Gitlab+P3C-PMD(Aliyun)标准化你团队的代码

# 1.背景简介：

项目越来越大，团队越来越大，代码越来越来多，然后就是各种五花八门的代码格式、代码规范。要做到代码规范我们可以采取代码交叉检视、IDEA/Eclipse安装插件自检、以及代码管理服务端做校验，且不说各种方法的利弊，本文直接采用服务端做校验(Gitlab)。

# 2.服务端gitlab校验原理简要说明：

利用gitlab的 (钩子)hooks机制，关于钩子有客户端、服务端的，本文仅描述服务端的钩子，更多关于钩子的资料参考：

<https://git-scm.com/book/zh/v2/%E8%87%AA%E5%AE%9A%E4%B9%89-Git-Git-%E9%92%A9%E5%AD%90>。

gitlab服务端会有三个主要钩子：**pre-receive**(处理客户端push动作时最先被调用的脚本，以非0值退出拒绝Push，可以用来做注释标准化、代码标准化等)、**update**(与pre-receive功能类似，为每个准备更新的分之各运行一次)、**post-receive**(是在push之后执行的脚本，可以用来调用后续的持续集成、发邮件通知等)。

而本文的用到的做代码规则校验的就是pre-receive的钩子，通过此钩子执行java –cp命令调用aliyun p3c-pmd的规则校验，然后检查如不符合规范返回错误信息到客户端(idea,git bash等)。

# 3. P3C-PMD

P3c阿里云的插件，github地址：<https://github.com/alibaba/p3c>

此处我们用到p3c-pmd插件，需要打包成jar文件，以便gitlab服务端使用。主要是受：

<https://www.jianshu.com/p/b87ca8615c9c>博文启发，结合该博文以及已经躺过的一些坑整理成一份新的文档

## 3.1 打包p3c-pmd，

使用最新2.0.0版本，Windows环境构建

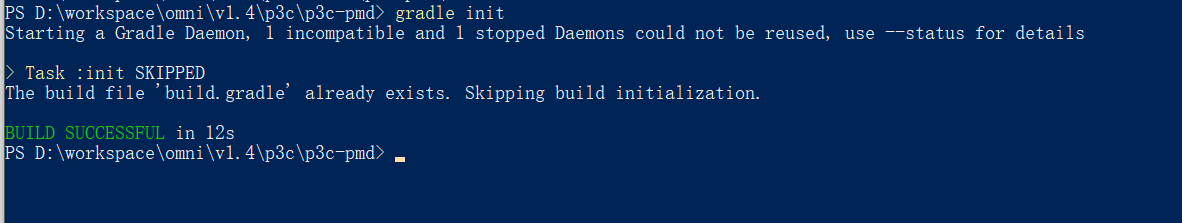
3.1.1 准备gradle环境和git环境(可选)，请自行百度安装gradle文档

3.1.2 github下载源码：git clone <https://github.com/alibaba/p3c.git> 或者直接到github上download zip文件

