《版本管理git回滚、重置、提交以及SourceTree复杂操作》

一、GIT简介

> Git是一个开源的分布式版本控制系统，可以 `有效` 、 `高速` 的处理从很小到非常大的项目版本管理。

- 对非线性开发有非常强的支持
- 分布式、协作式开发处理代码冲突的能力强
- 数据结构更优，更适合管理大规模工程
- 社区更活跃，生态更完整

二、命令行基本操作

1、Git基本配置

`git config` 可以配置git的参数，可以使用 `git config --list` 查看已经配置的git参数。

配置用户名及邮箱

> 在使用Git提交前，必须配置用户名和邮箱，这些信息会永久保存到历史记录中。

```
1   git config --global user.name "cpy"
2   git config --global user.email xxxx@qq.com
```

2、创建Git仓库

直接调用 `git init` 初始化当前目录，即创建Git仓库。

3、获得Git仓库

如果需要克隆远程仓库，可以使用 `git clone` ，例如：

```
1   git clone http://gitlab.zq.com:8888/fattan/frontend.git
```

4、提交更新

git文件状态

- **git status**

  通常提交前先检查下修改了什么内容，当前Git目录下各文件的状态。

```
xixi@DESKTOP-HPT8NH9 MINGW64 ~/Desktop/git/git (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        style.css

nothing added to commit but untracked files present (use "git add" to track)
```

- **git add**

  可以添加文件或者目录，也可以使用通配符。比如：

```
1    git add Readme.md     # add file only
2    git add *.cpp         # add all cpp files
3    git add /home/code/   # add all files in /home/code
```

  多文件提交：使用$ git add . 注意最后的“.”不要忘记。一次性提交项目下所有文件到版本库

- **git diff**

  `git diff` 可以查看当前目录的所有修改。

- **git commit -m**

  文件添加到版本库

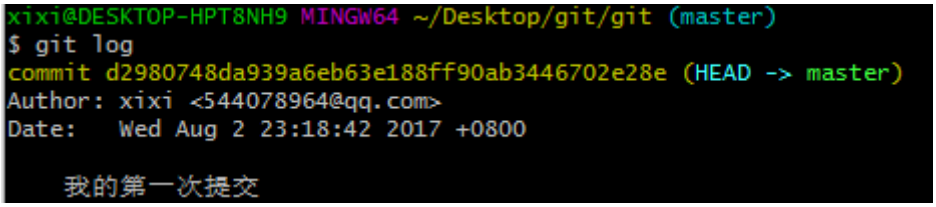
```
xixi@DESKTOP-HPT8NH9 MINGW64 ~/Desktop/git/git (master)
$ git commit -m "我的第一次提交"
[master (root-commit) d298074] 我的第一次提交
 1 file changed, 11 insertions(+)
 create mode 100644 gitdemo.html
```

- **git rm**

  `git rm` 会把文件从当前目录删除（不会保存删除的文件）。如果需要从Git仓库中删除，但保留在当前工作目录中，亦即从跟踪清单中删除，可以使用 `git rm --cached readme.md` 。

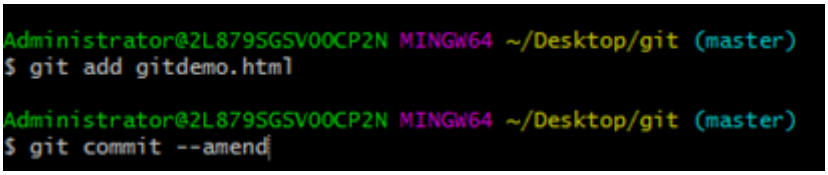
5、提交历史查看

> 执行命令 `git log` 后，提交人、邮箱、时间、备注信息显示出来，如图：

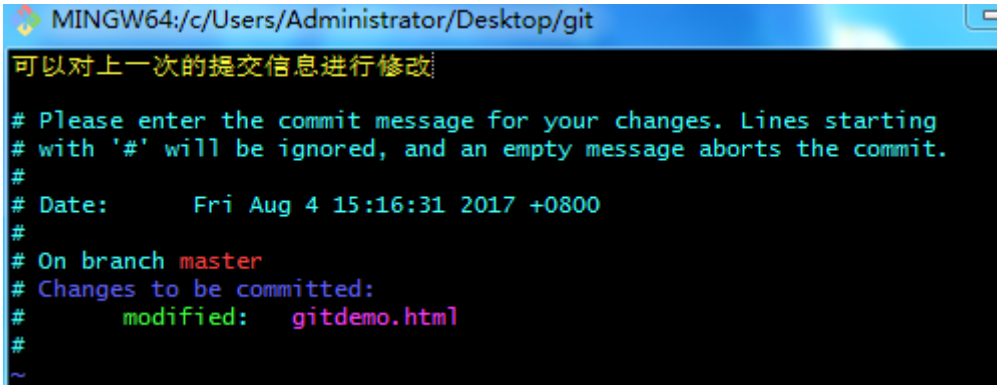


6、撤销操作

> 本地仓库的代码还原操作叫做 `撤销`
>
> 修改最后一次版本区提交：提交信息写错了。想要撤消刚才的提交操作，
>
> `$ git commit --amend` 选项重新提交，如图所示：



> 对文件进行修改放入缓存区之后 `$ git commit -amend` 进入修改区域



修改完成后 按下 `Esc` 键 使用 `:wq` 退出不要忘了前面的

7、代码回滚

- **在工作区的代码**

> 丢弃文件操作

```
1   git checkout -- a.txt  # 丢弃某个文件
2
3   git checkout -- .      # 丢弃全部
```

- **代码git add到缓存区，并未commit提交**

> `回滚缓存区文件操作`，此命令仅改变暂存区，并不改变工作区，这意味着在无任何其他操作的情况下，工作区中的实际文件同该命令运行之前无任何变化

```
git reset HEAD .          # 回滚暂存所有

git reset HEAD a.txt      # 回滚暂存某个文件
```

- **git commit到本地分支、但没有git push到远程**

> `回滚本地分支文件操作`，此命令是在git commit到本地分支之后、但没有git push到远程时的操作。

```
git log                           # 得到你需要回退一次提交的commit id
git reset --hard <commit_id>      # 回到其中你想要的某个版
git reset --hard HEAD^            # 回到最新的一次提交
git reset HEAD^                   # 此时代码保留，回到 git add 之前
```

- **git push把修改提交到远程仓库**

1）通过git reset是直接删除指定的commit

```
git log                           # 得到你需要回退一次提交的commit id
git reset --hard <commit_id>      # 强制返回上一次提交的源码状态
git push origin HEAD --force      # 强制提交一次，之前错误的提交就从远程仓库删除
```

2）通过git revert是用一次新的commit来回滚之前的commit

```
git log                  # 得到你需要回退一次提交的commit id
git revert <commit_id>   # 撤销指定的版本，撤销也会作为一次提交进行保存
```

3） git revert 和 git reset的区别

  - `git revert`是用一次新的commit来回滚之前的commit，此次提交之前的commit都会被保留；
  - `git reset`是回到某次提交，提交及之前的commit都会被保留，但是此commit id之后的修改都会被删除

8、远程仓库

> 可以使用`git remote`查看当前的远程库。`git remote -v`可以显示对应的克隆地址。（对于多个远程仓库很有用）

1. **添加远程仓库**

```
git remote add [short_name] [url]    # 可以添加新的远程仓库。
```

2. **从远程仓库抓取数据**

```
git fetch [remote-name]      # 可以从远程仓库抓取数据到本地。
```

也可以使用 `git pull`

3. **推送数据到远程仓库**

```
git push [remote_name] [branch_name]     # 默认使用origin和master
```

4. **查看远程仓库信息**

```
git remote show origin
```

5. **远程仓库的删除和重命名**

```
git remote rename [old_name] [new_name]
git remote rm [remote_name]
```

9、 分支

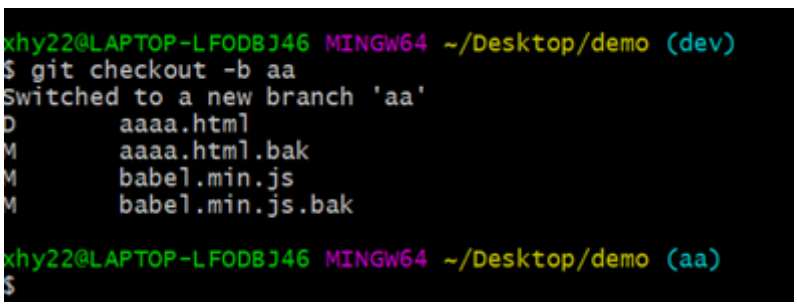
1. **显示所有分支**

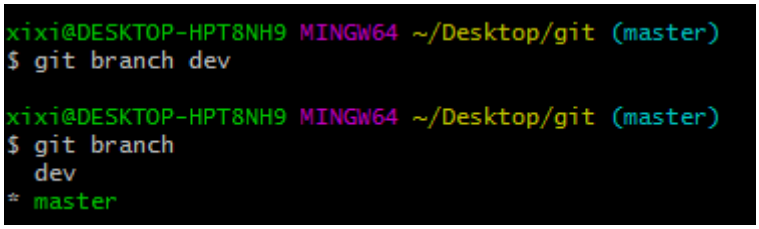
使用 `git branch` 可显示当前所有分支。可以使用--merged和--no-merged查看已经合并、未合并的分支。

2. **创建及切换分支**

可以使用下面命令直接切换并创建分支

```
git checkout -b testing # 创建testing 分支
git branch testing      # 创建testing 分支
git checkout testing    # 切换到testing分支
```





注意：切换分支时请保持工作目录没有未提交的修改。Git鼓励使用分支，处理完问题之后合并分支即可

3. **分支合并**

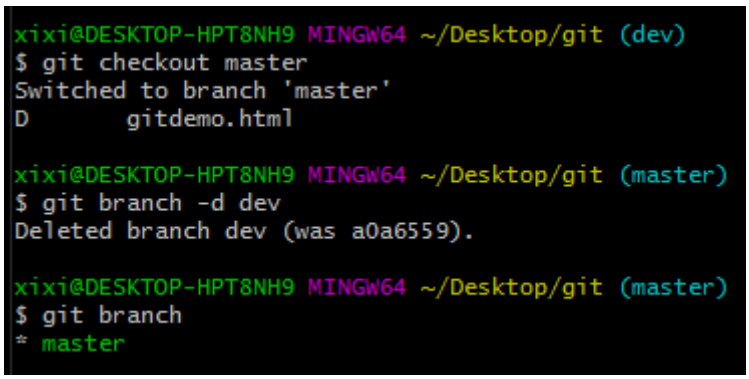
将dev分支合并到master（主分支）上，需要通过下面命令

```
$ git checkout master
$ git merge hotfix
```

