目录

[修改代码 2](#_Toc459384585)

[Case1. 修改代码时，每个需求单独分支，然后在提交的时候分别提交，可以保证多个需求之间的修改不冲突 2](#_Toc459384586)

[Case2. 多个需求同时操作 3](#_Toc459384587)

[Case3. 代码的编辑历史 3](#_Toc459384588)

[Case4. ADD代码遇到的CASE 4](#_Toc459384589)

[Case5. 误操作git reset –hard后如何恢复 4](#_Toc459384590)

[提交代码 5](#_Toc459384591)

[Case1. 补充提交 5](#_Toc459384592)

[Case2. 修改提交前几次提交的commit message. 5](#_Toc459384593)

[Case3. 合并多条提交信息 6](#_Toc459384594)

[同步代码 7](#_Toc459384595)

[Case1. 本地有未提交的修改导致pull失败 7](#_Toc459384596)

[Case2. 两种同步代码的区别 7](#_Toc459384597)

[Case3. 同步代码时遇到冲突情况怎么办？ 9](#_Toc459384598)

[PUSH 11](#_Toc459384599)

[Case1. 上传前的检查 11](#_Toc459384600)

[Case2. Miss-ChangeId 11](#_Toc459384601)

[Case3. repo upload fail 11](#_Toc459384602)

[其它 11](#_Toc459384603)

[1. 映射远程分支 11](#_Toc459384604)

[2. 小技巧 12](#_Toc459384605)

[3. 提交Commit Message 12](#_Toc459384606)

[4. 如何避开Config文件修改冲突 12](#_Toc459384607)

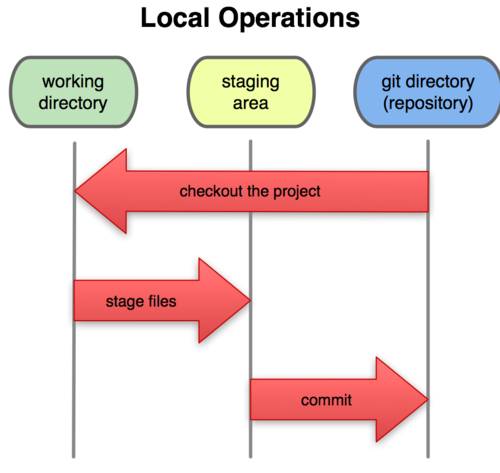
[总结 13](#_Toc459384608)

**需要补充的：**

**Git exclude 去除不相干的问题**

**Git diff**

**首先介绍一个git的仓库结构**



总结了工作中遇到的14种情况，几乎每种case都有人遇到过，主要通过：[修改代码](#_修改代码)，[提交代码](#_提交代码)，[同步代码](#_同步代码)，[PUSH](#_PUSH)等几个过程来分析每个case。

**重点：同步过程解决代码冲突，PUSH过程中FAIL的情况**

# 修改代码

## 修改代码时，每个需求单独分支，然后在提交的时候分别提交，可以保证多个需求之间的修改不冲突



经常会遇到一些case。

1. 多个需求同时操作

修改某个需求A的时候，会有些紧急的需要B要处理，但是由于前一个需求A能不完善，会响应到B功能的实现。

按以往的经验，可能会把这些修改的文件，Copy到单独的目录，等B需求完成后，再COPY回来，接着处理。

**灵活使用GIT分支来解决：**

创建FeatureA，当遇到突发情况时，临时add/commit 该分支的修改。

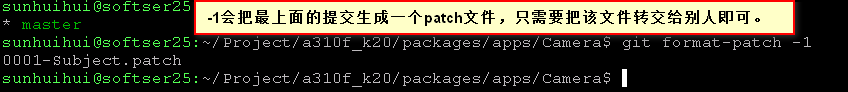
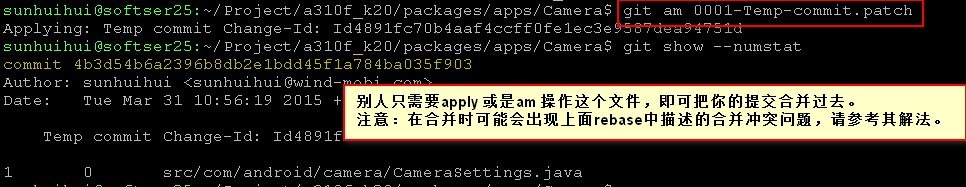
再从PDU3下另切分支FeatureB，单独在FeatureB分支修改提交，最后再合回到PDU3来提交。

