



9、分支操作

0、简单介绍

(0) 文档



Git 教程 - v1.0.pdf

2 MB



git笔记.doc

200 KB

(1) 廖雪峰由浅入深GIT教程

- <https://www.liaoxuefeng.com/>

(2) 阮一峰GIT远程操作

- http://www.ruanyifeng.com/blog/2014/06/git_remote.html

(3)

- <https://git-scm.com/book/zh/v2>

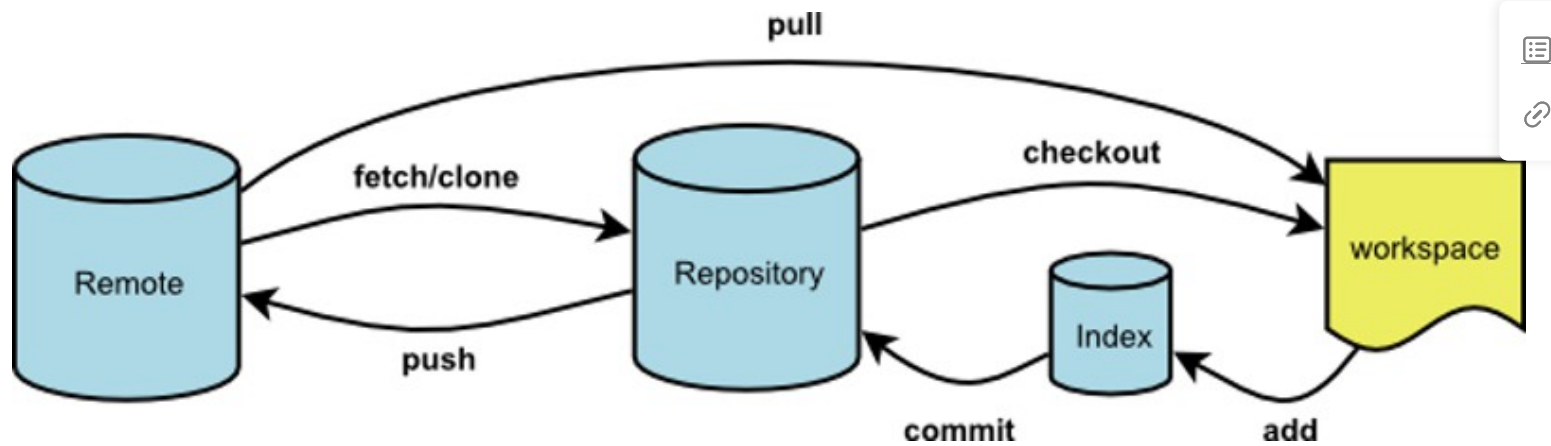
(4) 知乎GIT文章

- <https://zhuanlan.zhihu.com/p/30044692>

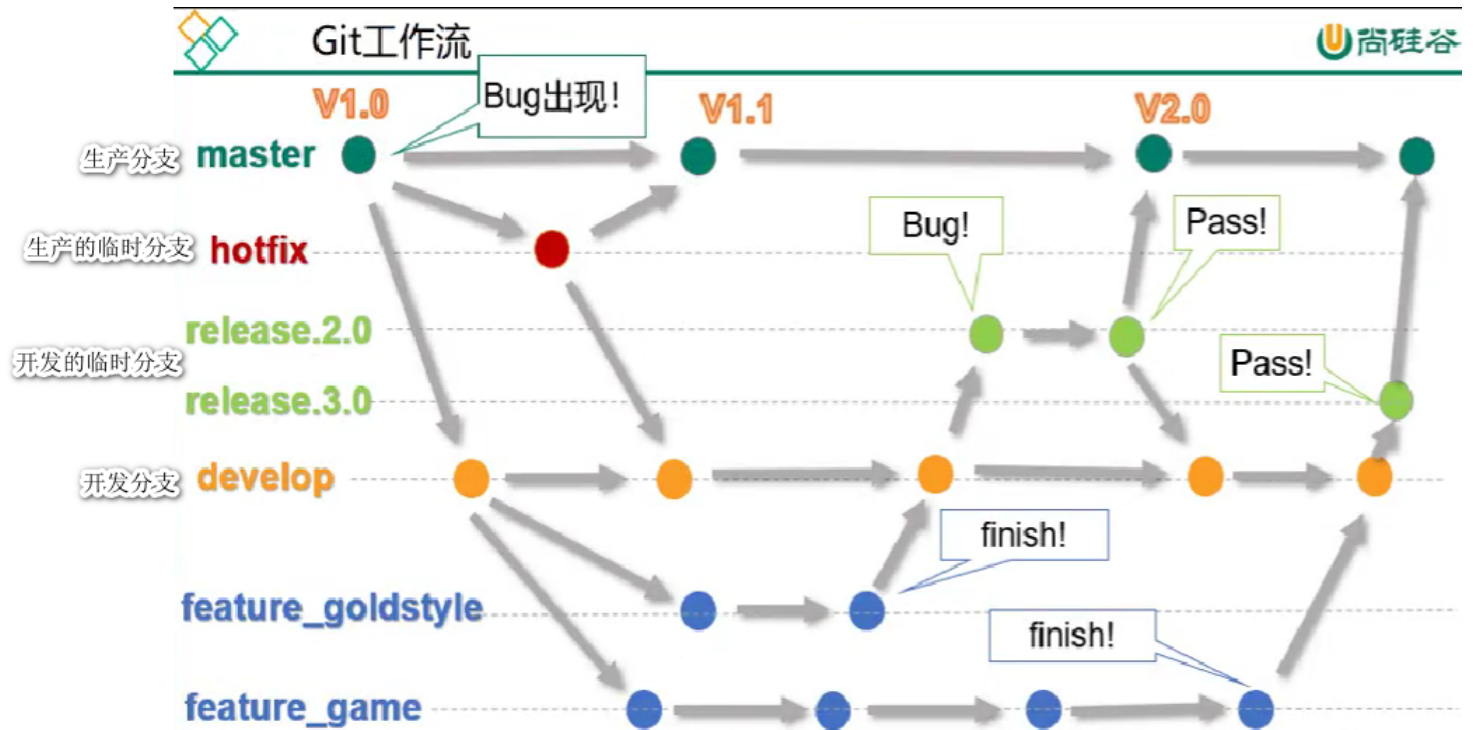
(5) git分支操作 及 远程新建分支后,本地查看不到

- <https://www.cnblogs.com/aaron-agu/p/10454788.html>

(6) git命令图解



(7) git实际开发中分支的创建和使用



1、branch,checkout,add,commit,push新建并推送分支

新建分支并切换到此分支，并将本地分支的修改提交到本地仓库，然后推送到远程库对应的远程分支上(先本地后远程)

- 分支、HEAD指针

每次提交，Git都把它们串成一条时间线，这条时间线就是一个分支，在Git里，这个分支叫主分支，即master指针指向的分支。

HEAD指针严格来说不是指向提交，而是指向分支（指针），只是说默认是一直指向了分支的最新提交版本位置，所以，HEAD指针指向的是当前分支（指针），和某个提交版本无关。如果从master分支（指针），切换到dev分支（指针），则HEAD指针指向的是dev分支（指针），默认是指向了dev分支（指针）的最新提交版本位置。

