分支管理与使用说明

(IDEA + GIT)

1. 简单说明：

Git 强大的分支能力（远超 CVS/SVN），能很好的应对团队并行开发，调联，测试，bug 修复等各种场景。能有效的降低团队成员之间的沟通成本与交叉影响。是目前使用最为广泛的分布式版本管理软件。作为开发、测试、项目管理人员会用、用好 Git 有着极大的意义。

2. 分支策略与规范：

参考开源软件的分支管理模式，并根据情况做适当简化：

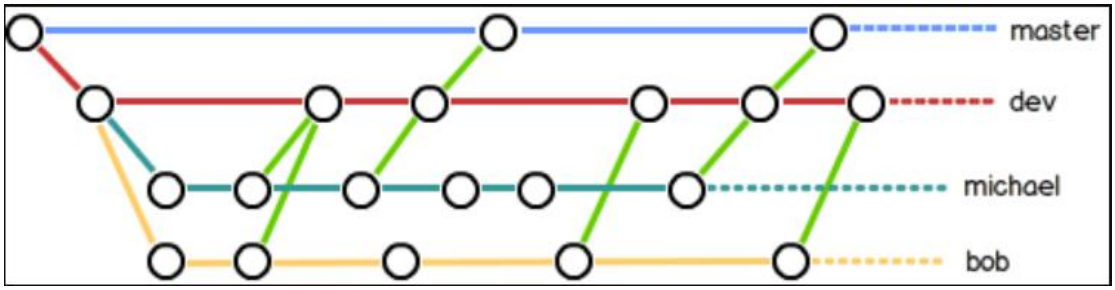
主分支	master	正式版本，都在这个主分支上发布
开发分支	develop	开发分支，永远是功能最新最全的分支
功能分支	feature-*	新功能分支，某个功能点正在开发阶段
修复分支	bugfix-*	修复线上代码的 bug

如何命名:

主干分支: master（由配置管理员创建）
开发分支: develop（由配置管理员创建）
特性分支：名称要以 feature-{中心/模块名}-特性名，例如：feature-order-createorder
修复分支：名称要以 bugfix-{中心/模块名}-修复名，例如：bugfix-base-pwdsafe-issue

【主分支 master】

实际开发中，一个仓库（通常只放一个项目）主要存在两条主分支：master 与 develop 分支。这个两个分支的生命周期是整个项目周期。就是说，自创建出来就不会删除，会随着项目的不断开发不断的往里面添加代码。master 分支是创建 git 仓库时自动生成的，随即我们就会从 master 分支创建 develop 分支，如下图所示。



master：这个分支最为稳定，这个分支代表项目处于可发布的状态。

例如王二狗向 master 分支合并了代码，那就意味着王二狗完成了此项目的一个待发布的版本，项目经理可以认为，此项目已经准备好发布新版本了。所以 master 分支不是随随便便就可以签入代码的地方，只有计划发布的版本功能在 develop 分支上全部完成，而且测试没有问题了才会合并到 master 上。

