**Gitlab 协同开发及代码评审**

[1 案例简介 3](#_Toc519257493)

[1.1 案例场景摘要 3](#_Toc519257494)

[1.2 gitlab账号 4](#_Toc519257495)

[2 Gitlab 分支的协同开发 5](#_Toc519257496)

[2.1 将本地的工程代码放在gitlab上托管 5](#_Toc519257497)

[2.1.1 先在gitlab上创建一个工程 spring-boot-redis 5](#_Toc519257498)

[2.2.2 Git Bash 至本地工程路径，初始化工程，新增代码，提交 5](#_Toc519257499)

[2.2.3 向远程增加一个origin 资源库（默认master分支） 5](#_Toc519257500)

[2.2.4 将提交push到远程origin资源库的master分支下面 5](#_Toc519257501)

[2.2 基于创建master分支创建一个develop分支 6](#_Toc519257502)

[2.2.1 Eclispe Git 克隆项目到本地 6](#_Toc519257503)

[2.2.2 Eclispe Git视图的资源库中选取Git Repository 7](#_Toc519257504)

[2.2.3创建一个标签 7](#_Toc519257505)

[2.2.4将标签推送至远程 8](#_Toc519257506)

[2.2.5 Gitlab上查看标签 10](#_Toc519257507)

[2.2.6 基于master分支创建 develop分支 10](#_Toc519257508)

[2.2.7 将新的develop分支推送至远程仓库origin的develop分支下 11](#_Toc519257509)

[2.3多用户基于功能创建特性分支 12](#_Toc519257510)

[2.3.1 用户创建各自的特新分支feature#61, feature#62 12](#_Toc519257511)

[2.3.1.1创建新的特新分支 12](#_Toc519257512)

[2.3.3.2 推送至远程 origin资源库的feature#61 分支下 13](#_Toc519257513)

[2.3.2 feature#62 开发完功能后，向远程提交更新并在gitlab提交合并请求 13](#_Toc519257514)

[2.3.2.1 feature#62代码评审后提交更新 13](#_Toc519257515)

[2.3.2.2 gitlab 提交申请合并feature#62 到develop分支 17](#_Toc519257516)

[2.3.3 合并操作 18](#_Toc519257517)

[2.3.3.1 gitlab完成代码评审分支合并 18](#_Toc519257518)

[2.3.3.2 git bash 命令行完成feature#62与develop 分支合并 19](#_Toc519257519)

[2.3.3.3 评审留痕 20](#_Toc519257520)

[2.3.4 lissa 用户从最新的 develop 分支拉一个特性分支feature#63 22](#_Toc519257521)

[2.3.5 Lissa开发完功能后开始更新提交 22](#_Toc519257522)

[2.3.6 在 Gitlab 上请求与develop分支合并 23](#_Toc519257523)

[2.3.7 创建合并请求之前比较 23](#_Toc519257524)

[2.3.8 合并feature63#分支到develop主干分支上 25](#_Toc519257525)

[2.3.9 feature#61 与最新develop 分支合并 27](#_Toc519257526)

[3 发布第一个release 28](#_Toc519257527)

[3.1 在最新develop.18.07.30基础上创建一个新分支release,标签18.08.06 29](#_Toc519257528)

[3.2 推送至远程 29](#_Toc519257529)

[4 合并develop至master主分支 30](#_Toc519257530)

[5 Gitlab Issue（议题）追踪问题、发布任务、缺陷报告 32](#_Toc519257531)

[6 结语 33](#_Toc519257532)

|  |  |  |
| --- | --- | --- |
| 作者 | 版本 | 备注 |
| 姜鹏 | V0.1 | 初稿 |

# 1 案例简介

## 1.1 案例场景摘要

案例演示gitlab 上的三个用户协作开发的场景。

Step1:用户root将代码工程放在远程gitlab服务器托管并在origin 资源库新建master分支；

Step2:用户root在master分支上拉一个develop分支；

Step3:Gitlab用户取代码取代码到本地，在develop主干分支上开发;

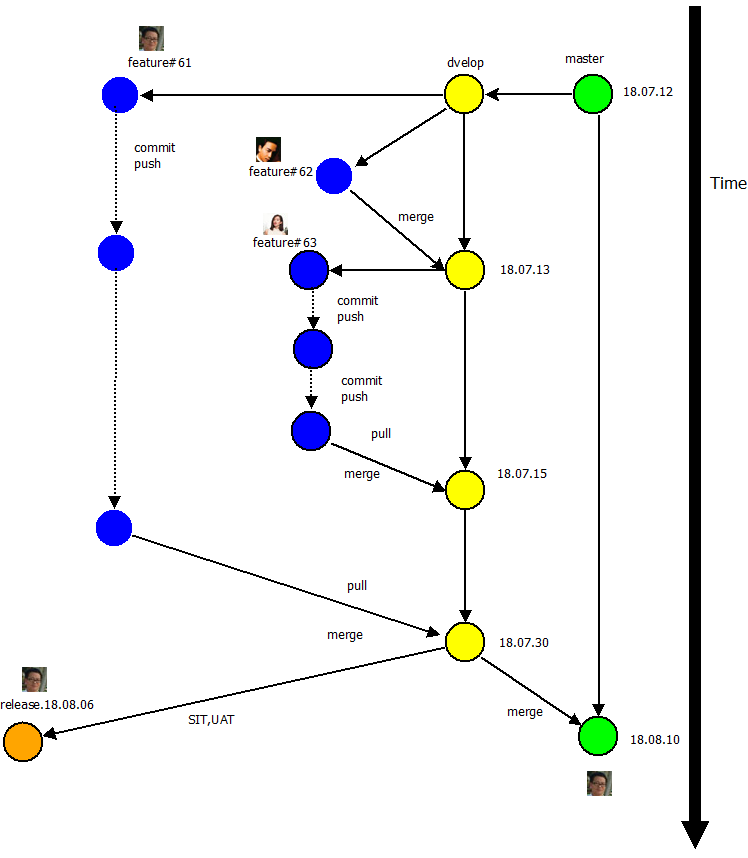
Step4:Gitlab用户 root 、jiangpeng、lissa分别在develop分支的基础上创建feature#61, feature#62, feature#63特性分支；

Step5:Gitlab用户 root 、jiangpeng、lissa先后将各自特性分支上开发完成的功能本地提交后并PUSH到远程的特性分支；

Step6:Gitlab用户 root 、jiangpeng、lissa先后切换到develop分支，将各自的特性分支合并到develop分支，Tag标记为18.07.30；

Step7: develop. 18.07.30分支，代码通过SIT,UAT测试后，在此分支上拉一个release 分支，Tag标记为18.08.06；

Step8: 将develop. 18.07.30分支合并到master分支，代码通过SIT,UAT测试后，标记为18.08.10；



## 1.2 gitlab账号

|  |  |  |  |
| --- | --- | --- | --- |
| 用户名 | 头像 | 角色 | Git分支 |
| root | C:\Users\Administrator\Desktop\pictures\me.jpg | 维护者 | feature#61, master, develop, release |
| jiangpeng | C:\Users\Administrator\Desktop\pictures\guorong.jpg | 开发人员 | feature#62 |
| lissa | C:\Users\Administrator\Desktop\pictures\lisa.jpg | 开发人员 | feature#63 |

# 2 Gitlab 分支的协同开发

## 2.1 将本地的工程代码放在gitlab上托管

### 2.1.1 先在gitlab上创建一个工程 spring-boot-redis

[**http://39.108.210.27/root/spring-boot-redis.git**](http://39.108.210.27/root/spring-boot-redis.git)

### 2.2.2 Git Bash 至本地工程路径，初始化工程，新增代码，提交

$git init

$git add .