3.1.3 进入p3c-pmd目录：



3.1.4 执行gradle init



3.1.4 编辑build.gradle文件

在文件末尾添加：

jar {

from {

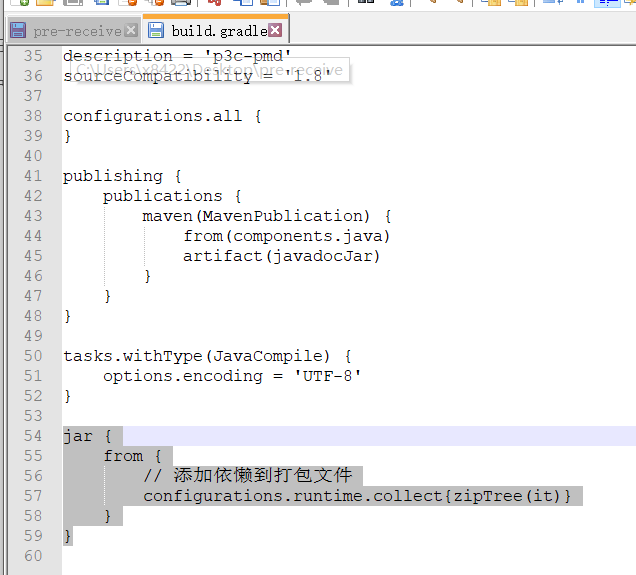
// 添加依懒到打包文件

configurations.runtime.collect{zipTree(it)}

}

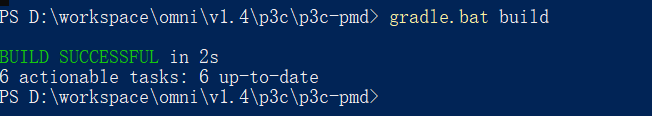
}

如下图所示：

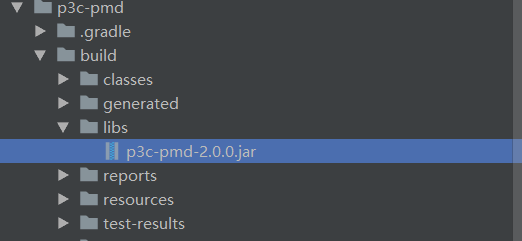


3.1.5 开始构建gradle

执行gradle.bat build



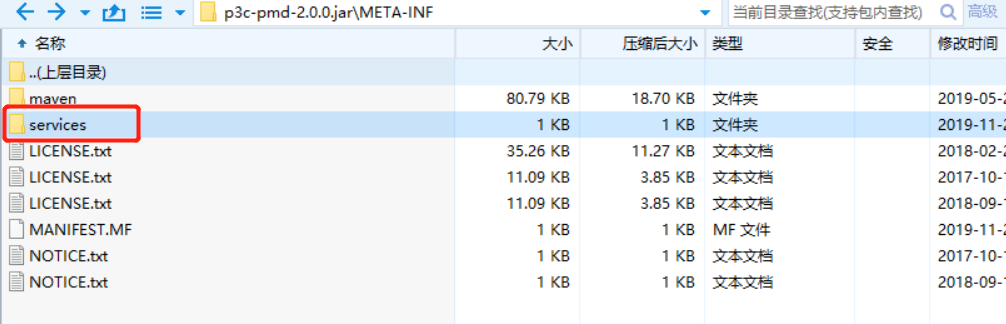
构建成功后会在 build\libs生成p3c-pmd-2.0.0.jar文件