【开发分支 develop】

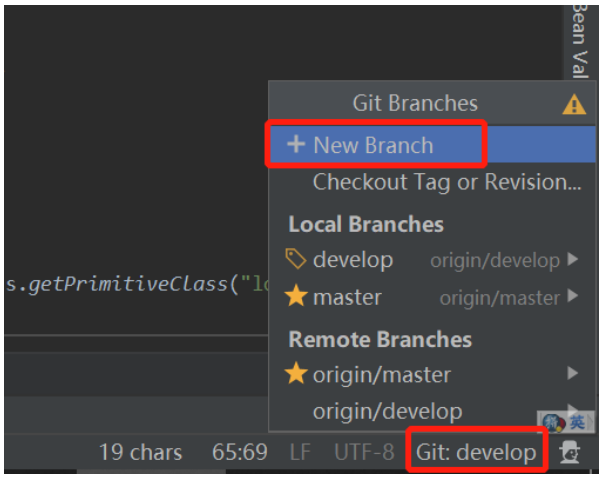
作为开发的分支，平行于 master 分支。

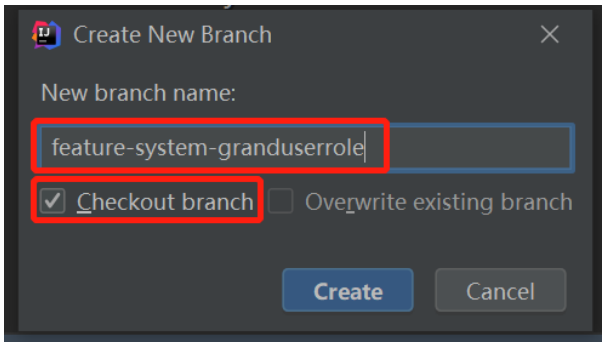
例如王二狗要开发一个注册功能，那么他就会从 develop 分支上创建一个 feature 分支 feature-register（后面讲），在 feature-register 分支上将注册功能完成后，将代码合并到 develop 分支上。这个 feature-register 就完成了它的使命，可以删除了。项目经理看王二狗效率很高啊，于是："二狗你顺带把登录功能也做了吧"。二狗心中暗暗骂道：日了个狗的，但是任务还的正常做，二狗就会重复上面的步骤：从 develop 分支上新创建一个名为 feature-login 的分支，喝杯咖啡继续开发，1 个小时后登录功能写好了，二狗又会将这个分支的代码合并回 develop 分支后将其删除。

【特性分支 feature】

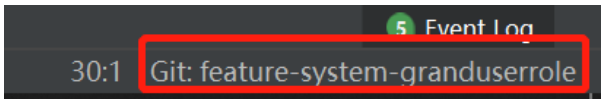
特性分支是为了程序员协同开发，以及应对项目的各种需求而存在的。这些分支都是为了解决某一个具体的问题而设立，当这个问题解决后，代码会 merge 回 develop 后删除。必须从 develop 分支创建，完成后合并回 develop 分支。