**相关GIT命令：**

git checkout –b

git cherry-pick

**Format-patch可以使用git format –number， git apply, git am**

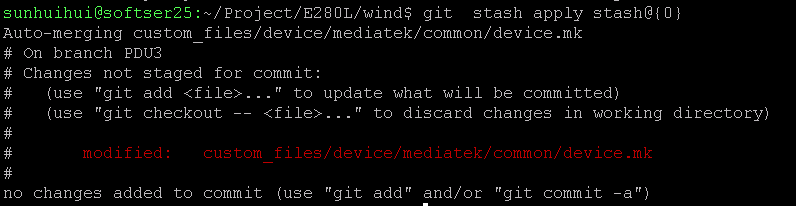


**使用Stash命令**

1. **git stash 临时将本分支的修改压入栈中**



1. **等FeatureB修改完成后，再从栈中把之前临时修改的内容取出来。**

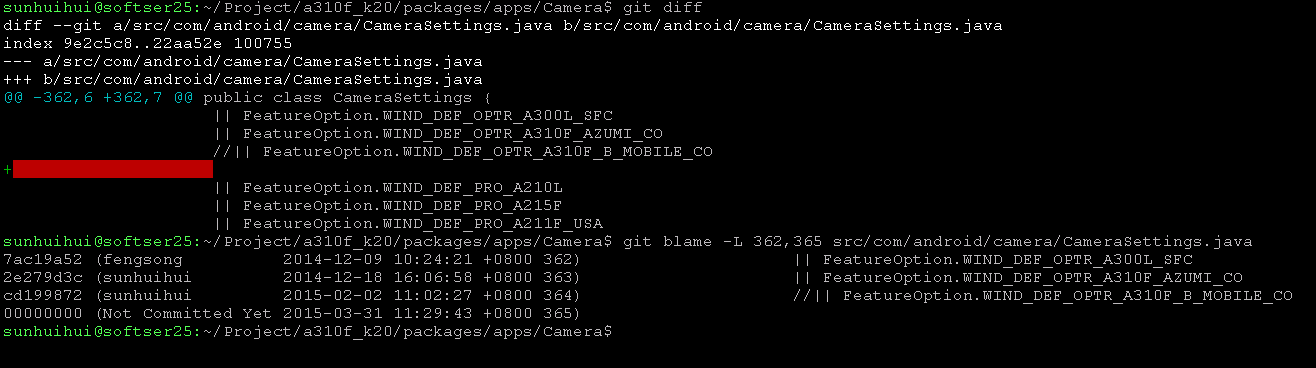


## **代码的编辑历史**

代码中有几行代码的情况需要知道是哪笔提交的，而不是整个文件的情况，毕竟git log file这样会产生很长的修改历史。

**Git命令：**

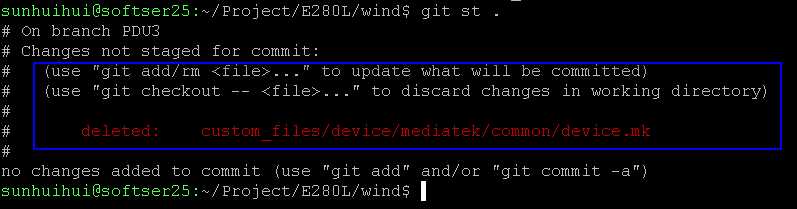
**git blame –L from, to file 这个命令很给力，也是经常要用到的。**



1. ADD代码遇到的CASE

git add –A 当遇到删除大量文件时，需要把删除的文件，从git 仓库中remove掉

需要调用git rm filePath.