合并之后可以使用 `git branch -d dev` 删除分支。如果合并时存在冲突，需要手工修改。

4. **删除分支**

```
git branch -d       #分支的删除
```



注意：要删除分支不能在当前分支删除，必须切换到其他分支才能删除

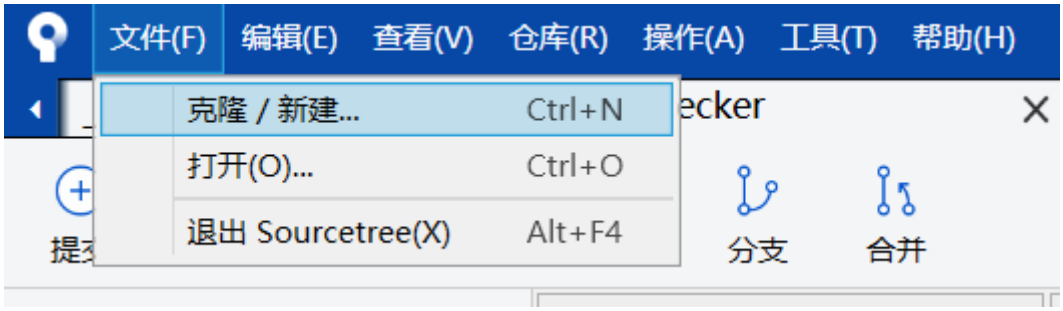
5. **修改分支名**

```
git branch -m       # 分支名字 新名字来进行分支的重命名
```