开发人员根据开发的功能点从 develop 分支创建一个 feature 分支。

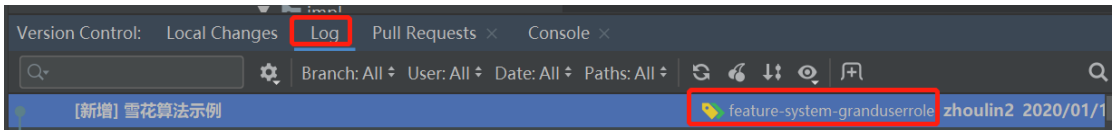




创建完成后，右下角会显示已经切换到最新创建的特性分支

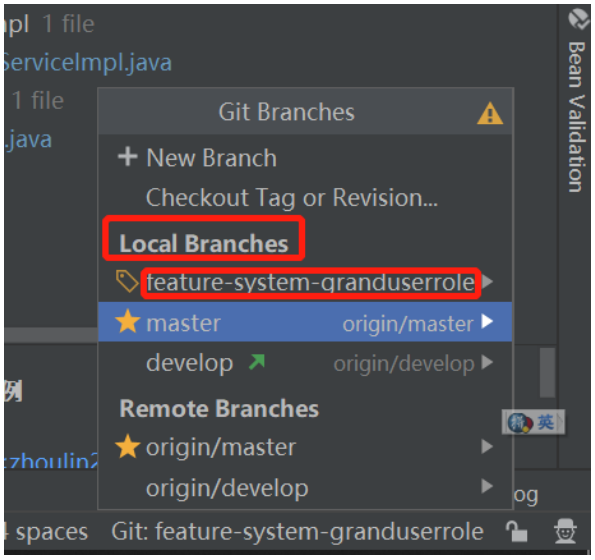


开发过程中新增、删除、修改的文件直接 commit 到特性分支上。



注：提交到特性分支。

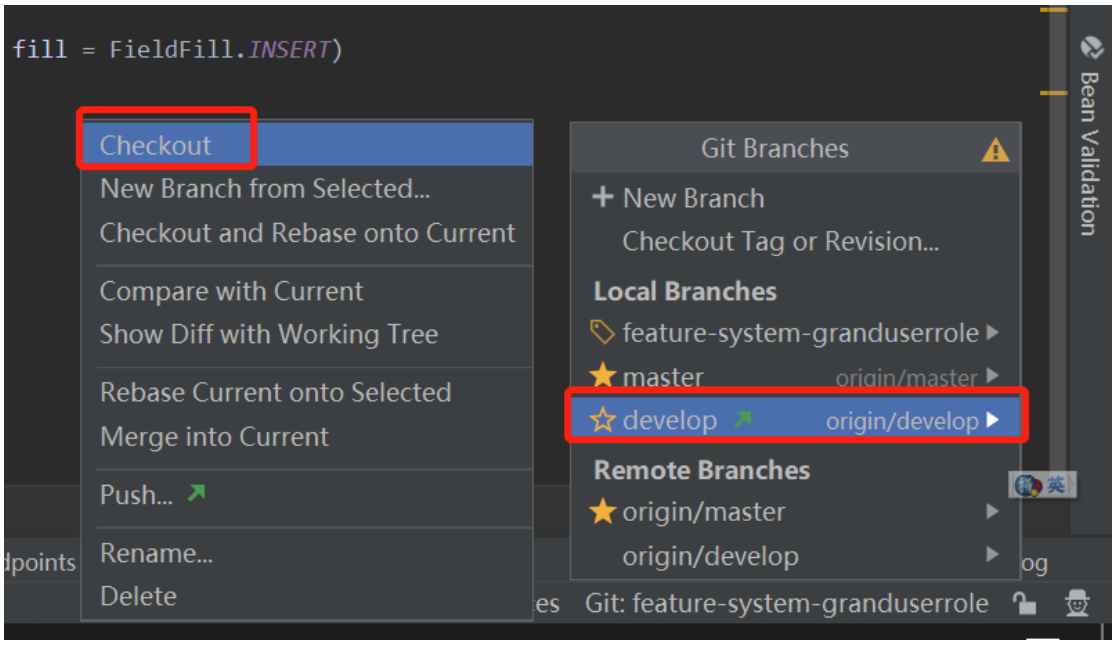
