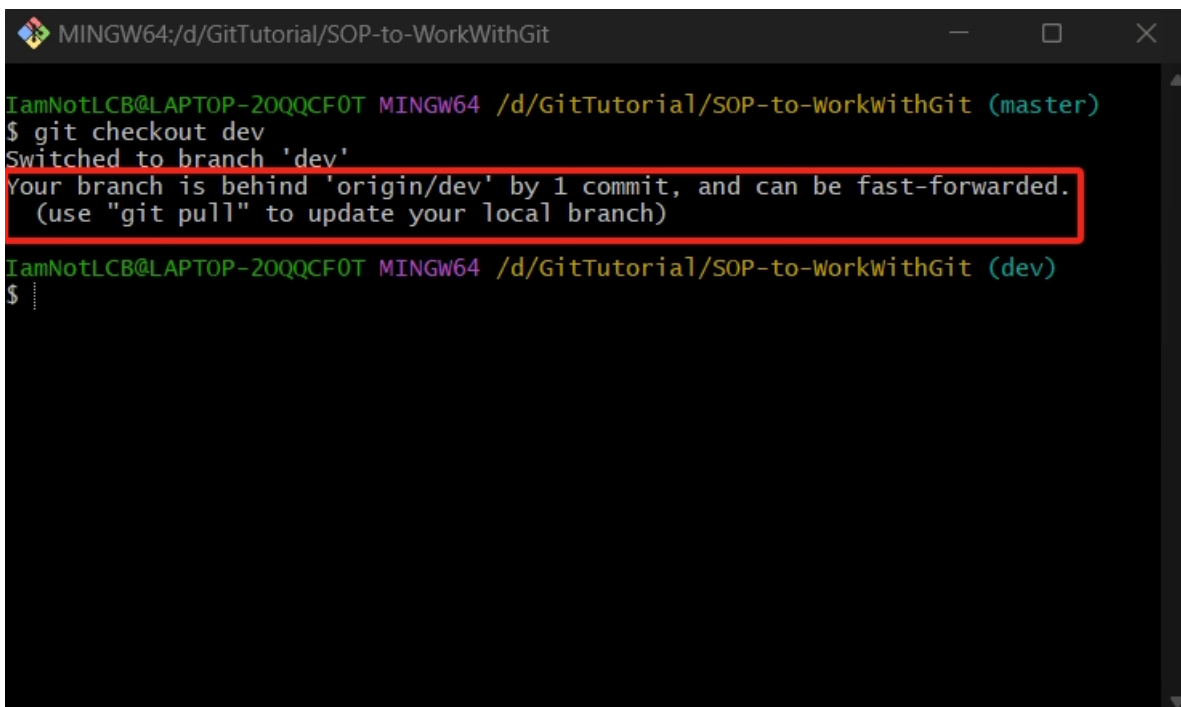
```
MINGW64:/d/GitTutorial/SOP-to-WorkWithGit
IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (master)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 997 bytes | 142.00 KiB/s, done.
From https://github.com/IamLCB/SOP-to-WorkWithGit
  ab4a876..a3f7763  master    -> origin/master
Updating ab4a876..a3f7763
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

在输出的指令中，会提示 哪些分支发生了修改与更新 以及 更新的文件。

当然，更常规的情况是我们使用多分支来进行合理的版本控制，那么在此时我们会先进行切换分支的操作，

使用指令：

```
git checkout <your-branch-name>
```



```
MINGW64:/d/GitTutorial/SOP-to-WorkWithGit
IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (master)
$ git checkout dev
Switched to branch 'dev'
Your branch is behind 'origin/dev' by 1 commit, and can be fast-forwarded.
(use "git pull" to update your local branch)
IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (dev)
$
```

如果你在第一次打开git bash的时候就使用过 `git pull` 指令，此时会提示：

