gitlab 使用教程

目录

注册

注册地址：http://gitlab.mrbird.cc/users/sign_in 即 （http://192.168.11.196/users/sign_in）
可以将 gitlab.mrbird.cc 设置到 hosts 里
邮箱用来接收验证邮件，修改密码，接受通知等。

注册好后，管理员加入项目组。然后就可以看到项目，进行协作开发了。自己也可以上传自己的项目，跟 GitHub 一样。

使用

1、建立项目&添加协作者

2、初始化项目

执行下面命令
cd existing_folder
git init
git remote add origin git@gitlab.mrbird.cc:2223/dev-team/febs-cloud-web.git
git add .
git commit -m '初始化项目' git push -u origin master

3、建立分支

3.1、主分支

每个项目的 dev 分支为开发分支,master 分支为主干分支
建立分支分支名称约束为'dev'，“create from”输入 master

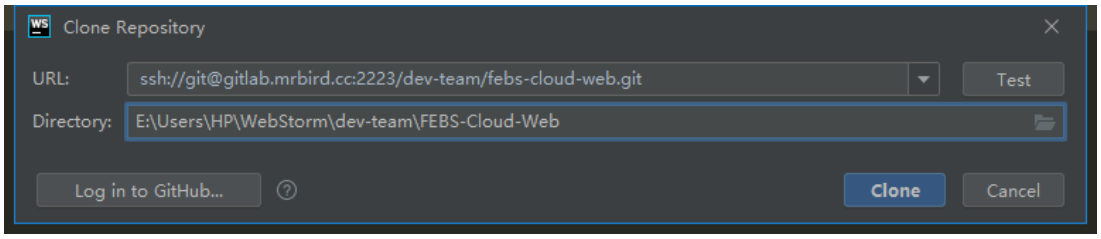
3.2、开发人员分支

建立开发者分支，分支名称约束为 dev_name 【例如“dev_ly”或者“dev_wss”，后面加上开发者自己的标示】，“create from”输入 dev，流程与上面所说一样

4、开发

4.1、拉取代码

首先创建一个项目名命名的文件夹，将代码 clone 到此目录中，并选择代码
git clone git@gitlab.mrbird.cc:2223/dev-team/febs-cloud-web.git



这里注意 Directory 中项目名字小写改大写。
先切换到 dev 分支
git checkout dev

创建开发者自身分支（其实这里最好是通过 gitlab 3.2 创建的分支来编辑代码） git checkout -b dev_name

4.2、开发以及提交

获取开发者分支后正常开发以及提交，开发功能结束后,发起一个 pull request 请求合并，经过审核将代码合并到分支代码（dev）中(一般直接合并到 dev)，测试稳定通过后合并到主干代码（master）中

4.3、合并开发者分支（dev_name）代码到开发分支（dev）

定位到 clone 的项目文件夹
cd FEBS-Web-Cloud
git add .
git status //查看文件状态
git commit -m "改动日志"
项目切换 dev_name 到 dev，然后执行 merge，第二行中的“--no-ff”一定要添加，否则不显示合并的 log
git checkout dev
这里多人协作的时候最好先 pull 一下 origin:dev 的代码
git pull origin dev
git merge --no-ff dev_name -m 'xxx' //显示 Already up-to-date.说明合并成功 git push origin dev //上传分支

4.4、合并到主干

最后测试稳定后将代码合并到主干，下面这一步由代码负责人（身份为 master 的管理者操作）
git checkout master git merge --no-ff dev -m 'xxx'//显示 Already up-to-date.说明合并成功 git push origin master//上传主干

两种管理方式

1、分支管理模式

开发阶段

1. 除了 master 分支创建一个供所有开发人员开发的 dev 分支；
2. 开发人员在 dev 分支上进行工作，随时随地 commit，每天 push 一次到服务器（在自己的 dev-name 中开发，合并到 dev,然后 push dev）；
3. push 代码前需要进行 pull 操作，因为有可能在之前有别的成员先进行了 push 操作，如果有冲突还需要进行冲突解决；
4. 每天上班后所有成员对 dev 进行 pull 操作，获取所有成员 push 的代码，有冲突需要解决；
5. 团队 Leader 每天将 dev 合并一次到 master。

