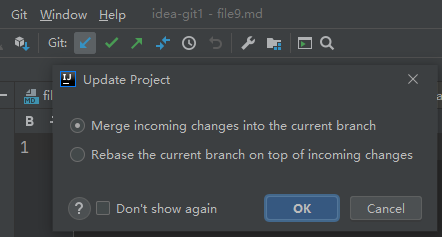
该文档主要对在Idea和Git的集成环境中，**某些容易混淆**的操作进行解释说明。

# Update Project



<https://www.jetbrains.com/help/idea/2020.3/sync-with-a-remote-repository.html#update>

If you have several project roots, or want to fetch changes from all branches each time you sync with the remote repository, you may find updating your project a more convenient option.

When you perform the update operation, IntelliJ IDEA fetches changes from all project roots and branches, and merges the tracked remote branches into your local working copy (equivalent to pull ).

Idea会拉取所有工程所有远程的更改，并合并更改到你的本地工作空间。

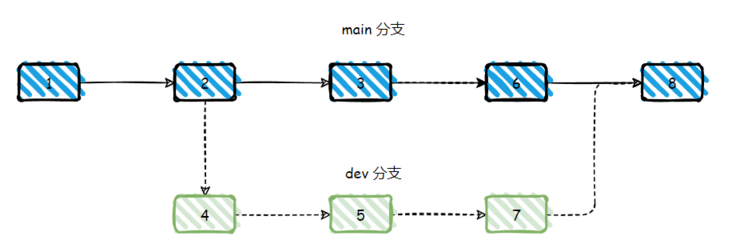
1. Git工具栏Remote文件下的分支信息，都会被更新；
2. 当前的工作空间Git Branch被与新的更改进行合并，合并方式（Merge还是Rebase）取决于你选择。

Rebase和Merge的区别，详见[Rebase vs Merge](#_Rebase vs Merge)。

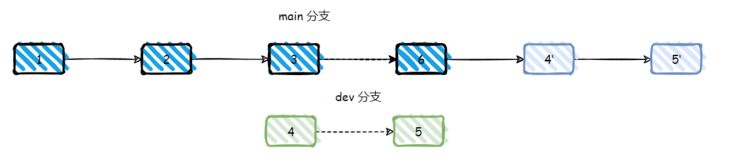
## Rebase vs Merge

一种是 Merge incoming changes into the current brance，而另一种则是 Rebase the current branch on top of incoming changes。

其中 Merge 的原理是找到两个分支的祖先 commit，然后将公共分支最新版合并到自己的分支，形成一个新的 commit 提交，用图表示如下。



Rebase 则是重新基于一个分支进行 commit，即将当前分支从祖先的 commit 后所提交的 commit 均撤销，将他们放到一个缓存中，然后基于一个分支的后面，将缓存的 commit 按照顺序以一个个新增到该分支后面，用图表示如下，将 4 和 5 所提交的 commit 放到缓存中，然后按顺序将其新增到 6 的后面。



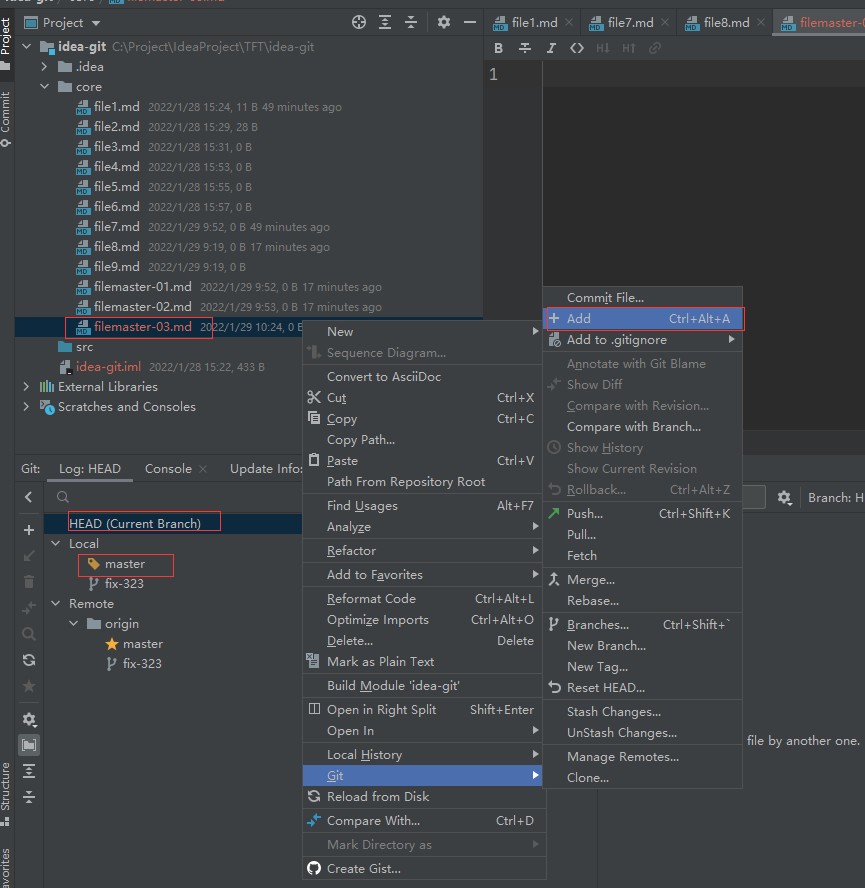
使用 Rebase 和 Merge 的基本原则可以总结如下三条：

1. 下游分支更新上游分支内容时使用 Rebase
2. 上游分支合并下游分支内容时使用 Merge
3. 更新当前分支内容时一定要使用 --rebase 参数

详细说明（解释），参考本地文档《git rebase vs merge 图文版.docx》，或链接https://zhuanlan.zhihu.com/p/34197548。