如果该特性不会与其它开发人员联调，特性分支仅保留在本地仓库，无需 push 到服务端，确保服务端分支结构的简单清晰。



注：feature-system-grantuserrole 分支仅存在于 Local 端，Remote 端（即别人）是看不到的。

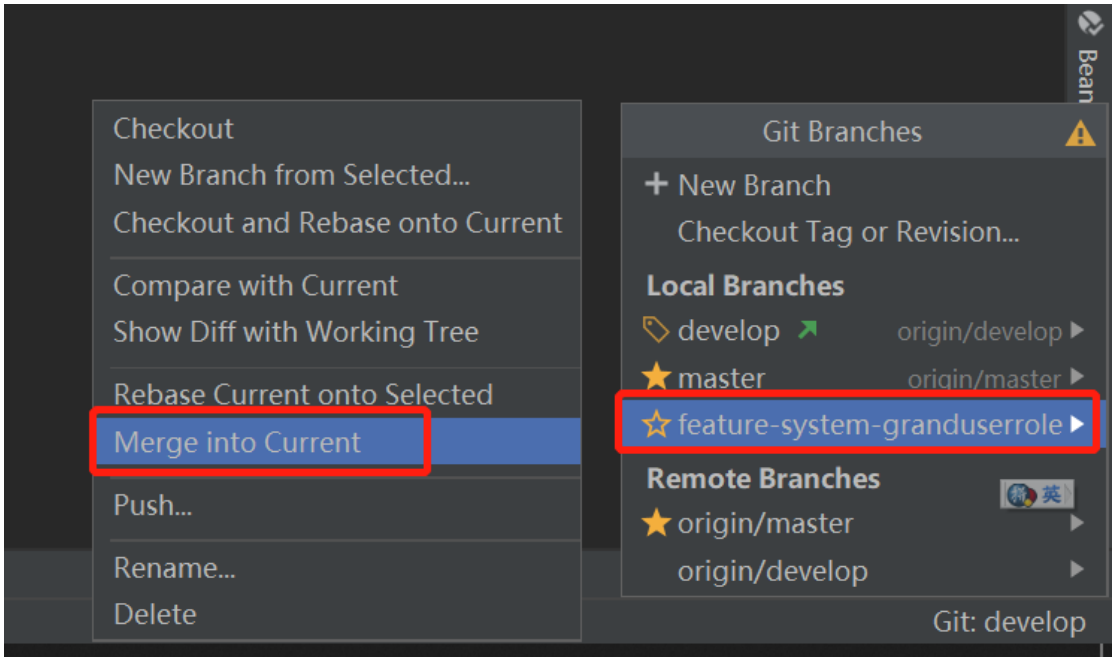
当功能点开发测试完成，测试通过，合并到 develop 分支去，可以将 feature 分支删除。