$git commit -m 'init project'

### 2.2.3 向远程增加一个origin 资源库（默认master分支）

$ git remote add origin <http://39.108.210.27/root/spring-boot-redis.git>

查看远程分支是否OK

$git remote –v

出现如下提示：

origin http://39.108.210.27/root/spring-boot-redis.git (fetch)

origin http://39.108.210.27/root/spring-boot-redis.git (push)

### 2.2.4 将提交push到远程origin资源库的master分支下面

$git push -u origin master

查看分支，运行：

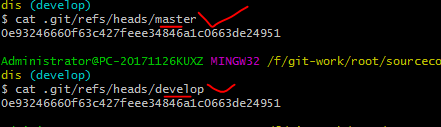
$git branch –a

****

Git Bash到工程目录查看头部信息：

$cat .git/refs/heads/develop

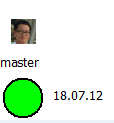
$cat .git/refs/heads/master



gitlab 上工程代码全部成功推送：



此时，Gitlab已有一个分支master

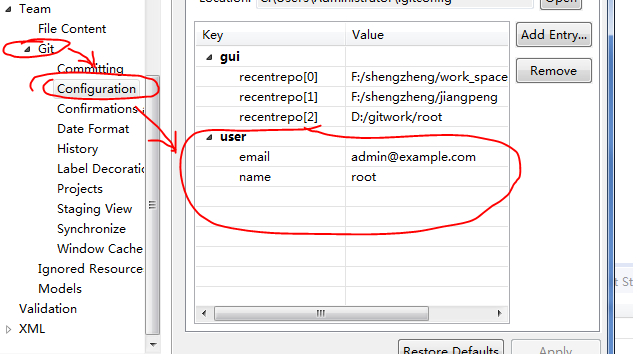


## 2.2 基于创建master分支创建一个develop分支

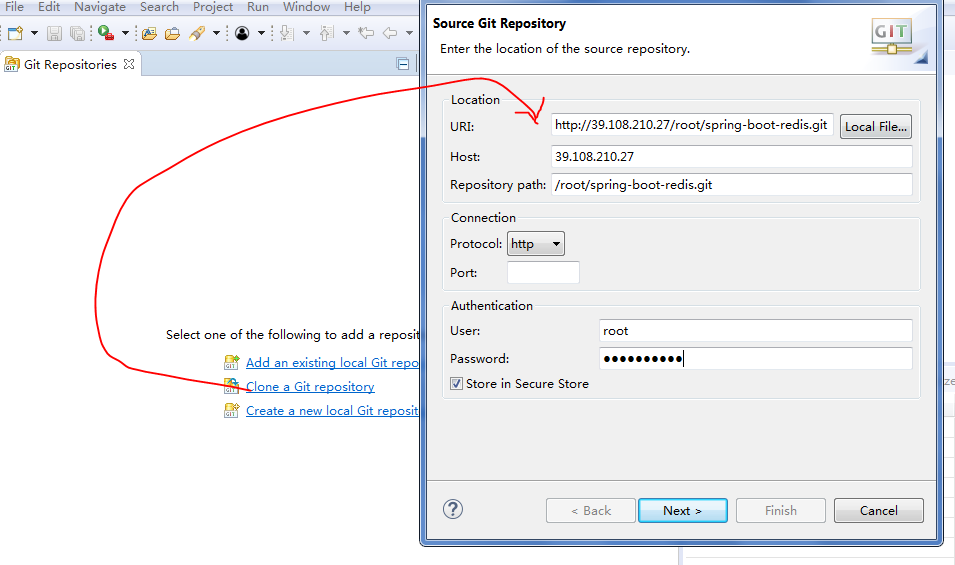
### 2.2.1 Eclispe Git 克隆项目到本地

先需要设置Gitlab的用户，如图所示：root用户也配置

在gitlab用户一致的账号和邮箱。

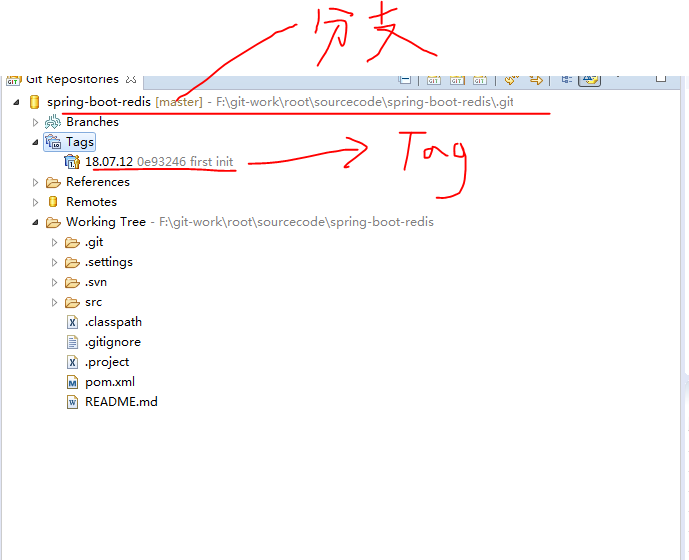
****

### 2.2.2 Eclispe Git视图的资源库中选取Git Repository

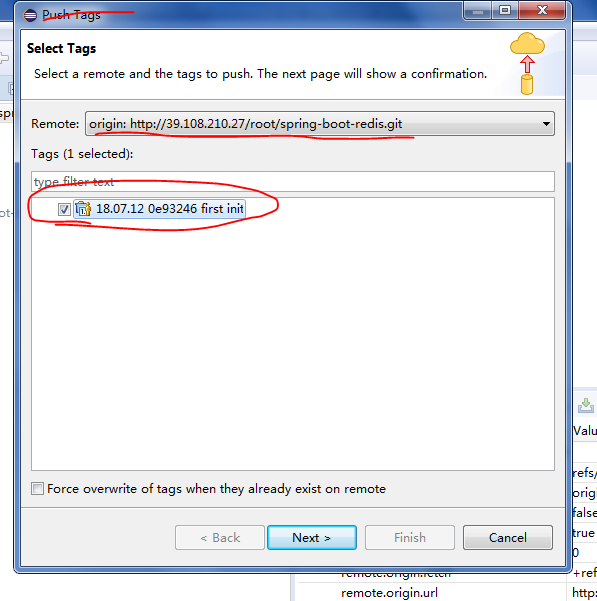
****

### 2.2.3创建一个标签

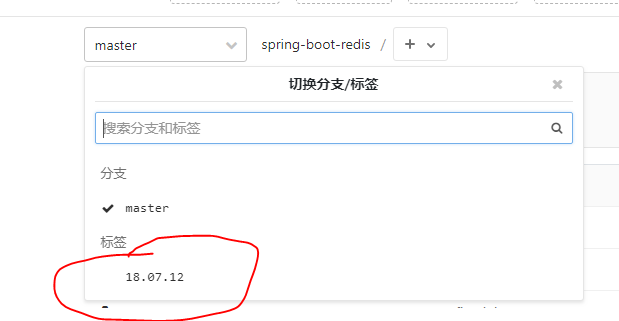
项目导入后，可以创建标签：

****

### 2.2.4将标签推送至远程

****

### 2.2.5 Gitlab上查看标签

****

### 2.2.6 基于master分支创建 develop分支

在新导入的项目位置运行： Git bash

查看当前的分支：

****

当前的分支为master

注意： 基于master分支创建develop分支，先要切换到master分支进行操作

$git checkout master

切换到master分支后，在运行：

$git checkout –b develop

也可以通过两个命令来实现：

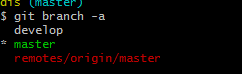
$git branch develop

$git checkout develop

查看分支：

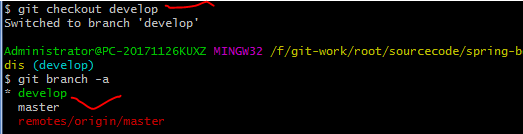
$git branch –a

发现本地develop 分支已创建好：

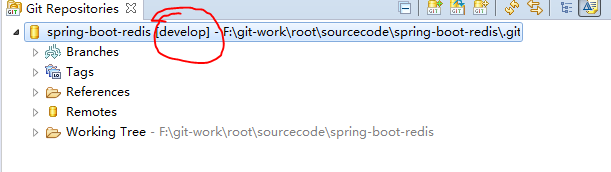
****

切换到develop分支

$git checkout develop

****

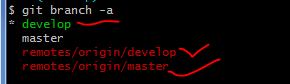
Eclipse 工程下的项目由master变成develop

****

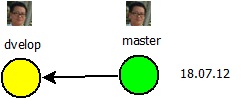
### 2.2.7 将新的develop分支推送至远程仓库origin的develop分支下