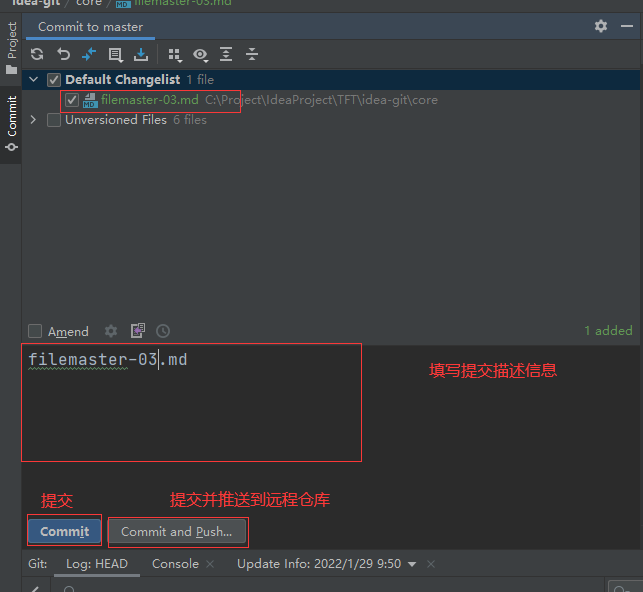
# Commit

提交代码，指将修改后的代码提交到本地缓冲（stage）区。

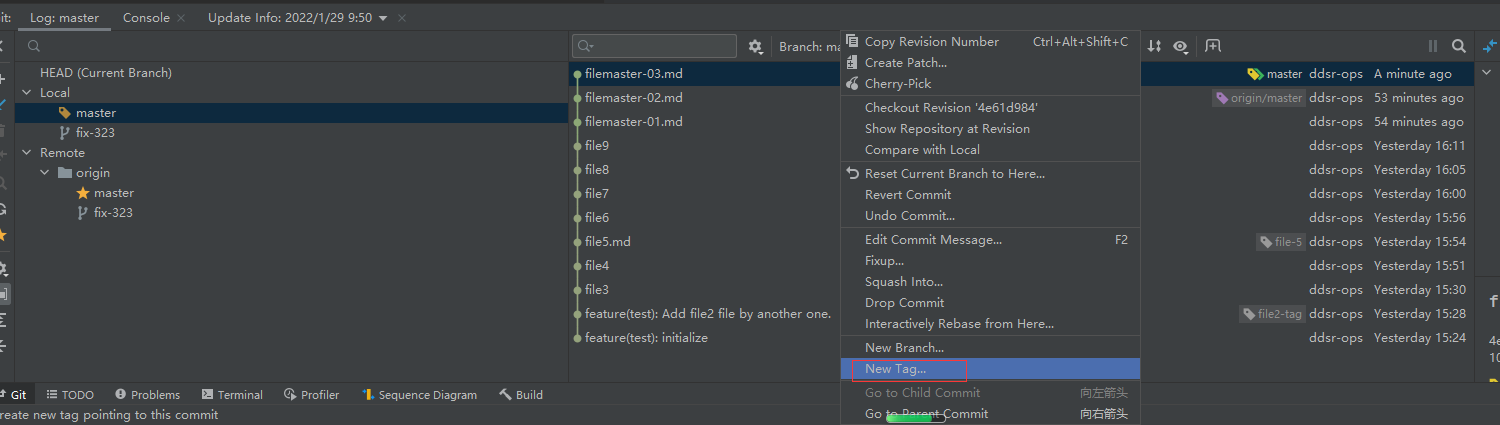


针对某个文件Add操作，Add后才能Commit到本地缓冲区（本地仓库）。Head指向当前分支，当前分支即Local-Master分支。

使用快捷键Ctrl + K, 可以快速提交到本地仓库。

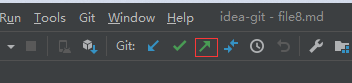


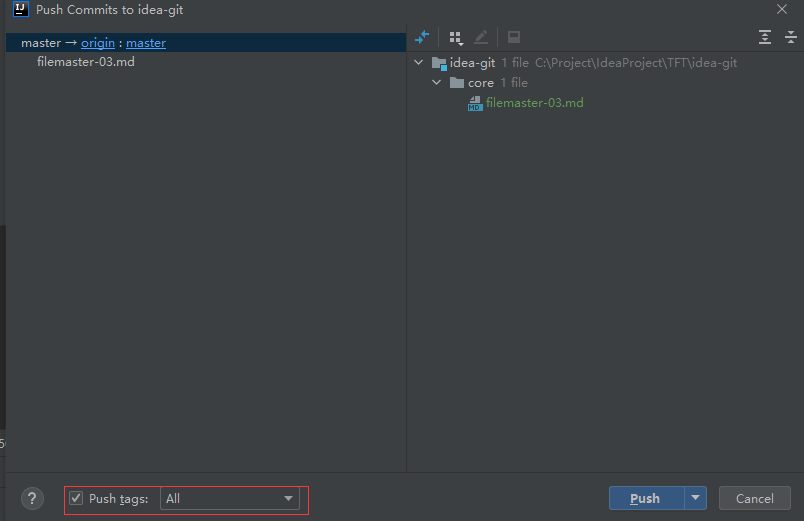
提交后，可以在本地仓库中看到提交记录，还可以针对该提交记录创建Tag。



选择后，输入Tag名称，要求Tag不能重复。

# Push

点击，或者使用快捷键Ctrl + Shift + K，Push代码至远程仓库。



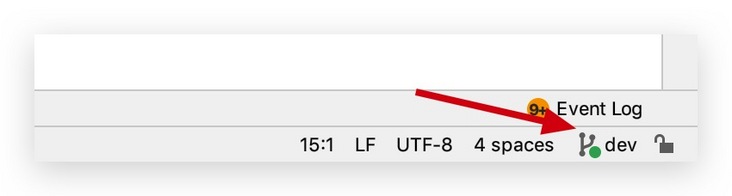
选择Push tags，tag将被同时推至远程仓库。

# Branch

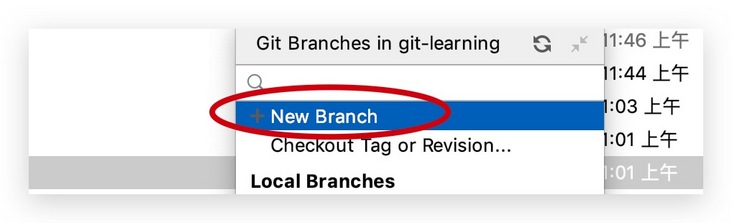
分支是git中非常重要的概念，假设没有git的情况下，我们系统上线了，在线上稳定运行，这时候你需要开发后续的功能但又想保留目前线上版本的代码，因为可能线上有紧急bug需要修复，你需要使用当时的版本进行修改，所以你可能会copy一份代码做备份，等新功能开发完，你还需要将这段时间在备份的代码上修改的代码整合过来，如果这个过程是人工进行的，非常容易出错，所以git提供了分支解决这个问题，线上运行的代码可以在主分支上，比如master分支，开发新功能可以从master分支切一个分支出来，比如dev分支，修复bug也可以从master分支上切一个分支出来，比如fix分支，当bug修复完毕，将fix分支整合(merge)到master分支上，当新功能开发完毕后，将dev分支整合到master分支，就完成了上述所有的操作。