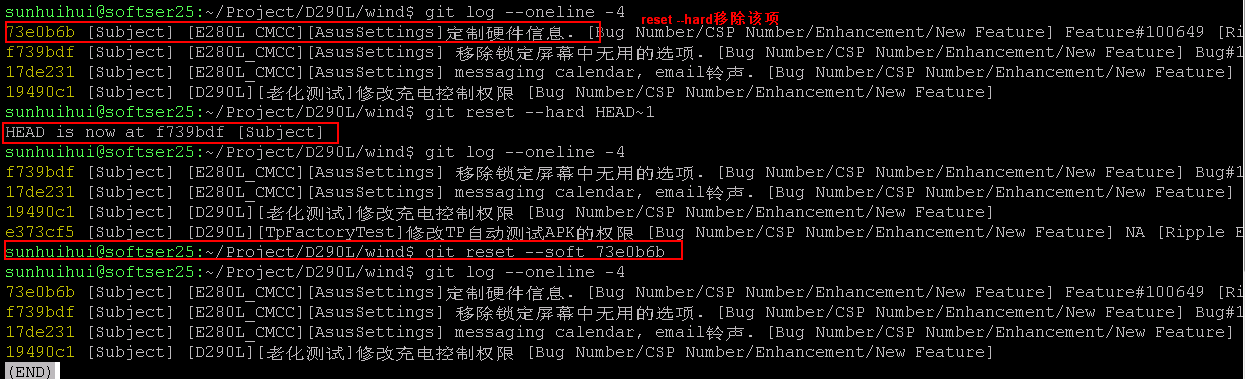
**git命令：**

**git add -A**

1. 误操作git reset –hard后如何恢复

**reset –hard误操作之后，如何恢复**

**git reset –soft SHA-1 这种情部适用于，知道上次reset的SHA-1 ID.**



**另一种情况适用于不知道撤消的commit ID。//TODO**

# 提交代码

提交代码注意事项：

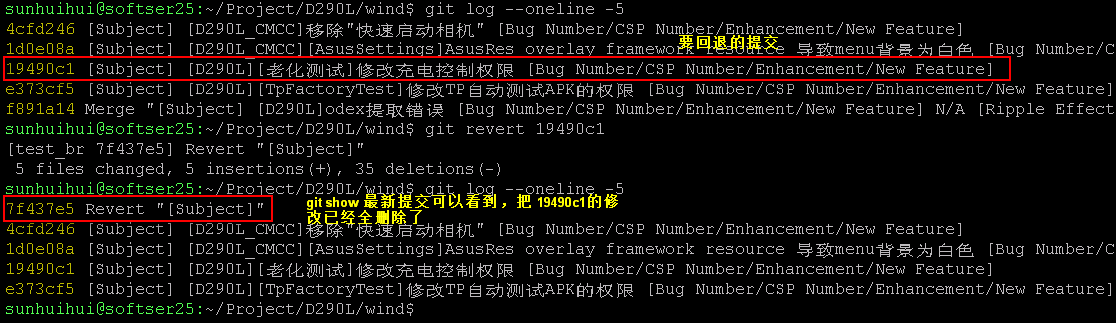
一个BUG和需求，一笔提交，并且尽可能的保证提交的完整性。

