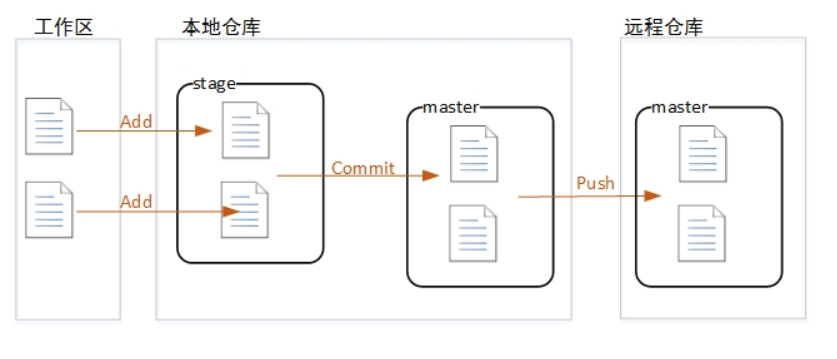
## git的使用的相关问题整理

一. #### 什么是工作区和暂存区



工作区：就是电脑里能看到的目录。

暂存区存在的意义：

使用git时候是否都是所有的修改全部提交了，根本没有考虑到多个修改文件，是和多个功能有关，而每一个功能应该单独做成一次提交，这样可以保证提交历史的清晰。否则，当你想要回滚历史的时候，你会无所适从，根本分不清每个版本包含了哪些功能，修复了哪些bug.而暂存区的作用就是为了，可以选择提交，比如你在开发B功能的时候，发现A功能还存在Bug，这时候就需要先修复A中的Bug，然后先提交修复的A中的Bug,然后再提交B功能开发的文件。这样就可以提高提交版本历史记录的清晰，方便回滚。而提交是原子性操作，文件的选择就交于暂存区去做，每一次提交都是一个完整的功能开发，保证commit的干净，降低commit的粒度。

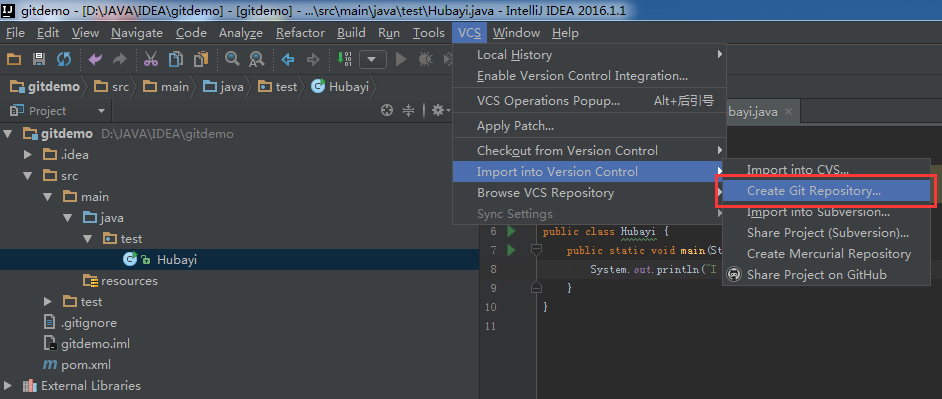
**idea打开git命令行快捷键 ：Alt + F12 （Terminal）**

**idea中git的使用，界面操作关键这几步就够用**：

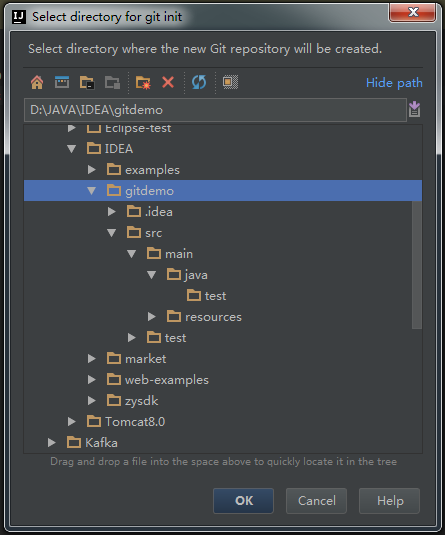
假设 组长 张三，组员 李四

场景一：张三创建项目并提交到远程Git仓库

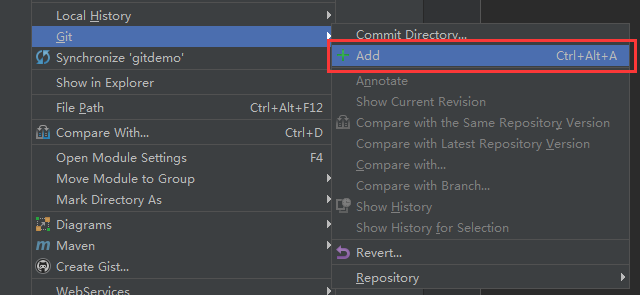
##创建好项目，选择VCS - > Import into Version Control -> Create Git Repository