## New Branch

在idea右下角显示当前分支



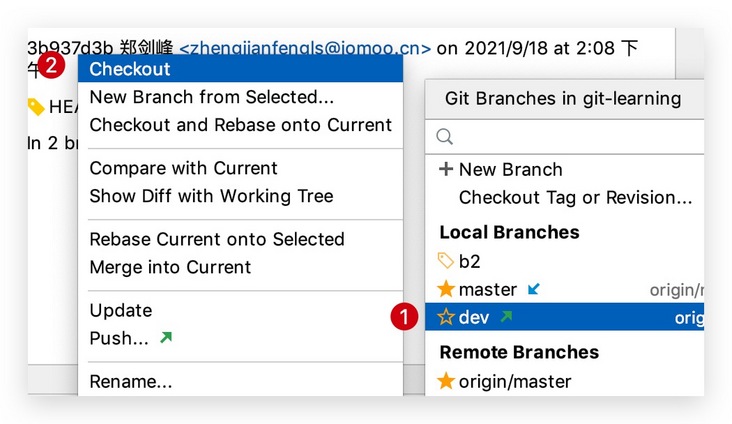
点开就可以在当前分支下新建一个分支



你也可以从某个commit切出一个分支，具体做法在commit日志上右键New Branch即可。

## 切换分支

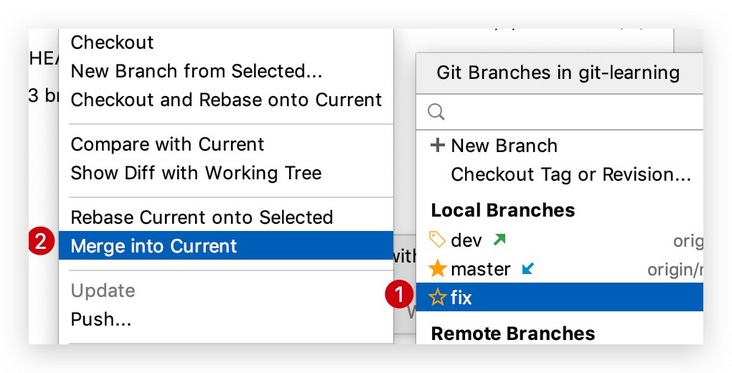
如果想要切换到其他分支，点击要切换的分支，选择Checkout



切换分支前需要把当前分支的修改提交或者暂存，暂存后续也会提到

## 分支合并

比如我要把fix分支合并到dev分支上，那么将当前分支切换到dev上，选择fix分支Merge into Current



合并过程可能会产生冲突，需要解决完冲突才能完成合并，解决冲突下文也会提到

合并完分支后，建议把fix分支进行删除，避免误将分支提交到远程仓库，导致分支混乱。

注意到上面菜单中在Merge上面还有一个Rebase Current onto Selected的选择，合并有两种方式，一种是merge，一种是rebase，这两种有什么区别呢，如下图，①是rebase方式合并，②和③是merge方式合并



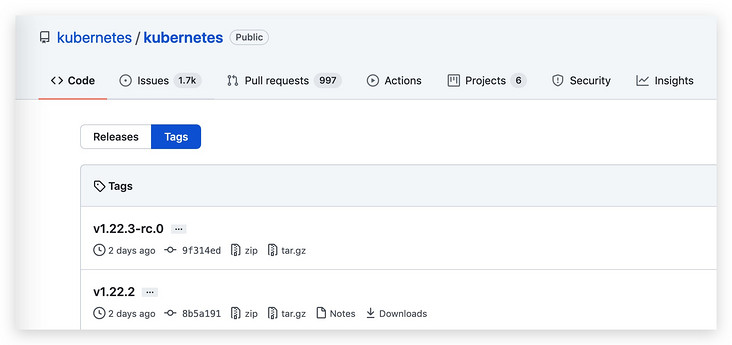
1. merge会保留分支提交信息
2. merge会创建一次Merge branch的commit信息
3. 相比merge，rebase会将分支上的commit整合到当前分支上，让整个代码周期更为清晰
4. 相比merge，rebase并不会产生多余的commit信息

具体用rebase和merge，各有各的好处，取决于项目的规范要求。

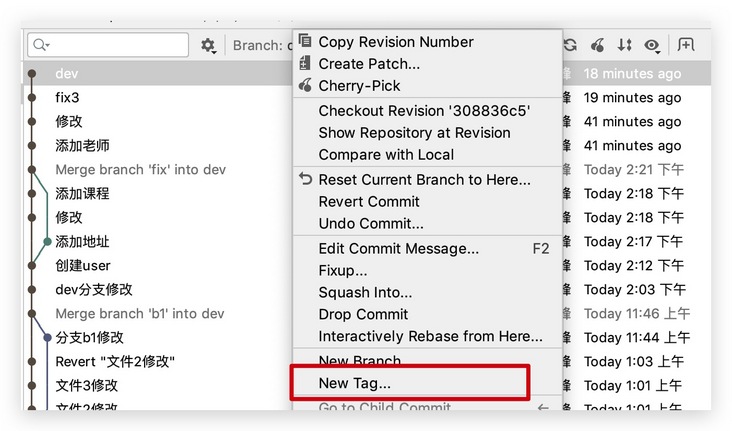
# Tag

## 创建标签

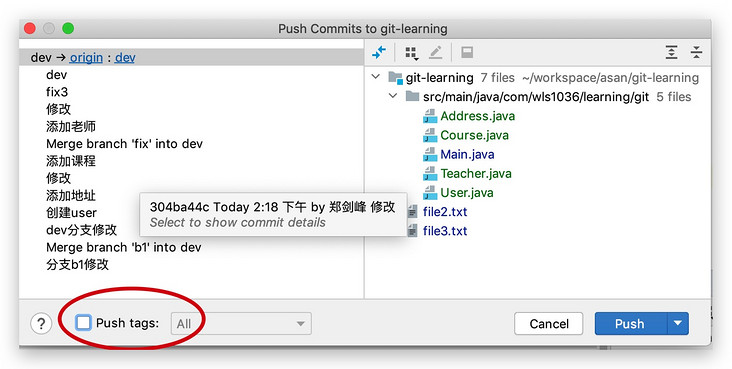
一般我们在发布一个新版本时会对项目打一个标签（tag），以表示其重要性，在gihub上我们也经常看到项目发布新版本时都会打上标签



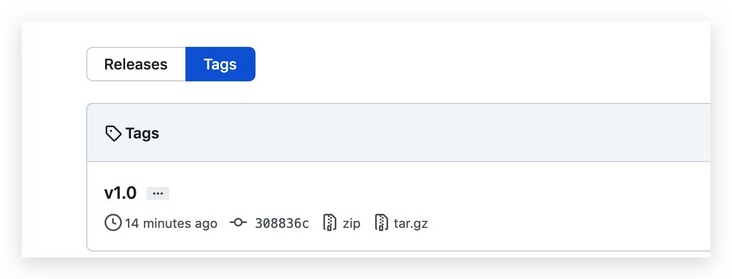
在commit的log上右键commit就为该commit创建一个标签，标签名称一般为版本号，比如v1.0 v1.1等



