

# 0.Idea中的git插件的命令列表

<https://www.jetbrains.com/help/idea/differences-viewer.html#diff-merge-viewer>

idea插件git 中文说明

每个git图像窗口左下角都有个问号，可以跳转至idea官网，有该窗口的说明

学了6小时左右感觉效率低，

一是没有官方教程、这个是个大问题，如果有那么事半功倍

二是英文命令单词懂，联合起来是懂非懂。可以拿去百度，或者百度翻译

提交ID其实是个Hash值

**cherry-pick和rebase 都已经完全理解了。**

## 1.0头部导航

### 1、Commit

即提交代码，打开提交窗口进行二次确认

同 【6.0左边导航栏commit】

### 2、Push

左上边 【原分支】 -> 远程仓库地址(可以下拉选择)：远程仓库的分支(可以手动修改)

左中间 提交日志

左下边 **Push tags**，这个理解就是对某个版本代码打标记，推送代码分支时，额外个标签，联想下github。  
这个tag分支也是可以下拉选择的

右上边 图形功能导航

点击某个文件，再点击左右箭头图形，打开进行本地仓和远程仓的改文件对比界面

上下箭头移动对比

**Jump to source**：跳转至源文件，即离开推送对比状态，可以【快速回到】待推送文件进行修改。

对比界面，右边本地仓库，左边远程仓库

对比界面控制选项1：

**side-by-side viewer**：并排对比，一行一行对比

**unified viewer**：单页面对比

对比界面控制选项2：

**Do git ignore**：所有非忽略文件

**trim whitespaces:** 每一行都不能忽略空格进行对比，智能辅助对比的。例如某行对比只有个空格差距，也要高亮出来

**ignore whitespaces:** 每一行只有空格的差异的话，无需高亮显示出来，即认为没有改变

**ignore whitespaces and empty lines:**

更多参考: <https://www.jetbrains.com/help/idea/differences-viewer.html#diff-merge-viewer>

对比界面控制选项3:

高亮控制

收缩未更改片段

同步滚动两个差异窗格

其他设置(列表)

方块图形: 即待推送文件列表的展示方式，按照目录和模块级别进行展示

选中某个文件，点击笔的图形: 快速跳转至源文件进行修改

**show Details:** 没发测试

收齐所有文件列表

全部展开文件列表

右中间 文件列表

右下边 推送、强制推送。取消

### 3、Update Project

`merge`和`rebase`这两种方式都可以合并代码。

**【1】、Merge the incoming changes into the current branch:**将传入的更改合并到当前分支中  
(常用)

**【2】、Rebase the current branch on top of the incoming changes:**在传入更改的基础上重新建立当前分支

第一点很好理解，经常用的，就是把远端合并到本地仓。

第二点也好理解。

**Rebase:** 变基，

拉远远程仓库的所有分支，更新到本地仓库

### 4、Pull

`git pull` 是 `git fetch + git merge FETCH_HEAD` 的缩写。

`git pull`就是先`fetch`，然后执行`merge` 操作，如果加`-rebase`参数，就是使用`git rebase` 代替`git merge`。

`Idea update project` 就是你可以选择到底是`merge` 还是 `rebase` 的`git pull`

### 5、Fetch

作用：拉取远端所有分支，存储到本地origin，这个origin是远端origin的一个副本。 Local origin

就是可以拉去远端的更新。

如果工作区当前分支就为dev\_1.0.0，远端同一个该分支dev1.0.0发生了新的提交。

那么本地fetch 该分支dev\_1.0.0时，该分支新的提交会被拉去下来，【存本地一个origin的以分支上】；但是不会合并(merge)。

merge就是将本地origin的某个分支合并本地仓库当前分支上，此时工作区文件发生了变化。

## 6、Merge

正如上面所说，merge就是将本地origin的某个分支合并本地仓库当前分支上，此时工作区文件发生了变化。

## 7、Rebase

理解rebase，得结合merge对比着看。

可以说 rebase和merge 是完成“同样的工作”，但是走了不同路线，留下了“不同的痕迹”。

同样的工作：新需求->拉取新分支->新分支合并到主分支

不同的痕迹：提交记录不一样

----- 上面理解是错的 -----

特性分支合并master分之前，别人抢先合并。

特性分支切换到master更新代码，

master切换到特性分支变基(rebase)到master最新节点。

切回master，把特性分支合并到master，再push

rebase就是说 多个人开发，每个人拉分支开发，有一些人分支任务先开发完，合并到master分支。自己这边慢于其他人合并master，于是我这边先git pull拉取master最新的，然后切换到我的分支，执行git rebase，变基到master的最新节点进行合并merge。这样最后的提交历史只有一条线。master那条线的记录