第一步：切换到 develop 分支



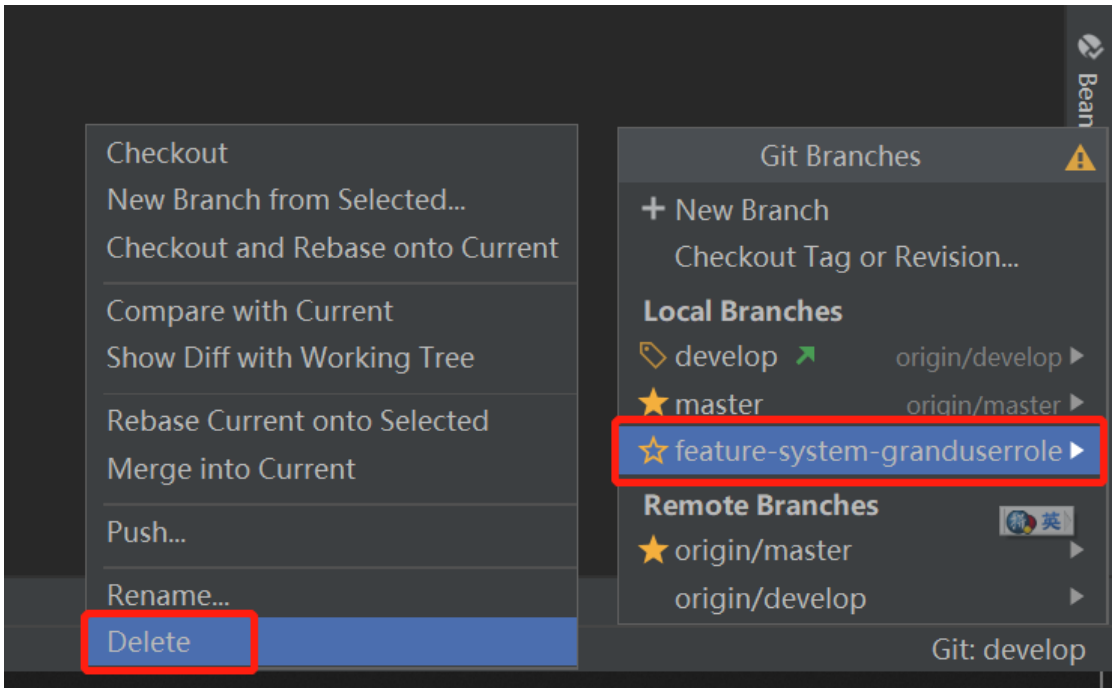
切换成功后，右下角显示当前在 develop 分支里

合并特性分支到 develop 分支之前，建议先对 develop 分支做 pull 更新操作，然后在 Merge into Current。

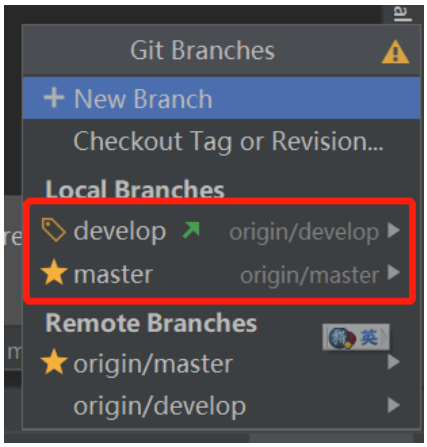


如果没有冲突，feature-system-grantuserrole 中的代码就会被合并到 develop 分支上了，合并成功后，需要 push 才能推送到远程仓库。