在push代码时可以选择将标签一起提交至远程仓库



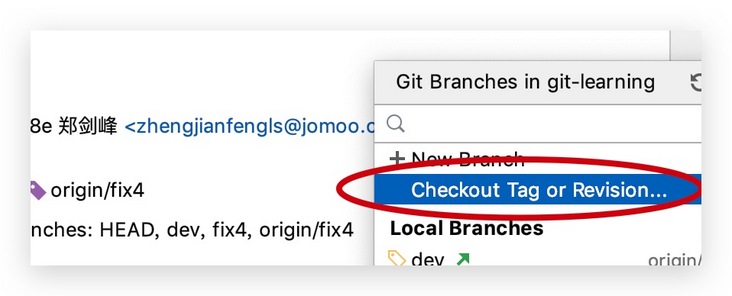
这样我们就能在远程仓库中看到标签，并且系统会自动将标签对应的commit当时的代码进行打包归档



## 根据标签拉取代码

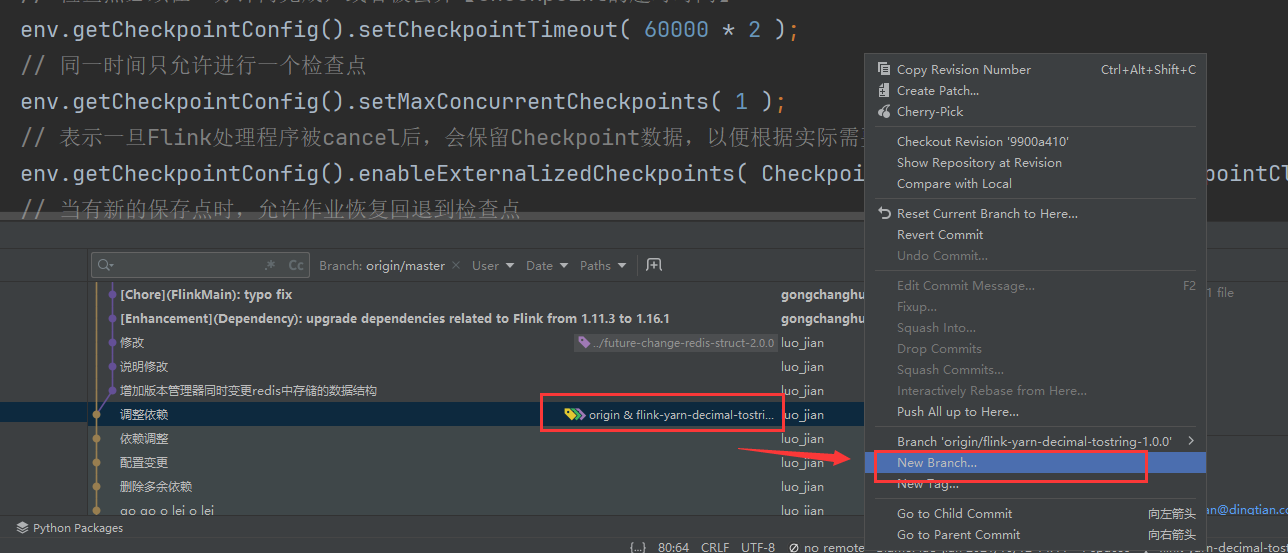
比如需要对v1.0的代码进行修复，我们需要拉取v1.0的代码，并且修复完毕后需要合并到当前分支

根据tag拉取代码



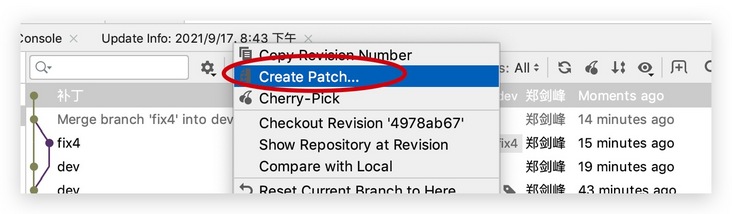
输入标签名称就可以拉取到代码，但此时的分支名称是分支对应的commit编号，说白了就是一个commit，我们需要在该分支上再创建一个分支才能对代码进行修改，创建的方法和上面一样，就是点击分支名称，New Branch即可，修改后提交，按照正常分支合并即可。