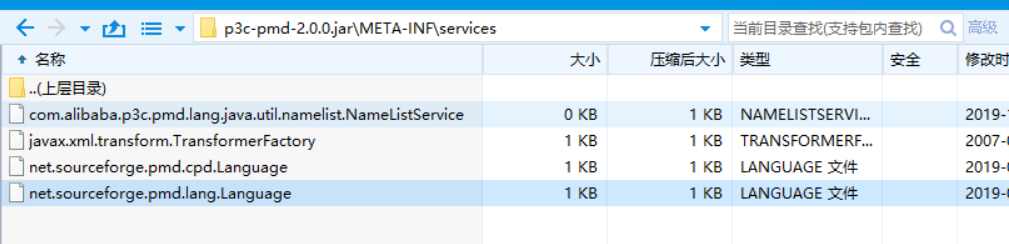
3.1.6 修改p3c-pmd-2.0.0.jar中META-INFO文件内容

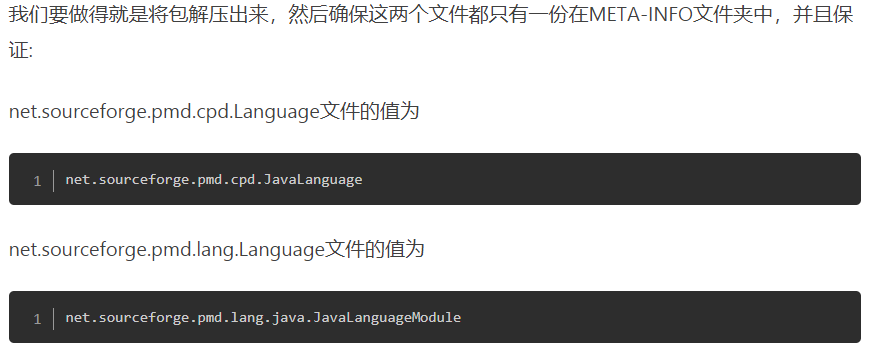
1).用压缩文件打开.jar文件：



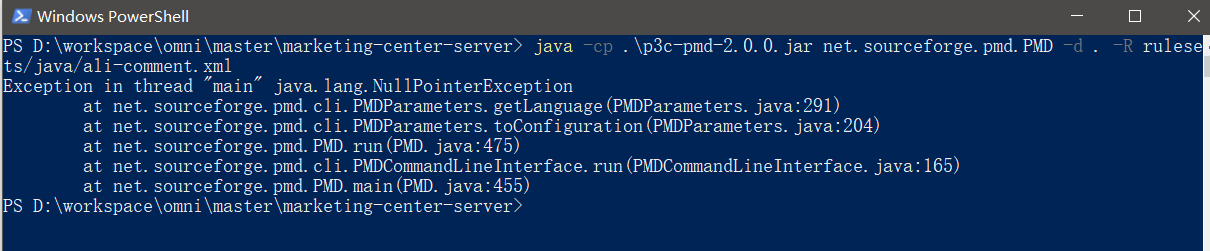


2).删掉重复的文件并确保里面的内容是正确的





3).主要用于解决：

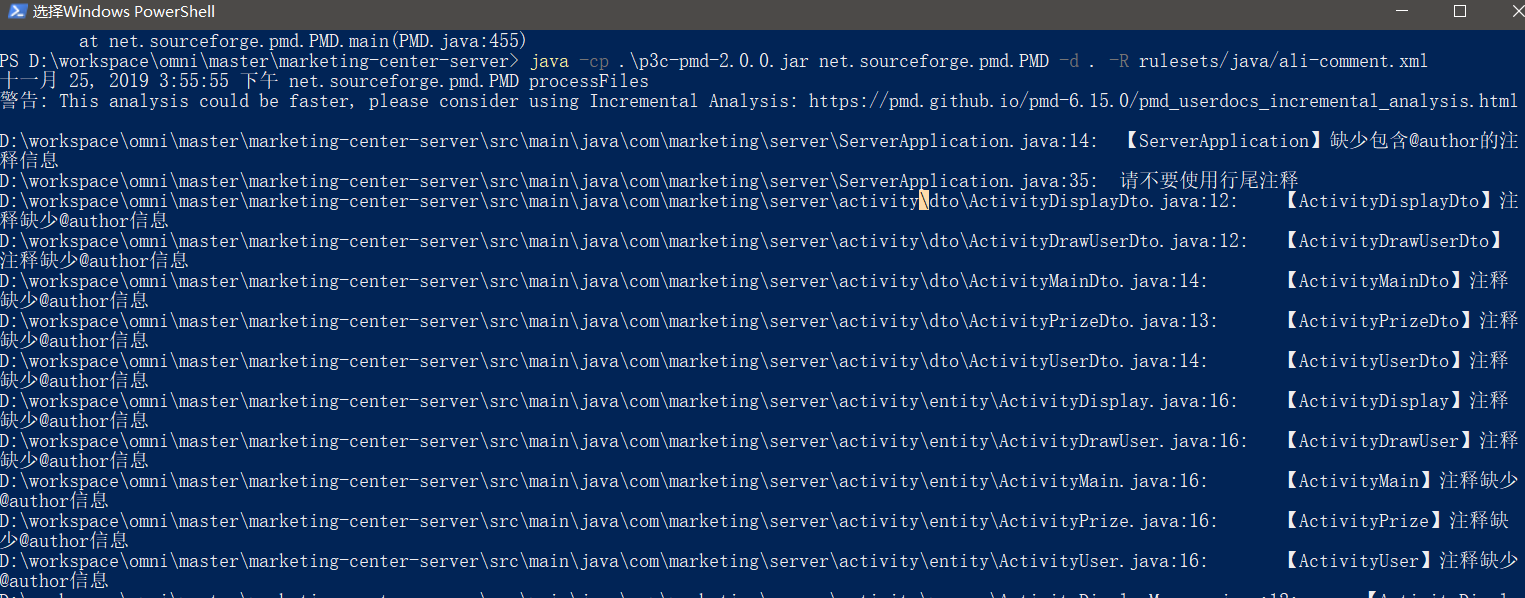




## 3.2 执行jar校验是否正确

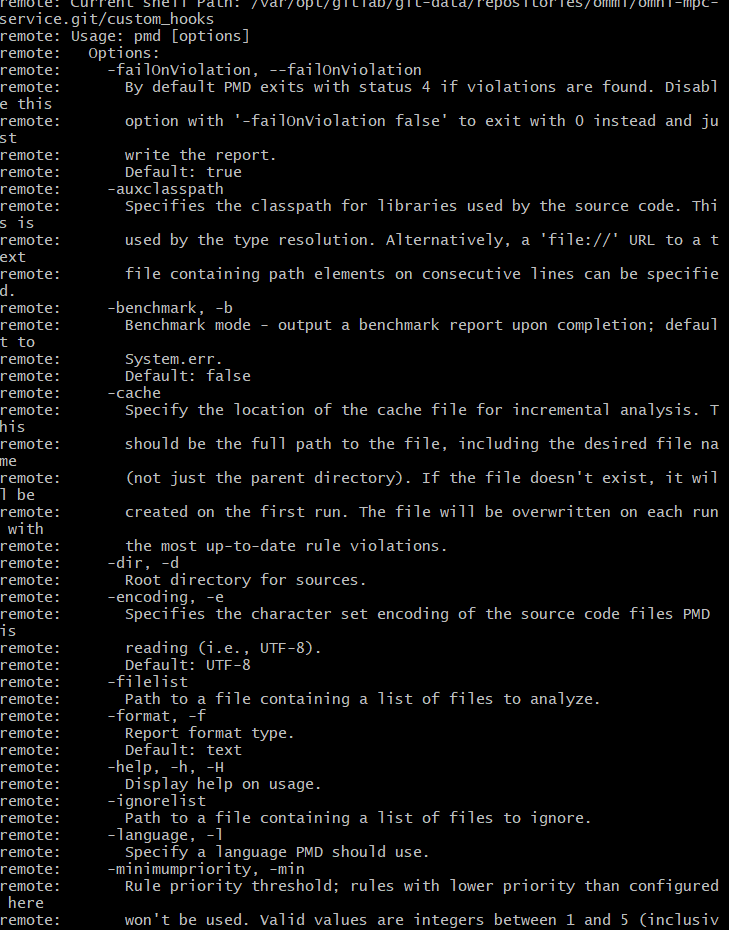
java -cp p3c-pmd-2.0.0.jar net.sourceforge.pmd.PMD -d . -R rulesets/java/ali-comment.xml