##接下来指定本地仓库的位置，例如这里选择了项目源代码同目录

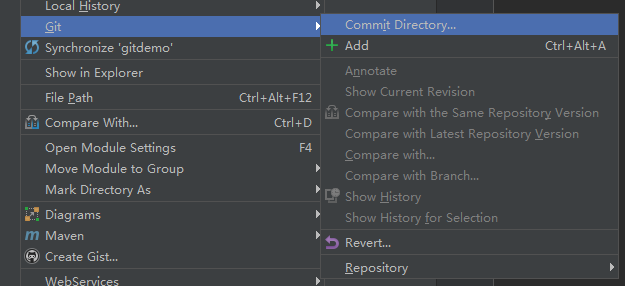


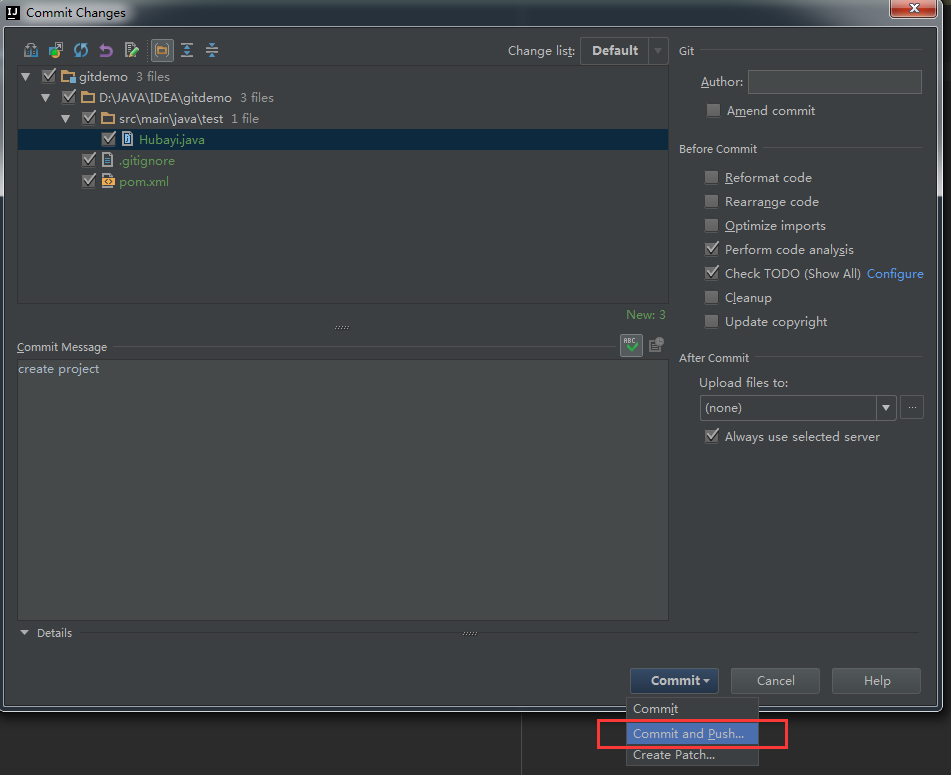
##点击OK后创建完成本地仓库，注意，这里仅仅是本地的。下面把项目源码添加到本地仓库。