可以选中Git提交日志的Tag位置，然后基于该Tag新建分支。

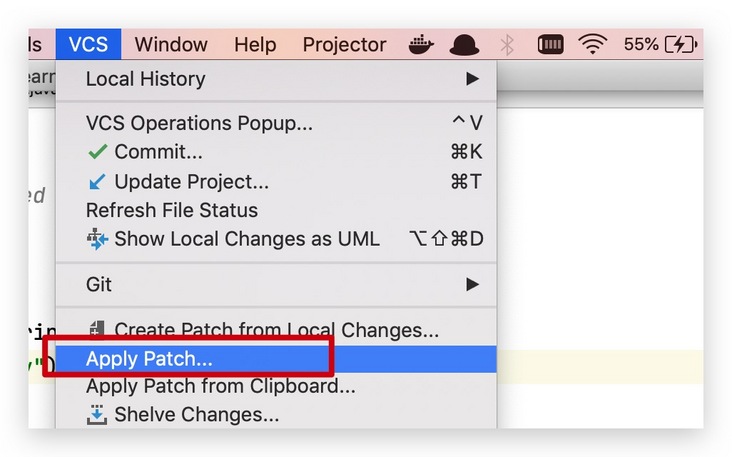


# Patch

可以为一个或多个commit创建一个补丁（Patch），idea会创建一个.patch的文件，里面记录了文件的变化



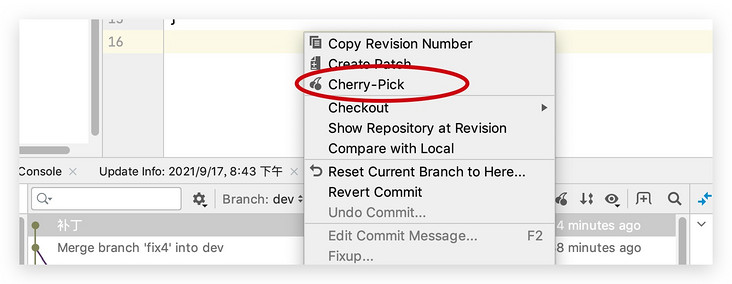
在分支上可以将补丁进行导入，VCS -> Apply Patch



打补丁也是一种合并的方式，也需解决代码冲突问题

**cherry-pick**

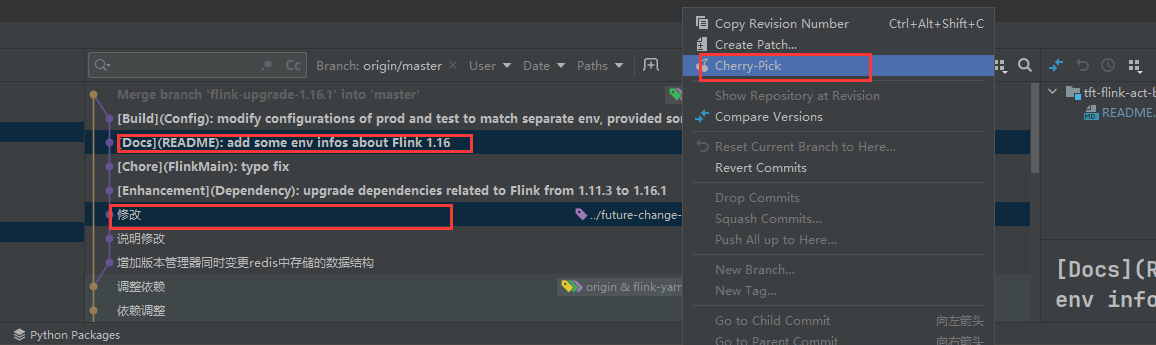
和打补丁类似，cherry-pick可以直接将其他分支的一个或者多个commit应用到当前分支，无需导出patch文件



Ref: <https://www.jetbrains.com/help/idea/apply-changes-from-one-branch-to-another.html#cherry-pick>

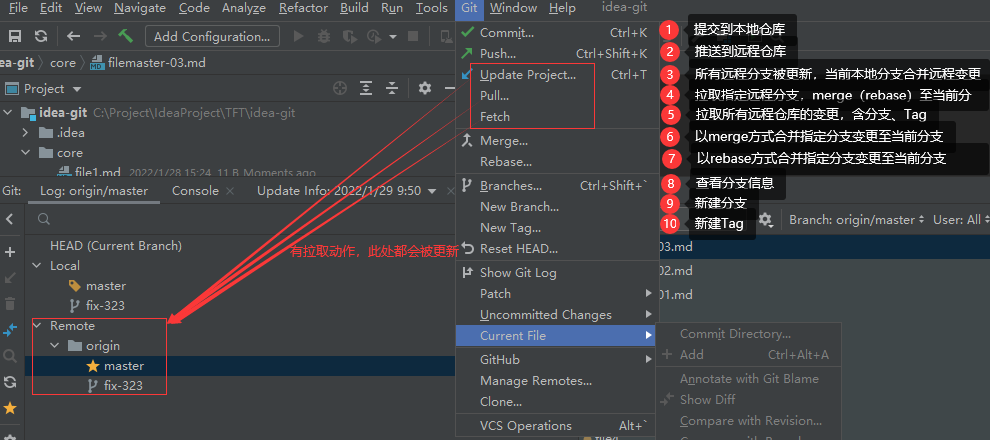
注意：cherry pick只是将**修改的内容**（commit），提交到当前工作分支（工作区），并不会将修改内容（commit）所在文件全部内容覆盖当前工作区的同名文件

在pick(选择) commit的内容时， 支持多个commit同时选择，如



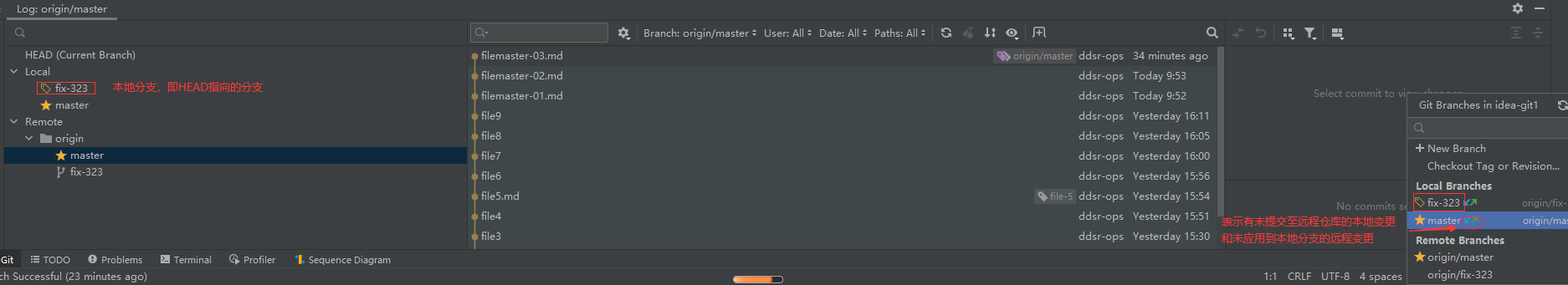
# Git Tool

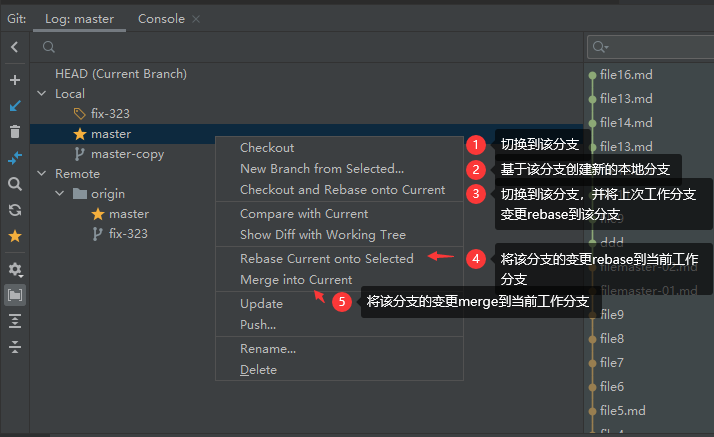
Git工具栏说明。



通常来说，Git工具栏的Commit/Push/Update Project使用较多，其他功能，一般不在此处使用，请参考[Git Bottom Tool](#_Git Bottom Tool)

# Git Bottom Tool





④和⑤目的都是一致，将选中分支的变更应用到当前工作分支。“应用”的方式不同，分别是rebase和merge。

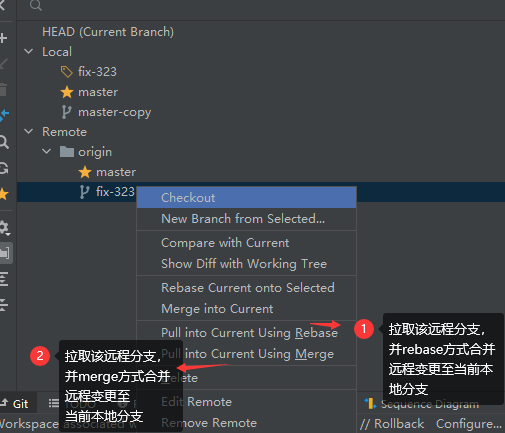
Entire git repository is local. git merge origin/branch merges the state of origin/branch known to local repository. It may be different on the server.