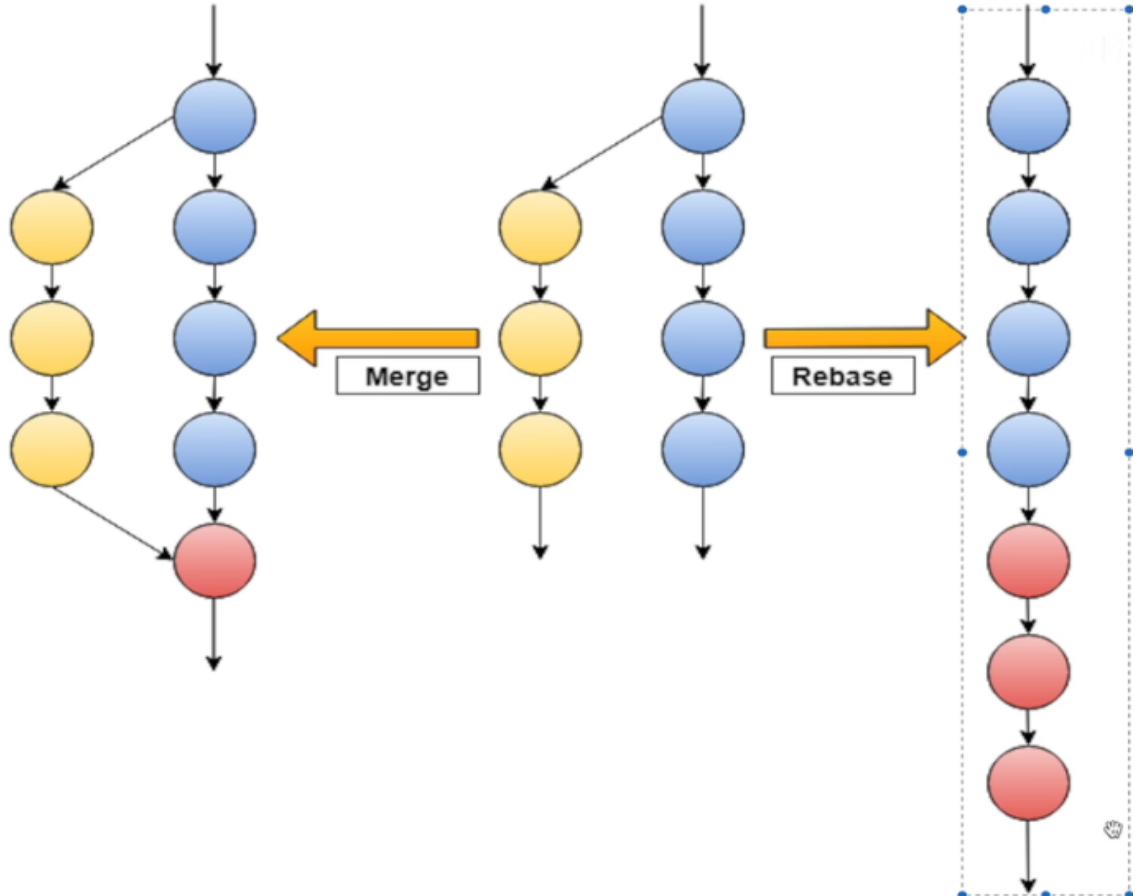
命令演示 A与B都采用rebase方式合并代码，按照时间书写命令

```
A git checkout b1
B git checkout b2
A git commit "c1"
A git merge b1 to master
A git push master
B git commit "c2"
B git commit "c3"
B git checkout master
B git pull master ## 把A的提交同步下来
B git checkout b2
B git rebase master
B git checkout master
B git merge b2
B git push master
```

rebase的一些选项：

```
--onto  
--interactive  
--rebase-merges  
--keep-empty  
--root
```

rebase的优缺点:



## 8、Branch/Branches

切换分支，或者新增分支

## 9、New Tag

新建一个tag

## 10、Reset HEAD

不太懂，等看完git基本原理再来看

**reset** 用于【已经提交的commit但是没有推送的commit的进行回退代码】，删除已有commit。  
通过移动这个HEAD(指针)，从而删除commit，从而回退代码。可以发现log没有了提交记录。

**revert** 用于【已经推送的commit进行回退代码】，新增commit覆盖提交

## 11、Show Git Log

打开git log 日志窗口，也就是左下角的git导航图标一个意思。

## 12、Patch

```
create patch from local changes 本地代码  
apply patch 导入补丁  
apply patch from clipboard 从剪贴板导入
```

**patch:** 即“补丁”

通过打补丁方式，把文件新的改变连带文件导出成一个.patch文件。

A开发了代码，有很多文件，但是不想提交，此时B开发要这些代码。

A开发就可以打补丁(patch)方式把代码给到B开发。

B开发直接导入就可以了(apply patch)，此时A开发的所有未提交的代码都会给到B开发。

此时B可以提交，也可以继续开发。

此过程 A开发和B开发通过文件方式实现了 代码的更新。传统是通过代码提交然后同步下载的。

此过程

patch的意义：

## 13、Uncommitted Changes

**Shelve Changes** 搁置更改      **Shelve:**上架/搁板/搁置

**show Shelf** 展示所有搁置

**Stash Changes** 储藏/存放改变

**Unstash Changes** 显示出所有的储藏

**Rollback** 回滚，回滚那些【距离上次提交】的后的所有修改

**Show Local Changes as UML**

**shelve**好像和**stash**是一样的功能，**shelve**是**idea**特有的，而**stash**是**git**的都是未提交的进行一个存储，方便切换分支后，再切换回来恢复代码的。因为未提交代码切换分支时，会被删除。**idea**有个**smart**

这两个都是针对【未提交】的代码进行保留的方式

## 14、Selected Files

当你选中一个文件 变为**Selected File**,  
当你选中多个文件 变为**Selected Files**,  
当你选中一个目录 变为**Selected directory**,  
当你选中多个目录 变为**Selected directories**,