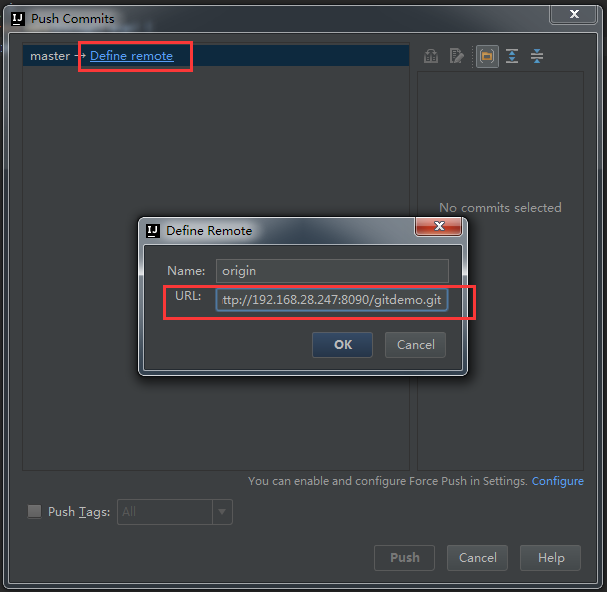
##IDEA中对操作做了一定的简化，Commit和Push可以在一步中完成。

具体操作，在项目上点击右键，选择Git菜单

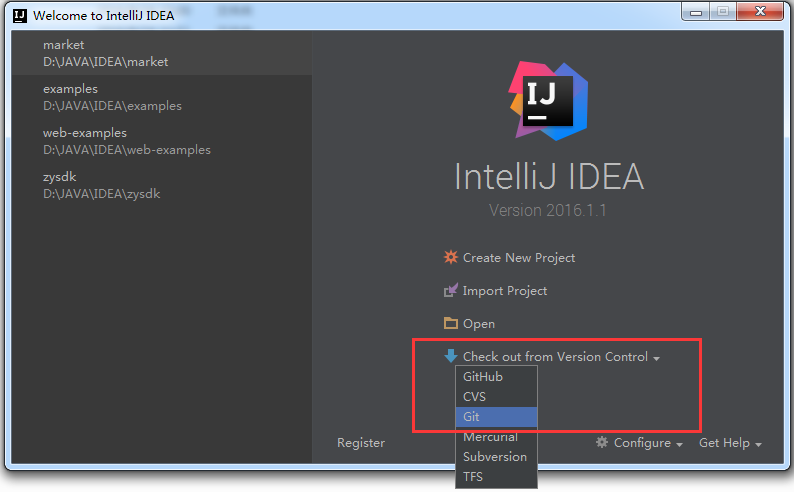




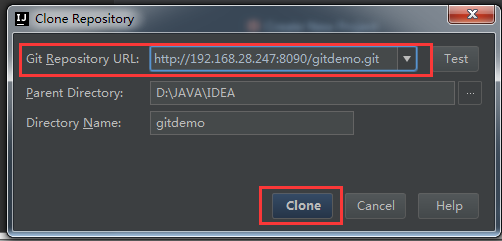
##因为是第一次提交，Push前需要指定远程仓库的地址。如下图，点击Define remote后，在弹出的窗口中输入远程仓库地址。



场景二：李四从远程Git仓库获取源码



输入张三Push时填写的远程仓库地址



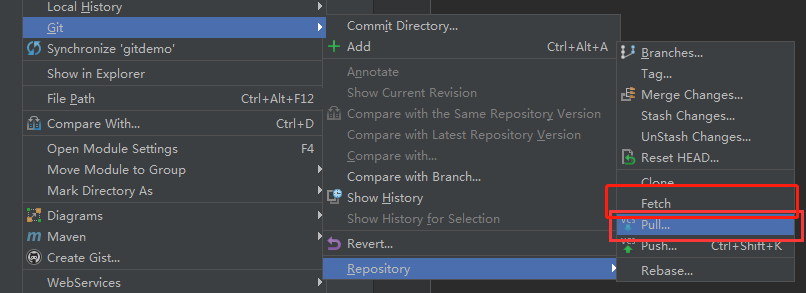
接下来按向导操作，即可把项目从远程仓库克隆到本地仓库和IDE工作区

场景三：李四修改部分代码并提交到远程Git仓库

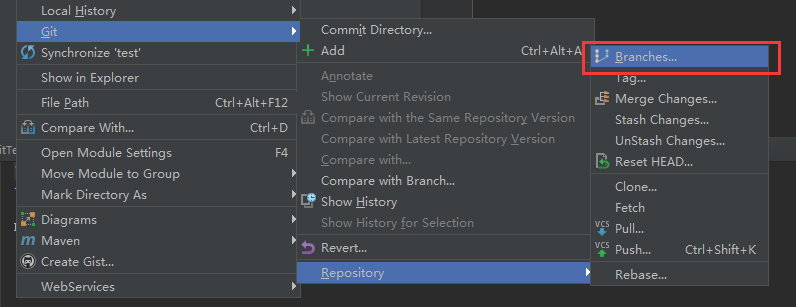
这个操作和首次提交的流程基本一致，分别是 Add -> Commit -> Push。参考场景一

场景四：张三从远程仓库获取李四的提交

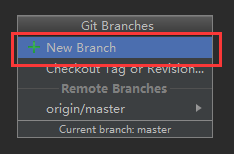
获取更新有两个命令：Fetch和Pull，Fetch是从远程仓库下载文件到本地的origin/master，然后可以手动对比修改决定是否合并到本地的master库。Pull则是直接下载并合并。如果各成员在工作中都执行修改前先更新的规范，则可以直接使用Pull方式以简化操作。