测试阶段

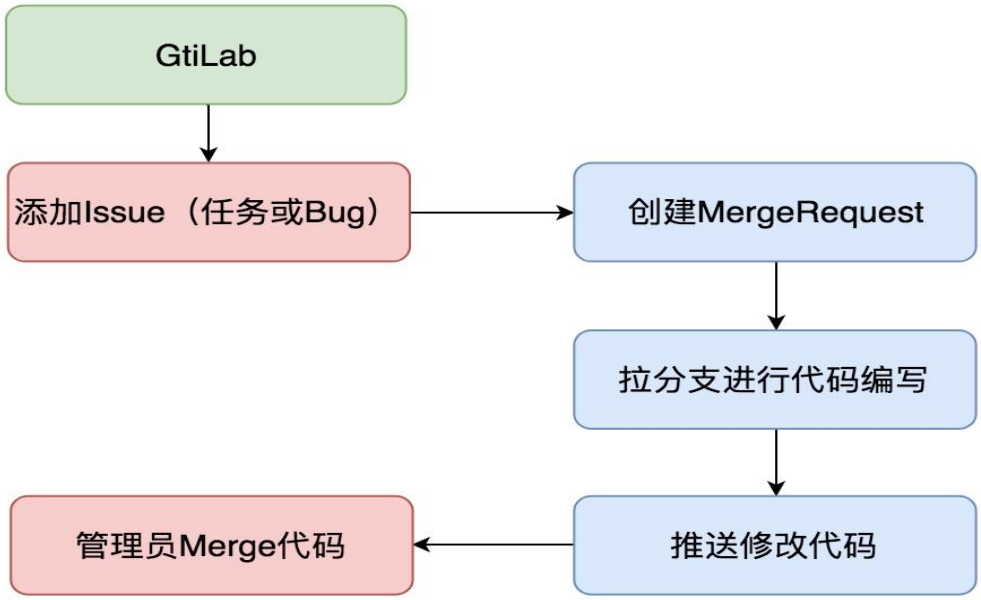
1. 测试进入后就需要添加 test 分支；
2. 在开发人员将代码 push 到 dev 分支后，可以在 dev 基础上创建 test 分支，测试人员以 test 分支搭建测试环境，开始测试；
3. 开发人员在接受到 bug 后，直接在测试分支上修改，然后让测试人员进行验证；
4. 每天团队 Leader 将测试分支上修改的 bug 合并到 dev 分支上，这样所有团队成员当天修复的 bug 都会在第二天被团队其他人 pull 下来；
5. 团队 Leader 每天将 dev 合并一次到 master。

上线阶段

1. 系统上线后试运行阶段会存在两种改动：bug 和优化需求，bug 通常当天解决晚上部署，优化需求通常周末部署；
2. bug 当天能修复的就直接在 test 分支上修复，然后进行测试，验证通过后合并到 master；
3. bug 当天不能修复的就针对该 bug 创建一个分支，修复完后合并到 test 分支进行测试，验证通过后合并到 master；
4. 每个优化需求都以 master 分支为基础创建一个 feature 分支，完成后合并到 dev 分支，开发人员可以先交叉测试，然后将 dev 合并到 test 进行测试，验证通过后合并到 master；
5. master 始终是一个干净的，可发布的分支。

2、Merge Request 模式

Merge Request 的开发流程



1. 需求或是 Bug 都是用 Issue 来表示；
2. 虽然 Issue 不支持多层级，但结合里程碑、标签等还是可以很好的对任务和 Bug 进行管理；
3. 管理员和团队成员都可以进行 Issue 的创建；
4. 任务的接收者对 Issue 创建 Merge Request；
5. 完成任务后推送代码到 Merge Request 对应的分支；
6. 管理员对代码进行 Merge。

总结

master 上是稳定的代码，一般在 dev 分支上做开发，当做好一个功能，完成一次 bug 修复后，再 push 到 origin:dev 。

FAQ

merge: dev_lang - not something we can merge
名字写错了，应该是 dev-lang