**Git 代码管理**

# 总览

Git或类似Git的代码管理平台（下简称Git）已经成为产品开发的重要管理工具，团队在协作开发，并持续更新代码时几乎离不开Git提供的代码管理服务。

Git使用可分支、可合并的工作流来记录产品开发的价值流动。Git提供了这样的协作框架，但具体产生什么样的工作流取决于团队的工作方式，Git并不保证工作流的清晰，如果团队的工作方式不规范，极有可能产生混乱的工作流，导致版本控制缺乏管理，引发多余的、痛苦地开发成本，并可能降低产品本身的可靠性。

为保证代码管理的规范性，版本的整洁，产品发布的可靠性，特制定本规则。

使用Git开发产品已经有十余年历史，关于在Git上开发产品的工作模型以趋于成熟，本规则主要依赖于前人的思路制定：https://nvie.com/posts/a-successful-git-branching-model/

按照本规则产生的工作流效果图为：



# Git代码管理细则

Git的工作流分为主工作流和支持工作流两大工作流，支持新功能开发、bug修复、版本发布

## Git的主工作流

Git的主工作流主要分成两个平行的主分支，master和develop。



### Master分支

Master分支里面的代码是稳定的，随时可以发布的代码，它来源于Develop分支。

### Develop分支

Develop分支里面的代码包含目前开发所有已经完成的功能特性，某个稳定的、且功能符合发布要求的Develop分支节点会作为下一次Master分支节点的来源。

## Git的支持工作流

Git的支持工作流由分支，Feature、Release和Hotfix构成。其中Feature分支用于新功能开发，Release和Hotfix都用于bug修复

### Feature分支

Feature分支是新功能开发的唯一途径，每个feature从开发时点的Develop分支出发新建一个当前feature的分支，当feature完成后，再合并回Develop分支，合并时需要把当前feature里所有的commit放到一起（--no-ff flag）。



### Release分支

Release分支用于支持版本的发布，Release分支由一个稳定的Develop分支节点生成，且此时Develop分支的该节点完成了所有发布需要的功能。此时Develop分支可以继续开发加入新的功能，而不必停下来等待发布完成以避免新功能影响发布进度和状态。Release分支修复测试后的bug，并把bug修复的代码合并的Develop分支，当Release分支通过测试后，再合并到master分支用于发布。

### Hotfix分支

Hotfix分支用于修复已经发布到master分支代码的Bug，Hotfix分支基于发现bug的master分支节点，修复后将代码同时合并到Develop分支和master分支。