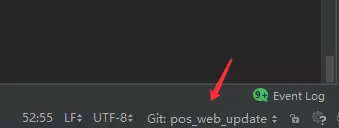
场景五：李四接受了一个新功能的任务，创建了一个分支并在分支上开发



选择New Branch并输入一个分支的名称

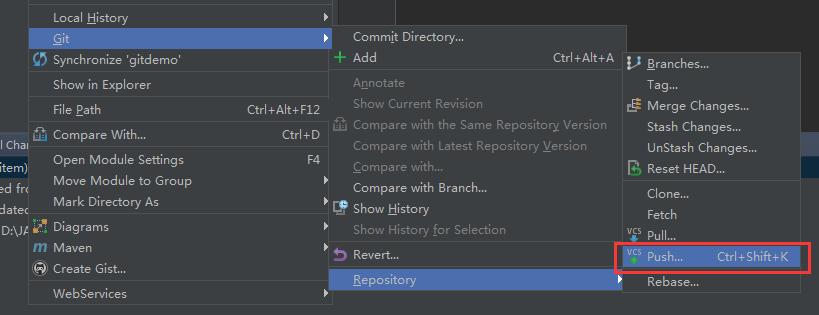


IDEA右下角查看所在分支，和checkout切换分支



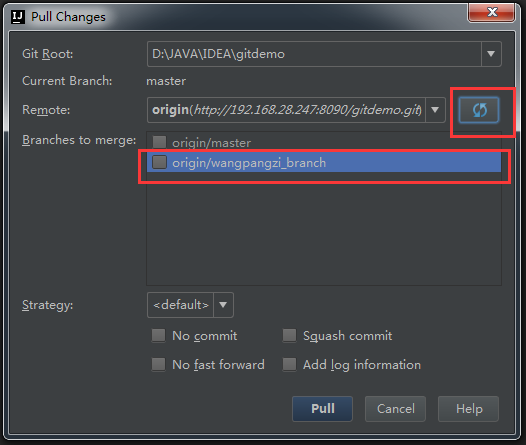
场景六：李四提交分支到远程仓库

切换到新建的分支，使用Push功能

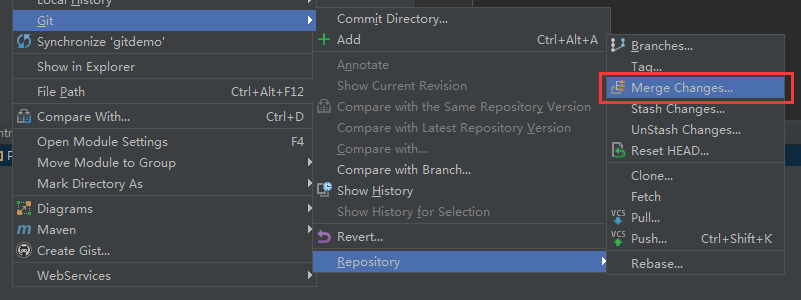


场景七：张三获取李四提交的分支

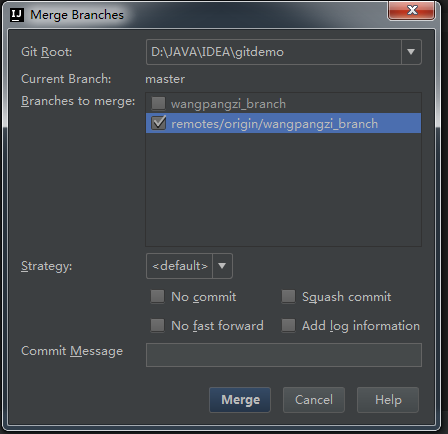
使用Pull功能打开更新窗口，点击Remote栏后面的刷新按钮，会在Branches to merge栏中刷新出新的分支。这里并不想做合并，所以不要选中任何分支，直接点击Pull按钮完成操作



场景八：张三把新分支合并到主干



选择要合并的分支，点击Merge完成



二. 分支的用途，如何合并分支

有一种情况，就是pull下了一个项目，然后创建分支写一个新功能，

发现线上的有个需要处理的bug，这时应该怎么操作来修改这个bug并push上去，

之后如果新功能合并到主分支时有冲突，在idea中应该怎么解决这个冲突

分支用途：

比如我们开发完了一个app上线了,接下那就是迭代功能开发了,如果上线的app出现了一个严重的bug,要你放下手头新功能的开发去解决这个bug,然后在发布一个新版本,如果你要是就在你要迭代功能的项目上进行修改发布的话,那肯定是不行，git分支就很好的解决了这个问题。

git checkout name 切换分支

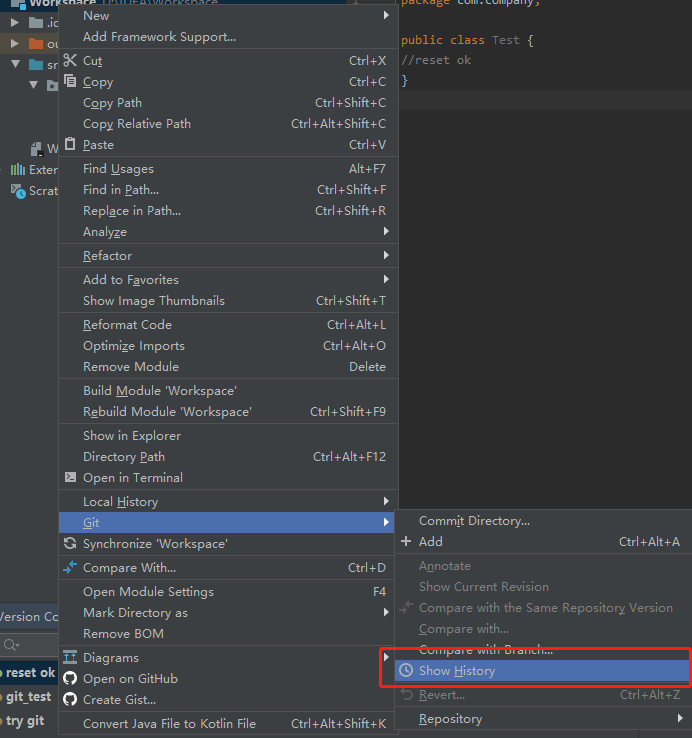
git merge name(要合过来的分支名)