**优点：**别的项目需要的话，可以直接把这个patch一次性合过去。参考git format-patch

**回退代码：**

大部分同事都是手动删除新加的代码，然后重新提交。

命令:revert



**注意**：这里会遇到一些冲突的问题

比如1个月前的某笔提交commitA， 现在需要把commitA回退掉，但是1个月中间有几个笔记修改了commitA的内容。

此时调用revert就会出现冲突的情况

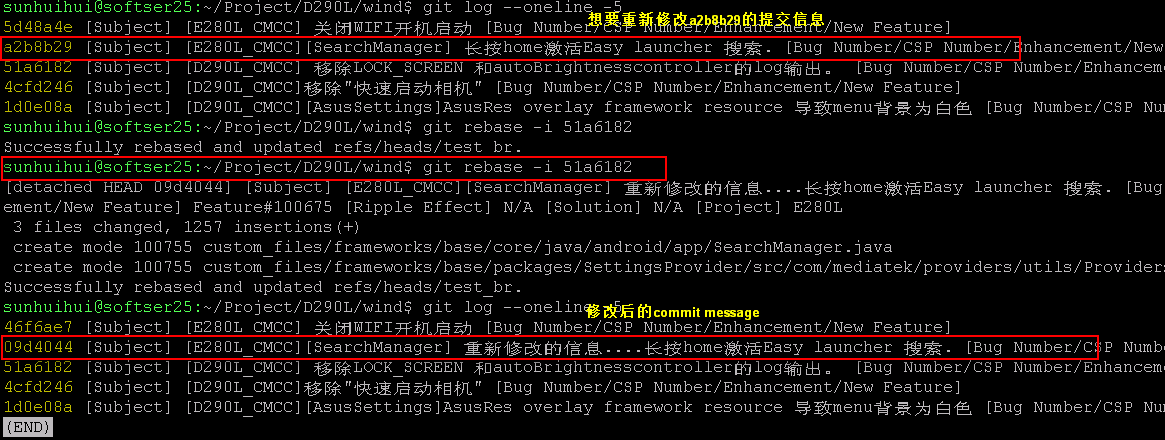
1. 补充提交

git commit –amend使用上次提交的commit信息

使用上次的提交信息，保证提交的一致性和完整性

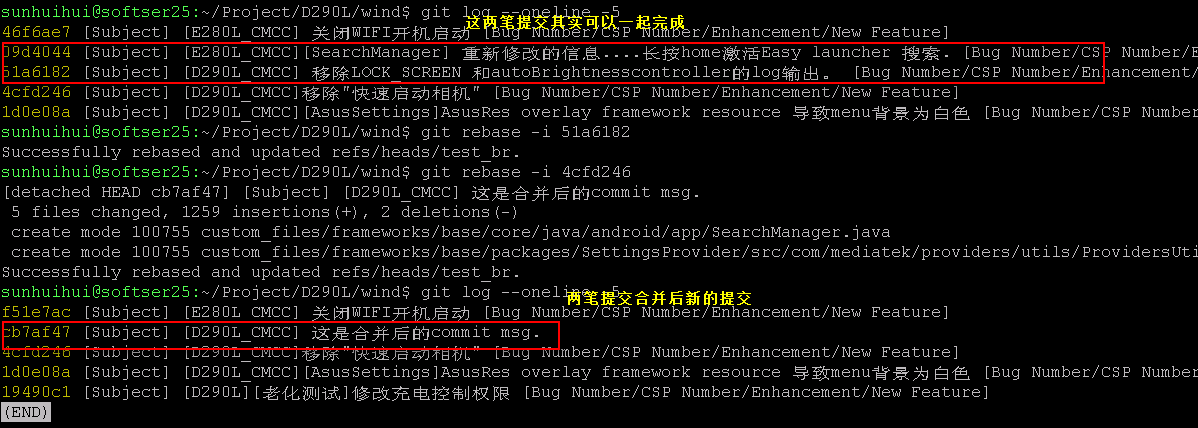
1. 修改提交前几次提交的commit message.