$git push origin develop

发现，远程分支多出一个develop分支

****

此时，分支状态是这样的：



## 2.3多用户基于功能创建特性分支

用户 root 基于develop分支创建feature#61；

用户 jiangpeng 基于develop分支创建feature#62；

用户jiangpeng开发完功能后，向develop合并；

用户 lisa基于develop分支创建feature#63；

用户lisa开发完功能后，向develop合并；

用户root开发完功能后，向develop合并，合并后develop最新分支develop18.07.30；

基于develop18.07.30拉(创建)一个分支做为发布release18.08.06；

### 2.3.1 用户创建各自的特新分支feature#61, feature#62

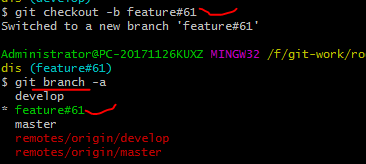
现以root用户为例：

先确保在develop分支的工作区间

$git checkout develop

#### 2.3.1.1创建新的特新分支

$git checkout –b feature#61

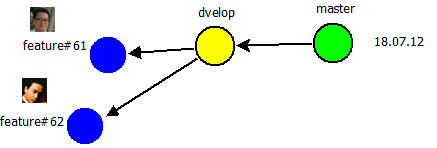


#### 2.3.3.2 推送至远程 origin资源库的feature#61 分支下

$git push origin feature#61

用户jiangpeng照以上步骤push feature#62到远程

此时的分支是状态是这样的：

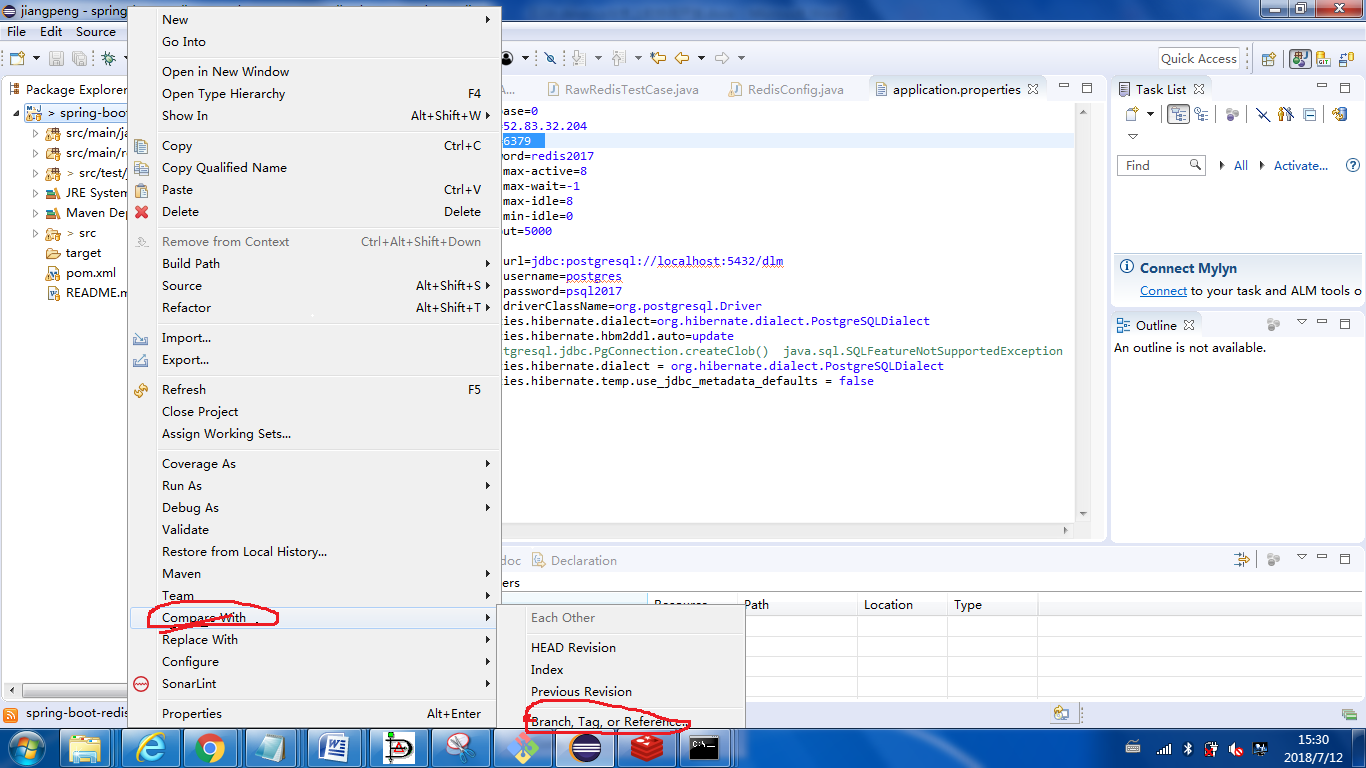


### 2.3.2 feature#62 开发完功能后，向远程提交更新并在gitlab提交合并请求

#### 2.3.2.1 feature#62代码评审后提交更新

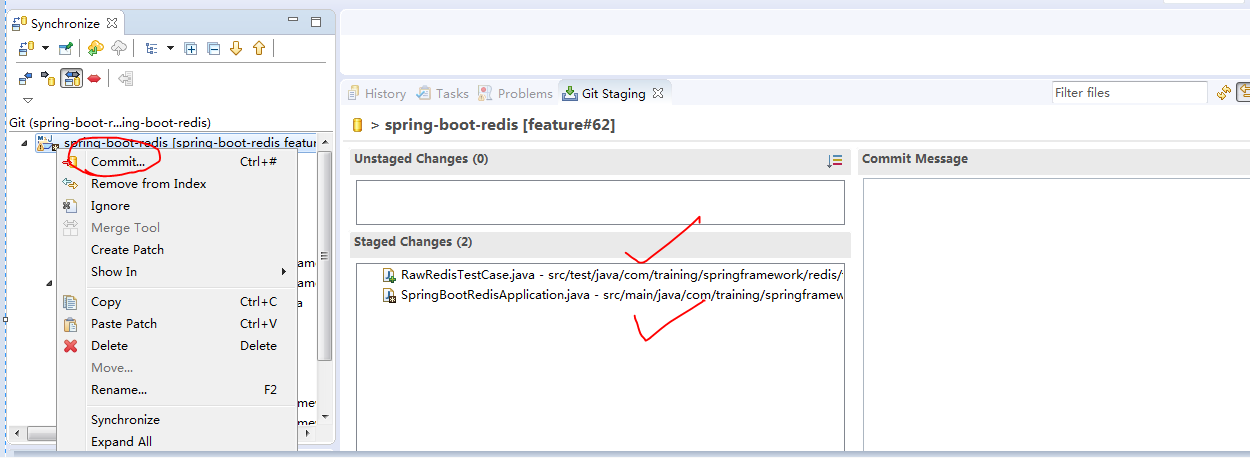
特新分支feature#62完成功能开发，采用一对一的方式交叉评审代码。在提交更新后填写固定模板的日志信息。日志信息记录本次评审情况。

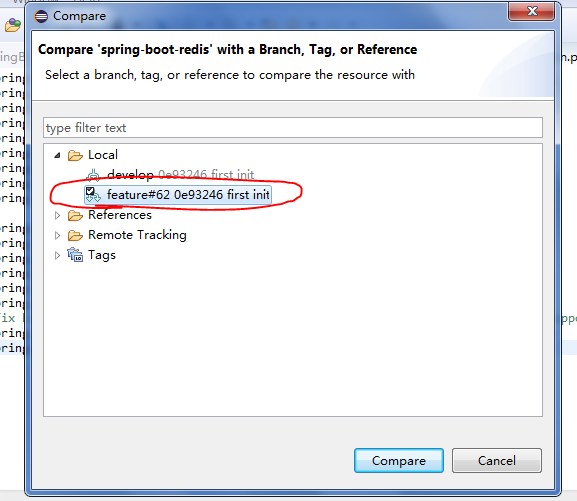
工程右键/Compare with/Branch , Tag or Reference:



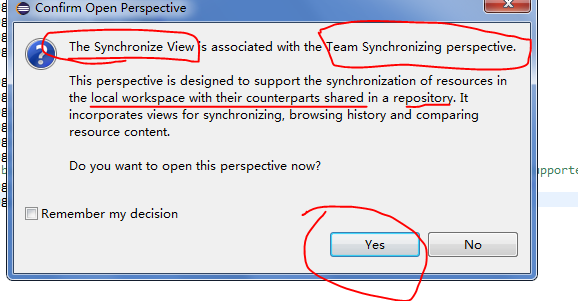
这儿选择： 与本地的 feature#62分支进行比较 ，可以比较自己本地的改动情况：