合并成功后，建议把特性分支删除，确保分支的干净清晰，因为特性开发完成后，已经没有存在的价值了。



删除后，在 Local 端已经看不到特性分支了。

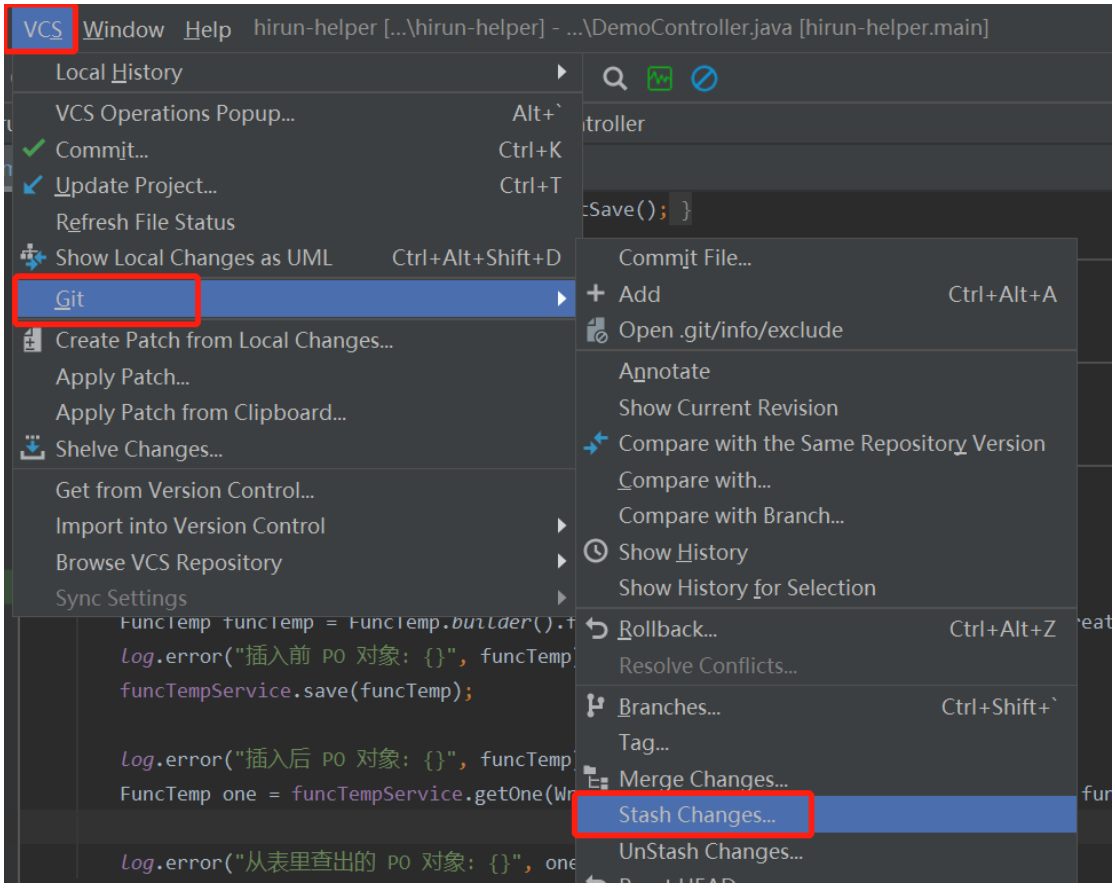


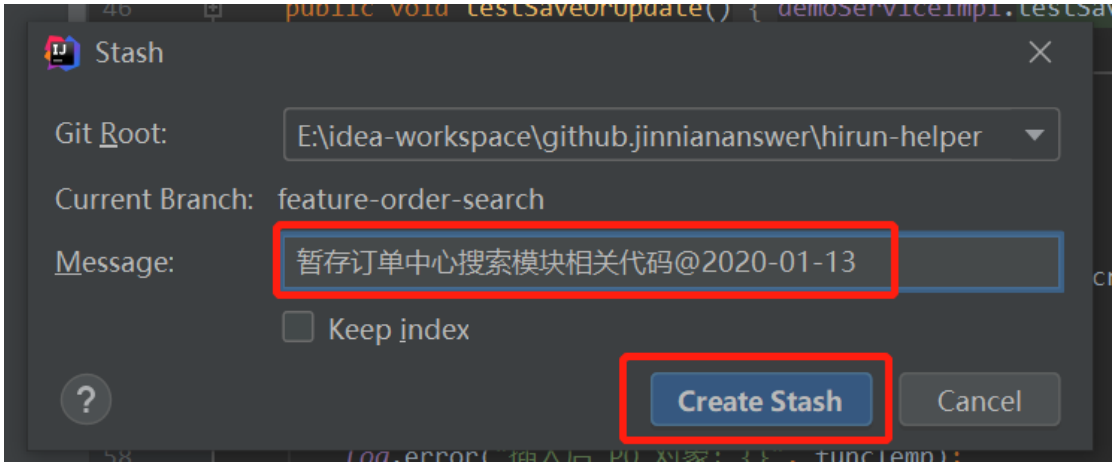
当需要发布一个版本来测试时，测试人员取最新的 develop 分支，进行集成测试。测试通过，将 develop 分支合并到主干，生产环境直接基于主干发布。