Git命令:git rebase –i commit SHA-1

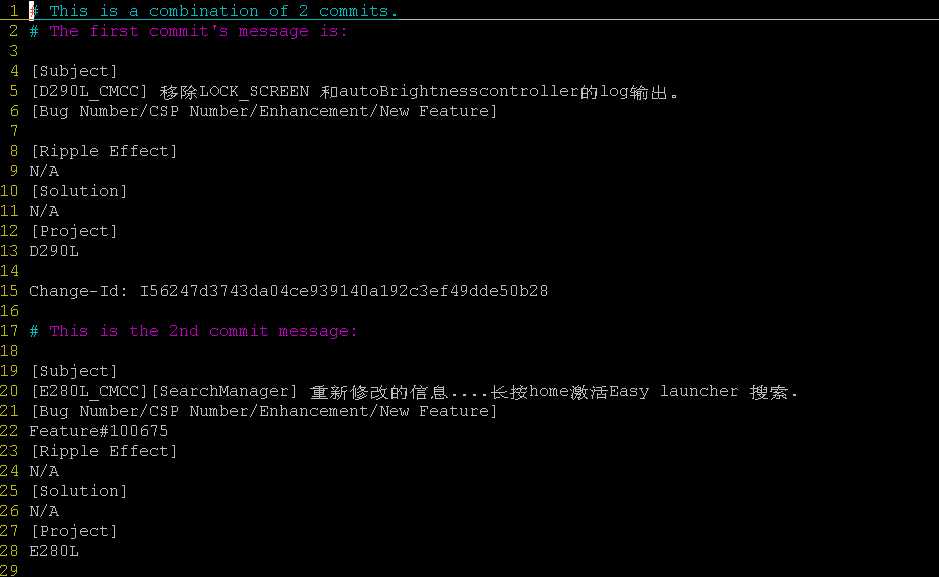


## **合并多条提交信息**

Git命令:git rebase –i commit SHA-1



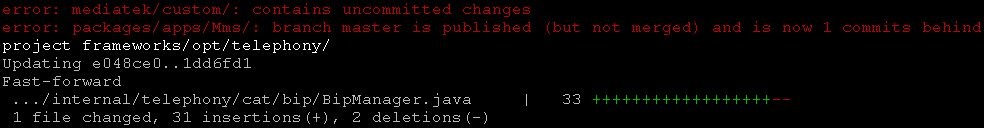
**操作步骤如下：**



# 同步代码

1. 本地有未提交的修改导致pull失败

repo 同步代码时经常会遇到这个问题，这是由于在拉代码时，本地有未提交的修改，就会导致这部分更新失败.



**解决方案**：看到红色异常后，对每个异常部分修改。

**推荐方案**：可能会有N个工程都会失败，推荐一个repo 命令

**Repo froall –c “git command” 该指令其它的功能，自行探索。**

该功能是指对repo 中manifest.xml文件中描述的所有工程都执行指定的git 命令,针对这个问题可以这样使用:

Repo forall –c “git checkout .” 对repo下所有的git工程，把本地修改都撤消。

**注意：使用该指令时，请确保有用的部分自行保存，该指令会把所有修改都回退掉。**

**这里会用到一个新的git 命令:**



1. 两种同步代码的区别

下面这个提交记录是比较清晰的，很容易看出每次提交的commit信息和ID

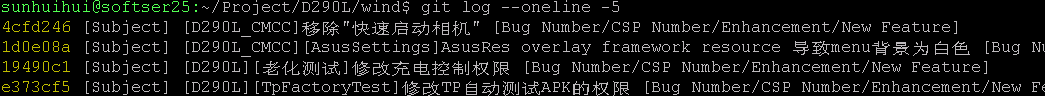


Figure 1

**但有时候会遇到下面的情况**

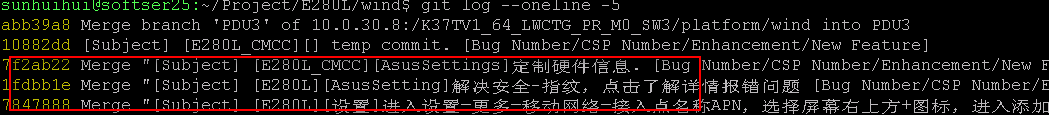
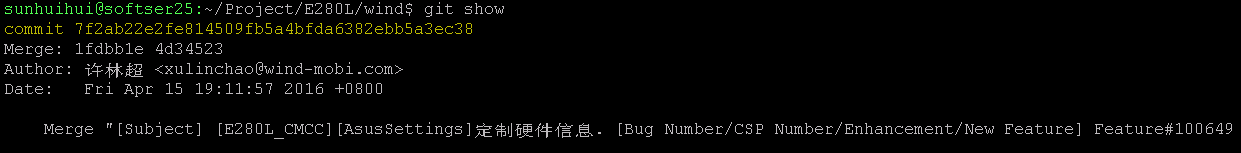