- 分离HEAD指针与分支，使HEAD指针指向此分支的某个提交版本



当然也可以修改HEAD指针不指向分支（指针）的最新版本（master,dev.,代表某个分支），而是指向此分支（指针）的某个版本（某个节点），这叫分离HEAD。

分离HEAD语法（和版本回退很相似，这里只是HEAD指针指向此分支的版本往回退）：

//HEAD分离并指向前一个节点

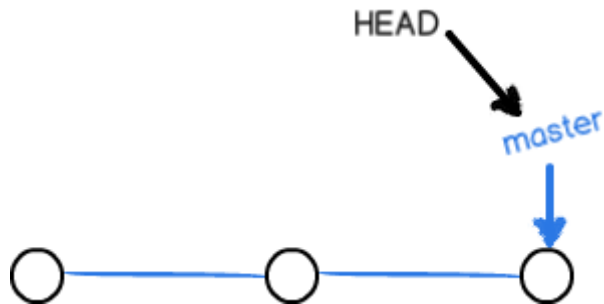
git checkout 分支名/HEAD^

//HEAD分离并指向前N个节点

git checkout 分支名~N

分离HEAD的目的

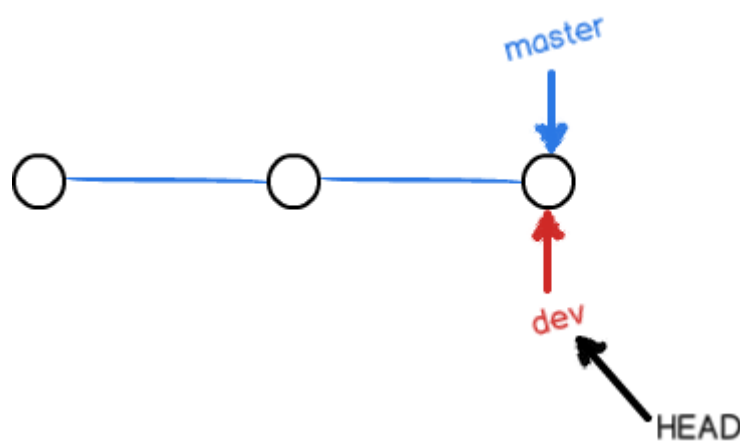
如果开发过程发现之前的提交有问题，此时可以将HEAD指针指向分支（指针）对应的节点，修改完毕后再提交，此时你肯定不希望再生成一个新的节点，而你只需在提交时加上--amend即可。git commit --amend



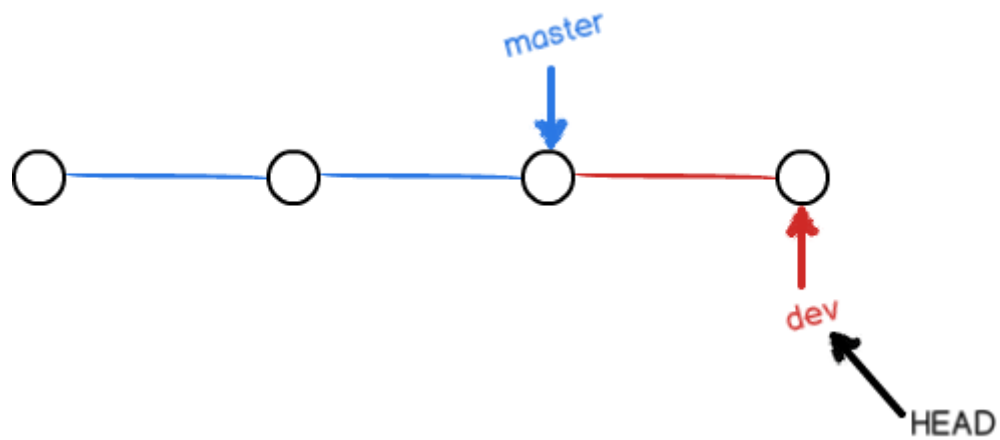
- 创建新分支的本质

Git创建一个分支很快，实质上只是增加一个dev指针（和master指针指向的其实是同一个分支），改改HEAD指针的指向（指向dev指针，dev指针指向了分支），工作区的文件都没有任何变化！

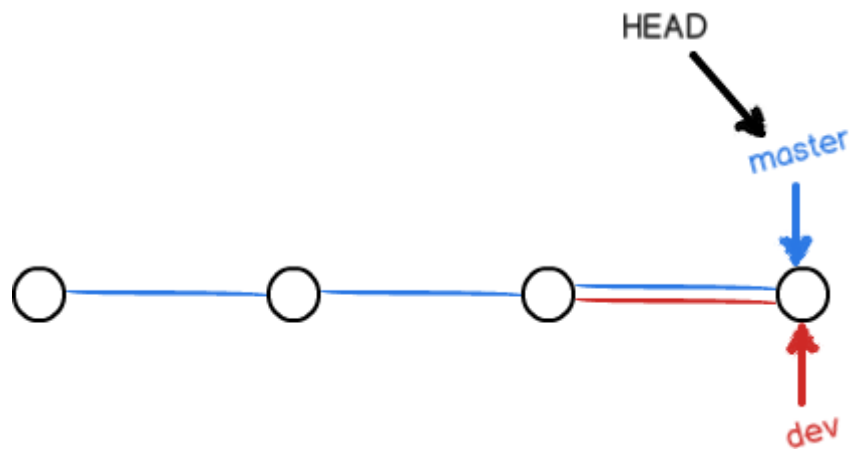
解释：以树比喻，可以把master看作是本地的必须存在的主干或者主分支，其他分支可以看作是master的分支或者分枝，是基于主干的分叉，其他分支最终都会合并至master分支。其实创建一个新的分支，只是创建一个dev指针，和master指针一样，都是指向此分支的最新提交版本。HEAD指针指向这个dev指针。