---》 我觉得要使用 stash储藏当前工作目录，切换到master分支上处理bug，改好后提交，push，再应用回自己储藏，继续开发。

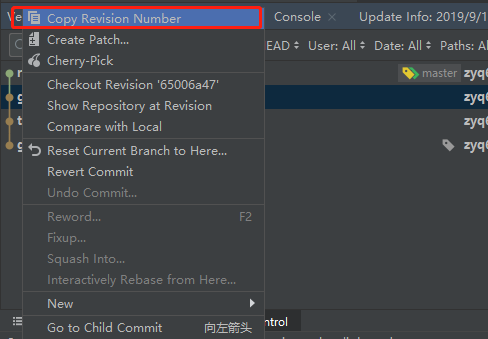
遇到版本冲突，首先想pull最新版本下来，进行对比合并，会解决大部分冲突问题

三. 如何优雅的在idea中进行版本回退，并push到远程主机

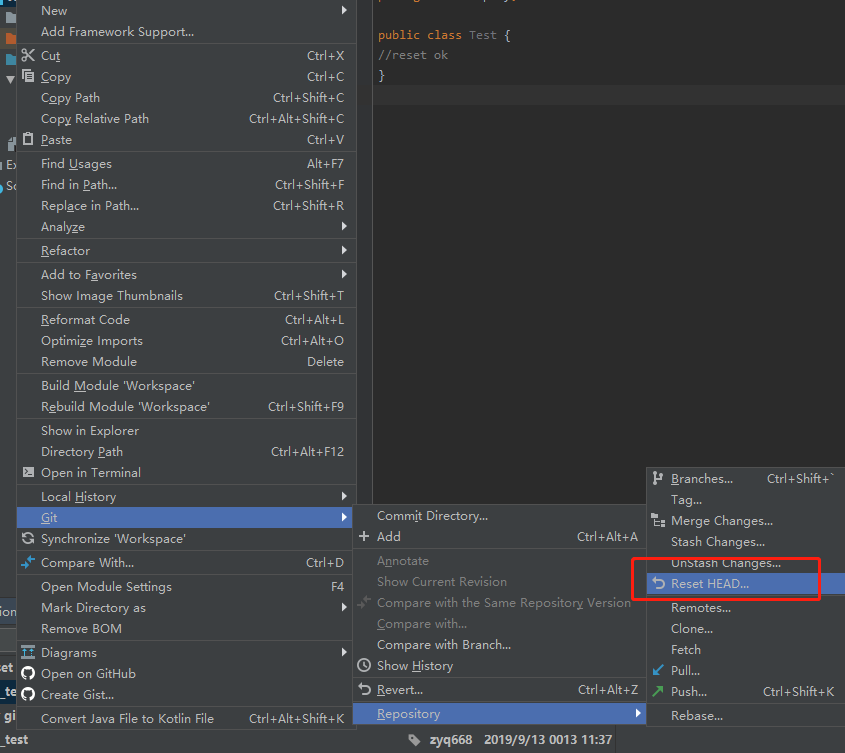
1.对项目右键git->Shwo History



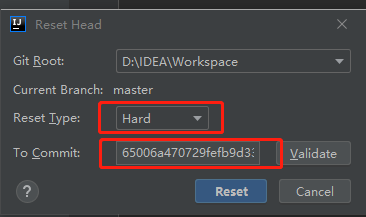
2.选择要回退的版本，右键，复制它的版本号（**最新的版本号也复制下，一会用**）



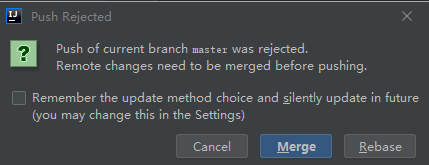
3.对项目右键git->Repository->Reset HEAD



4.填上回退版本号

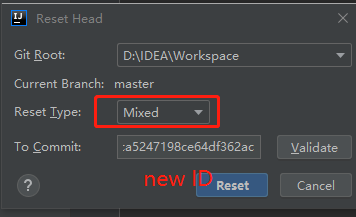


5.此时进行push，会与远程仓库版本冲突



取消掉即可，介绍个处理冲突的方法

a. 右击项目依次选中：Git->Repository->Reset HEAD



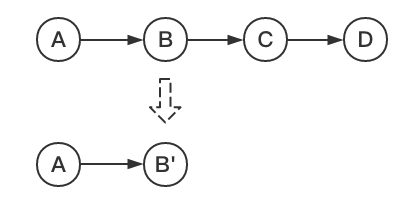
b. 选中Reset Type:**Mixed,** To Commit:**2746f428……**；然后点击Reset按钮

c. 这时你会发现，最新版本有回到newVersion。但是代码还是oldVersion的代码，这时候重push到远程仓库就不会版本冲突了

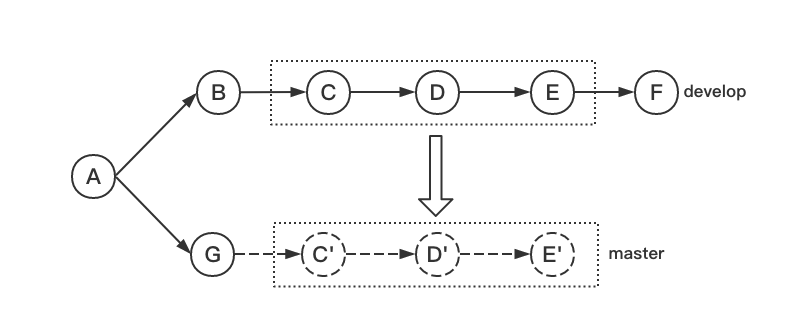
四. git rebase 的作用，应该如何使用

它的作用简要概括为：可以对某一段线性提交历史进行编辑、删除、复制、粘贴；因此，合理使用rebase命令可以使我们的提交历史干净、简洁！

**1.合并多个commit为一个完整的commit**



**2.将某一段commit粘贴到另一个分支上**



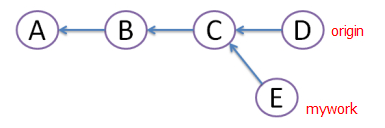
git rebase 详解<https://www.jianshu.com/p/4a8f4af4e803>