Figure 2

**Figure1:** 里面的每一笔提交git show 都可以明显看到修改的内容

**Figure2:** git show显示结果



**以上情况出现的原因：**

git pull 和git pull –rebase都是从服务器拉取最新代码，但是稍微有点区别。



**如何避免这种情况出现：**这种情况的产生，需要自已本地提交时，尽量使用git pull –rebase

服务器的代码在merge人员操作时

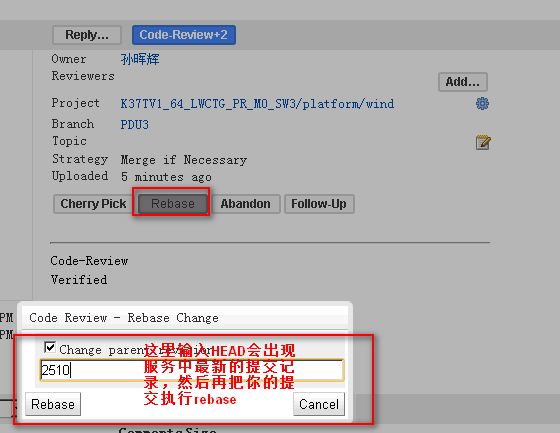




**有时候会遇到，明明自己在提交前已经pull –rebase了，提交完了，依然还是会有merge记录**

**原因：**当你pull –rebase完，push到服务器后，到merge人员把你的代码真正merge的时间中，有别人的代码提交入库了。

**如何避免这种情况：**



**以上过程中还会出现另一个问题：merge失败**

在merge代码之前，有别人和你修改了同一个文件

**总结：**

Git pull –rebase 产生的Log提交记录要比git pull 看起来更加的直观明了，推荐此方法。

看看git pull –rebase的变化，这里有行log ，意思是把本地提交放到提交的最上面。



1. 同步代码时遇到冲突情况怎么办？

**经常在更新代码时，repo sync 或是 git pull (--rebase) 时会遇到修改冲突的问题，这是因为服务器他人的提交和本地修改了同一行导致的，因此在合并时电脑傻傻的不明白该怎么办了。**



图 1

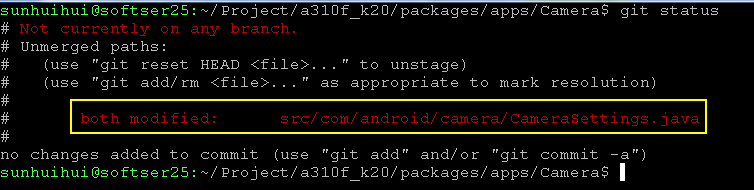


图 2



图 3



图 4

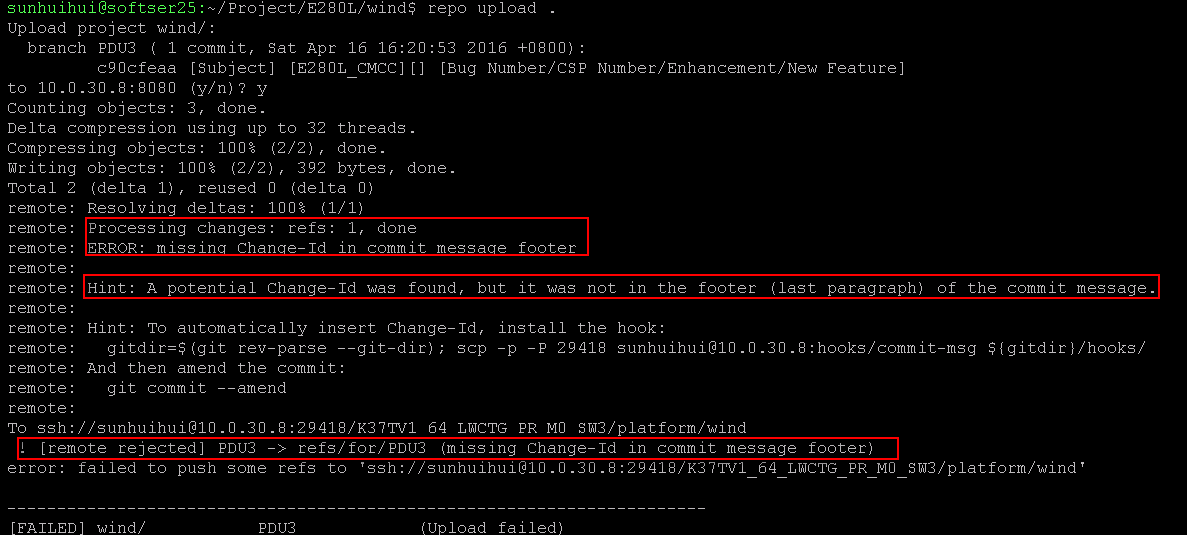
# PUSH

1. 上传前的检查

push前最好自己git show 查看当前修改，确保修改内容的style和正确性

**Git show 可以查看指定提交的修改内容，但有时候不需要关注修改了哪些东西，只需要知道修改的文名。这时可以使用git show –numstat**



1. Miss-ChangeId