当HEAD指针指向dev指针时，从现在开始，对工作区的修改和提交就是针对dev指针了，比如新提交一次后，dev指针往前移动一步，而master指针不变。即更改dev指针指向的分支的时间标记点，同时使HEAD指向当前dev指针指向分支的最新时间标记点。（准确地说，是通过添加新的时间标记点来增长）

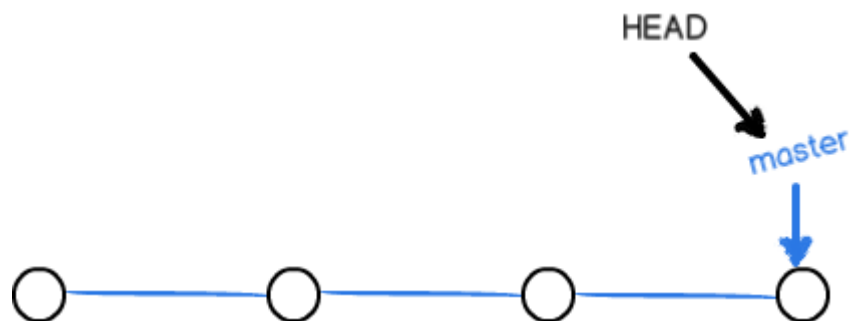


我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指针指向dev指针指向的分支的当前提交，就完成了合并。



所以Git合并分支也很快！就改改指针，工作区内容也不变！

合并完分支后，甚至可以删除dev分支。删除dev分支实质是把dev指针给删掉，删掉后，我们就剩下了一条master指针指向的分支。



```
1 //新建本地分支
2 git branch dbg_lichen_star
3 //切换到新分支
4 git checkout dbg_lichen_star
5 或者
6 //新建并同时切换到新分支
7 git checkout -b dbg_lichen_star
8
9 //使用新的git switch命令切换分支, 比git checkout要更容易理解。
10 我们注意到切换分支使用git checkout <branch>, 而前面讲过的撤销
    修改则是git checkout -- <file>, 同一个命令, 有两种作用, 确实有
    点令人迷惑。
```

11

复制代码

```
12 实际上, 切换分支这个动作, 用switch更科学。因此, 最新版本的Git提供了新的git switch命令来切换分支:
13
14 创建并切换到新的dev分支, 可以使用:
15 $ git switch -c dev //或git checkout -b dev
16
17 直接切换到已有的master分支, 可以使用:
18 $ git switch master //git checkout dev
19
20 //将工作区的修改添加到暂存区
21 git add 文件名.后缀名 //将某文件添加到暂存区
22 git add . //将当前目录及子目录的文件添加到暂存区
23
24 //将暂存区的修改提交到本地仓库
25 git commit -m "提交注释";
26
27 //推送本地分支到远程(远程分支, 在推送到远程仓库前对其他人是不可见的)
28 git push <远程主机名> <本地分支名>:<远程分支名>
29 git push origin dbg_lichen_star:dbg_lichen_star
30 或 git push -u origin dbg_lichen_star:dbg_lichen_star
   // -u在笔记8中讲过, 下面暂时都没用-u
31
32
33 //如果省略远程分支名, 则表示将本地分支推送与之存在"追踪关系"的远程分支(通常两者同名), 如果该远程分支不存在, 则会被新建。
34 git push origin dbg_lichen_star
35
36 //如果省略本地分支名, 则表示删除指定的远程分支, 因为这等同于推送一个空的本地分支到远程分支。
37 git push origin :dbg_lichen_star
38 等同于
```




```
39 git push origin --delete dbg_lichen_star
40
41 //如果当前分支与远程分支之间存在追踪关系，则本地分支和远程分支都可以省略。即将当前本地分支内容推送到origin主机的对应远程分支。
42 git push origin
43
44 //如果当前分支只有一个追踪分支，那么主机名都可以省略。
45 git push
46
47 //如果当前分支与多个主机存在追踪关系，则可以使用-u选项指定一个默认主机（这里使用默认主机是origin），将本地的dbg_lichen_star分支推送到origin主机，同时指定origin为默认主机，后面就可以不加任何参数使用git push了。
48 git push -u origin dbg_lichen_star
49
50
```

javascript >

1-2、git fetch --prune远程分支拉取到本地（拉取前此时还没有本地分支）

-

```
//根据远程仓库origin已经存在的远程分支,创建指定名称的本地分支
dbg_lichen_star1
git checkout -b dbg_lichen_star1 origin/dbg_lichen_star
```