```
Your branch is behind 'origin/<your-branch-name>' by x commit, and can be fast-forwarded.
```

其中，最重要的是 *can be fast-forwarded*，这个意味着可以快速合并，那么我们此时再使用 *git pull* 指令就可以快速保持本地内容与远程仓库一致了，当然，有时候会出现 *cannot* 的情况，我们会在文章最后讲解处理 *merge conflict* 的问题。

此时，你就可以正常的开始你的开发工作了！

0x02 开发完成后提交工作-status, add, commit, push

假设我们现在在 `dev` 分支下完成了开发工作，那么可以使用 *git status* 指令查看文件更新状况，在 `bash` 内使用

```
git status
```

```
MINGW64:/d/GitTutorial/SOP-to-WorkWithGit
IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (dev)
$ git status
On branch dev
Your branch is up to date with 'origin/dev'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        This is MY DEV WORK

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

IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (dev)
$ |
```

通过结果可知，分为两类文件：

1. **Changes not staged for commit**：意味着更新的文件，也就是说原来就有，发生了更改
2. **Untracked files**：意味着新增的文件，以前没有，现在有了

那么接下来，我们需要把这些更改加入 **提交暂存区**，使用 *git add* 命令：

```
git add <your-file-name> (当然，your-file-name 可以使用一个 . 来快速添加所有文件)
```

```
IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (dev)
$ git add .

IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (dev)
$ git status
On branch dev
Your branch is up to date with 'origin/dev'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   This is MY DEV WORK

IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (dev)
$ |
```

注意，这里我们使用了 `git add .` 来快速添加所有文件，然后我们再使用 `git status` 命令查看状态，可以发现文件都变成了绿色，**Changes to be committed**，可以进行下一步的提交操作了。

下一步，我们将 **提交暂存区** 正式提交，使用 *git commit* 命令：

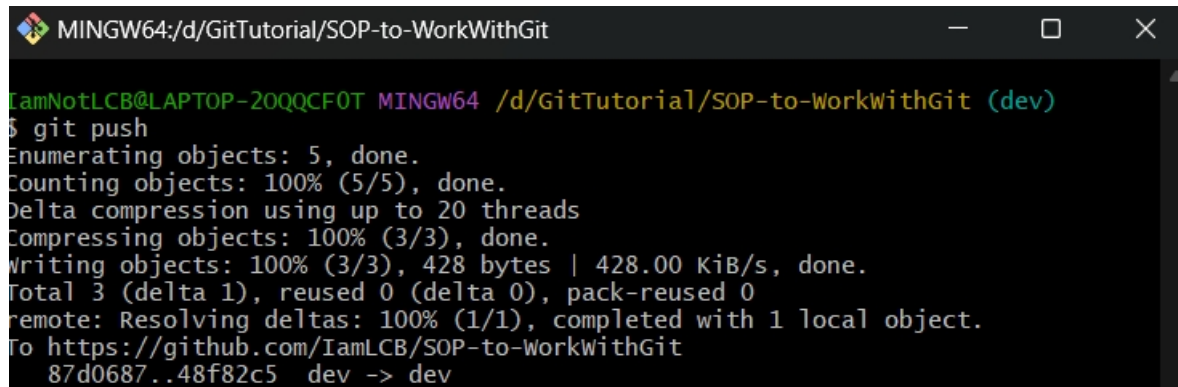
虽然大家需要有一定的英文读写能力，但是还是建议大家写中文的log/message，方便大家阅读。

```
IamNotLCB@LAPTOP-20QCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (dev)
$ git commit
[dev 48f82c5] 更新了README，提交DEV WORK。
2 files changed, 2 insertions(+)
create mode 100644 This is MY DEV WORK
```

退出vim后，就会自动提交commit。

现在我们已经成功的把我们的工作提交到了**本地仓库**，那么接下来我们就需要将本地的工作**提交到远端仓库**，使用 **git push**：

git push

A terminal window titled 'MINGW64:/d/GitTutorial/SOP-to-WorkWithGit' showing the output of the 'git push' command. The output indicates that 5 objects were enumerated, 100% of objects were counted, and 3 objects were compressed and written. The push was successful to the remote repository 'https://github.com/IamLCB/SOP-to-WorkWithGit' at commit '87d0687..48f82c5 dev -> dev'.