【修复分支 bugfix】

系统上线后，难免遇到 bug，需要紧急修复。
如果手头正在某个特性分支上开发，但还未开发完，如果冒然切换到其它分支，会把当前修改的文件带过去。所以…

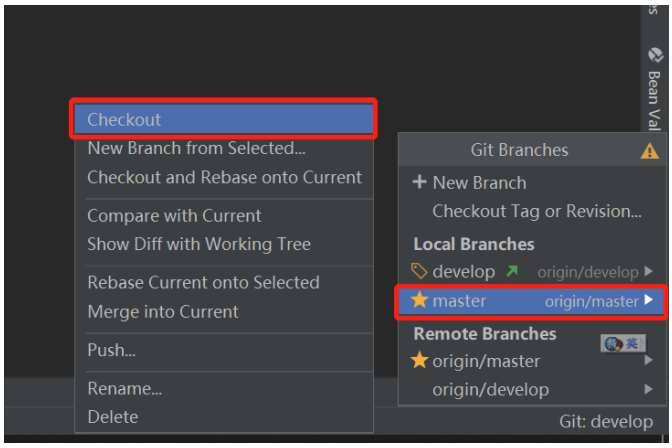
第一步：将当前修改藏起来（Stash Changes）

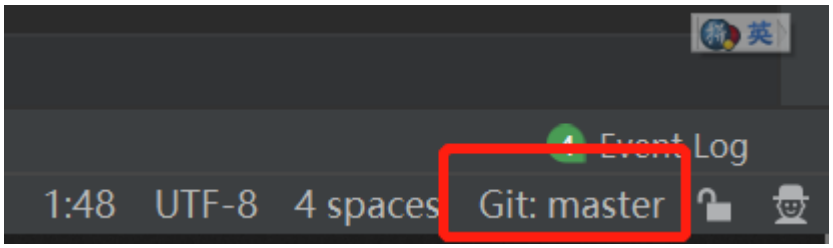




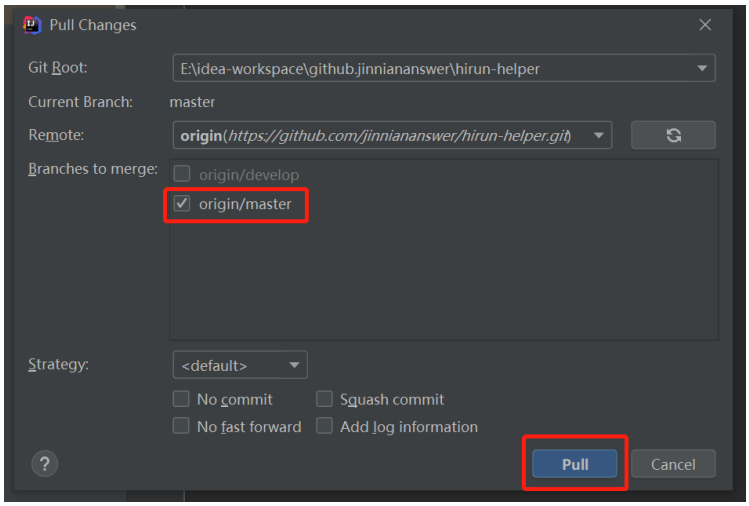
点击 Create Stash 按钮后，你会发现当前工作区内的代码恢复了原样。

第二步：切换到 master 分支：

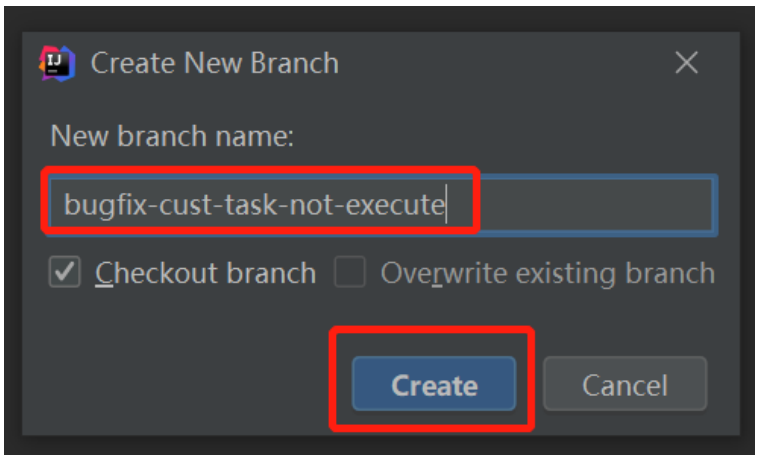


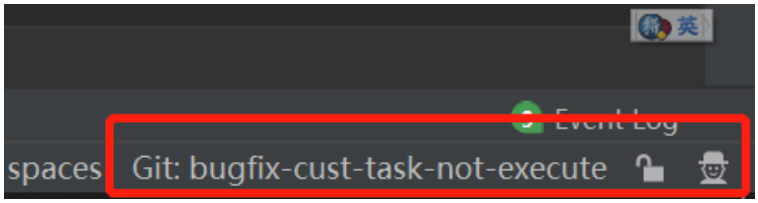


第三步：pull master 分支最新代码。



第三步：基于 master 分支创建 bugfix- 修复分支。



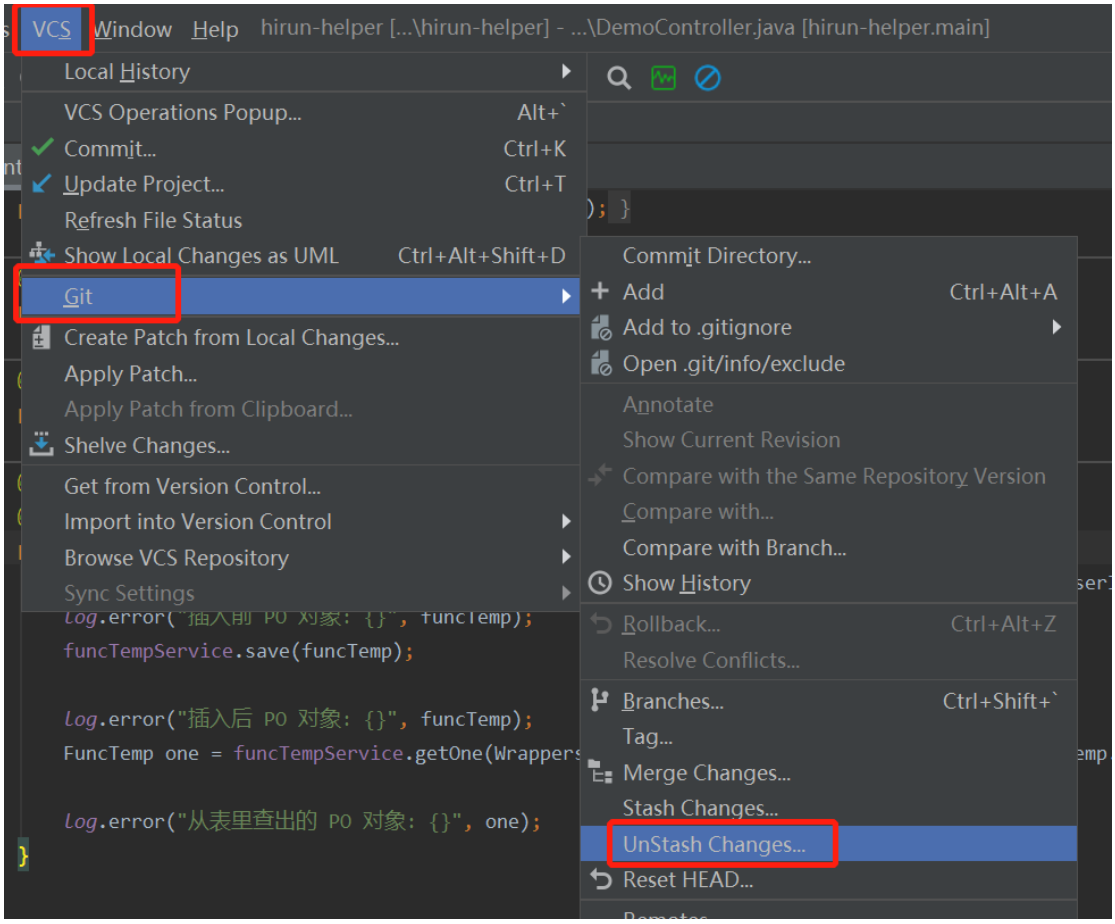


第四步:

修改的代码先 commit 到 bugfix-cust-task-not-execute 分支，再 merge 到 master 分支和 develop 分支。具体 merge 方法可以参考前面。

第五步:

如果前面有做过暂存代码（Stash Changes）操作，这时可以切换到原有的特性分支，通过恢复暂存代码（Unstash Changes）。从而完美衔接上前续工作。


VCS Window Help hirun-helper [...\hirun-helper] - ...\DemoController.java [hirun-helper.main]
Local History
VCS Operations Popup... Alt+`
Commit... Ctrl+K
Update Project... Ctrl+T
Refresh File Status
Show Local Changes as UML Ctrl+Alt+Shift+D
Git
Create Patch from Local Changes...
Apply Patch...
Apply Patch from Clipboard...
Shelve Changes...
Get from Version Control...
Import into Version Control
Browse VCS Repository
Sync Settings
Commit Directory...
Add Ctrl+Alt+A
Add to .gitignore
Open .git/info/exclude
Annotate
Show Current Revision
Compare with the Same Repository Version
Compare with...
Compare with Branch...
Show History
Rollback... Ctrl+Alt+Z
Resolve Conflicts...
Branches... Ctrl+Shift+`
Tag...
Merge Changes...
Stash Changes...
UnStash Changes...
Reset HEAD...
funcTempService.save(funcTemp);
log.error("插入后 PO 对象: {}", funcTemp);
FuncTemp one = funcTempService.getOne(Wrappers
log.error("从表里查出的 PO 对象: {}", one);