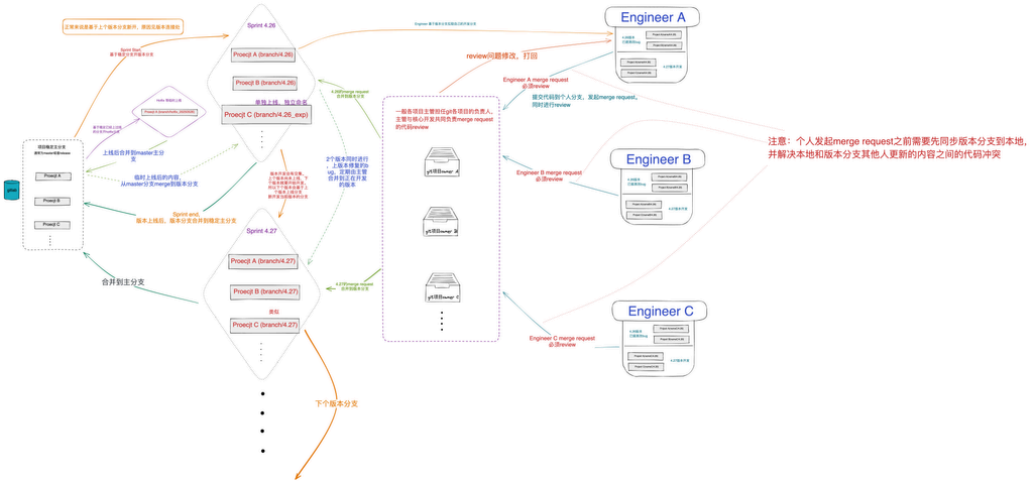
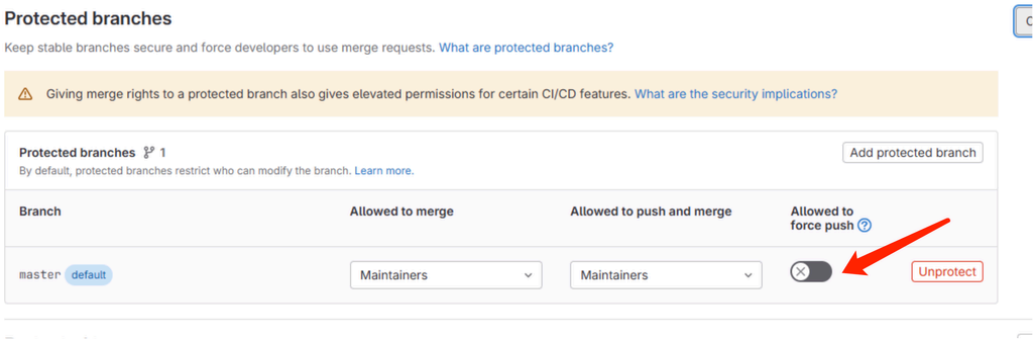
代码分支管理与提交流程规范

分支管理流程图（原稿修改请滑动到底部）：



一、代码分支创建

所有的用于上线的分支（版本分支、hotfix分支）都应该关闭force push.



1.版本代码上线分支

版本开发，最后统一跟版本上线

版本定版开始，根据版本号，在各个项目基于项目的主分支(一般为master或者release)统一开设版本分支。命名格式为：branch/{sprint版本号}，如：branch/4.27