```
IamNotLCB@LAPTOP-20QCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (dev)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 20 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 428 bytes | 428.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/IamLCB/SOP-to-WorkWithGit
87d0687..48f82c5 dev -> dev
```

如果没有什么大问题，一般就会成功push，此时你已经把你的工作成功提交到了远端仓库，也就完成了一次标准的工作流程。当然，对于团队管理者，还会涉及到合并分支、处理冲突的问题，我们会在下一节当中解释。

0x03 团队管理者-merge&merge conflict

那么现在假设，我们手下的员工完成了工作的提交，他们提交到了 **dev** 分支下，现在，我需要把dev分支与main合并，也就是说 **把dev分支下更新的内容提交到main分支**（由于笔者的默认设置，本教程中主分支是master分支），那么我们就需要使用 **git merge**，此处我们展示一种最简单的方式：

1. git checkout master
2. git merge <your-branch-name>


```
MINGW64:/d/GitTutorial/SOP-to-WorkWithGit
IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (master)
$ git merge dev
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.

IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (master|MERGING)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

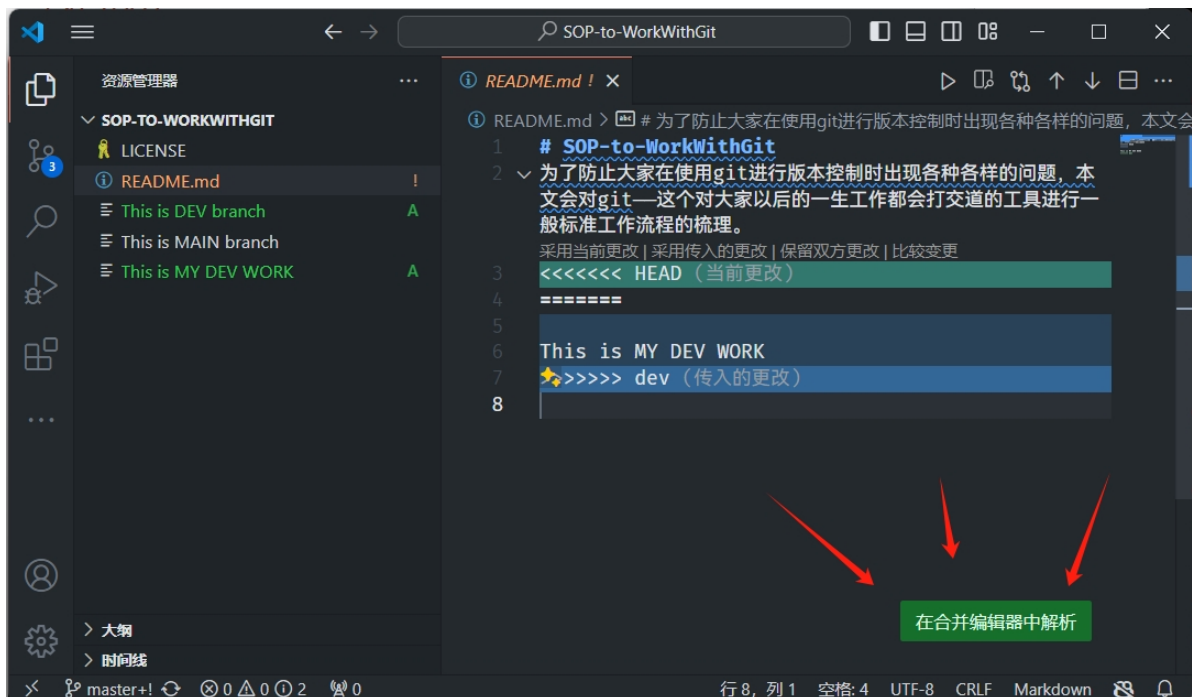
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:
  new file:   This is DEV branch
  new file:   This is MY DEV WORK