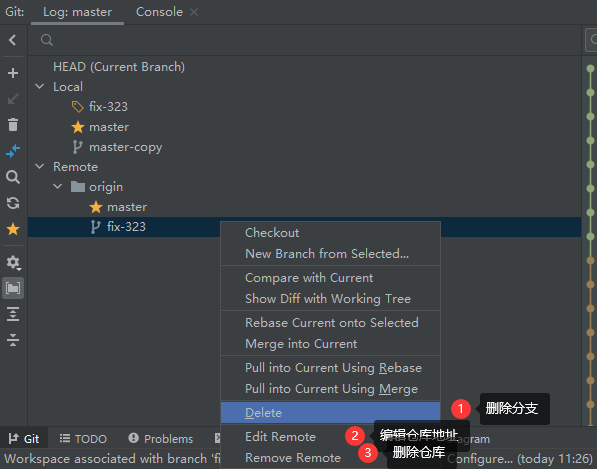
整个Git仓库是本地的，git merge origin/branch命令意指合并origin/branch状态到本地仓库。origin/branch状态也许和服务器端的不一致。

Pull first updates the local state of origin/branch to match the server, and then merges. That is why git pull is a different command.

We first tried to substitute Merge with the Pull, but had to put the Merge back

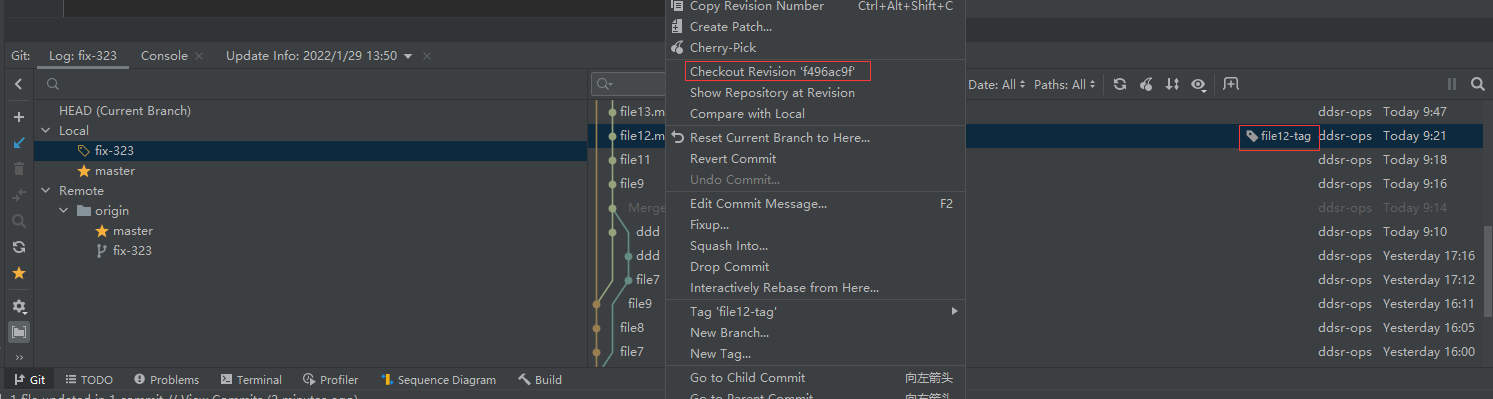
Ref： <https://youtrack.jetbrains.com/issue/IDEA-266647>

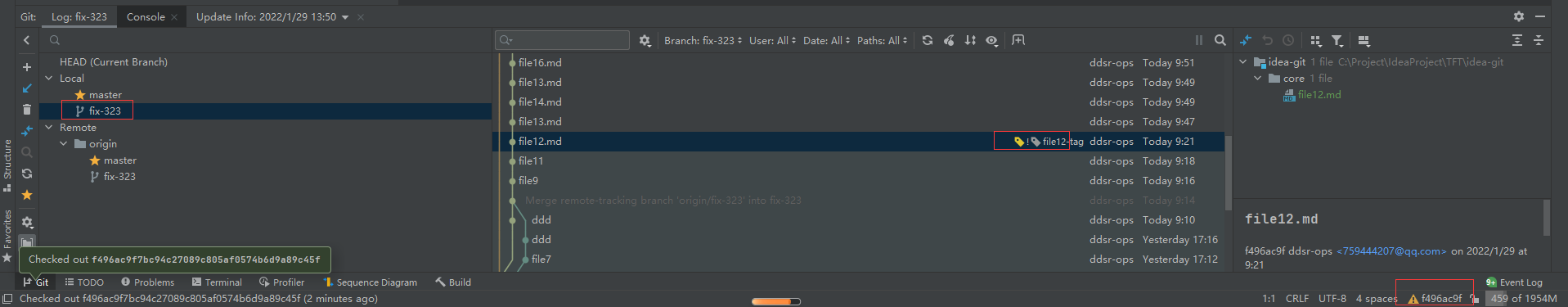




# Check Out By Tag

根据分支的某个Tag，check out（切换分支），这个分支可以是本地分支，也可以是远程分支。





虽然checkout, 但是没有本地分支来保留该分支的修改，所以需要创建新的本地分支（直接在右下方新建分支即可），还可以将新建的分支推送到远程仓库。如果不新建本地分支来保留本地更改，切换到其他分支将丢弃这部分更改！

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -c with the switch command. Example:

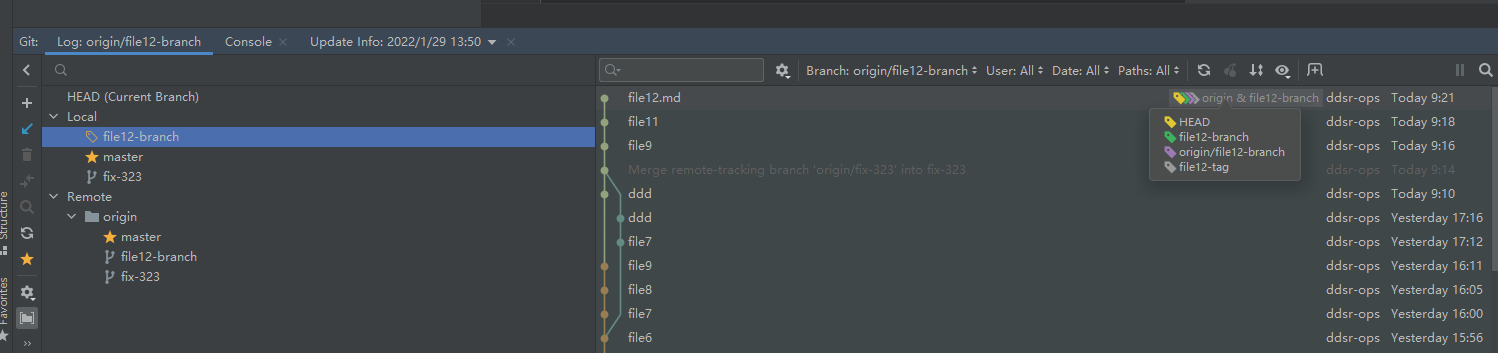
git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at f496ac9 file12.md