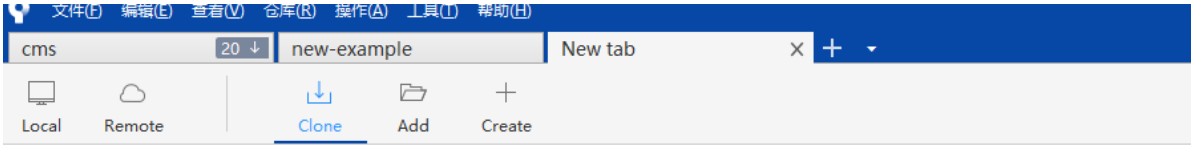
三、Source Tree 操作

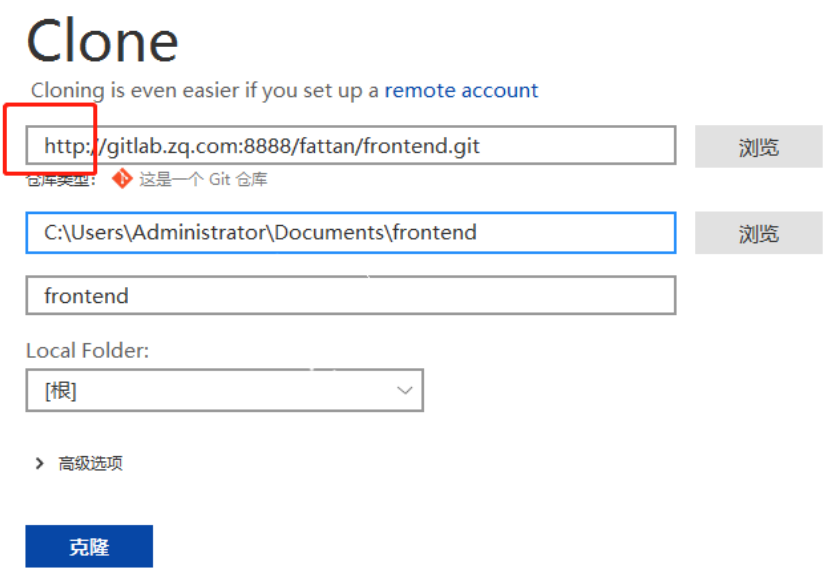
1、拉取代码

> Sourcetree顶部菜单栏"文件" > “克隆/新建”。

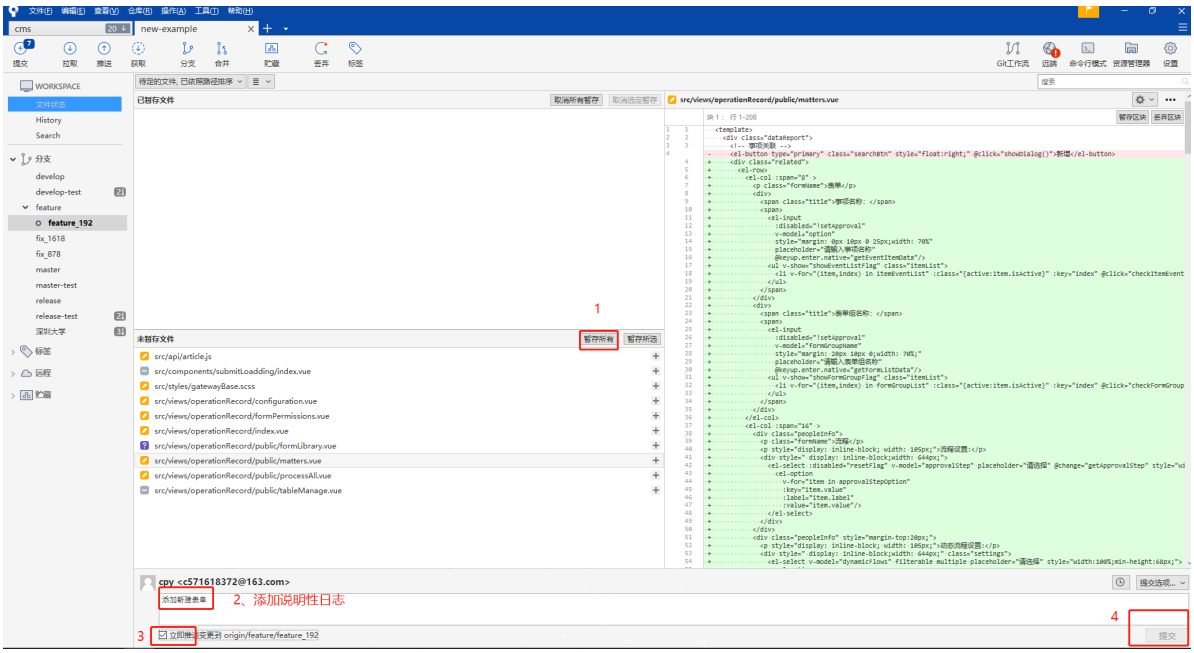


> 在弹出的Tab页中，选择默认的"Clone"，输入获取到的Git地址。
>
> 注意：这里使用的HTTP格式的地址



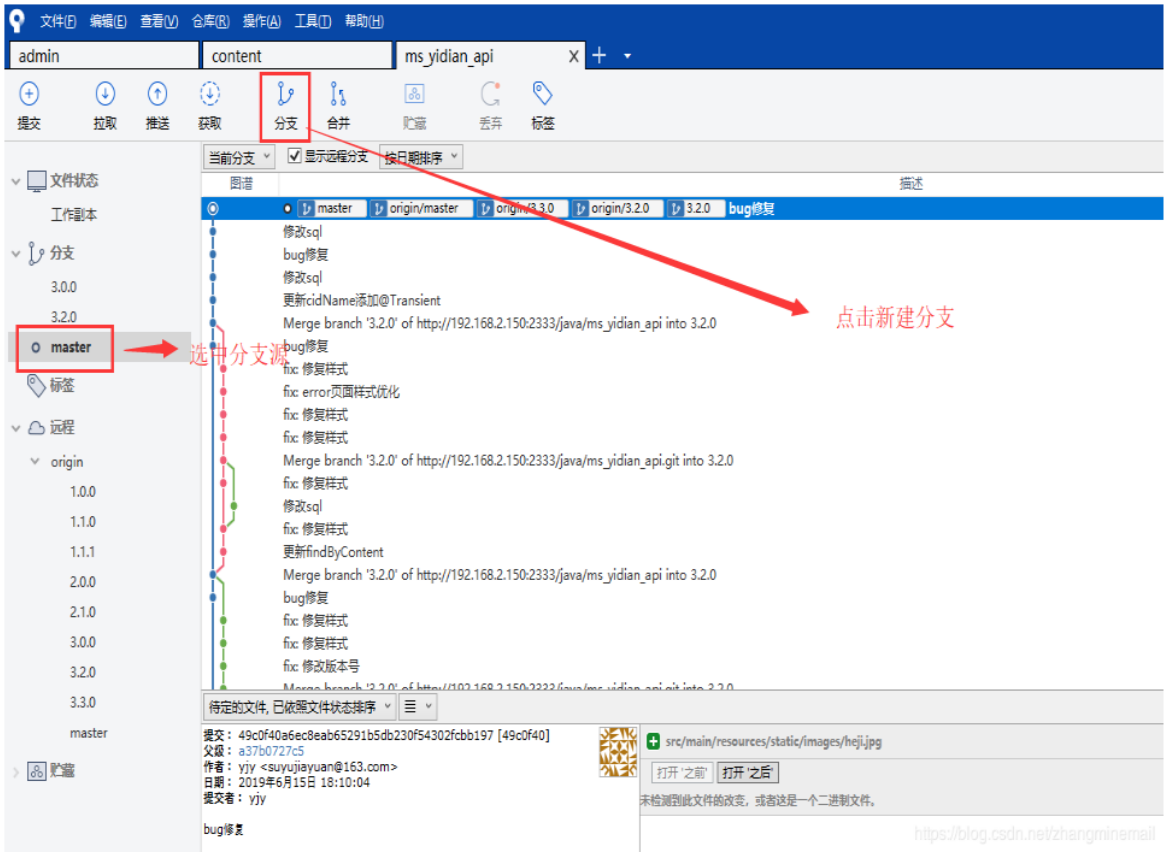


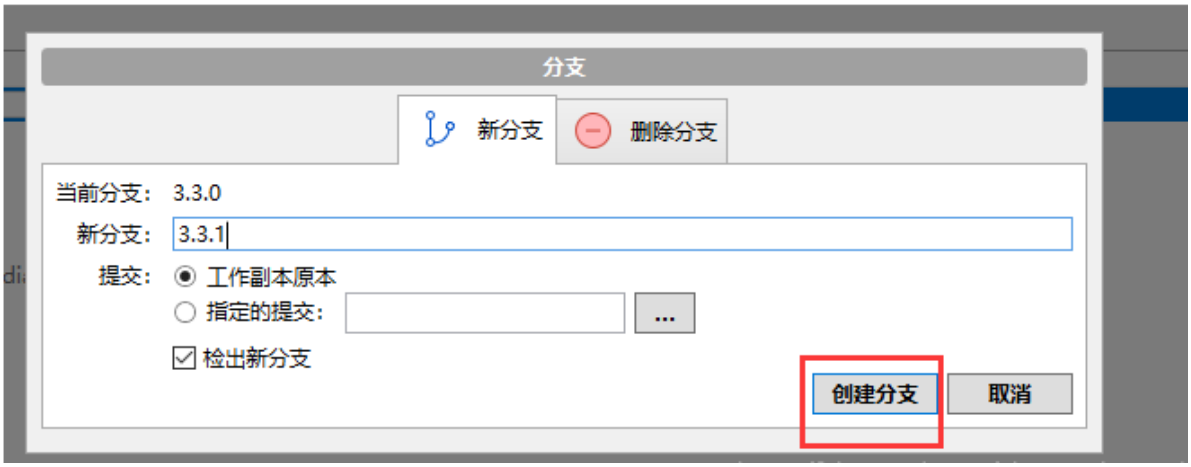
2、提交代码



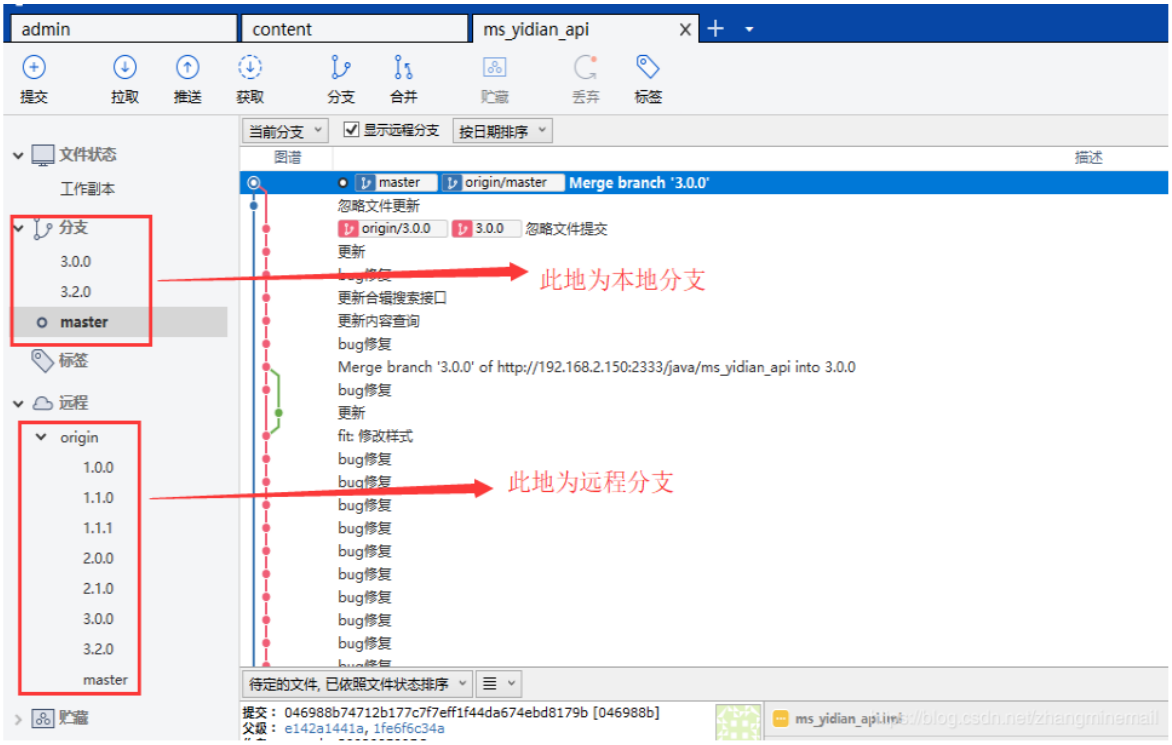
命令行	Sourcetree
git add	暂存
git commit	提交
git push	推送