右键 Add to Index/ Commit 跳出如下界面：“staged changes”列表中为更新的所有文件列表。

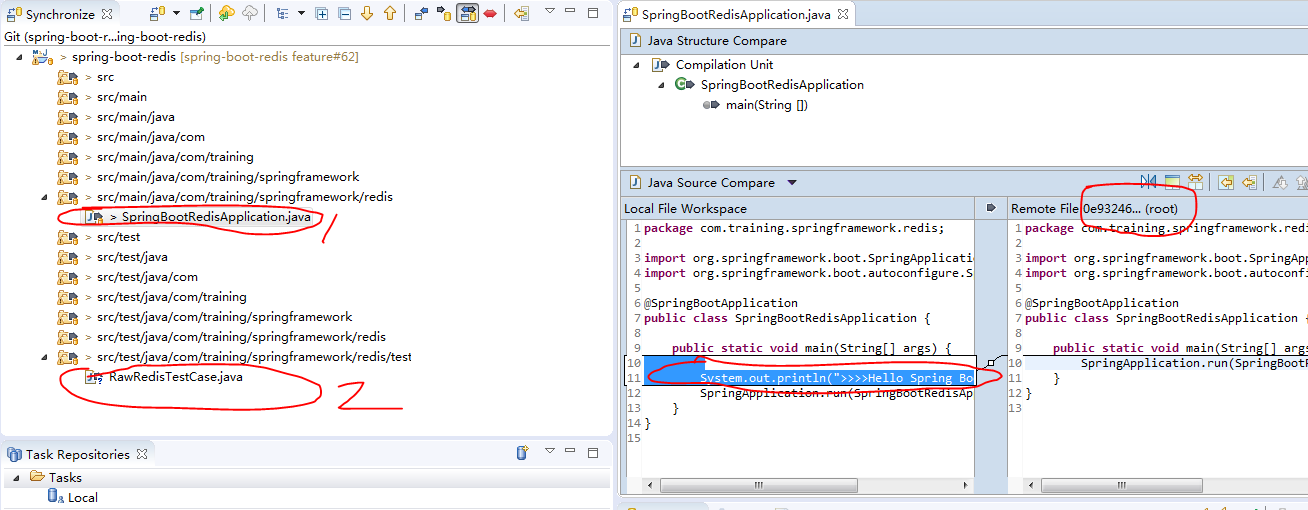




点 compare后， 将出现“同步视图”的窗口， 点击“YES”.

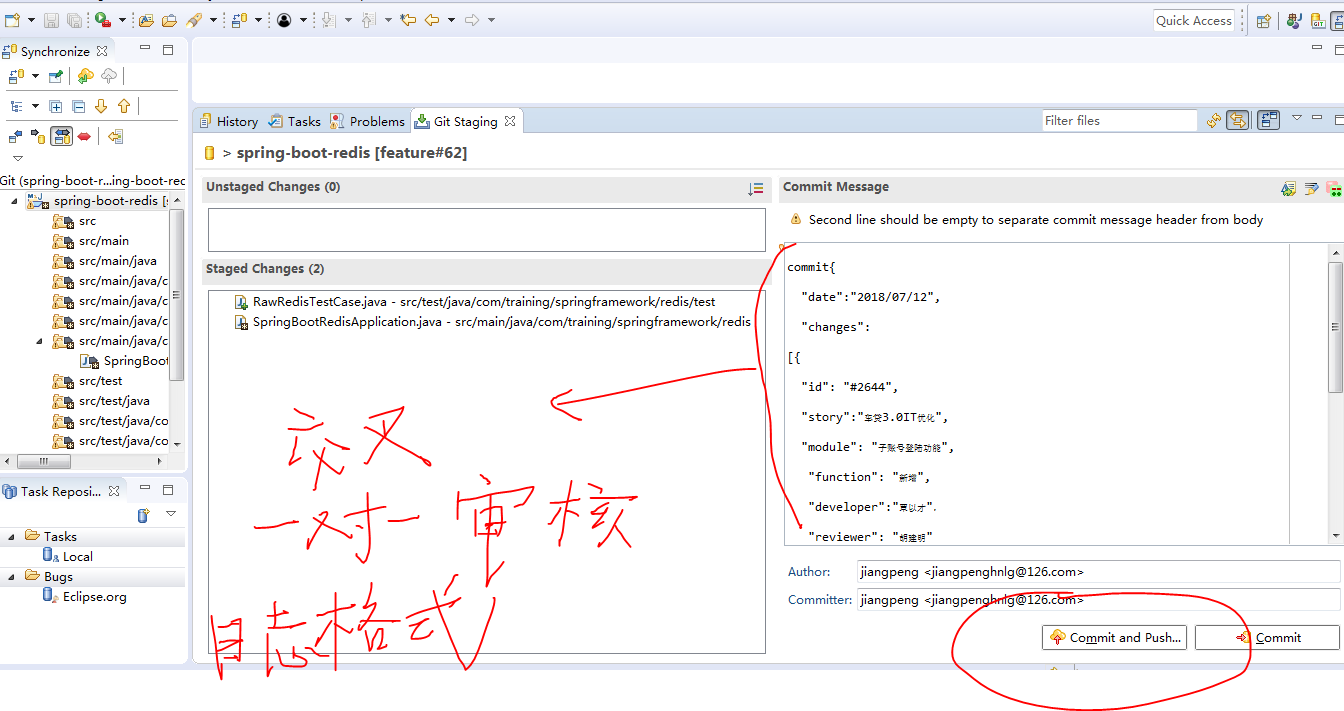


进入 ”Team 同步视图“ ， 可以清晰看到本地相对与远程 “develop”分支的改动情况。



代码审核无误后，在 “Commit Message” 文本框内输入固定格式的评审结果模板。

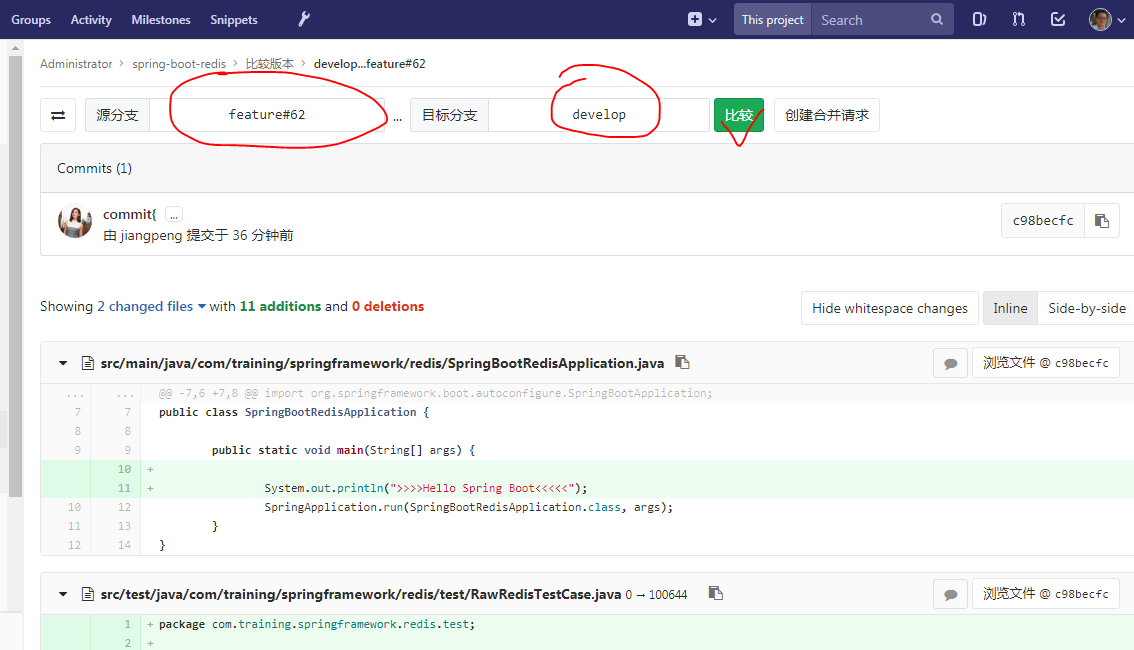
点“Commit and Push ” 提交并PUSH到远程feature#2 。