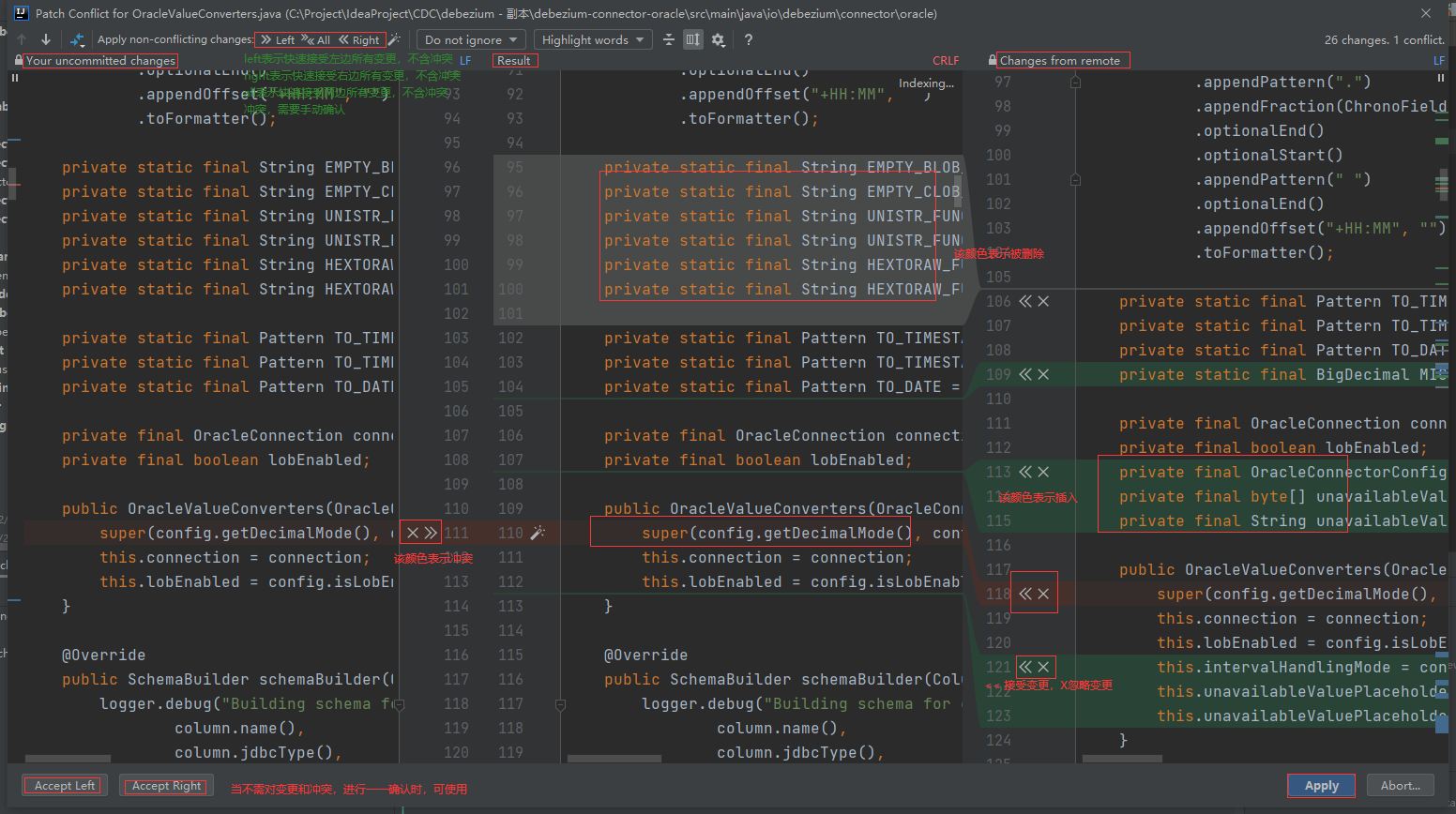


Idea相关的Tag操作，可以参考

<https://www.jetbrains.com/help/idea/use-tags-to-mark-specific-commits.html#checkout-tag>

# Resolve Conflict

冲突，一般发生在合并代码时，需解决掉冲突部分，才能完成整个代码的合并。



# Undo

git回滚代码有两种方式

reset

revert

这两种有什么区别呢，假设我们提交了三次commit

commit1 -> commit2 -> commit3

如果我想会滚到commit2，如果用reset的话，那么commit2之后的所有修改都会被丢弃，也就是把commit2之后的commit全部砍掉了，revert有点像undo，会把commit2做的操作反做一遍，也就是undo，并且生成一个新的commit，变成

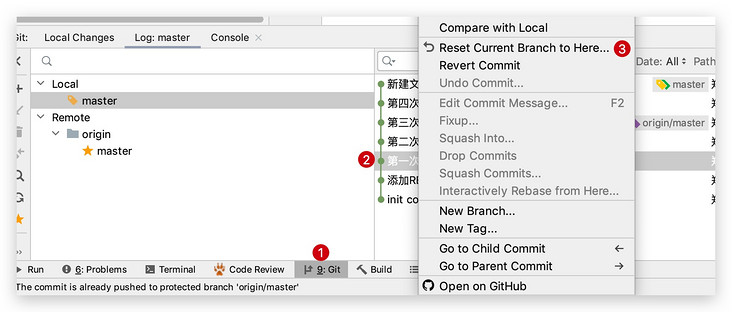
commit1 -> commit2 -> commit3 -> commit4

具体应用在什么场景呢，举个例子，比如我们在调试的时候可能要加一些调试语句，比如system.out，当调试完毕，我们希望能够移除这些调试语句就可以在调试前提交一次commit，调试完毕后reset到该次commit就行，当然你也可以通过分支来实现，调试完毕把分支删掉即可。

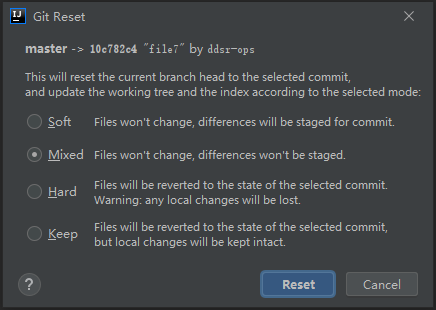
我们的系统在不断的迭代，突然某一天，产品经理说之前已经上线的一个功能不要了，那么就可以找到该功能对应的commit进行revert，这样既移除对应的功能又能保留后续的新功能。