3、新建分支

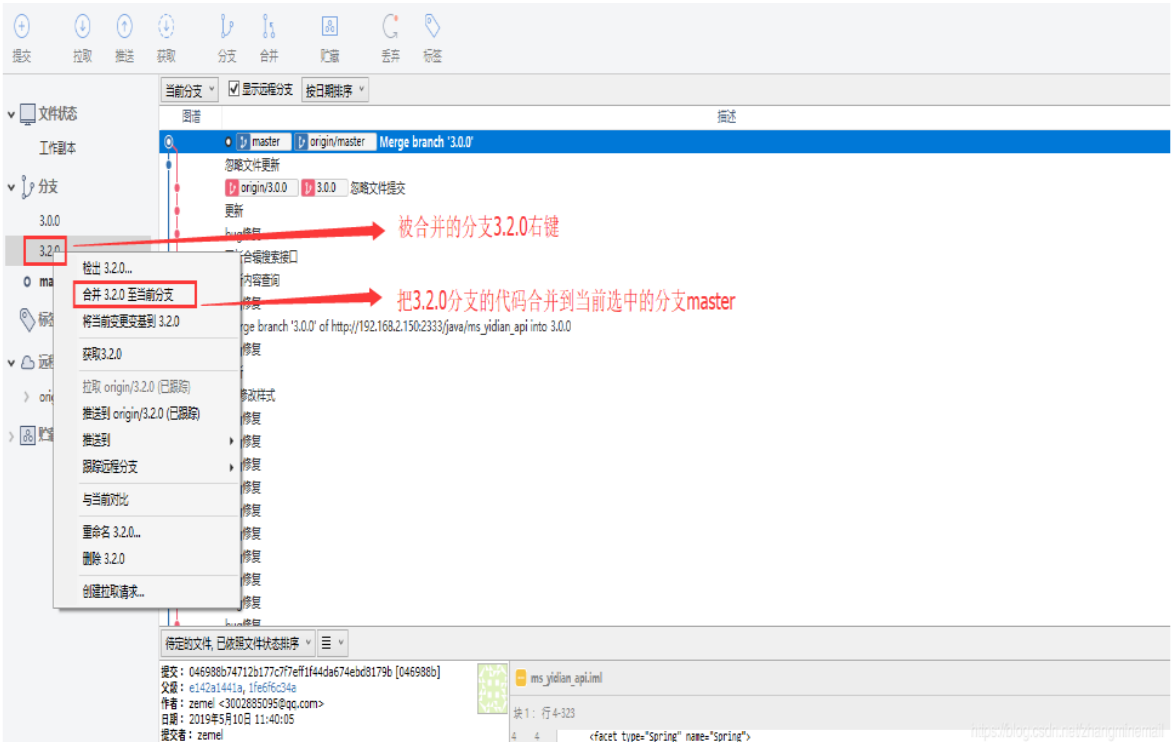


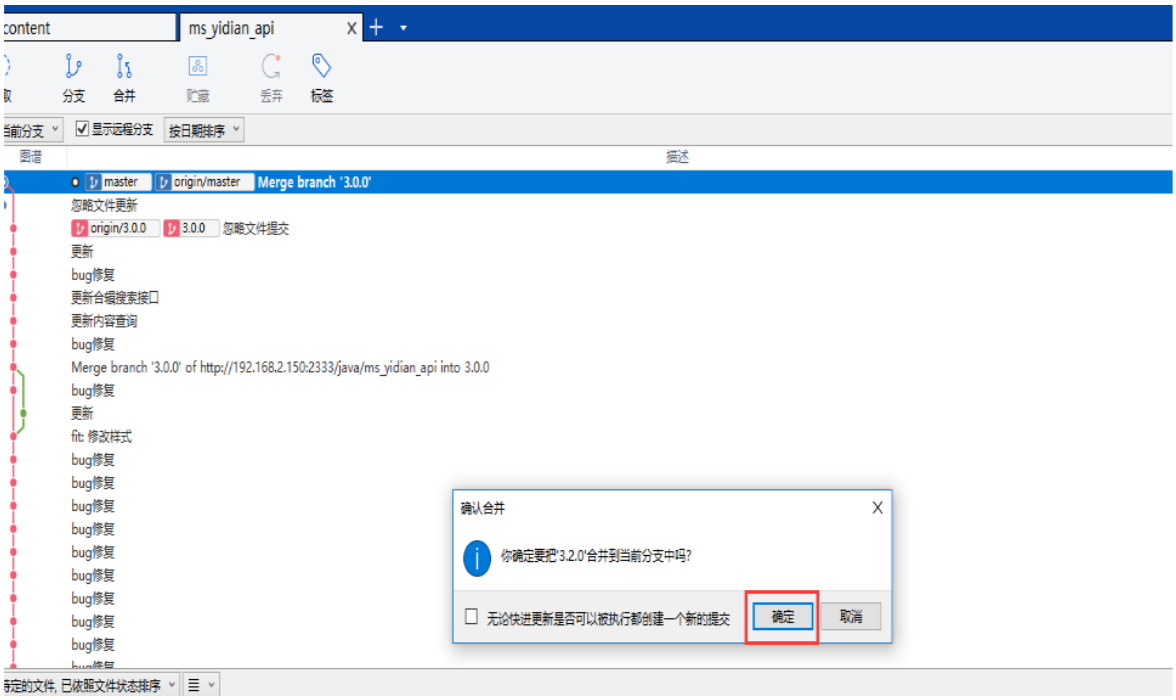


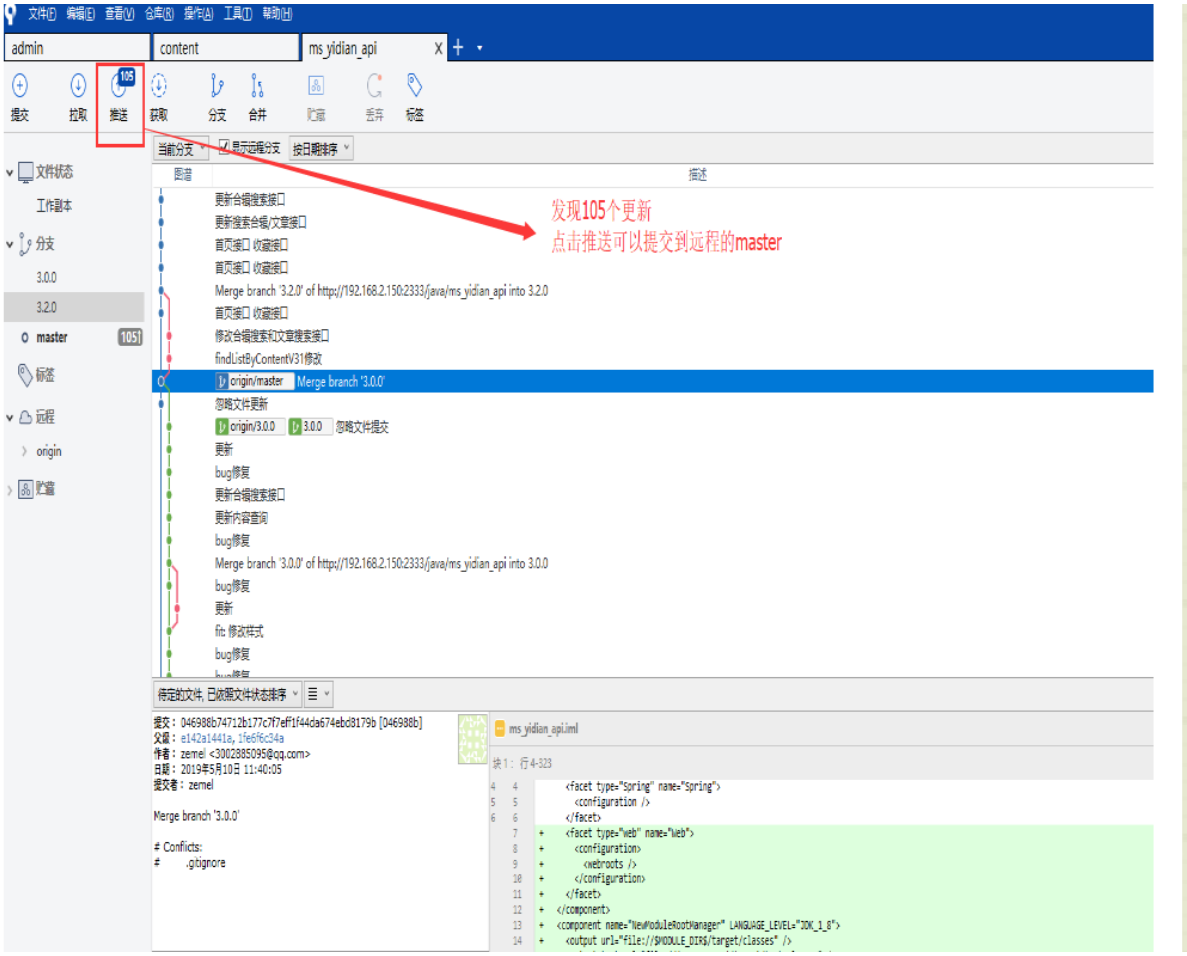
4、合并分支

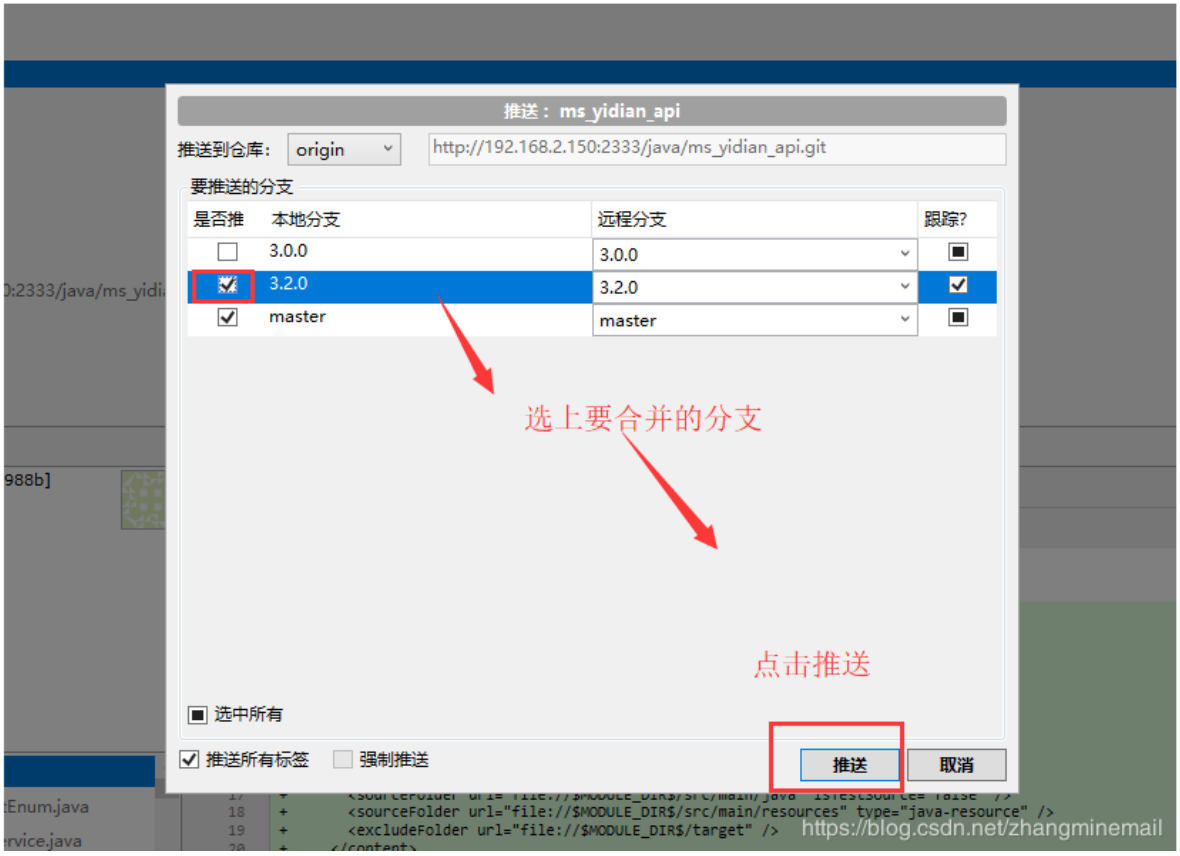


> 把【3.2.0】合并到【master】，分支切换到master


被合并的分支3.2.0右键
把3.2.0分支的代码合并到当前选中的分支master
检出 3.2.0...
合并 3.2.0 至当前分支
将当前变更基到 3.2.0
获取3.2.0
拉取 origin/3.2.0 (已跟踪)
推送到 origin/3.2.0 (已跟踪)
推送到
跟踪远程分支
与当前对比
重命名 3.2.0...
删除 3.2.0
创建拉取请求...
master
origin/master
Merge branch '3.0.0'
忽略文件更新
origin/3.0.0
3.0.0
忽略文件提交
更新
bug修复
更新合福搜索接口
更新内容查询
Merge branch '3.0.0' of http://192.168.2.150:2333/java/ms_yidian_api into 3.0.0
fit: 修改样式


ms_yidian_api
master
origin/master
Merge branch '3.0.0'
忽略文件更新
origin/3.0.0
3.0.0
忽略文件提交
更新
bug修复
更新合福搜索接口
更新内容查询
bug修复
Merge branch '3.0.0' of http://192.168.2.150:2333/java/ms_yidian_api into 3.0.0
bug修复
更新
fit: 修改样式
确认合并
你确定要把'3.2.0'合并到当前分支中吗?
无论快进更新是否可以被执行都创建一个新的提交
确定
取消

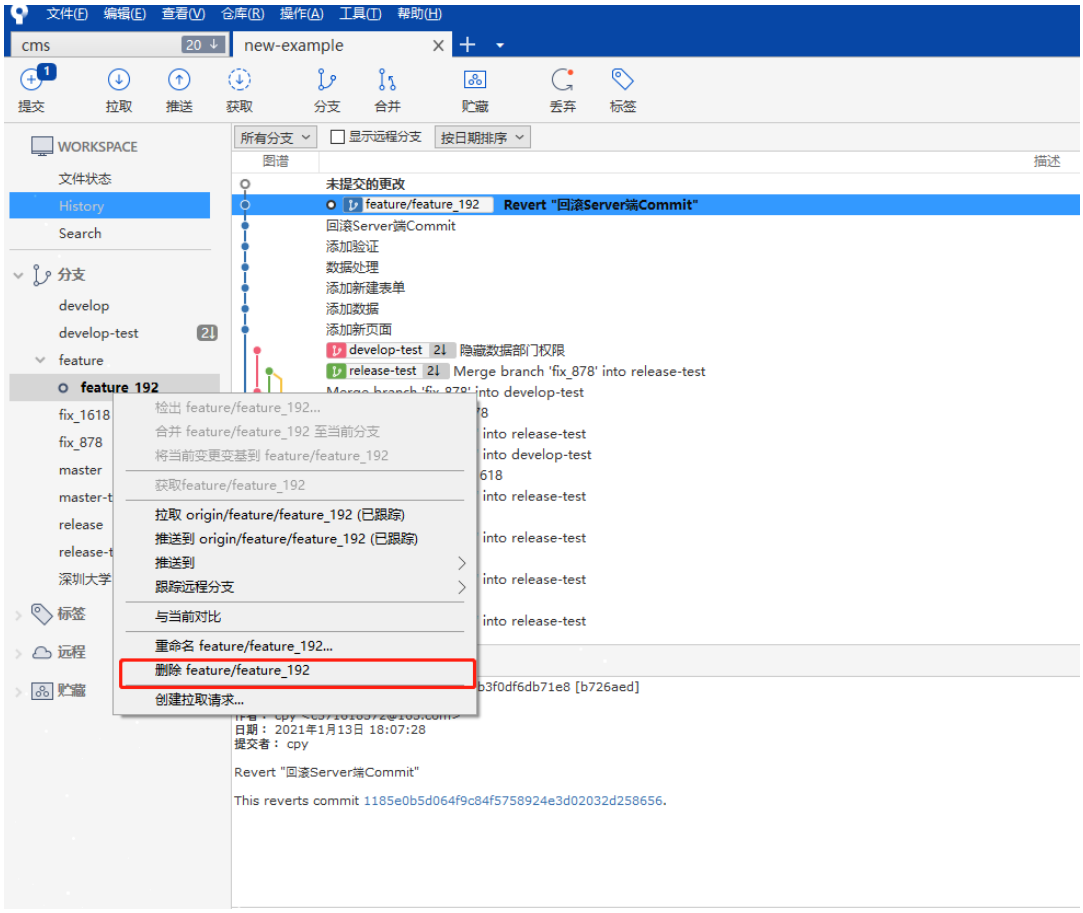

admin
content
ms_yidian_api
提交
拉取
推送
获取
分支
合并
标签
发现105个更新
点击推送可以提交到远程的master


推送： ms_yidian_api
推送到仓库:
origin
http://192.168.2.150:2333/java/ms_yidian_api.git
要推送的分支
是否推
本地分支
远程分支
跟踪?
3.0.0
3.2.0
master
选上要合并的分支
点击推送
选中所有
推送所有标签
强制推送
推送
取消