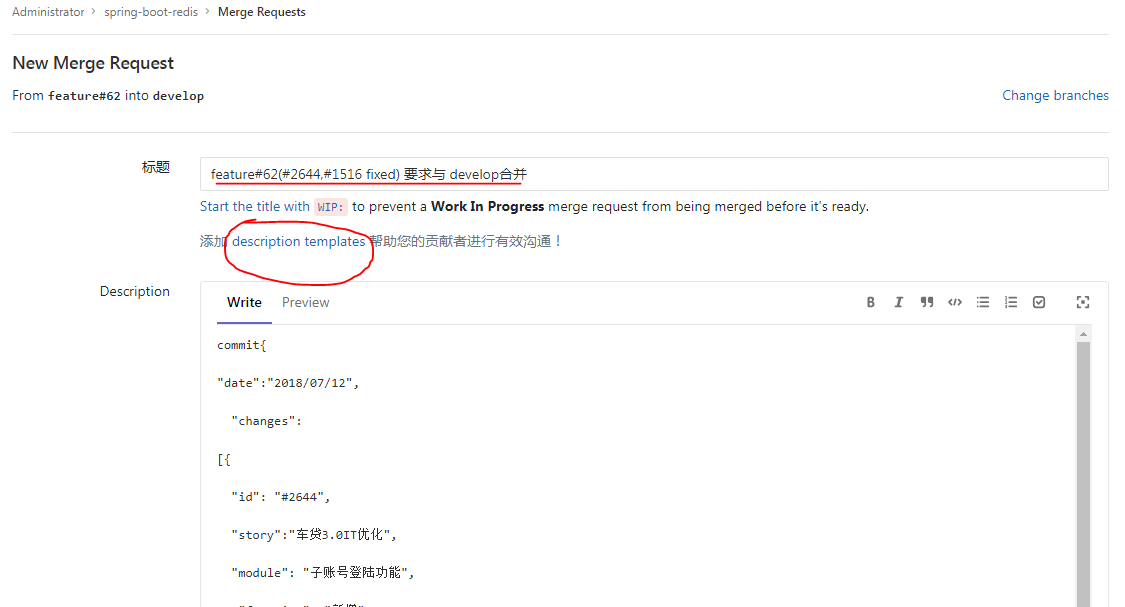


用户jiangpeng开发完功能后，将feature#62 与develop 合并。

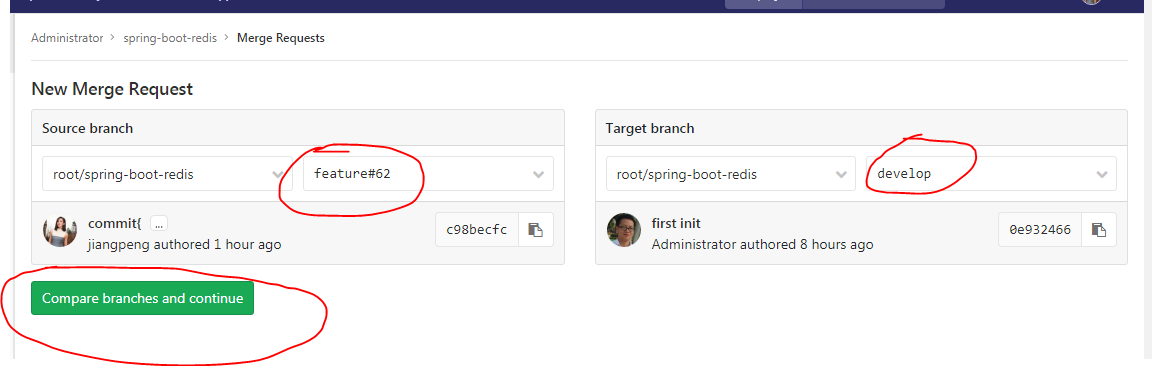
#### 2.3.2.2 gitlab 提交申请合并feature#62 到develop分支

合并前可以通过Gitlab 将 feature#62 分支与develop分支进行比较





可以创建“描述模板”来统一格式。



点击 “ 比较并继续“。

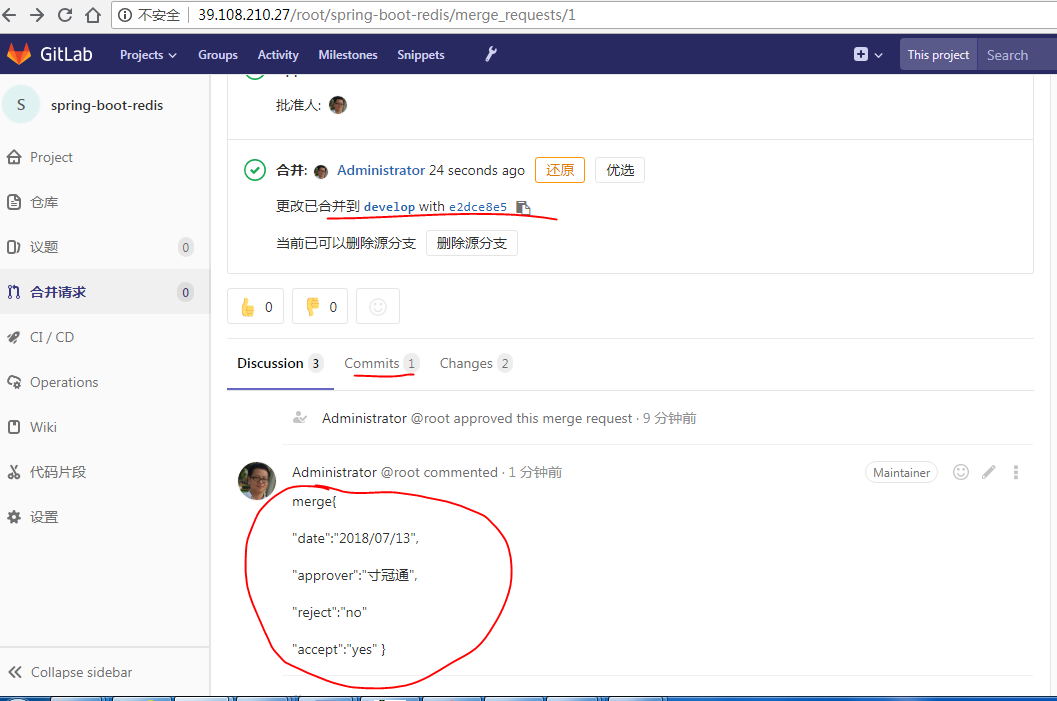
### 2.3.3 合并操作

feature#62与develop 分支的合并操作有两种实现方式：

直接在gitlab 界面操作或者直接Git Bash 命令行操作。

#### 2.3.3.1 gitlab完成代码评审分支合并

批准人登陆gitlab, 点击”合并请求” ，在合并前同样可以比较、评审两分支的代码，确认可以合并后，点击“合并“，完成feature#62与develop分支的合并，并提交评论。提交的评论可以按照固定的模板。如下图：



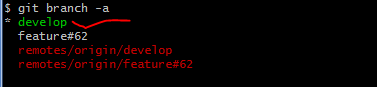
评论内容模板包含评审日期，批准人，拒绝还是接受此次合并申请， 拒绝或接受此次评审的理由等，评论内容模板作为强制项。

#### 2.3.3.2 git bash 命令行完成feature#62与develop 分支合并

先切换到develop分支：

$git checkout develop

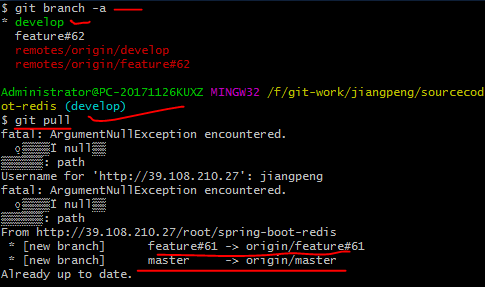
$git branch –a



本地develop 分支先pull 远程最新代码

在合并前，这个动作很重要，先必须确保本地develop已经同步到远程develop分支的最新代码。

$ git pull



执行合并：

$git merger --no-ff feature#62

合并成功后，将本地develop分支推送至远程origin资源库的develop分支下。

$git push origin develop

#### 2.3.3.3 评审留痕

在“活动”选项卡中，点击“合并事件” 可以查看评审记录：

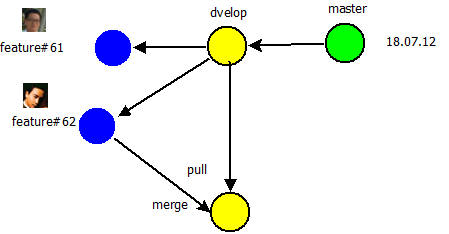


点击连接查看评审记录：



可以看出，除合并评论日志外，前面的提交更新中的日志也可查看。作为代码评审的重要依据之一。

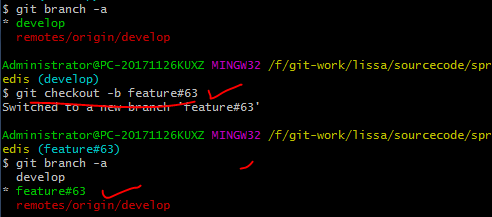
feature#62 与 develop合并后, Git Tree 状态变成了这样：