## Reset

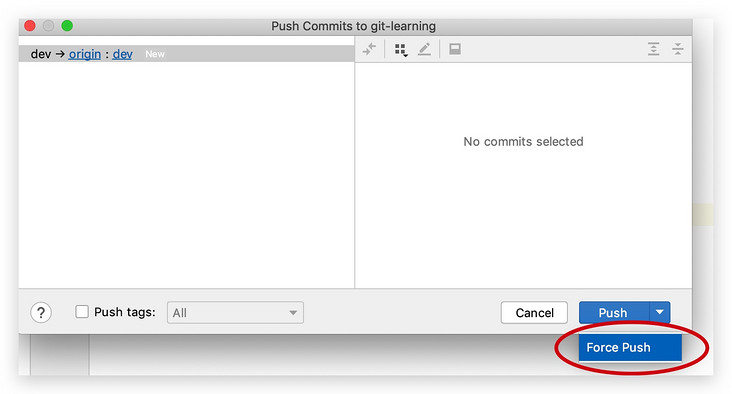
在idea地步工具栏，点击Git工具栏，切换到Log标签页，可以看到本地commit日志和远程commit日志，在指定commit上右键选择Reset Current Branch to Hear就可以以reset的方式回滚到该commit上



有四种回滚的模式



选择Hard模式，将会清除本地版本，强制回滚到指定commit状态，但是通过reset是无法进行push操作，因为本地的版本比远程版本要低，此时可以强制push到远程分支



idea默认是禁止在master分支上进行强制push，如果是master分支需要强制push可以在终端使用命令行 git push --force

force push是一种非常不好的习惯，除非你清楚的意识到force push带来的后果，不然千万不要执行force push

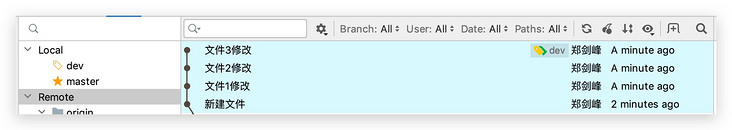
## Revert

revert可以将指定的commit所做的修改全部撤销掉，而不影响其他commit的修改，假设我们有三个文件，分别为file1.txt，file2.txt，file3.txt，现在提交三个commit，三个commit分别为

Commit1：向file1.txt中添加内容

Commit2：向file2.txt中添加内容

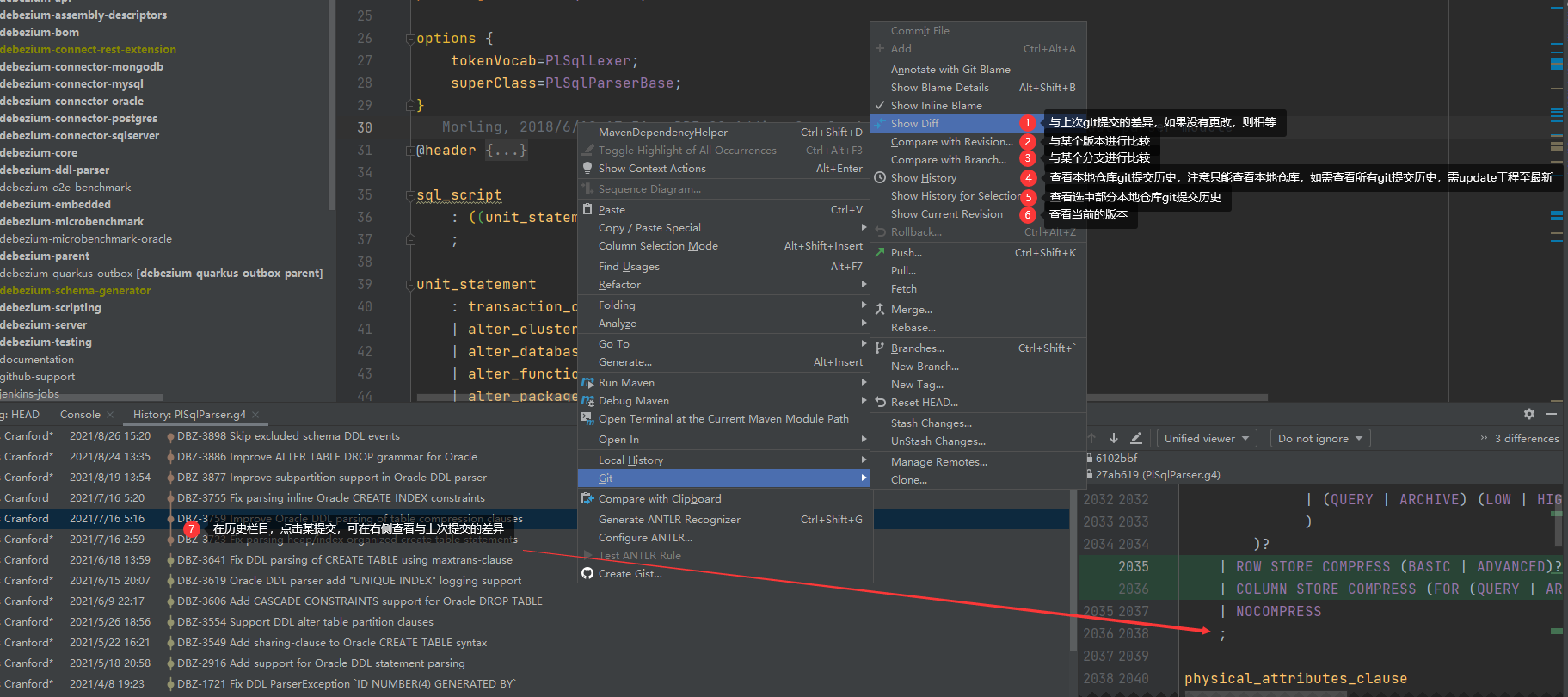
Commit3：向file3.txt中添加内容



现在我们在Commit2：文件2修改这个commit上面右键选择Revert Commit，这时候会弹出提交窗口，idea会自动提交一次新的commit，commit后我们发现文件2的内容没有了，也就是撤销了commit2修改的内容，commit后执行push操作就能推送到远程仓库了，因为是生成了新的commit，所以无需force push就能提交到远程仓库

# 鼠标右键Git

在编辑器中，鼠标右键弹出的git操作选项：



其中local history，表示本地针对该文件的修改记录，与git无关。

# 底部git工具栏

下图为idea底部工具栏，查看远程git分支的变更情况。