5、删除分支

- **本地删除**

> 右键目标分支，删除即可
>
> 注意：这里是本地删除，不影响远程仓库。如果切换目标分支时，不能进行删除操作。



- **远程删除**

> 右键目标分支，删除即可。【慎重】

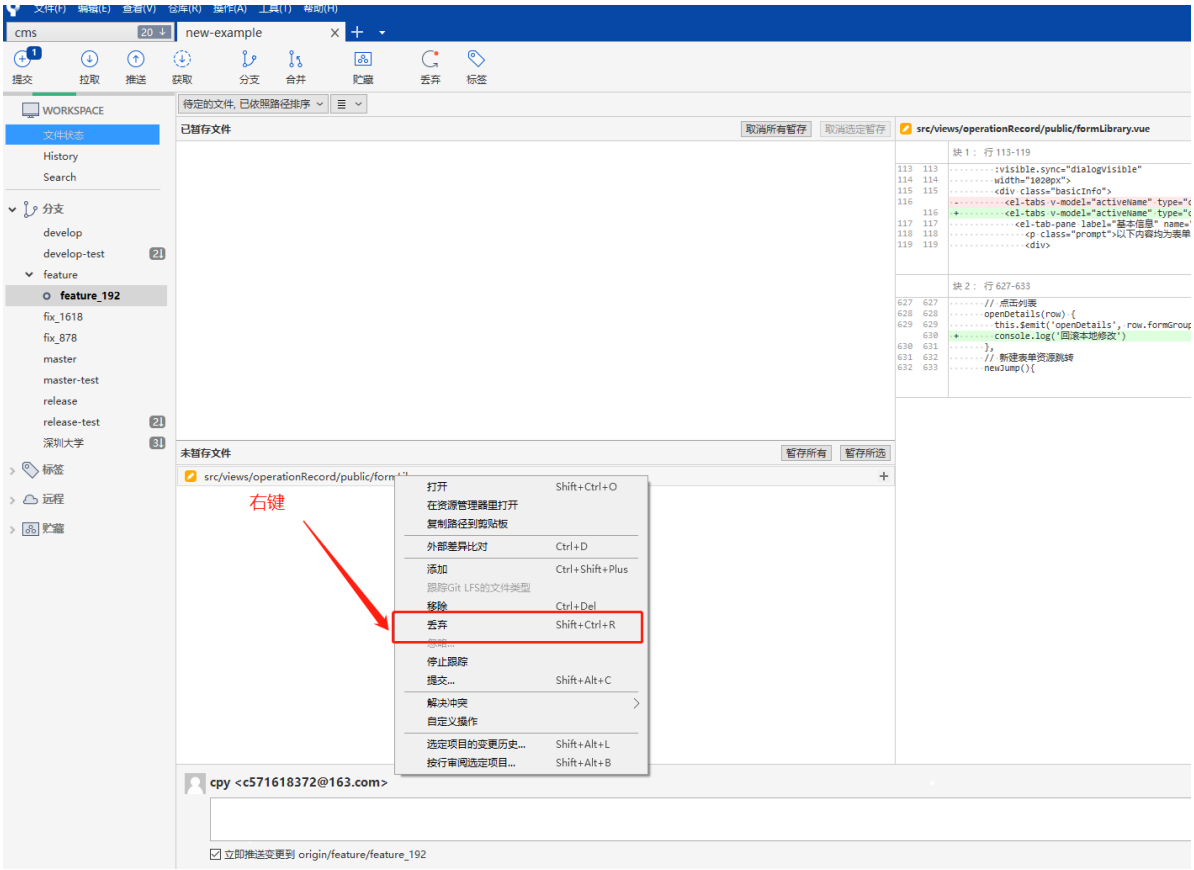
6、回滚代码

这里的回滚大概分为三种：

- 回滚本地修改。（此时还未进行本地Commit）
- 回滚本地的Commit。（此时还未推送到远程Git服务器）
- 回滚已经推送到Server端的Commit。