lofty前后2个版本是有一定交集，如4.26版本全面提测后，4.27版本已开始开发，所以4.27版本会基于上个版本4.26开分支。

跟版本中开发，跨多个版本延后上线，或者当前版本提前上线

基于项目的主分支(一般为master或者release)统一开设版本独立上线的分支

命名格式：branch/{sprint版本号}_{需求名}，如：branch/4.26_exp

2.engineer开发分支

所有参与项目的开发人员，基于版本分支创建自己的该项目的开发分支。

命名格式：{开发英文name}/{版本分支}，如：uranus/4.26，或者uranus/4.26_exp

3.Hotfix分支

线上bug修复或者**当天需求上线（如改文案）**，基于稳定版本（master或者release）分支，以及日期，创建hotfix分支

命名格式：branch/hotfix_{date}，如：branch/hotfix_20250605

二、代码提交与合并

1.版本开发

除hotfix外，所有版本开发都是基于个人分支进行。版本个人分支，开发应该养成习惯，定期merge版本分支到个人分支，每日至少1次，并提交个人开发部分代码，避免多人开发同一个项目代码长期不合并，从而导致差异化过大，花费大量时间合并代码。

需求在个人分支基本完成，提交merge request，合并到版本上线分支。

- merge request必须先在本地将版本上线分支最新代码合并到本地，并解决代码冲突。
- 所有的merge request代码必须review，主管或者团队核心开发，模块负责人，均可参与review。且通过teams群组@负责人进行代码review，及时提醒，避免review等待时期过长。
- review包括代码需求逻辑实现，设计架构模式，性能预估，扩展性，方法、类、变量命名，注释等等。某一行，或者整体结论。使用英文进行在gitlab上备注清楚。
- 代码合并到主分支后应当自动触发sonar代码自动检查任务，辅助进行代码检查。（尚未全部配置上，还在完善中）
- review完成后，review人员告知项目负责人点击合并代码。将开发分支合并到版本上线分支上。
- 除个人分支，所有的提交都应该是经过review的，禁止直接push到关键分支。

2.多分支合并同步

版本上线或者hotfix上线后，一般观察2~3d无严重线上问题无需回滚后。合并分支到master或者release稳定分支。合并完成后，将master或者release分支，再合并同步到正在开发的版本分支上。

比如：4.26版本上线后，把4.26版本内容合并到master分支，再把master分支合并到4.27版本分支。

因为4.27版本是基于4.26版本创建，4.27版本创建后。4.26版本依然会有bug修复等更改代码，所以通常并不会等版本上线后才完全同步到4.27，主管会定期把4.26版本分支直接同步merge到4.27版本上。避免时间太久差异过大。

三、分支清理

超过1年的，非master或者release分支，应该清理，节省项目空间，提升git服务性能。

四、现状、难点及调整

现状难点：

lofty微服务加lambda服务，总计超过300+，平均每个后端负责超过5个以上的模块服务。所以每个开发都有自己负责的项目，作为项目owner，负责该业务模块的分支创建和提交，通常为了提速，只在比较重要的模块或者大型项目才进行review及merge request. 特别是团

队骨干及核心开发，负责业务较多。直接push代码情况较为严重，需要调整。

当前lofty总计git Project数量统计

lofty CRM & AI backend	lofty SITE & data backend	lofty FE	Billing	Flow	LW CN
84	83	27	10	11	13

1. git分支较多，通常每次版本涉及git地址项目，CRM + SITE + FE 总计50个左右。版本开始，各主管，需要负责版本分支的创建的，开发没权限，也不能创建，主管需要投入一定时间处理。
2. review及代码合并，比较大的项目可能会进行线下多人review，以及小的改动提交，有问题会当面叫上开发直接告知，效率会比较高，也比较节约review人的时间。缺点在于没有review记录。

改进调整计划：

1. 调整所有gitlab项目开发权限，所有开发都只保留develop拉取权限，确保版本分支无直接push条件，只能通过merge request提交。 deadline: `2025年6月6日`
2. gitlab项目整合，减少git项目分支，根据业务相关性，把关联性强的业务整合到一个git Project下，通过不同的CI打包不同的服务即可。减少git地址的数量，从而降低管理难度。 旧服务处理 deadline: `2025年9月30日`，后续新服务按照此要求设计。
3. review机制，AI辅助，sonar全量集成。 持续进行

分支管理及代码提交合并流程图（原稿）：

```
Cannot GET /draw/
```

App is not responding. Wait or cancel?