Gerrit使用规范

1.分支管理规范

本机master上的代码坚决不动,开发的代码永远在本机切换新的分支进行新的业务开发(只让master保持跟服务器master上的同步，能省去很多不必要的问题)。下面是一个正常情况下的栗子🌰

git checkout master

git pull #拉远端最新版本代码到本机

git checkout -b dev #假设你要创建一个叫dev的分支进行开发

git rebase master #合并远端最新代码到开发的分支（常见是在gerrit后台review后merge时发生冲突后需要本机rebase后再次追加提交）

#如果有冲突（rebase后会有提示），手动解决冲突后，进行以下操作

git add -A

git rebase - -continue

#本地代码已经可以提交到gerrit，执行以下push操作，（注意不能直接像以前那样push到远端的master上，而是一个特殊的gerrit分支）

git push origin HEAD:refs/for/master

如果你在master上status出现了以下情况：

(master) git status

On branch master

Your branch is ahead of 'origin/master' by 2 commits.

(use "git push" to publish your local commits)

nothing to commit, working tree clean￼

出现这个说明没有严格按照以上规范，这时需要通过git reset 进行回滚后再pull（回滚前确保这上面的代码再gerrit上已有，已有的分支可以checkout下来继续在个人分支上继续开发）

2.commit规范

a.多个开发任务并行时，不要都提交在一个commit里（相互有依赖的可以）

b.需要在gerrit中同时维护多个未merge的分支（同时提交多个commit，或者上一个提交因为一些原因还没有merge又需要再提交一个新的），对于每个分支均从master上checkout一个新的分支进行开发，不在一个开发分支里commit多个，这样能最大程度避免依赖。避免出现你的后提交的代码需要上线，但所依赖的代码还不能merge的情况。

c.提交的subject规范：1.格式：“关键字 正文” 2.关键字，固定以下几个(如有新的需要定义找CTO协商)：added(新增,简写add)、fixed(修复,简写fix)、changed(修改原来的业务逻辑,简写change或cg)、upgraded(优化或组件升级,简写upgrade或up) 3.正文：解释是什么和为什么的 。例如：“cg 同步数据接口增加几个表单项”，“fix 同步数据接口缺乏必要校验”

3.所有提交必须review代码再入库

我们使用gerrit的最大的好处，就在与可以review代码，可以看到我们本次提交所涉及的改动，对比老代码能更容易发现本次改动的疏漏或bug。

因此提出建议规范：自己的代码无论改动多少，+2前必须先给指定人review+1（组员的代码给组长review，组长的代码自己指定组内他人review）。自己review可以在研发过程中的任何时候，不一定要等到开发完了再review。