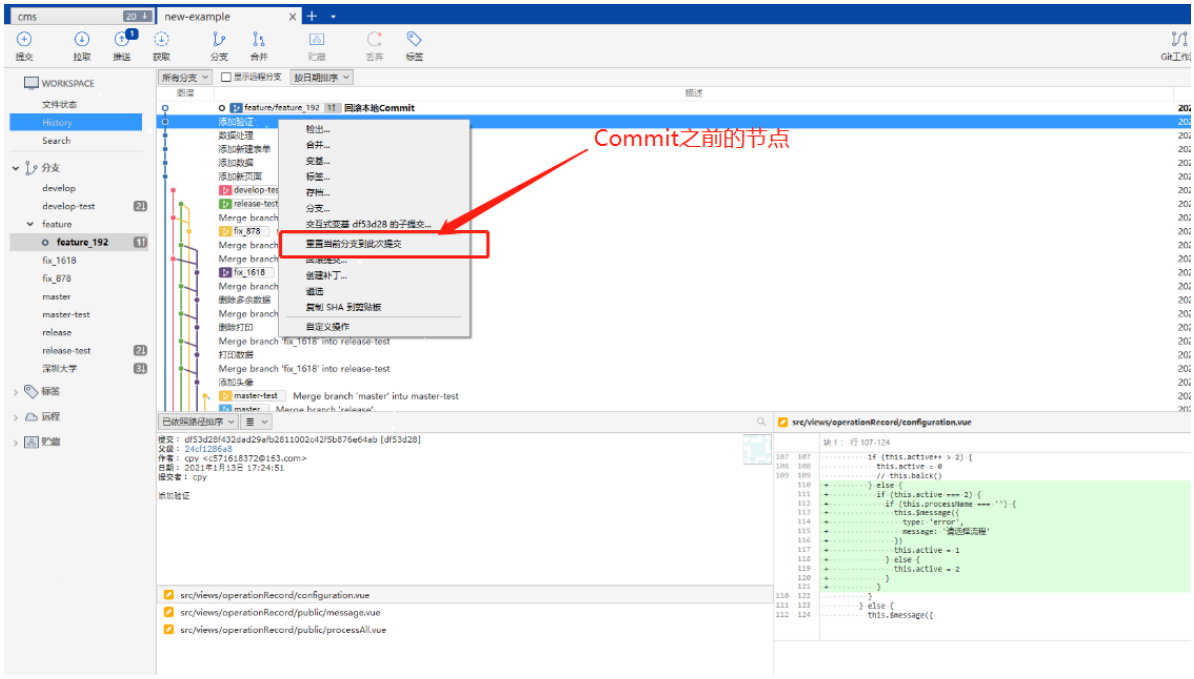
6.1 回滚本地修改

> 只需要在未暂存区选中对应的文件，右键选择"丢弃"即可

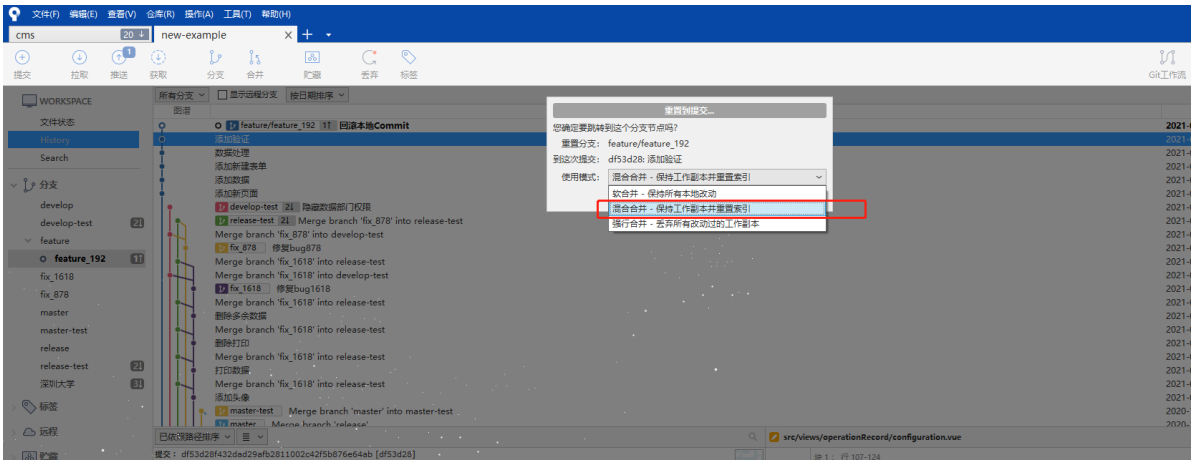


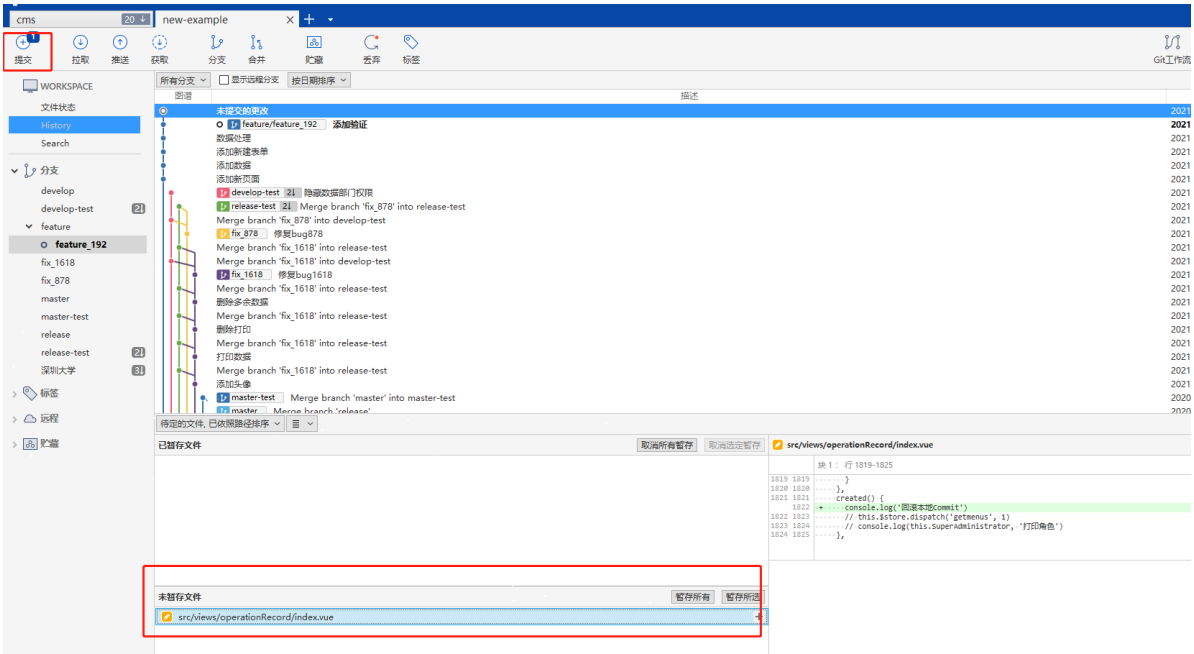
6.2回滚本地Commit

> 需要选取所要回滚Commit之前的那个Commit节点



> 弹出的交互框中，三个选项中推荐直接使用默认的混合合并，然后点击确认即可



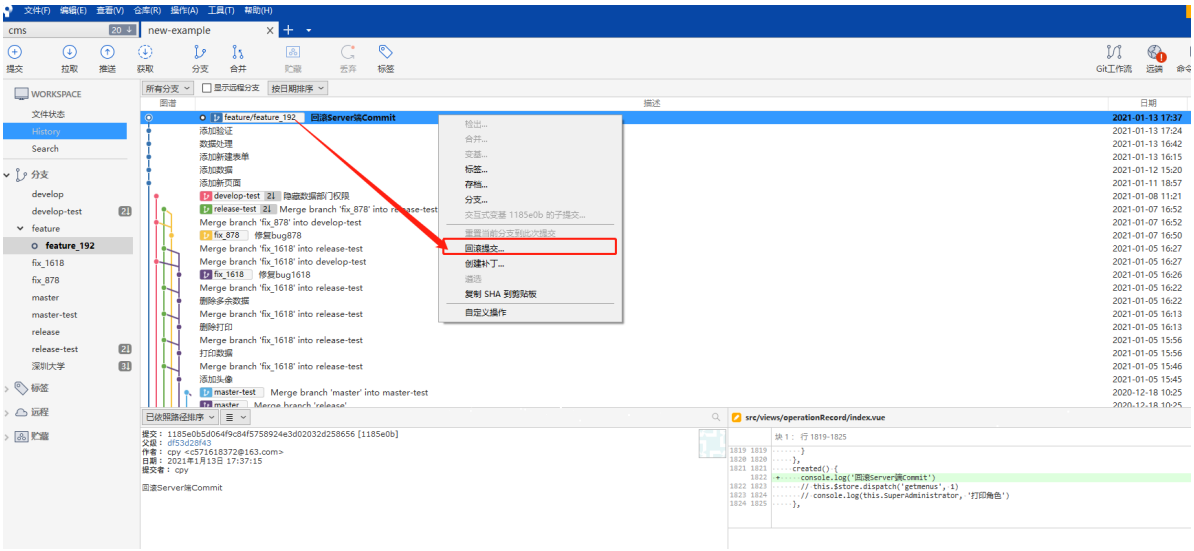


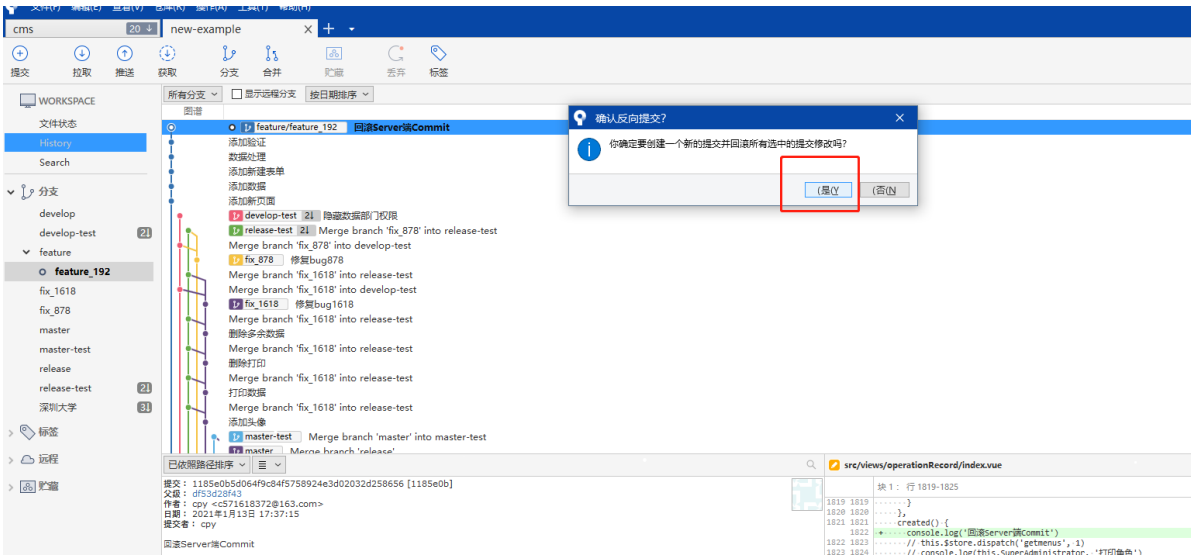
使用模式说明

选项	含义
软合并	回退到暂存区
混合合并	回退到未暂存区（默认）
强行合并	直接把提交的文件reset （最好不要用）

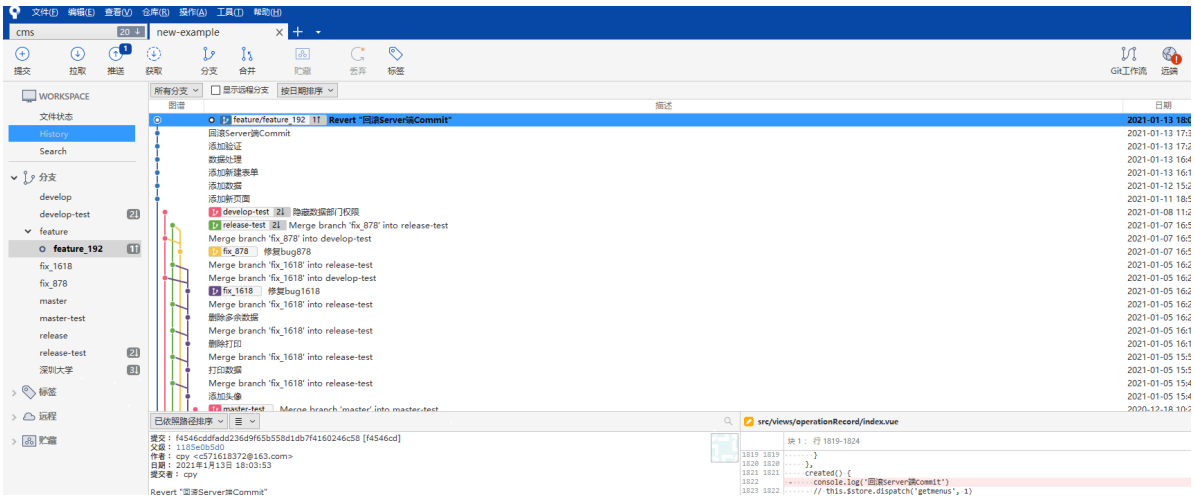
6.3回滚Server端Commit

> 首选需要选中想要进行回滚的Commit节点，如下：



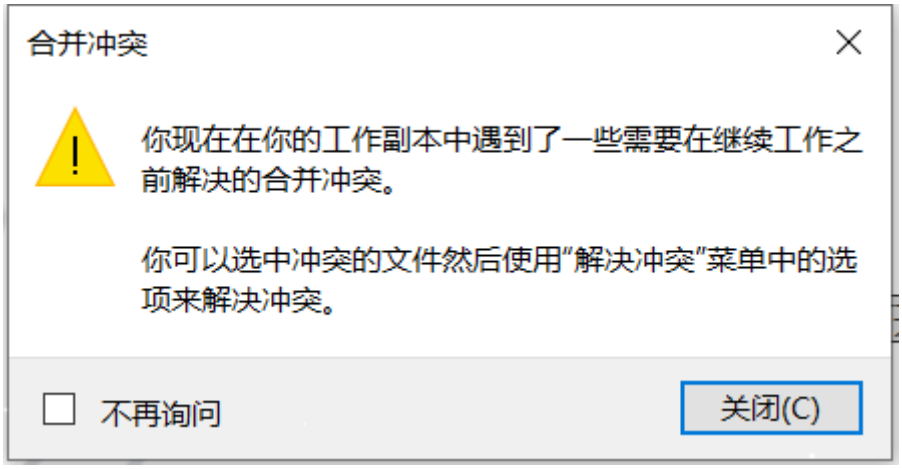


> 确认之后将自动生成类似下图中的Commit，然后推送即可



7、解决冲突

由于某些原因发生代码冲突的问题，这类问题在Sourcetree中表现如下：



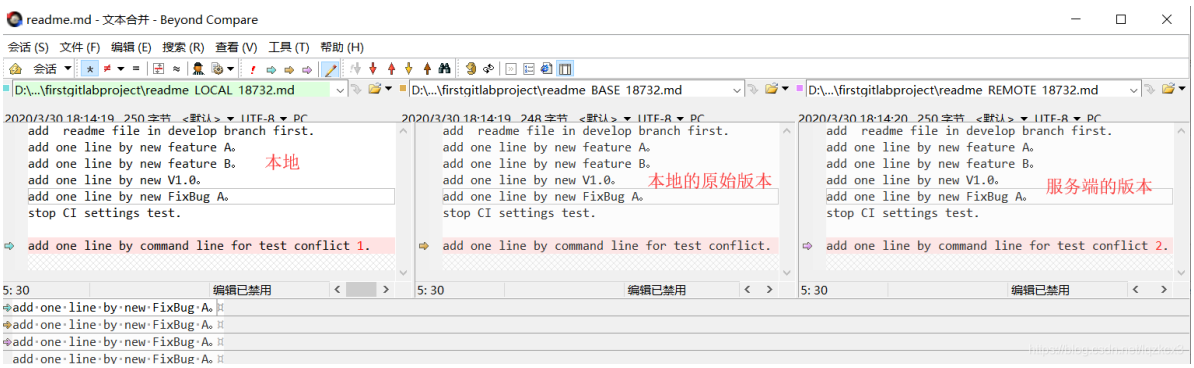
解决方法：

1）本地代码中手动解决；

2）外部合并工具：Beyond Compare

下载地址：**https://www.beyondcompare.cc/**

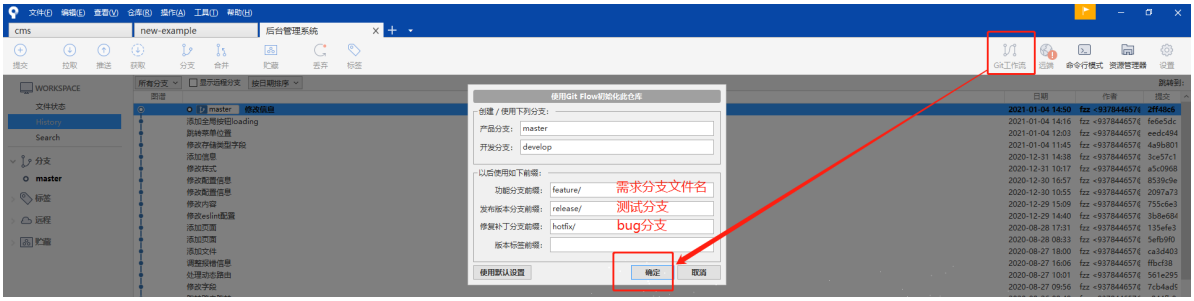
右键选中发生冲突的文件，最终选择打开外部合并工具，解决完冲突之后，再次提交文件即可。



8、git工作流

> 如果想要开发新的需求，或者修复bug，可使用该工具git工作流。

1）需要对项目流进行初始化



直接点“确定”，获取develop分支源码

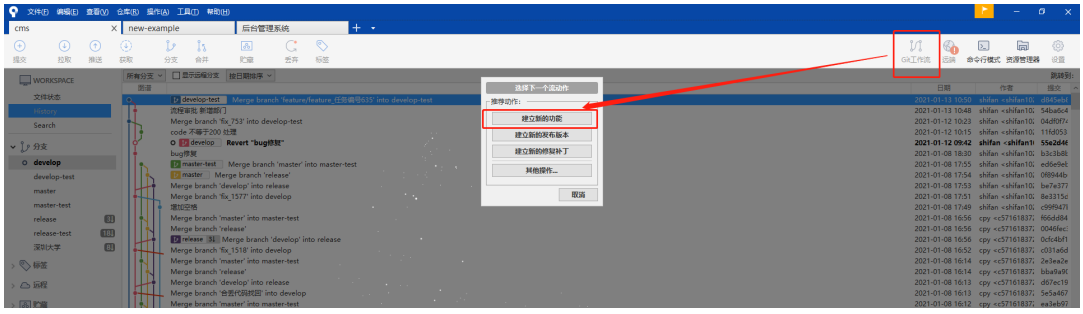
2）分支共有5种类型

- master，最终发布版本，整个项目中有且只有一个
- develop，项目的开发分支，原则上项目中有且只有一个
- feature，功能分支，用于开发一个新的功能
- release，预发布版本，介于develop和master之间的一个版本，主要用于测试
- hotfix，修复补丁，用于修复master上的bug，直接作用于master