或者

```
//拉取别人已经推送到远程的分支到本地,然后再切换到此分支
git fetch --prune
git checkout xxx
```

2-1、git remote查看远程所有主机

```
git remote
```

2-2、git branch查看本地所有分支, 远程所有分支

复制代码

```
1 (1) 所有
2 git branch -a (all) //查看本地和远程的所有分支(本地分支和远程
  分支没有以映射方式显示, 竖向列表显示, 远程分支以remotes/origin主
  机标识)
3 git branch -vv //查看本地分支和远程分支(本地分支和远程分支横向
  以映射方式显示, 可显示出本地分支是否有对应远程分支及是否同步。没有
  对应则gone, 没有同步则behind, ahead, diverged等)
4
5
```

```
6 (2) 本地
7 git branch //查看本地所有分支
8 git branch -v (abbrev) //查看本地所有分支(带提交版本号及备注)
9
10 (3) 远程
11 git branch -r //查看远程所有分支
12
13 (4) 自定义匹配分支
14 git branch --all | grep remotes | grep -v origin/HEAD//
    只显示远程分支, 并且不显示匹配到origin/HEAD字符串的分支
15
16 grep 管道命令
17 没有选项时:
18     只显示匹配到的字符串所在的行。可使用grep管道命令多次匹配 |
    grep xxx |grep xxx ....
19 常用选项:
20     -E : 开启扩展 (Extend) 的正则表达式。
21     -me : 忽略大小写 (ignore case) 。
22     -v : 反过来 (invert) , 只打印没有匹配的, 而匹配的反而不打印。
23     -n : 显示行号
24     -w : 被匹配的文本只能是单词, 而不能是单词中的某一部分, 如文本
    中有liker, 而我搜寻的只是like, 就可以使用-w选项来避免匹配liker
25     -c : 显示总共有多少行被匹配到了, 而不是显示被匹配到的内容, 注
    意如果同时使用-cv选项是显示有多少行没有被匹配到。
26     -o : 只显示被模式匹配到的字符串。
27     --color :将匹配到的内容以颜色高亮显示。
28     -A n: 显示匹配到的字符串所在的行及其后n行, after
29     -B n: 显示匹配到的字符串所在的行及其前n行, before
30     -C n: 显示匹配到的字符串所在的行及其前后各n行, context
31
32
33 git branch --merged //查看哪些分支已经合并入当前分支
```



3、

①查看当前分支的工作空间的文件状态

②隐藏不提交的内容

③合并本分支内容(只会合并本分支已经提交的内容)到当前主分支master上，然后推送到远程服务器

④删除此分支

⑤恢复隐藏的内容(可以恢复到本分支，也可以恢复到其它任意分支，前提是同一个项目不同分支)

⑥将promoter分支的指定文件或指定的文件夹下的所有文件，直接覆盖掉当前master分支的对应文件或文件夹下的所有文件

复制代码

```
1 //1
2 git status
3
4 //2 使用目的：分支代码修改后未提交时，进行切换分支会报错，可以先隐藏起来
5 git stash
6
7 //3、需要先切换到主分支master，再将分支dbg_lichen_star分支的提交合并到当前所在主分支中，然后推送master到远程
8 git merge dbg_lichen_star
9 git push
10
```

```
11 // 4、若合并后，本地分支dbg_lichen_star不再需要，可以删除（-d删除本地分支）
12 $ git branch -d dbg_lichen_star
13 
14 
15 // 5、先切换到需要恢复内容的分支上（暂存区内容与哪条分支及有无分支无关，目的是合并分支时，若本地有正在开发的代码，可以进行隐藏，则只合并提交的代码。分支删除后暂存区内容也存在，暂存区内容可以恢复到任何分支）
16 git checkout 切换到需要恢复内容到的那个分支名
17 git stash pop
18 
19 // 6、但严格来说，这似乎不是合并，而是用另一分支"source_branch"的指定文件直接覆盖当前分支的相应文件。
20 git checkout source_branch <paths>...
```