得到如下提示，表名正确可用：



p3c-pmd 几个参数说明：-d 待校验的源文件目录，-R 校验规则，-f 检测结果输出格式。

Github没有明确说明各种参数，只不过执行命令错误会提示各种参数以及使用方法：



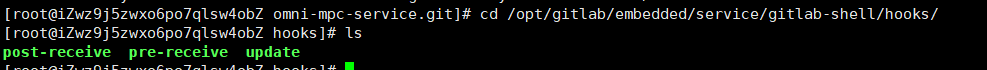
# 4. gitlab + p3c-pmd配合使用

## 4.1 gitlab hooks配置

4.1.1 gitlab hooks配置分全局(所有项目)和针对项目配置

官方文档说明地址：<https://docs.gitlab.com/ee/administration/custom_hooks.html#setup>

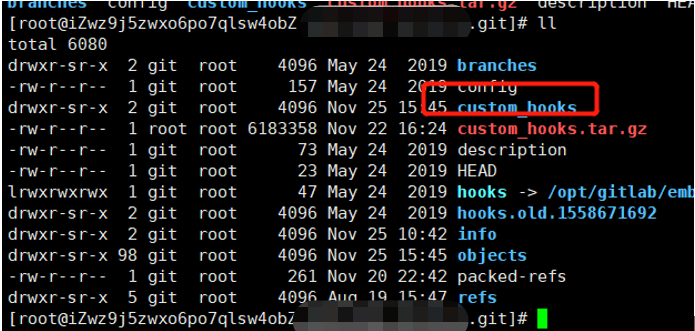
全局配置路径：/opt/gitlab/embedded/service/gitlab-shell/hooks/