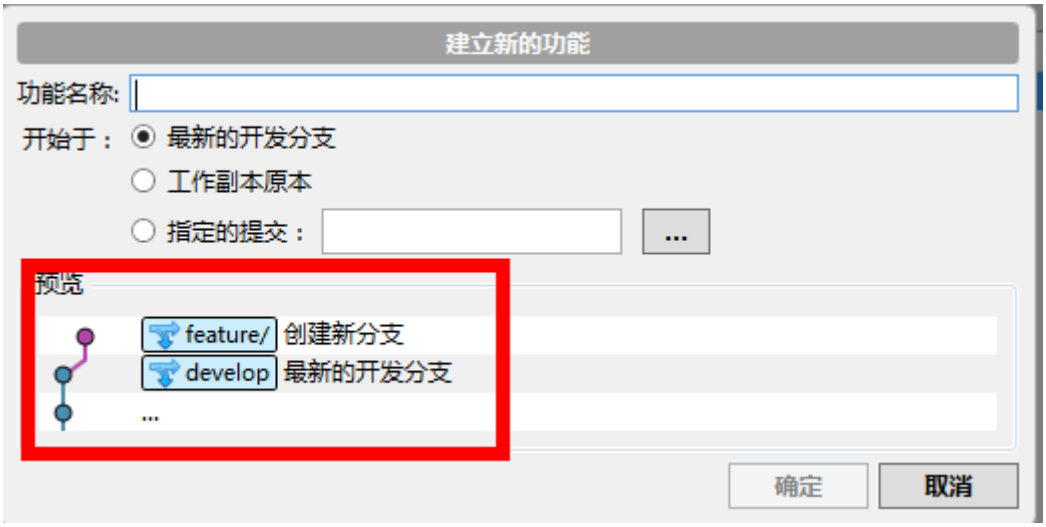
3）需求分支feature

> 当开发中需要增加一个新的功能时，可新建feature分支，用于增加新功能，并且不影响开发中的develop源码，当新功能增加完成后，完成feature分支，将新功能合并到develop中，更新develop上的代码

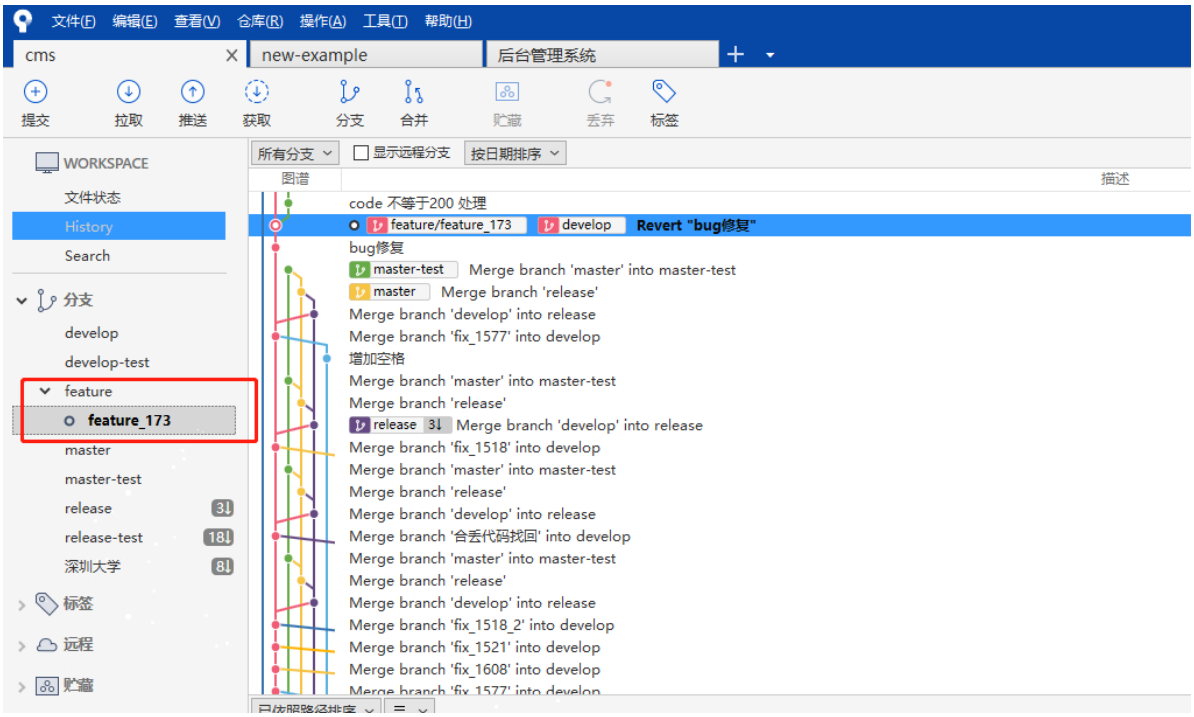
- 新建feature。首先当前开发分支指向develop，点击“Git工作流”



- 选择“建立新的分支”

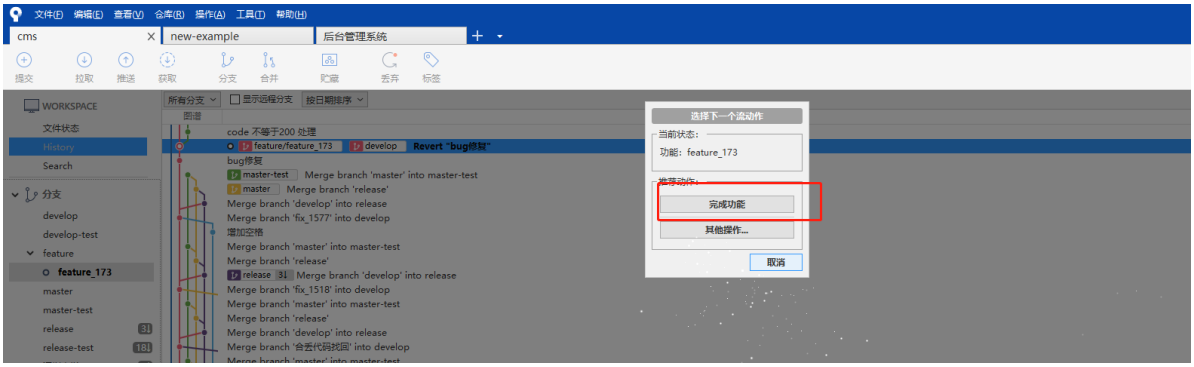


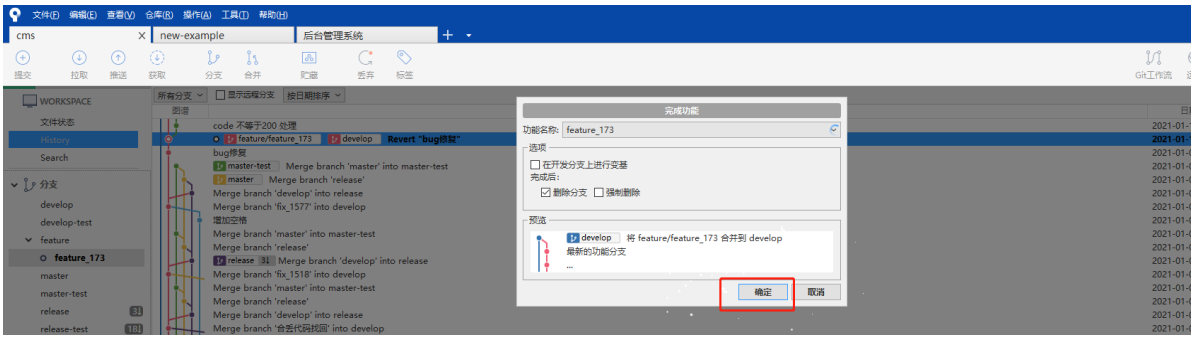
> 在预览中可看到，feature分支是从develop分出的，输入功能名称，点击确定，项目结构中增加feature分支，并且当前开发分支指向新建的feature分支，在该分支下进行开发任务，并提交



- 成feature开发后，将feature中的源码合并到develop分支，点击“Git工作流”，选择“完成功能”，点击确定，进行提交合并

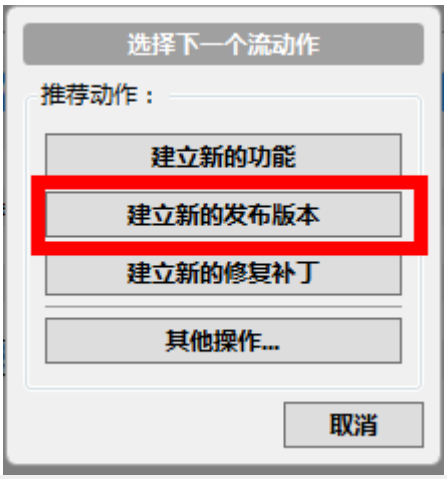
> 当多人协作开发时，可能会出现，不同人员对同一文件进行操作，从而引起合并冲突，对这种情况进行模拟，在当前新建两个feature，分别对feature文件进行修改，然后分别合并