扩展选项都是 **git** 的基本操作，显示历史、显示对比。添加至忽略文件等

**Add to .gitignore** 有两个选项

**Add to .gitignore** 这个是对公共忽略文件的处理，因为会改变 **.gitignore** 这个文件，这个文件必须提交。例如 **classs** 文件目录

**.git/info/exclude** 这个适合个人的文件忽略版本控制，例如自己写的测试类。使用这个选项的前提是，这个文件不能加入版本控制

加入版本控制的文件后，这个命令失效。例如，自己写的验证自己想法的东西。

这个颜色淡黄色，这个不受切换分支影响，始终在那里

## 15、Gittee

集成 **gitee** 的插件

**share project on gitee** 将工作目录推送到 **gitee**

**sync fork**

**create gist** 创建主旨。

类似于 **git** 是一个 **git** 库，可以 **fork** 和 **clone**

暂时用不到，详细可以看下增链接 -

<https://www.cnblogs.com/leader755/p/14284716.html>

**open on gitee:** 通过浏览器跳转至 **gitee** 上项目主页

## 16、GitHub

**share project on GitHub** 将工作目录推送到 **GitHub**

**create gist** 创建主旨。

## 17、Manage Remotes

管理远程仓库的地址的，可以有多个

## 18、Clone

拉去远程仓代码到本地仓的

## 19、VCS Operations

1、就是版本控制的一些操作选项列表

像**commit**

**rollback** 这里再次强调，回滚只是 【回滚那些【距离上次提交】的所有修改】

2、还有

**localhistory**

**put label**

这个**put label** 就是给本地历史打一个标签名字用于区分用的

## 2.0底部右下角

### 1.0 新分支和新Tag操作

**New Branch** 以当前分支为基础拉去新的分支

**Checkout Tag or Revision** 暂时不知道

## 2.0 本地仓库分支列表

Local Branches : 本地分支列表

选择当前分支：

**New Branch from selected** 以当前分支为基础拉去分支，连本地【未提交、未跟踪】的也会同步到新分支

**Show Diff with Working Tree** 对当前分支显示出哪些修改了没有提交的，以树的结构展示出来

**Update** 相当于 `git pull +当前分支`； 这个是【远程仓】拉去，不是【本地远程仓】，会更新本地仓同时更新本地远程仓

**Push** 就是常规的push

**Rename** 就是修改本地仓库名字，远程仓库名字还是之前那个。

例如**dev\_2.0.0**改为**dev\_2.0.1**，这意味着本地之前的被改了一个分支，内部其实改了个**id**，同时还是跟本地远程**dev\_2.0.0**挂钩的。

如果此时在本地远程的**dev\_2.0.0**再次**checkout**一次，就会在本地多出一个**dev\_2.0.0**那么就意味着本地有两个分支

(**dev\_2.0.0**和**dev\_2.0.1**)和本地远程**dev\_2.0.0**挂钩，其实相当于本地拉了(两次远程**dev\_2.0.0**)，此时**dev\_2.0.1**相当于一个**dev\_2.0.0**的副本。

这里理解**dev\_2.0.0**和**dev\_2.0.1**相当于“双胞胎”，但是“姓名”不一样

另外要说明下 当分支名字后面有一个蓝色箭头，表示本地远程仓已经拉去下来了当前分支的最新代码，本地仓还没有更新到远程的最新的代码

也就是，本地代码不是最新的，直接点击**update**就可以同步到本地仓库，蓝色箭头也消失了

选择非当前分支：

**Checkout** 重新检出本地远程仓的代码到本地

**New Branch from Select** 以当前分支为基准新建分支到本地仓，名字不能和本地仓已有的分支重名。

**Check and rebase onto Current** A分支切换到B分支，B分支变基，其提交记录在A分支提交记录的后面。这个待验证

**compare with Current** 看起来比较分支不同，但是点击了没反应  
**Show diff with work Tree** 通过树状方式比较两个分支代码的不同，蓝色表示有更新，灰色表示一方没有此文件。

**Rebase Current onto Selected** 变基(rebase)

一个用户局可以验证rebase。

创建一个master分支，根据这个master分支创建一个dev分支

master修改两次，提交两次，

切换到dev分支修改两次，提交两次

就在dev分支上，选择master，执行**Rebase Current onto Selected**，

会发现dev分支已经把master的分支的代码合并过来了。同时日志曲线由两条变为一条。

【这里注意一下，**rebase**和**merge**都是合并】，举例说明：1、A分支要把master分支的代码拿过来，直接在A上选择master

执行**rebase**;2、A分支代码要合入master，切换到master上，选择A分支执行**merge**。

**rebase on/merge to** 一个是**on**(我要在谁前面)，一个是**to**(我要跟在谁后面)

**Merge Select into current** 合并分支代码(merge)

**update** 同上面

**push** 同上面

**rename** 同上面

**Delete** 删除当前用户本地库的当前选择分支，本地远程序当前选择分支还存在的

## 3.0 本地远程仓库列表

Remote Branches: 不能直接修改，只能通过本地仓库合并之类方式进行操作，这个其实是远程仓的一个副本，存留在本地而已。

不能删除

远程仓库列表：

**Checkout** 同上

**New Branch from Select** 同上

**compare with Current** 同上

**Show diff with work Tree** 同上

**Rebase Current onto Selected** 同上

**Merge Select into current** 同上

**Pull into Current using Rebase**

**Pull into Current using Rebase**

**Delete** 删除当前用户【本地远程分支】，同时同步删除【远程序的该分支】。会自动提交到远程序删除命令

## 3.0底部左下角Git

## 1.0 左边树目录

## 2.0 中间日志流水线

### 1 头部导航可选按钮

用于筛选过滤想要的日志

**Branch:** 选择某个分支的日志

User

Date

Paths

两个曲线箭头 : refresh

樱桃: cherry-pick

主线和分支线: 首先显示传入的提交, 不太明白

像眼睛的图像: 展示设置

Compact references

Align References to left

Show Tag Names

Show Commit Timestamp

Collapse Linear Branches

Highlight

My Commit

Merge commit

Current Branches

Not Cherry-Picked Commits

Show Columns

易驱线包着+号: Open new Git Log Tab, 打开一个新的日志窗口

## 2 commit右键选项列表

右键某个commit记录, 选项有

**Copy Reversion Number** 可以拿到提交id, 也就是commitID

**create patch**

**cherry-pick** 任意挑选commitID进行合并到当前分支

用于把某些提交commit, 合入到当前分支的代码中。会新增一个提交ID。

idea的log记录的commitID可以用ctrl进行多选, 然后进行挨个解决冲突, 进行cherry-pick;