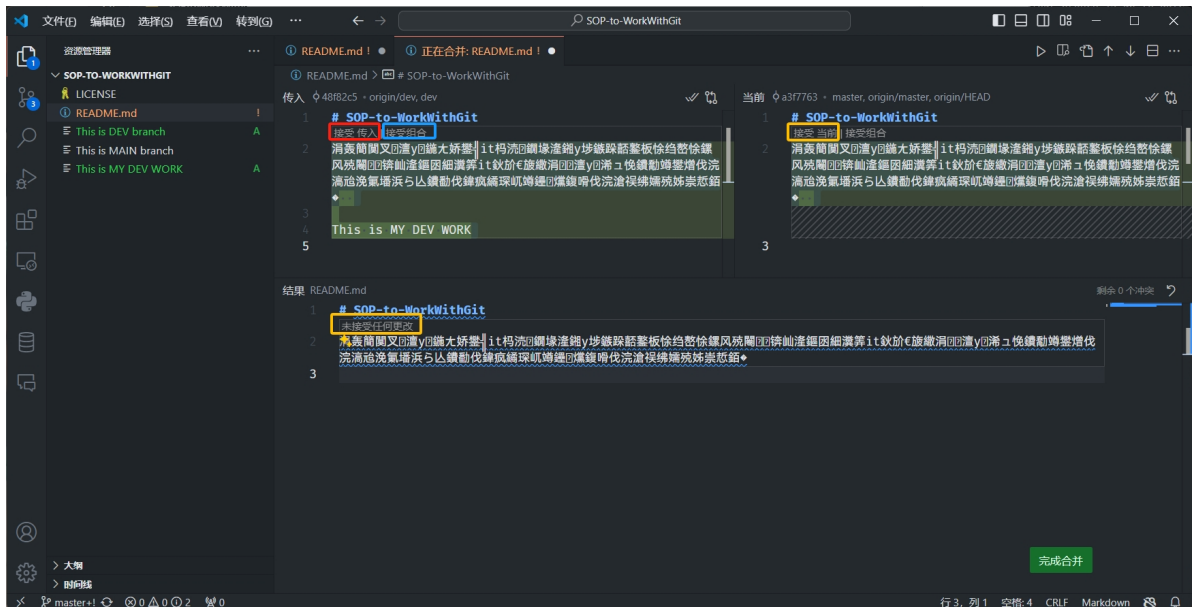
Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   README.md

IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (master|MERGING)
$ |
```

非常不巧，我们此处出现了 **CONFLICT** 的情况，也就是说，正如图中展示的，**git status** 告诉我们，对于 `README.md` 这个文件，master分支和dev分支都做了修改，而且修改的内容不一样（一般也不会运气好到两个人修改的完全一样），现在我们要处理 **merge conflict** 的情况，我们可以使用 **VSCode** 打开这个文件：



打开之后，什么都别管，直接点击进入 **在合并编辑器中解析**，就能进入到非常方便的图形化界面



(由于一些编码问题，所有的中文都变成了乱码，简单来说，DEV分支内的README.md比master分支中，结尾多了一行空行，一行This is MY DEV WORK)

那么，如图所示，你可以选择**接受 传入**、**接受 当前** 或是 **接受 组合**，来快速解决合并问题，当然，没有一种方案可以解决所有问题，在实际情况下，更可能的是你甚至需要手动修改代码文件来处理 **merge conflict**

```
IamNotLCB@LAPTOP-20QQCF0T MINGW64 /d/GitTutorial/SOP-to-WorkWithGit (master|Merging)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:
  modified:   README.md
  new file:   This is DEV branch
  new file:   This is MY DEV WORK
```

在解决 **merge conflict** 后，再查看status，就能看见所有的都变绿了，那么接下来直接使用

`git commit` 来提交你的merge操作并使用 `git push` 推送至远端仓库。