### 2.3.4 lissa 用户从最新的 develop 分支拉一个特性分支feature#63

创建分支feature#63

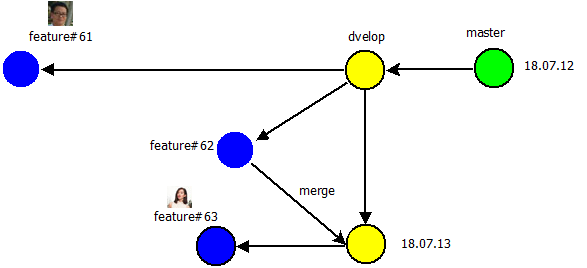
$git checkout –b feature#63



推向远程：

$git push origin feature#63

feature#63 分支创建后, Git Tree 状态变成了这样：



### 2.3.5 Lissa开发完功能后开始更新提交

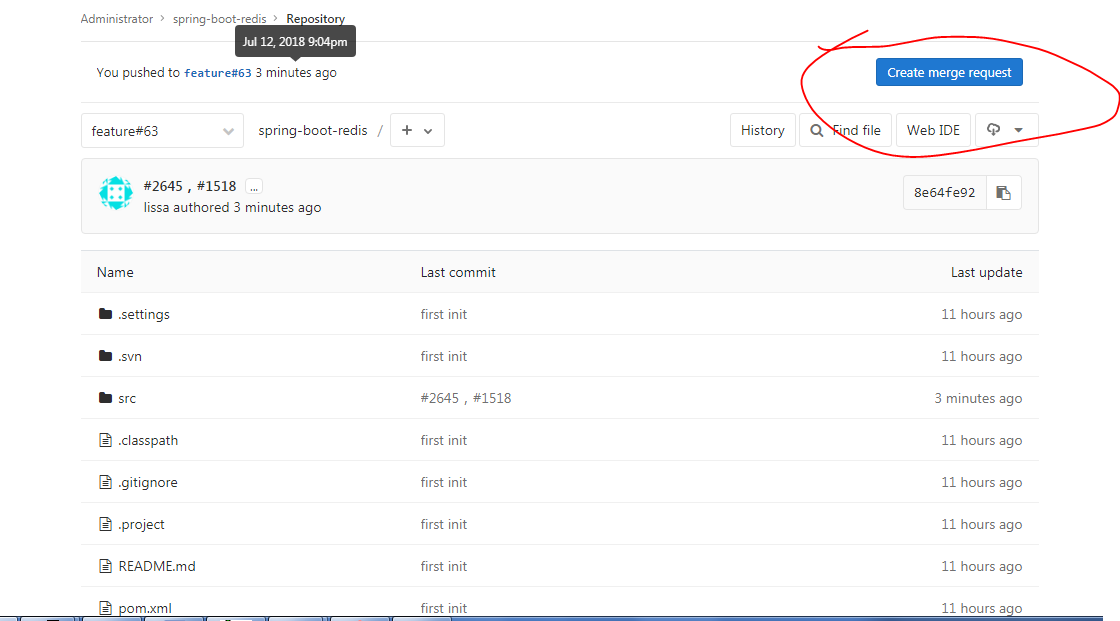
提交代码前要求其他开发人员对代码进行评审。

$git add .

$git add –m “ 日志内容模板“

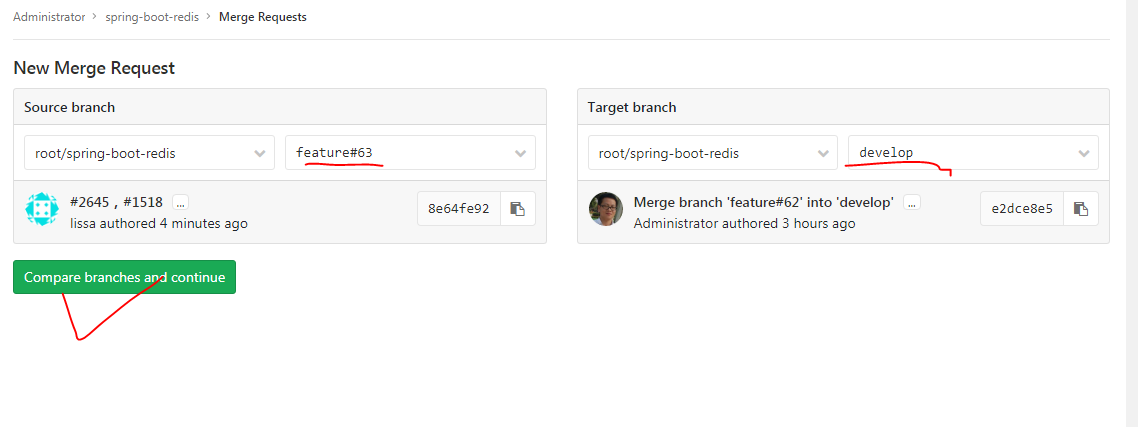
$git push origin feature#63

### 2.3.6 在 Gitlab 上请求与develop分支合并

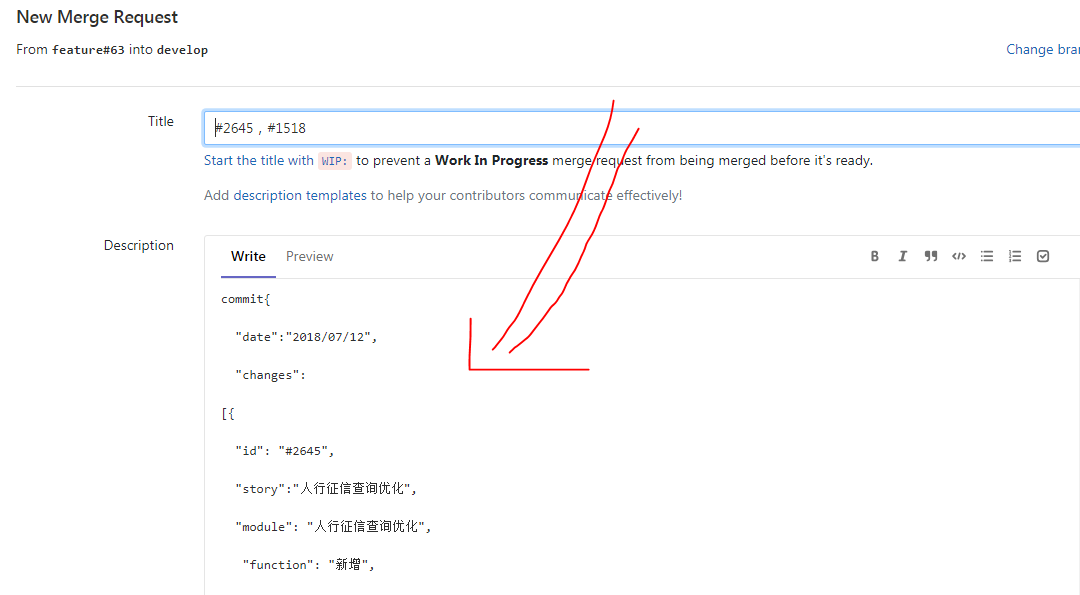


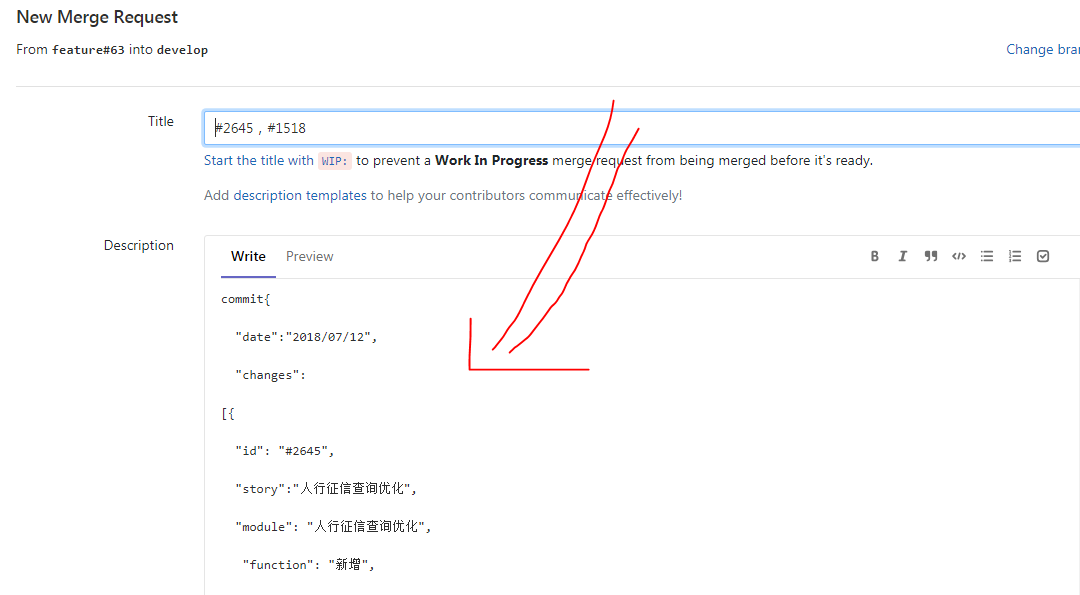
### 2.3.7 创建合并请求之前比较

合并前，开发人员1对1完成代码评审， 登陆gitlab, 找到合并的请求，点“比较分支并继续”。具体操作《2.3.2 feature#62 开发完功能后，向远程提交更新并在gitlab提交合并请求》已作介绍。