javascript >



```

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace4/sinovac_order_server (promoter)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace4/sinovac_order_server (master)
$ git checkout promoter d:/idea_workspace4/sinovac_order_server/src/main/java/com/scmsafe/sinovac/model/entity/region
Updated 5 paths from eb8ebcb

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace4/sinovac_order_server (master)
$ git checkout promoter d:/idea_workspace4/sinovac_order_server/src/main/java/com/scmsafe/sinovac/controller/pc/region
Updated 2 paths from eb8ebcb

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace4/sinovac_order_server (master)
$ git checkout promoter d:/idea_workspace4/sinovac_order_server/src/main/java/com/scmsafe/sinovac/service/region
Updated 2 paths from eb8ebcb

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace4/sinovac_order_server (master)
$ git checkout promoter d:/idea_workspace4/sinovac_order_server/src/main/java/com/scmsafe/sinovac/mapper
Updated 12 paths from eb8ebcb

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace4/sinovac_order_server (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   src/main/java/com/scmsafe/sinovac/controller/pc/region/AreasController.java
    modified:   src/main/java/com/scmsafe/sinovac/controller/pc/region/RegionController.java
    new file:   src/main/java/com/scmsafe/sinovac/mapper/AreasMapper.java
    new file:   src/main/java/com/scmsafe/sinovac/model/entity/region/dto/Areas.java
    new file:   src/main/java/com/scmsafe/sinovac/model/entity/region/vo/AreasFullVO.java
    new file:   src/main/java/com/scmsafe/sinovac/model/entity/region/vo/AreasVO.java
    new file:   src/main/java/com/scmsafe/sinovac/model/entity/region/vo/ImportAreasVO.java
    new file:   src/main/java/com/scmsafe/sinovac/model/entity/region/vo/ProvinceVO.java
    new file:   src/main/java/com/scmsafe/sinovac/service/region/IAreasService.java
    new file:   src/main/java/com/scmsafe/sinovac/service/region/impl/AreasServiceImpl.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace4/sinovac_order_server (master)

```

4-1、删除远程分支(不影响暂存区内容，暂存区内容可以pop到任何分支中)

1、删除远程分支

复制代码

- 1
- 2 // 要删除dbg_lichen_star的远程分支，我们必须先把分支切换到其它分支(这里使用主分支master)，因为你现在所在的分支就是dbg_lichen_star，在这个分支下，是不能删除它的

```
3 git checkout master //必须先切换到主分支
4
5
6
7 //删除与本地分支追踪的远程分支。（若远程分支名与本地分支名一致，可
  省略远程分支名）
8 git push origin --delete dbg_lichen_star
9
10 等同于
11 //如果省略本地分支名，则表示删除指定的远程分支，因为这等同于推送一
   个空的本地分支到远程分支。
12 git push origin :dbg_lichen_star
13
14 等同于
15 //解释：其中<branchname>为本地分支名 -D强制删除，相当于 --
   delete --force
16 git branch -d|-D -r origin/<branchname>
17
18 //删除本地追踪的远程分支。
19 git branch -d -r origin/dbg_lichen_star
```

swift >

4-2、删除本地分支（一般先删除远程分支后，才再删除本地分支）

```
1 git checkout master //必须先切换到其它分支(如master)，才能去删
  除dbg_lichen_star本地分支
2
```