针对项目的配置路径：

/var/opt/gitlab/git-data/repositories/${group}/${project\_name}.git/

需要在该文件下创建custom\_hooks目录，不要试图直接修改hooks目录下的东西，是针对全局的。

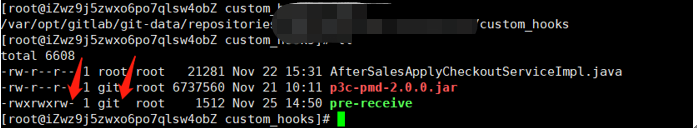


本次以项目为单位配置代码校验，即在custom\_hooks

## 4.2 配置p3c-pmd-2.0.2.jar

4.2.1拷贝3步骤所生成的p3c-pmd-2.0.0.jar文件到 custom\_hooks目录

4.2.2 pre-receive文件，并保持776可执行权限以及仅属于git用户



4.2.3 编辑里面的内容：

|  |
| --- |
| EJECT=0  JAVA\_HOME=/usr/local/java8  BASE\_PATH=$(cd `dirname $0`; pwd)  while read oldrev newrev refname; do  #上一个版本 logid  echo 'old version:'${oldrev}  #当前版本 logid  echo 'new version:'${newrev}  #当前提交的分之  echo 'branc:'${refname}    #文件删除  if [ "$oldrev" = "0000000000000000000000000000000000000000" ];then  oldrev="${newrev}^"  fi    #对比文件，取更新的文件  FILES=`git diff --name-only ${oldrev} ${newrev} | grep -e "\.java$"`  if [ -n "$FILES" ]; then  # 临时路径  TEMPDIR=$BASE\_PATH/"tmp"  # 移更新的文件到临时文件目录  for FILE in ${FILES}; do  mkdir -p "${TEMPDIR}/`dirname ${FILE}`" >/dev/null  git show $newrev:$FILE > ${TEMPDIR}/${FILE}  done;    #仅取main目录下的文件，排除test目录下的文件  MAIN\_JAVA\_PATH=$TEMPDIR'/src/main'  #echo 'Temp update files path: '$MAIN\_JAVA\_PATH  #FILES\_TO\_CHECK=`find $MAIN\_JAVA\_PATH -name '\*.java'`  #echo 'Check files:'${FILES\_TO\_CHECK}  echo 'Aliyun p3c-pmd check starting.....'  #echo 'Current shell Path:' $BASE\_PATH  #echo 'JAVA\_HOME:' $JAVA\_HOME  #echo 'Root directory for java sources: '$MAIN\_JAVA\_PATH  # 执行p3c-pmd检查  $JAVA\_HOME/bin/java -Dpmd.language=en -cp $BASE\_PATH/p3c-pmd-2.0.0.jar net.sourceforge.pmd.PMD -d $MAIN\_JAVA\_PATH -R rulesets/java/ali-comment.xml,rulesets/java/ali-concurrent.xml,rulesets/java/ali-constant.xml,rulesets/java/ali-exception.xml,rulesets/java/ali-flowcontrol.xml,rulesets/java/ali-naming.xml,rulesets/java/ali-oop.xml,rulesets/java/ali-orm.xml,rulesets/java/ali-other.xml,rulesets/java/ali-set.xml -f text  REJECT=$?  echo 'Aliyun p3c-pmd check end...'  #REJECT=1  echo $REJECT  rm -rf $TEMPDIR  fi  done  exit $REJECT |