git pull 和 git pull - rebase 的区别

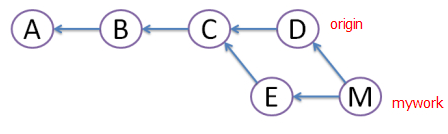
git pull ≈ git fetch + git merge

git pull -rebase ≈ git fetch + git rebase

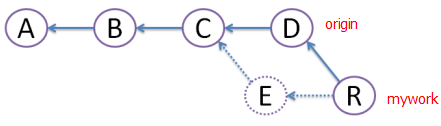
重点看 git merge 和 git rebase的区别



在远程分支origin的基础上创建一个名为"mywork"的分支并提交了，同时有其他人在"origin"上做了一些修改并提交了，就会发生冲突，以下两种方法解决：



用git pull命令把"origin"分支上的修改pull下来与本地提交合并（merge）成版本M，但这样会形成图中的菱形，让人很困惑。



git rebase会创建一个新的提交R，R的文件内容和上面M的一样，但我们将E提交废除，当它不存在（图中用虚线表示）。这种删除，避免了菱形的产生，保持提交曲线为直线，让大家易于理解。

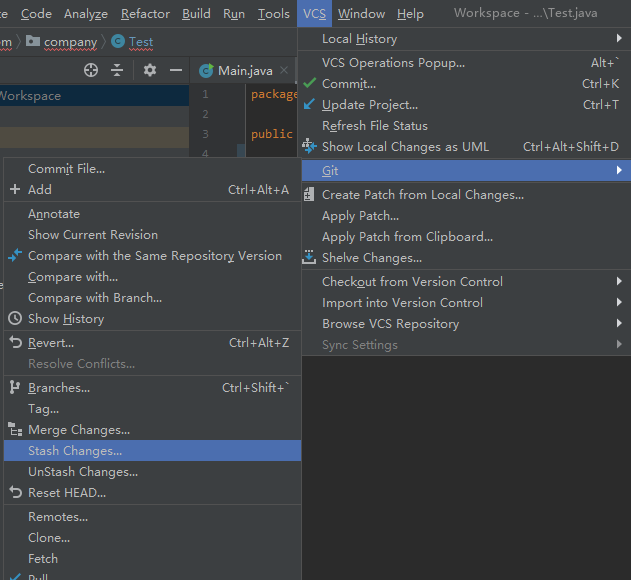
五. git stash 的使用（Stashing 储藏）

经常有这样的事情发生，当你正在进行项目中某一部分的工作，里面的东西处于一个比较杂乱的状态，而你想转到其他分支上进行一些工作。问题是，你不想提交进行了一半的工作，否则以后你无法回到这个工作点。解决这个问题的办法就是git stash命令。

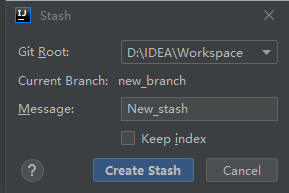
“‘储藏”“可以获取你工作目录的中间状态——也就是你修改过的被追踪的文件和暂存的变更——并将它保存到一个未完结变更的堆栈中，随时可以重新应用。

**5.1常用操作IDEA中的stash**：

1.Git->Stash Changes

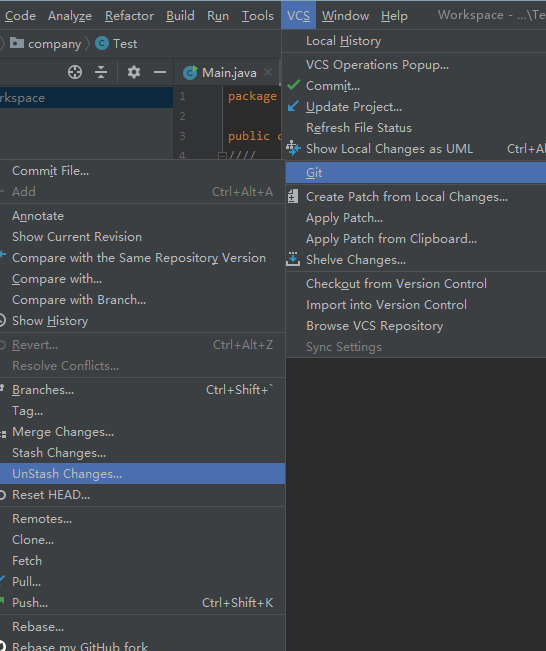


2.编辑备注信息，创建stash，之后会发现代码已经恢复成原来的状态，存储成功

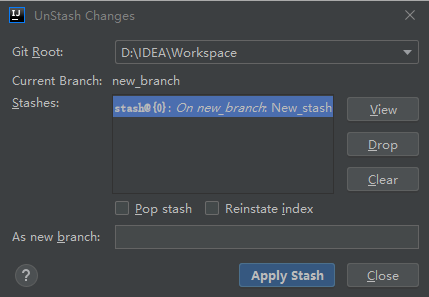


3.在处理完其他bug后，回到自己的分支，恢复stash

Git-> UnStash Changes...