## Commit规范

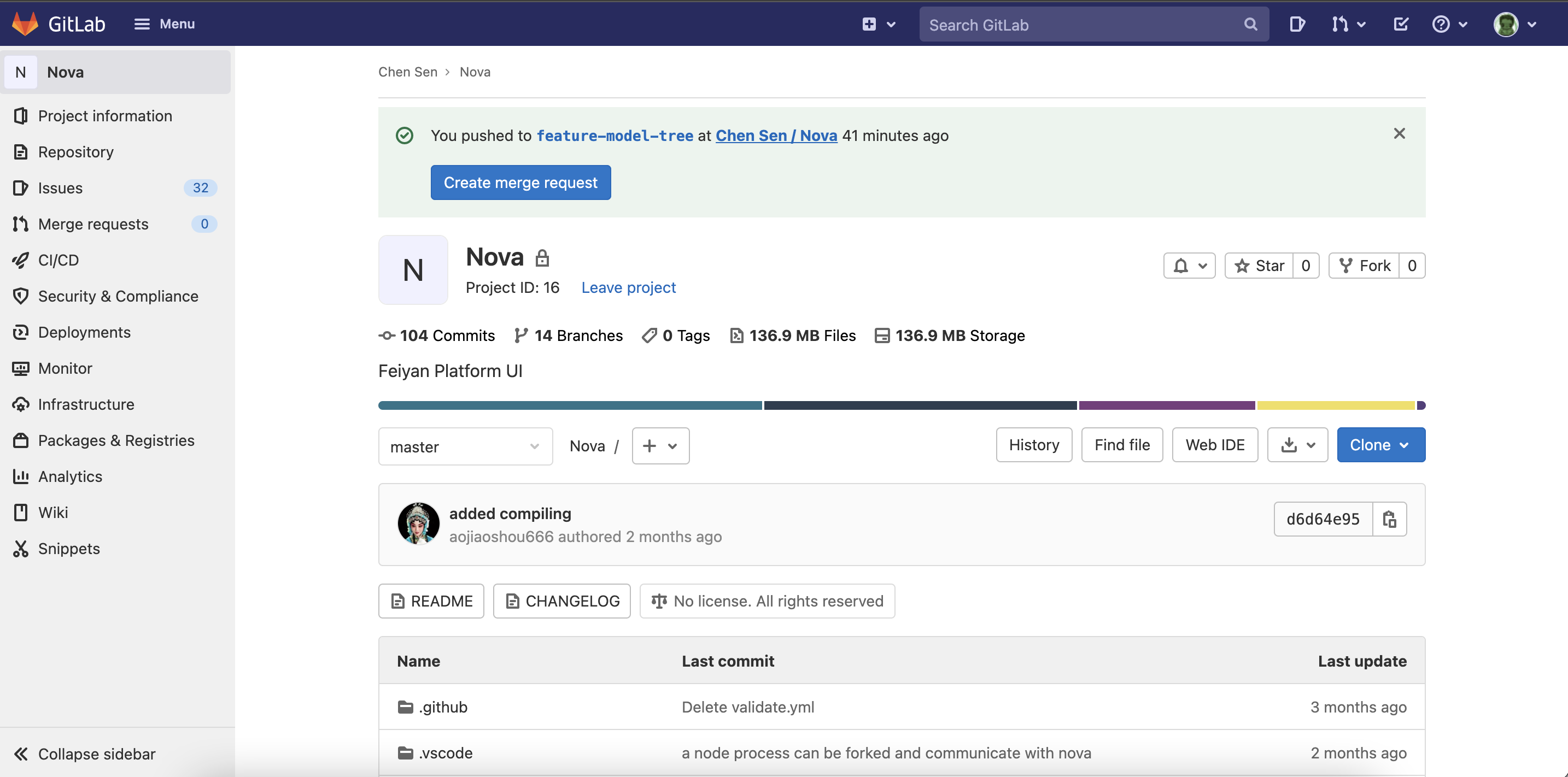
当我们修改代码后，需要像对应分支更新修改后的代码时，需要执行Commit操作。Commit让我们在分支上的所有工作都可以跟踪。每条Commit有配有一条信息用于描述Commit带来的变化。我们在Commit时，必须附上清晰的Commit信息，以便于团队其他成员能快速理解这条Commit带来的变化。Commit前务必运行npm run lint检查语法代码规范问题，修正好再提交。