**checkout Revision 'qweqwe'** 拿到commitID的前几位, 其实前几位足够区分这个版本号的唯一性了。

**Show Repository at Revision** 展示这个提交版本的仓库, 会在右边看到树状的文件目录, 再写文件只在这个提交id之前的所有内容

**compare with local** 【(当前选择的提交id)的内容】和【最新提交的工作目录(包括未提交、新提交等)的差异】,

未跟踪不会参与对比, local 也就是当前最新提交id的状况。

**Reset Current Branch to Here** 回退提交到当前选择分支. reset对应的操作, 相关commitID都会被删除。

**soft** 文件不会改变, 不同的部分会被暂存(staged), 等待提交。也就是文件, 或者新的  
改变会被放在暂存区

(**staged**)，也就是相较与之前版本有新增文件的，变为待提交。

**hard** 新增文件会丢失，原文件与修改回来，选择文件或者文件目录，都会有样子，本地文件会原封不动

**Revert Commit** 回退提交 最新提交移动到选择的提交上，以新的提交去改变，而不是删除提交。也就是**reset**和**revert**的区别实现。

会保留之前的**commitID**，可以在历史提交记录看到，不会被删除

**undo Commit** 撤销提交，撤销上一次提交。改变的文件会暂存，新增文件会被暂存，提交**id**会被删除。HEAD回到上一次提交。

**Edit commit** 可以修改选中的**commitID**的提交信息

**Fixup** 暂时不清楚...

**Squash into** 挤进

**Drop commit** 删除提交，选中的**commitID**的对应提交的所有内容会被删除，记录也会被删除。也不会暂存。直接没了。可以通过

**Local history**找回来

**interactively rebase from here** 变基相关

**New Branch** 以选中的**commitID**为最近提交内容，这些内容作为拉取新分支的起始点。

**new tag** 同上

**go to Parent commit** 以当前选中**commitID**为基准，【鼠标会切换】到上一次提交的**commitID**上

**go to child commit** 以当前选中**commitID**为基准，【鼠标会切换】到下一次提交的**commitID**上

**open on gitee** 外部跳转链接

### 3.0右边选择某个日志流水线的commit记录

就会显示出来**当前提交与之前提交的差异文件**，可以进行对比，右键 **show Diff in a new tab**

右键选项：

**Show Diff** 点了没反应

**Show diff in new Tab** 可以对比看有啥不同和上一次提交。（左边前一次提交，右边当前提交）

**compare with local**

**compare before with local**

**edit soure** 快速跳转至源文件

**open repository version** 跳转至源文件仓库

**revert selected changes**

**cherry-pick selected changes**

**history up to here** 过滤出这个文件提交的历史记录，从一开始到当前提交的记录都会被过滤出来。

**show changes to parents**

## 4.0右击击文件内容-Git

跟头部导航基本一样

## 5.0工作目录的文件右击选择Git

跟头部导航基本一样

## 6.0左边导航栏commit

amend :修正、修订

修正一下最新的一次提交。类似于把提交ID给替换掉了，提交信息也可以更改

## 7.0 idea中集成git的.gitignore文件的原理

需不需要自己额外新建一个在工作目录中？.idea的文件中自带这个文件

## 8.0idea中git颜色管理文件状态

棕色	已忽略
红色	待追踪
绿色	已追踪+已添加至暂存区+待提交至仓库
白色	已追踪+已提交至仓库
蓝色	已追踪+提交过再次被修改（自动添加至暂存区）+待将修改提交至仓库

棕色的也可以提交，提交后和其他文件一样被git管理起来了，此时.gitignore里面的对这个文件的【忽略作用已经失效了】。