4.应用回对应的stash（pop：应用并删除stash）完毕



**5.2常用git stash相关命令**：

注意⚠️直接执行git satsh 是不会保证当前文件都储藏起来，要把需要的文件git add添加后中的才会被储藏！

（1）**git stash** save "save message"  : 执行存储时，添加备注，方便查找，只有git stash 也可以的，但查找时不方便识别。

（2）**git stash list** ：查看stash了哪些存储

（3）**git stash show** ：显示做了哪些改动，默认show第一个存储,如果要显示其他存贮，后面加stash@{$num}，比如第二个 git stash show stash@{1}

（4）**git stash show -p** : 显示第一个存储的改动，如果想显示其他存储，命令：git stash show  stash@{$num}  -p ，比如第二个：git stash show  stash@{1}  -p

（5）**git stash apply** :应用某个存储,但不会把存储从存储列表中删除，默认使用第一个存储,即stash@{0}，如果要使用其他个，git stash apply stash@{$num} ， 比如第二个：git stash apply stash@{1}

（6）**git stash pop** ：命令恢复之前缓存的工作目录，将缓存堆栈中的对应stash删除，并将对应修改应用到当前的工作目录下,默认为第一个stash,即stash@{0}，如果要应用并删除其他stash，命令：git stash pop stash@{$num} ，比如应用并删除第二个：git stash pop stash@{1}

（7）**git stash drop** stash@{$num} ：丢弃stash@{$num}存储，从列表中删除这个存储

（8）**git stash clear** ：删除所有缓存的stash

（9）**git stash branch name** : 创建储藏的分支，避免重新应用时会遇到一些问题，应用工作成功便会丢弃储藏

**5.2在这里**，存储时感觉需要add的文件太多了，还有个非常有用的小技巧：

one：add 那些你不想备份的文件（例如： git add file1.js, file2.js）

two：调用 git stash –keep-index。只会备份那些没有被add的文件。

three：调用 git reset 取消已经add的文件的备份，继续自己的工作。

六.git reset的作用：

**A. 文件从暂存区回退到工作区，撤销add**

**B. 版本回退**

例如：如果想取消某个add的文件，可以使用该命令来进行撤销操作

　　**撤消add：**git reset 文件名

　　**撤消所有add的文件：**git reset HEAD .

　　**撤消某个文件或文件夹：**git reset HEAD 文件（夹）名

　　**把从cache中删除的文件，重新添加到cache中：** git add -f 文件名

版本回退：

**1. git reset  --mixed commit的id**

还原到commit的id，（git reset 默认是mixed ）此commit之后的文件变成modified红色（即还没add的状态）（不影响本地文件）

**2. git reset  --soft commit的id**

还原到commit的id，此commit之后的文件变成modified绿色（即add（勾选中）的状态）（不影响本地文件）

**3. git reset --hard commit的id**

还原到commit的id，此commit之后的文件都被还原（本地文件被替换）

**4. 对reset进行撤销**

睡了一觉，第二天醒来，我们后悔了，又想回到reset之前的状态，该怎么做呢？

Git提供了一个命令git reflog用来记录每一次的操作，即使是reset掉的commit记录，reflog都会有记录，所以只需找到对应的commitId，以便确定要回到的哪个版本