例子：

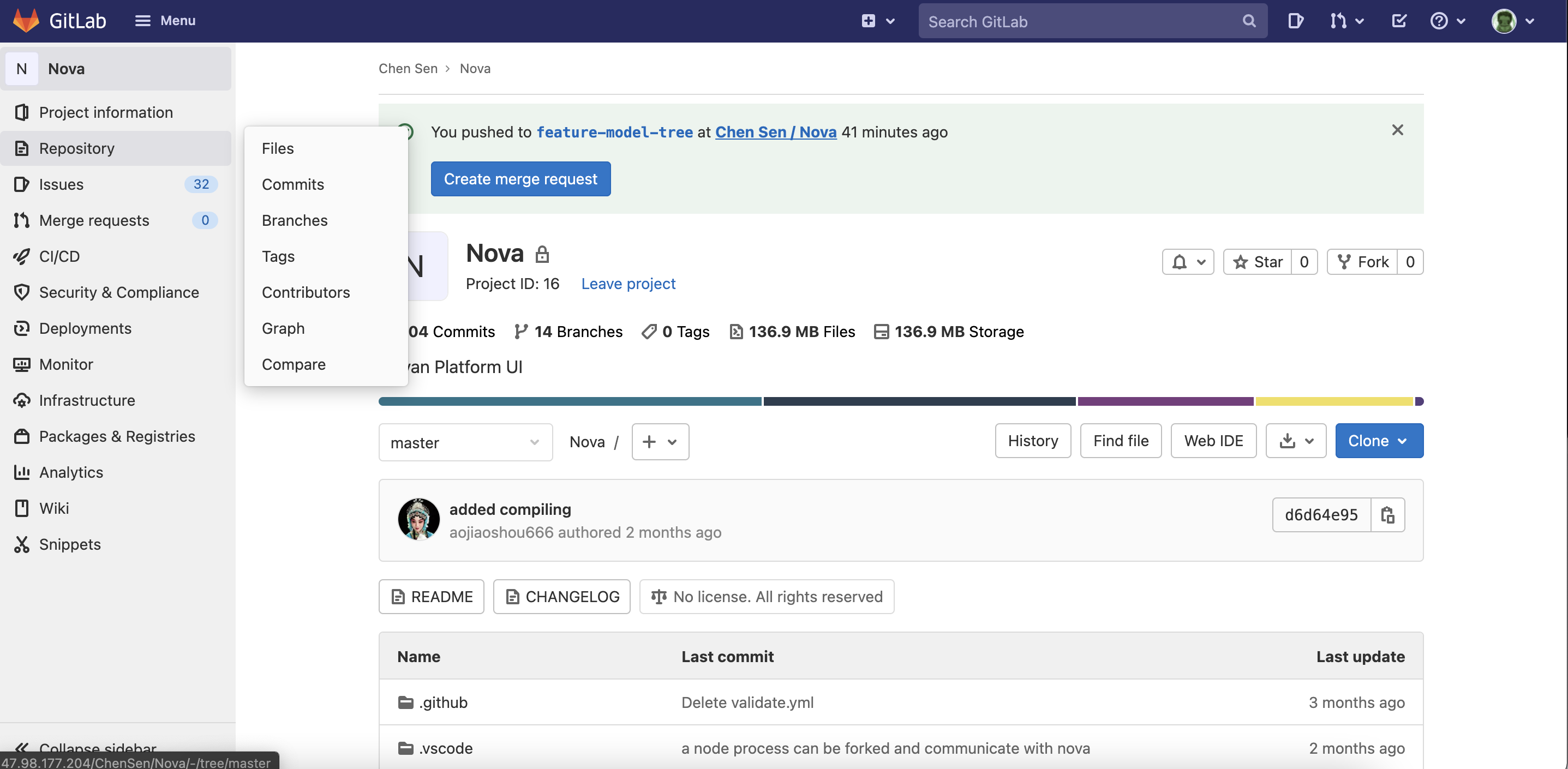
## Merge Request规范

Gitlab采用Merge Request合并分支。我们平时的工作主要涉及往develop分支上提交Merge Request的情况。下面介绍我们会常用的两种新建Merge Request方式。其他方式请参考官方文档[Creating merge requests | GitLab](https://docs.gitlab.com/ee/user/project/merge_requests/creating_merge_requests.html)