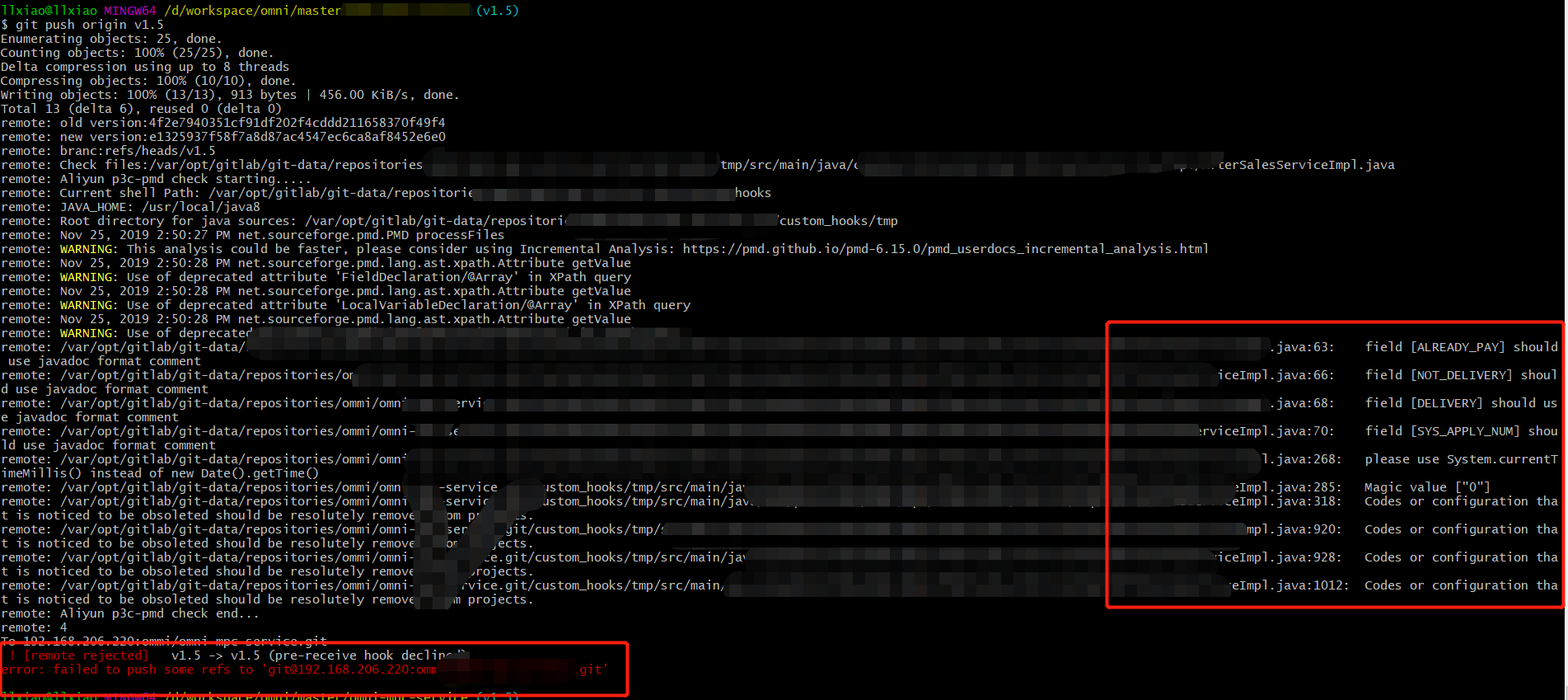
参考文件pre-recive文件和jar:



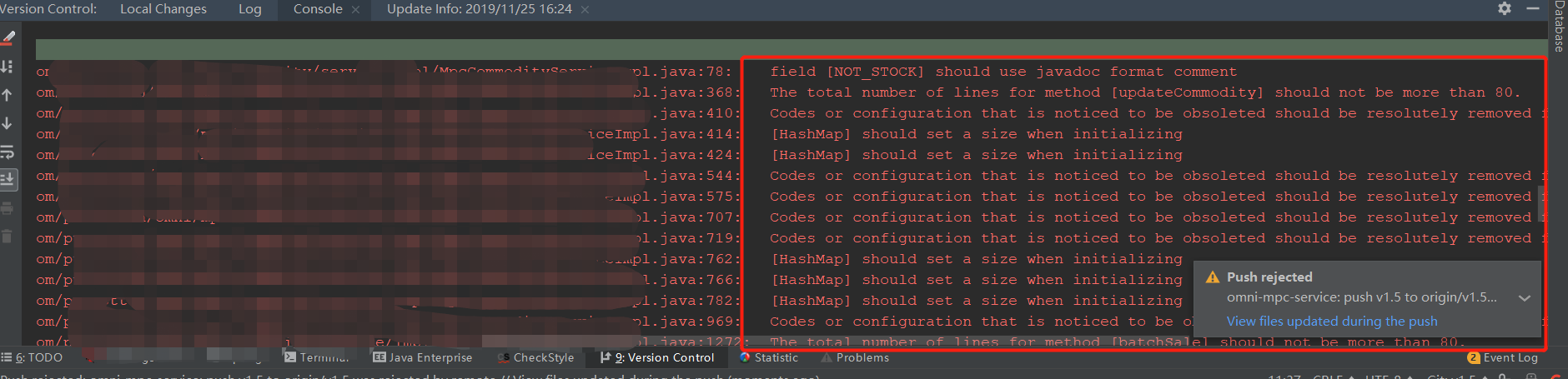
***注意的坑：JAVA\_HOME的设置、BASE\_PATH对比更新后的待检查临时文件的目录、以及Dpmd.language=en参数设置成英文(暂无法处理返回到客户端乱码)，以上两个环境变量配置错误需要自己调试该这个校验的命令***

## 4.3 客户端提交测试

Git bash提交测试服返回结果：



IDEA 返回结果：



修改所有返回错误后再次push返回成功的