七.git add . 和 git add -A 的区别：

git add -A和 git add .   git add -u在功能上看似很相近，但还是存在一点差别

简单说就是

**·**git add -A  提交所有变化

**·**  git add -u  提交被修改(modified)和被删除(deleted)文件，不包括新文件(new)

**·**  git add .  提交新文件(new)和被修改(modified)文件，不包括被删除(deleted)文件

**八.git diff 的用法:**

1. **不带参数的调用**

git diff filename
这种是比较 工作区和暂存区

1. **比较暂存区与最新本地版本库**

git diff --cached filename

1. **比较工作区和最新版本**

git diff HEAD filename

1. **比较工作区与指定的 commit -id 的差异**

git diff commit-id filename

1. **比较暂存区与指定 commit-id 的差异**

git diff --cached commit-id filename

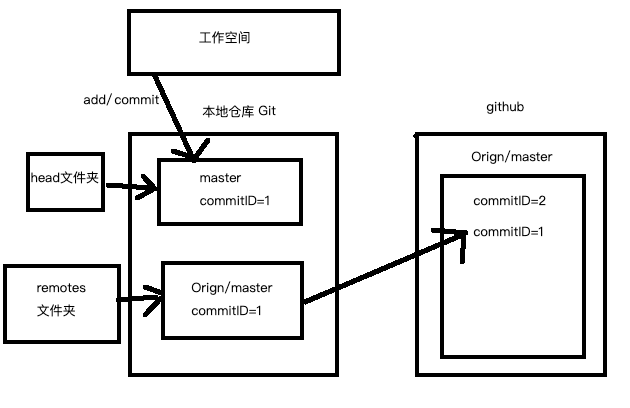
1. **比较两个 commit -id 的差异**

git diff commit-id commit-id

**九.git fetch 和 git pull 的区别:**

**git pull ≈ git fetch + git merge**

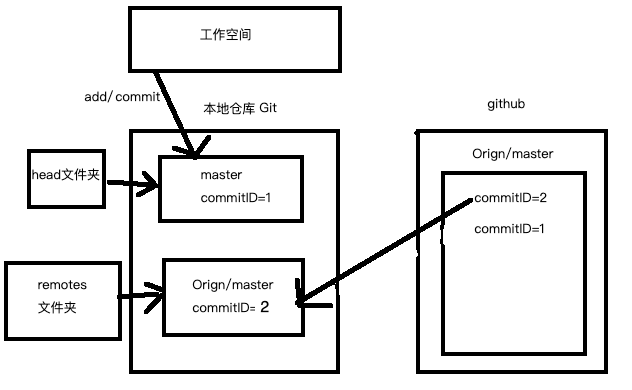
首先假设我们本地仓库的 master 分支上 commit ID =1 ，orign/mastter中的commit ID =1 ;这时候远程仓库有人更新了github ogirn库中master分支上的代码，新的代码版本号commit ID =2 ,那么在github上 orign/master的commitID=2，然后我们要更新代码。



**git fetch**

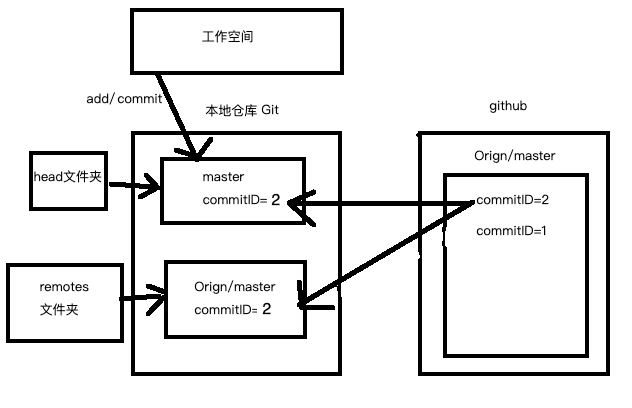
使用git fetch更新代码，本地的库中master的commitID不变，还是等于1。但是与git上面关联的那个orign/master的commit ID变成了2。这时候我们本地相当于存储了两个代码的版本号，我们还要通过merge去合并这两个不同的代码版本，如果这两个版本都修改了同一处的代码，这时候merge就会出现冲突，然后我们解决冲突之后就生成了一个新的代码版本。

这时候本地的代码版本可能就变成了commit ID=3，即生成了一个新的代码版本。



相当于fetch的时候本地的master没有变化，但是与远程仓关联的那个版本号被更新了，我们接下来就是在本地合并这两个版本号的代码。

**git pull**



相对简单暴力，使用git pull的会将本地的代码更新至远程仓库里面最新的代码版本

**由此可见，git pull看起来像git fetch+git merge，但是根据commit ID来看的话，他们实际的实现原理是不一样的，各有优劣。**

**将下载（fetch）和合并（merge）放到一个命令里的一个弊端是，你的本地工作目录在未经确认的情况下就会被远程分支更新。当然，除非你关闭所有的安全选项，否则git pull在你本地工作目录还不至于造成不可挽回的损失，但很多时候我们宁愿做的慢一些，也不愿意返工重来。**

**简单命令行操作：**

详解：<https://blog.csdn.net/prike/article/details/50040723>

git init 库的初始化（之后就可以git status查看状态了）

git status 查看暂存区状态

git add filename 添加进暂存区

git commit -m "信息说明，版本说明" 提交到本地仓库

git reset HEAD filename 清空暂存区

git checkout -- filename 刷新工作区

git log 记录日志

git reflog 详细日志

git reset --hard commitID 回到该id版本

git rm filename 清空本地工作区

清空缓存还要 用空的工作区进行commit

**远程仓库：**

在GitHub官网上settings，ssh，设置访问权限。

ssh-keygen -t rsa -C "youremail@example.com" 命令行中输入，一直回车即可

ll 详细目录信息

cd ~ 回到用户目录

pwd 显示当前目录

cd .ssh/ 进入ssh文件夹

ll 显示一下详细目录

cat id\_rsa.pub 找到这个打开它，复制里面的密钥放在官网上

ssh -T git@github.com 测试是否连接成功

**Git更换push提交地址：**

git remote rm origin 删除远程地址

git remote add origin http://~~~~~.git

命令行创建GitHub存储库：

echo“#Test\_git”>> README.md

git init

git add README.md

git commit -m“first commit”

git remote add origin git@github.com:zyq668 / Test\_git.git

git push -u origin master

命令行推送现有存储库：

git remote add origin git@github.com:zyq668/Helloworld.git 对接github仓库

git push -u origin master 推到远程服务器 origin主机 -u指定默认主机

**克隆远程代码：**

例 git clone git@github.com:zyq668/Test\_git.git

**标签管理：**

git tag 查看所有标签

git tag name 创建标签

git tag -a name -m "comment" 指定提交信息

git tag -d name 删除标签

git push origin name 标签发布

**分支管理：**

git branch 查看现有分支

git branch name 创建分支

git checkout name 切换分支

git branch -d name 删除分支

git merge name(要合过来的分支名)

**常见问题**

1.在IDEA中使用Terminal，没有自动补全文件和关键词功能

解决：官网下载git

IDEA中 File -> Setting -> Tools -> Terminal -> shell path 这里导入bash.exe路径

如 D:\Git\bin\bash.exe

重启IDEA即可