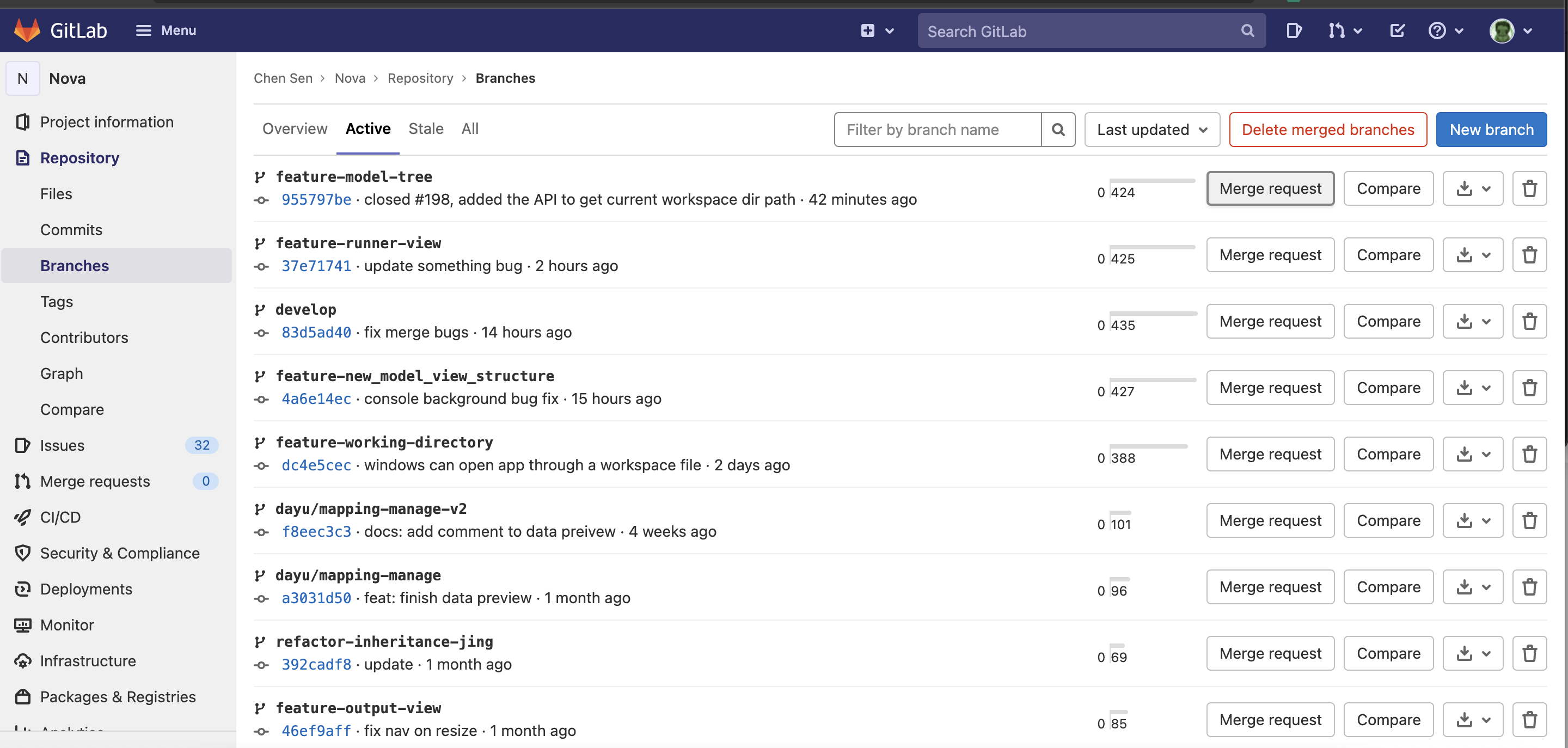
1. 在VSCode里Push code到 Remote的 feature分支后，打开Gitlab项目页面会自动提示是否Create merge request。



2. 随时在左侧导航栏选择Repository > Branches进入Branches页面



在这里选择想要新建Create merget request的分支点击 左侧的Merge request按钮



进入新建Merge requet页面后，点击上方的Change branches将target branch换成develop。按提示填写description，简要说明该feature或bugfix的功能。然后选择Assignee和Reviewer提交。Reviewer只负责代码的审阅，而Assignee还负责最终的合并。关于Assignee和Reviewer的区别，参考官方文档。[Assignee](https://docs.gitlab.com/ee/user/project/merge_requests/getting_started.html#assignee) 和[Reviewer](https://docs.gitlab.com/ee/user/project/merge_requests/getting_started.html#reviewer) 。