复制代码

```
3 git branch -d dbg_lichen_star //删除本地分支，因为远程分支已经
删除此时会提示本地分支没有被合并，若需保留本地分支内容那就合并到其它
分支，否则直接强制删除就好（-D 强制删除）
```

javascript >



4-3、git branch -m重命名分支

复制代码

```
1 //方法1：先重命名本地分支，删除远程分支，推送新命名的本地分支到远
程
2
3 //①、只重命名本地分支，使用-M则表示强制重命名。（此时只有本地分
支，远程还没有推送的情况）
4 git branch -m|-M <oldbranch> <newbranch>
5
6 //②、重命名远程分支（假如本地与远程分支同名）
7 a. 重命名本地分支
8 git branch -m old-local-branch-name new-local-
branch-name
9 b. 删除远程旧分支
10 git push origin :old-local-branch-name
11 c. 推送新命名的本地分支（不写远程分支名，默认和本地分支名一样）
12 git push -u origin new-local-branch-name
13
14 =====
15
16 //方法2：先重命名本地分支，推送新命名的分支，删除远程旧分支
17 a. 本地分支重命名
18 git branch -m oldLocalName newLocalName
```



```
19 b. 将重命名后的分支推送到远程
20     git push -u origin newLocalName
21 c. 删除远程的旧分支
22     git push --delete origin oldLocalName
23
24 =====
25
26
27 //方法3: 若只是本地分支名和远程分支名称不一致, 只修改本地分支名称
    和远程保持一致即可, 前提是已经建立关联的分支
28 git branch -m oldLocalName newLocalName //newLocalName和
    远程分支名称相同
```

javascript >

5、git log查看提交历史

```
1
2 //从最近到最远的显示提交日志
3 git log
4
5 *****按照关键字查询显示部分提交日志*****
6 //显示添加或移除了某个关键字的相关提交 (代码文件中内容检索查询)
7 git log -S "代码片段"
8
9 //仅显示含指定备注信息的提交(提交时的备注检索查询)
10 git log --grep "提交时的备注信息"
11
12 //仅显示指定作者的提交
13 git log --author "作者名称"
14
15 //仅显示指定时间之前的提交
```

复制代码

```
16 git log --before "yyyy-MM-dd HH:mm:ss"
17
18 //仅显示指定时间之后的提交
19 git log --after "yyyy-MM-dd HH:mm:ss"
20
21 //注意: -G是查正则格式的文本内容
22 git log -G "你要查的正则格式的文本内容" -p
23
24 //查找此文件的提交历史记录。
25 //使用方式: 先切换到你文件所在的目录, 然后再使用命令查找。
26 git log -p "文件名"
27 比如: git log -p "test.java"
28
29 *****显示全部提交日志
   *****
30 //显示提交日志: 包括提交版本号, 提交作者, 提交日期, 版本备注信息,
   几个文件变更, 总添加行数的动作次数, 总移除行数的动作次数。
31 git log --shortstat
32
33 //显示提交日志: 包括提交版本号, 提交作者, 提交日期, 版本备注信息,
   几个文件变更, 并列出行每个变更的文件及文件的添加移除行数的动作次数之
   和(+代表添加一行的动作, -代表移除一行的动作), 总添加行数的动作次
   数, 总移除行数的动作次数。
34 git log --stat
35
36
37
38 *****显示最近的n条提交日志
   *****
39 //仅显示最近的 n 条提交
40 git log -n //git log -3 , 最近的 3 条提交
41
```