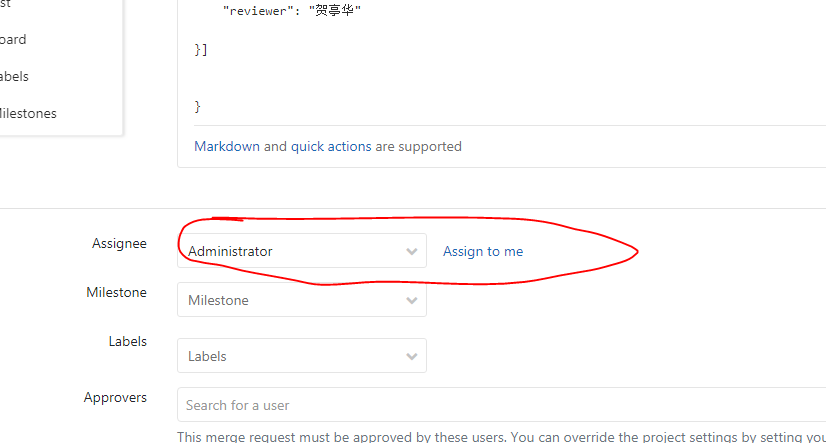


查看提交日志记录。





指定合并审批人员：



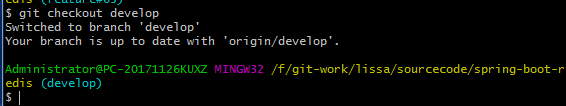
### 2.3.8 合并feature63#分支到develop主干分支上

此章节与《2.3.3 合并操作》相同。

这儿用仅介绍Git Bash 命令行实现合并分支：

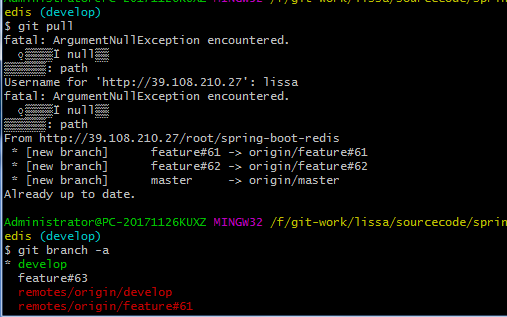
务必先切换到本地develop 分支

$git checkout develop



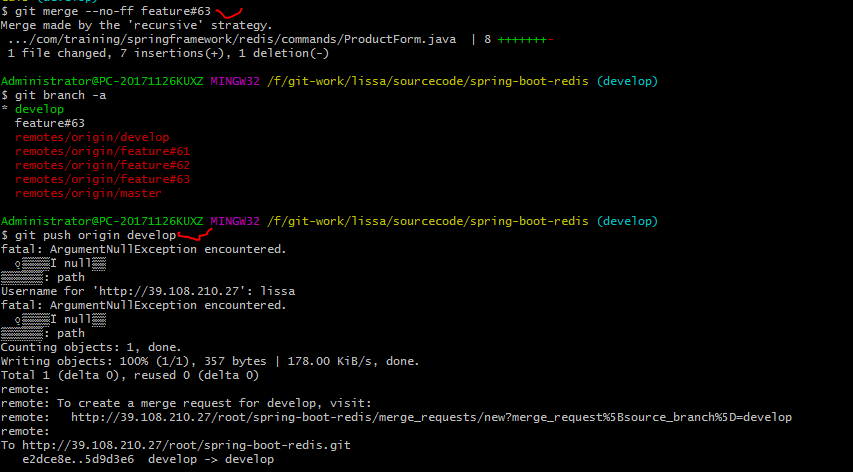
更新本地的develop 分支

$git pull

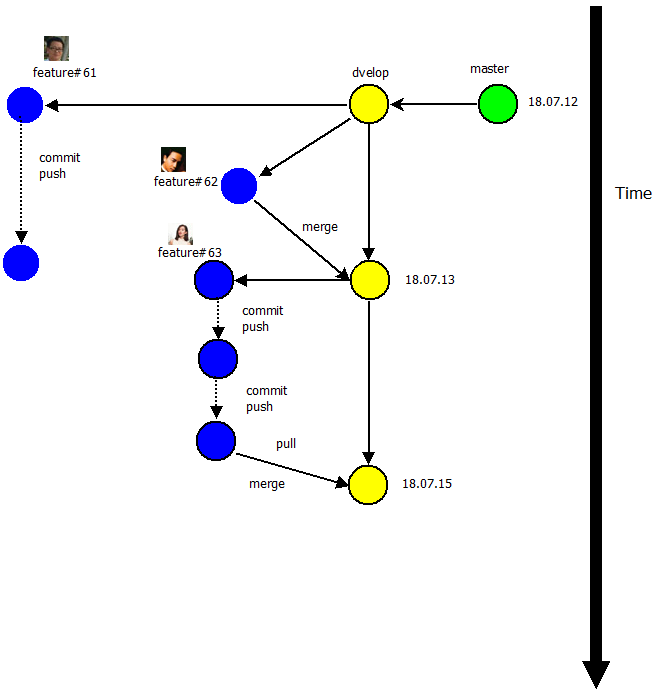


合并feature63#分支到develop主干分支上

$git merge –no-ff feature#63

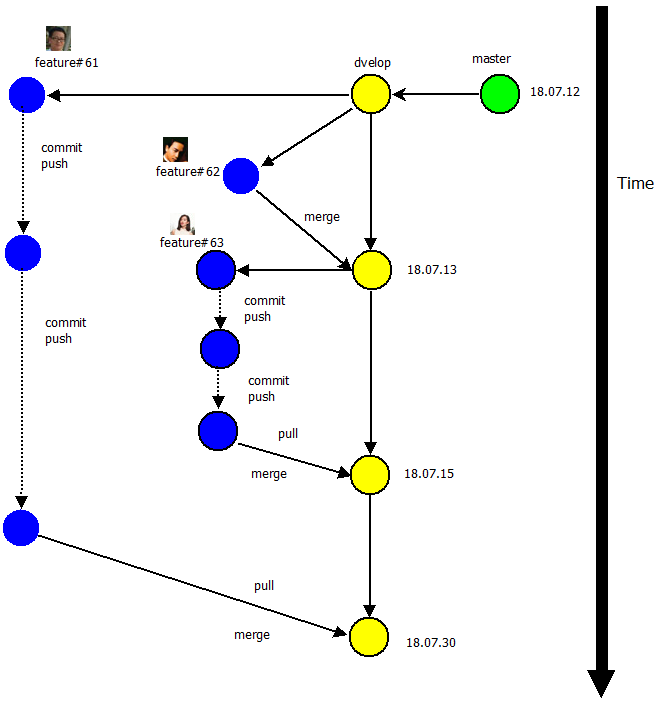


合并后的分支状态是这样子：



### 2.3.9 feature#61 与最新develop 分支合并

这儿不再累述合并分支操作。 请参见前面的合并操作。合并后的git 状态是这样的：



# 3 发布第一个release

前面的特新分支feature#61, feature#62, feature#63已完成了功能的开发。最后都合并到干分支develop上， 目前develop上最新的分支为develop.18.07.30，测试人员对分支代码做develop.18.07.30SIT,UAT测试，通过测试后，我们将发布一个稳定的版本release.18.08.06。

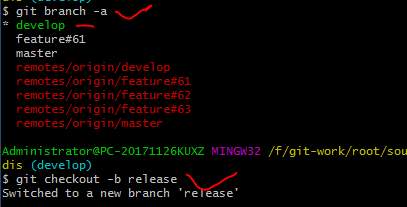
## 3.1 在最新develop.18.07.30基础上创建一个新分支release,标签18.08.06