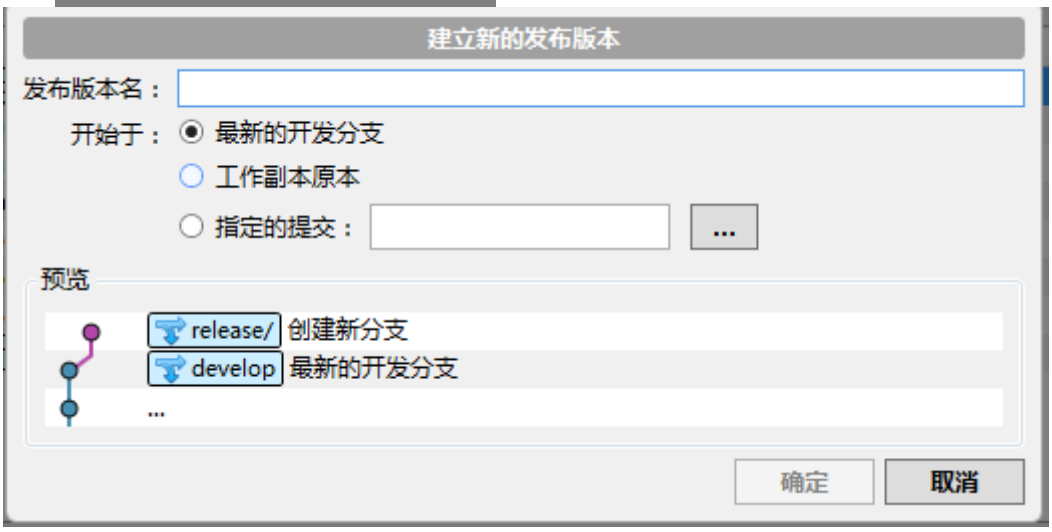




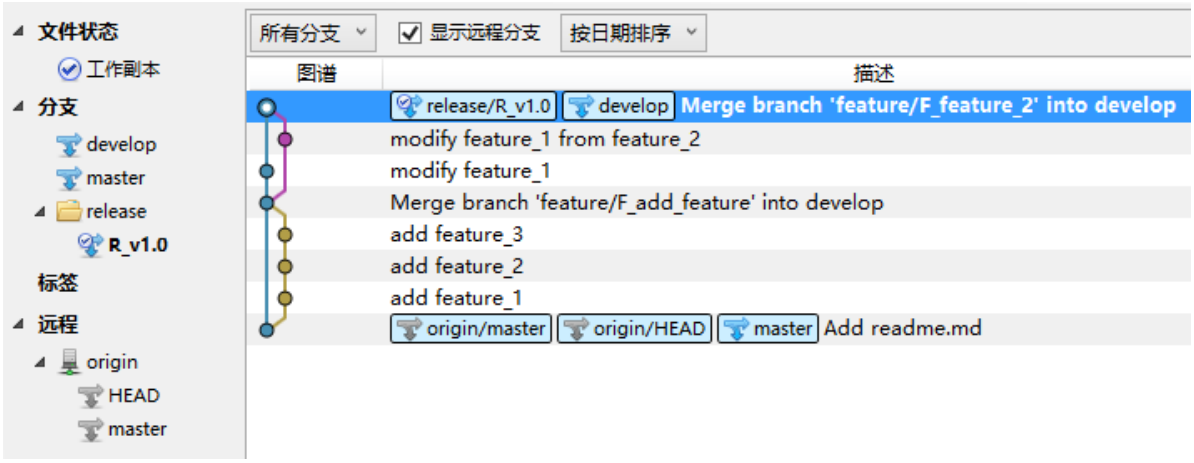
4） 预发布测试阶段

> 可以从develop分支，建立release分支，进入预发布测试阶段。点击“Git工作流”，选择“建立新的发布版本”





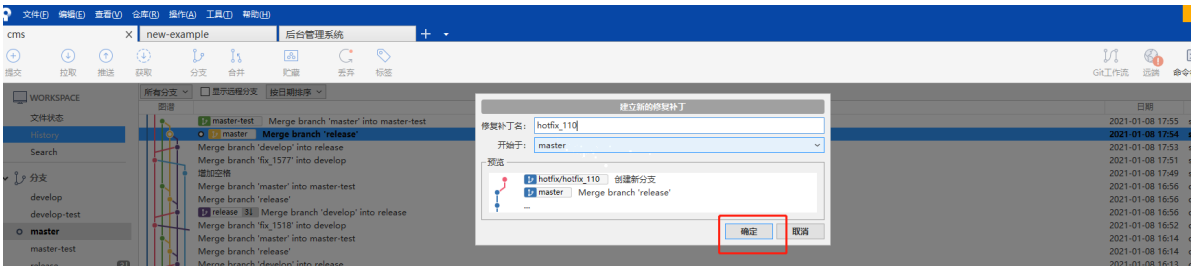
> release是从develop分出的，输入发布版本名‘R_v1.0’，点击确定



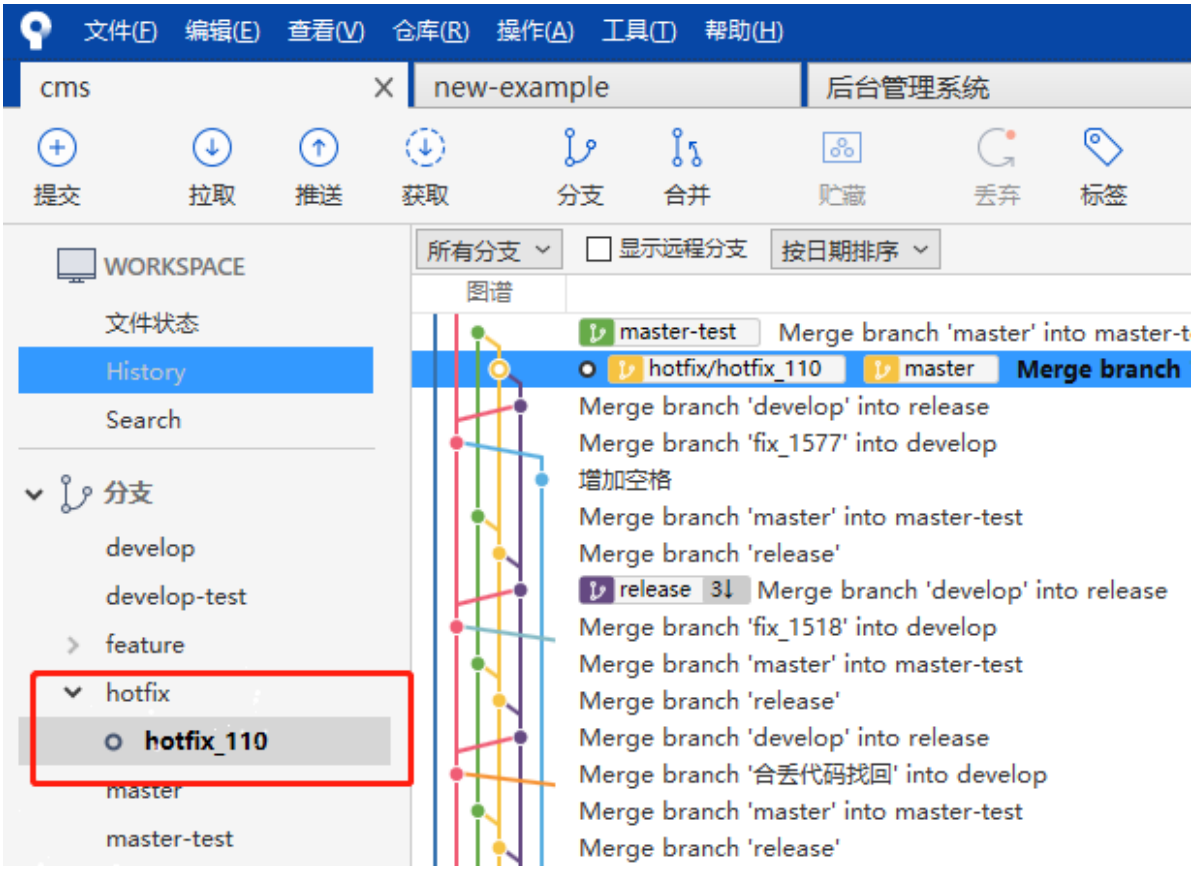
`R_v1.0为阶段性发布版本，主要用于发布前进行测试，后续的开发工作仍旧在develop上进行，如果在测试过程中发现问题，直接在release上进行修改，修改完成后进行提交`

5) 补丁hotfix

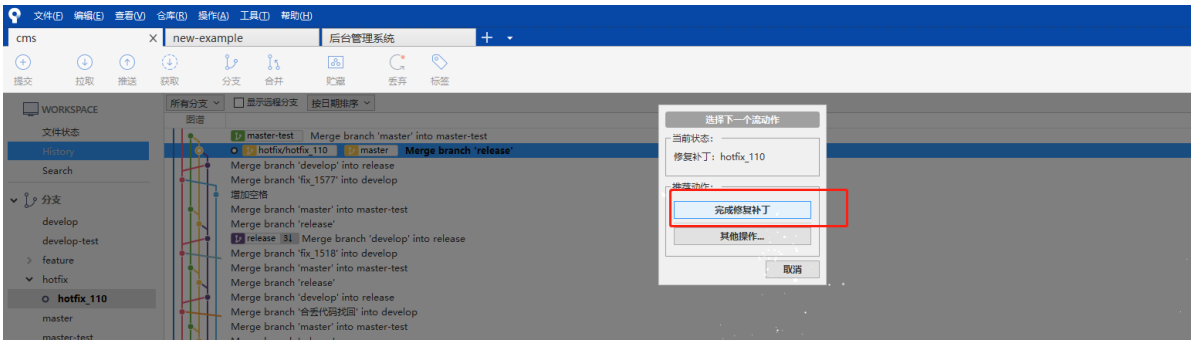
> 当正式版本出现问题时，需要进行问题的修改，可以在master分支建立修改补丁hotfix。将当前分支切换到master，点击“Git工作流”，选择“建立新的修复补丁”

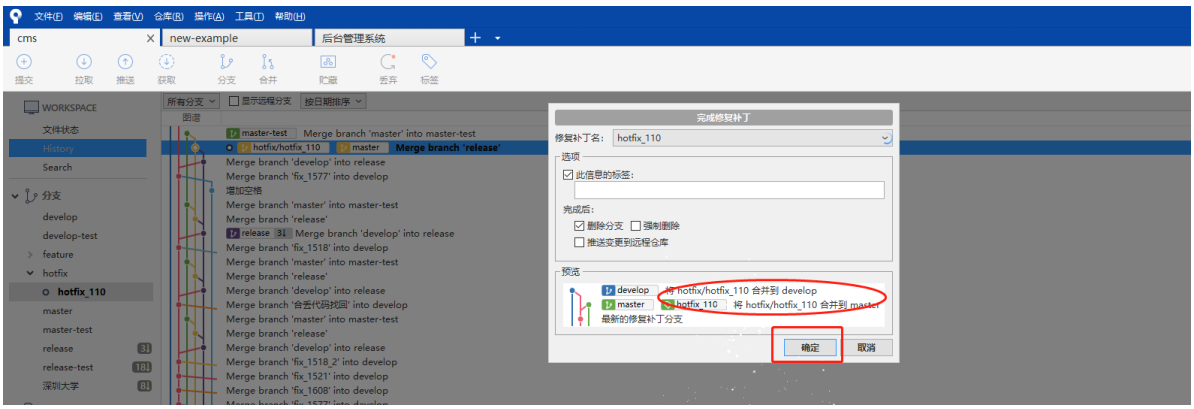


> 点击确认即可



修复完成后，可向master和develop分别合并，点击确定，完成分支合并





补充：从远程拉取分支，在该分支下完成开发，应“双击”远程的该分支，出现“检出新分支”，之后点击“确定”，就将远程分支拉取到本地了。

至此，版本管理git回滚、重置、提交以及SourceTree复杂操作全部完成！！！