2. repo upload fail

可能是编码错误，这个情景无法复现；有人遇到了及时提醒然后把问题总结一下。

# 其它

## 映射远程分支

git branch –set-upstream local\_branch remote\_branch\_name

## 小技巧

编辑在framework/base及其它目录下的代码时，可以先把本地的git add 并且commit，然后再编辑。

**好处：**

在编辑完成时，此次修改可以明显的看到，并且可以灵活的使用git 来控制此次的修改，不会与以前的修改冲突。

**缺点：**

合MTK patch，repo sync 后，临时提交的东西可能会与服务器的冲突造成sync失败，遇到这种情况，只要回退掉temp commit重新pull对应目录即可。

## 提交Commit Message

当提交commit Message时需要把修改的文件前面#解开，有经常会遇到修改了大量文件，如何快速操作?

Vim:

Linefrom ,lineTo s/src/des/g

lineFrom, lineTo d

## 如何避开Config文件修改冲突

因为Config文件，经常会被多个人修改，这样经常会出现config文件修改冲突导致merge失败。

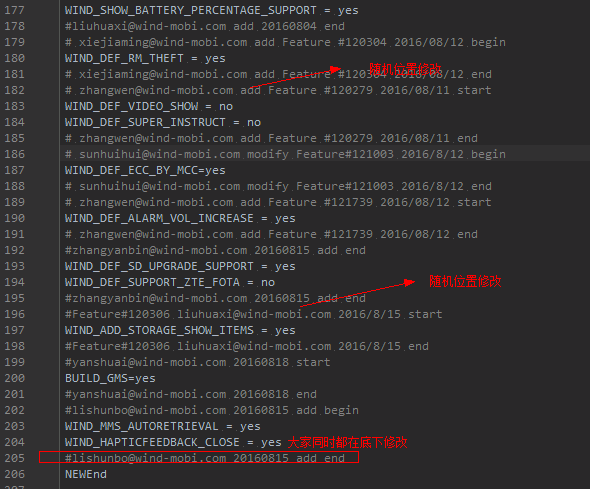
给大家提供个小技巧，可以降低冲突的概率

冲突原因：

冲突大部分原因是在底部添加新宏导致的，几个人同时在底部加了新宏，那么一定会出现冲突。

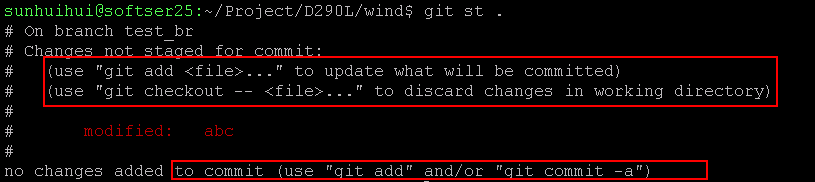
解决方案：

基于这个原因，比如A在底下加了宏，B加宏的时候，可以避开底部，添加到CONFIG的中间，这样就可以避开冲突。

如果大家都在中间位置随机添加，那么就可以大大降低config文件冲突的概率

# 总结

几乎每一步git 操作，都会有相应的提示（如下图），多观察操作后的输出内容，很快就可以运用自如。



Figure