切换至最新develop分支

$git checkout develop

创建发布分支release

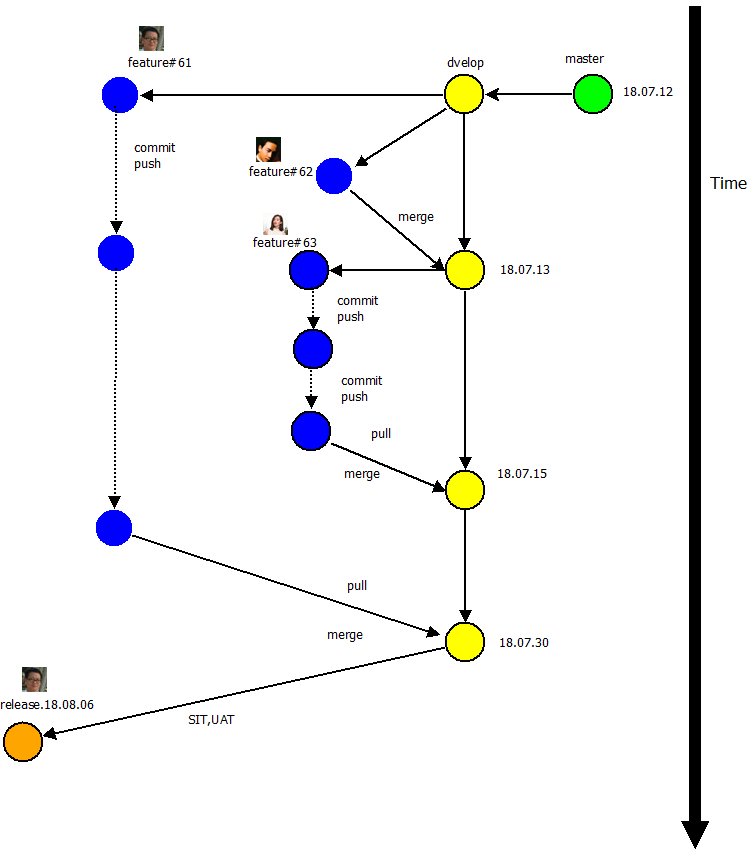
$git checkout –b release



## 3.2 推送至远程

$git push origin release

至此，第一个release版本成功发布。此时的GIT状态是这样的：



# 4 合并develop至master主分支

这里以管理员账号在Git Bash 下操作：

切换本地master工作区：

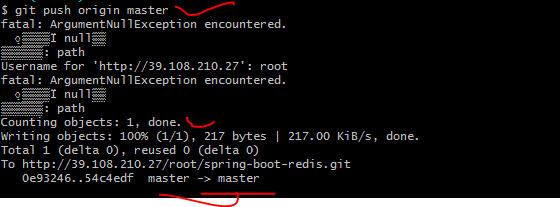
$git checkout master

合并develop到本地master分支：

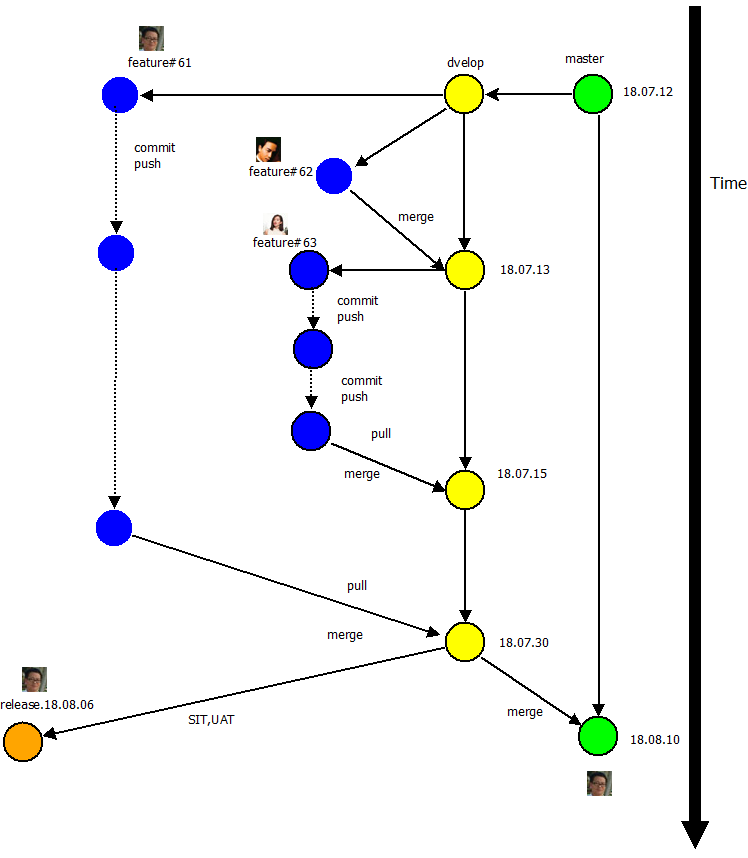
$git merge –no-ff develop

将本地的master分支推送至远程origin资源库的master分支下

$git push origin master

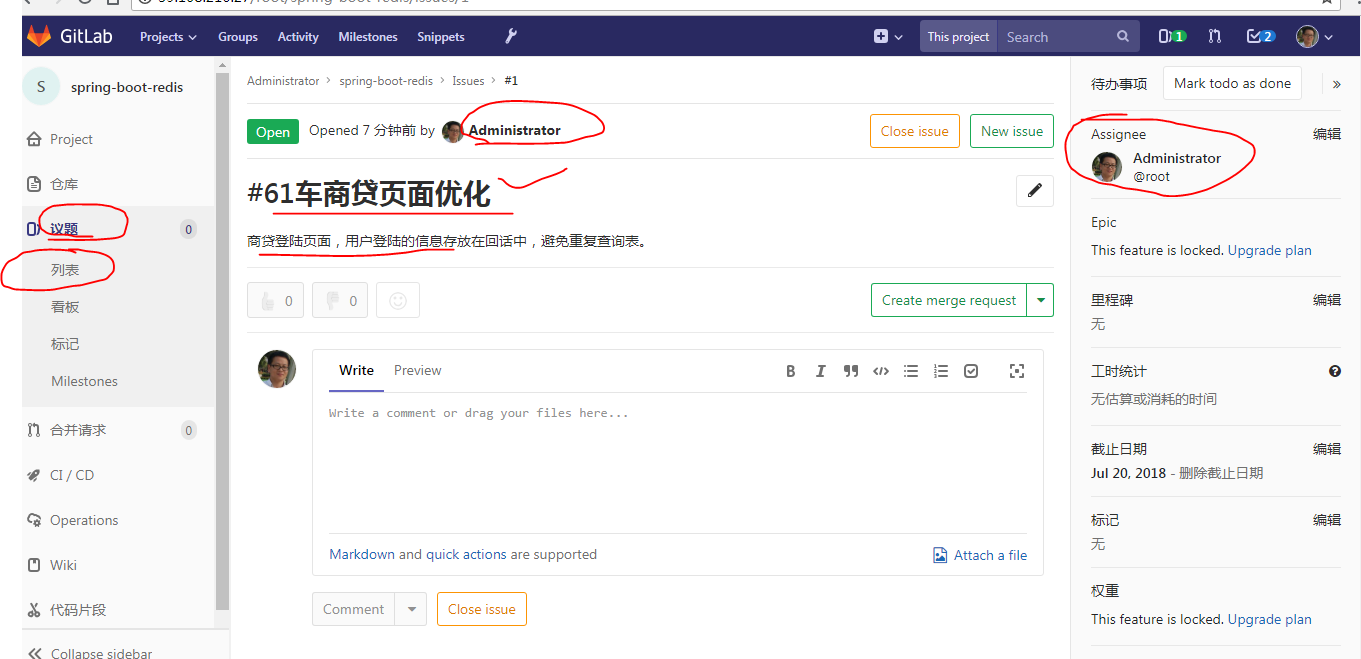


合并后的GIT 状态是这样的：



# 5 Gitlab Issue（议题）追踪问题、发布任务、缺陷报告

通过gitlab issue 功能，可以方便管理软件项目，包括软件需求设计、发布任务、跟踪一个系统缺陷。如下图： 控制面板的左侧，点击“　议题/列表“，可以发布一个议题。如：　可以为特新分支发布一个议题，描述其功能需求。



可以看出一个议题的发起人，包括议题的处理人。还可以将一个议题以milestone里程碑为单位归类。里程碑可以代表一个分支的版本、一个功能点、或者一个问题、BUG缺陷等。可以关闭议题，也可以将此议题分配给其他成员。

# 6 结语

本案例从Gitlab托管上一个空白项目开始，以多用户在develop上协作开发，在各自特性分支上完成开发后，提交更新到远程分支前如何一对一进行代码评审，后再与develop分支合并的迭代过程,在执行“分支合并“前如何进行两个分支代码的比较及评审。在所有的特新分支与develop分支进行合并后，如何基于最新develop分支发布一个release版本，最后通过develop分支与master分支合并，实现了协同开发。为增强读者对Git,Gitlab代码管理的理解，注重实操的同时提供了每一个GIT操作后GIT Tree的各版本分支的迭代图。