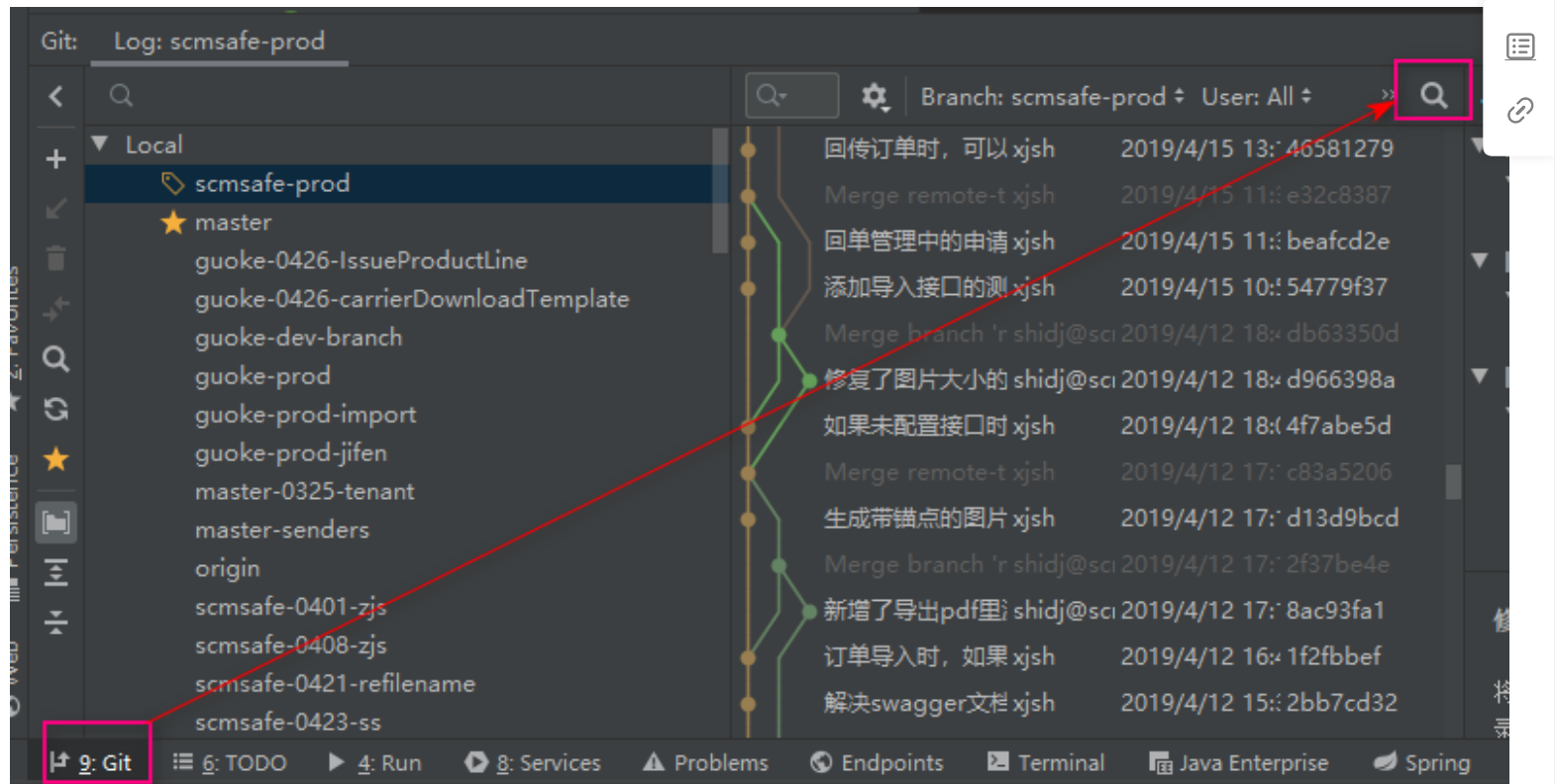
```
42 // 显示最近n次提交有哪些文件被修改
43 git log -n --name-status (这里n是数字, 代表按照最新时间倒着显示此分支几条提交日志)
44
45
46
47 *****显示某个提交的日志*****
48 // 只显示某个提交的修改
49 git show commit-id
50
51 // 只显示某个版本中的具体某个文件的修改
52 git show commit-id filename
53
54 // 定位到某个提交版本
55 先通过git log -S "代码片段"或git log --grep "提交时的备注信息" 命令, 过滤拿到所需commit_id
56 idea左下角的Git, 打开图形化的git历史记录, 右上角有搜索框, 输入commit_id, 回车即可定位。具体见下图
57
58
59 *****格式化提交日志*****
60 // 格式化的显示日志(以列表方式显示提交日志, 不是以树状显示)
61 git log --pretty=oneline
62
63 // 格式化后以树状的显示提交日志(显示提交日志的全部序列号)
64 git log --graph --pretty=oneline
65
66 // 格式化后以树状的显示提交日志(只显示提交日志的前7位序列号)
67 git log --graph --pretty=oneline --abbrev-commit
68
69
70
```



```
71 主要参数选项如下:
72     -p: 按补丁显示每个更新间的差异
73
74     --stat: 显示每次更新的修改文件的统计信息
75
76     --shortstat: 只显示--stat中最后的行数添加修改删除统计
77
78     --name-only: 尽在已修改的提交信息后显示文件清单
79
80     --name-status: 显示新增、修改和删除的文件清单
81
82     --abbrev-commit: 仅显示SHA-1的前几个字符, 而非所有的40个
83 字符
84
85     --relative-date: 使用较短的相对时间显示 (例如: "two weeks
86 ago")
87
88     --graph: 显示ASCII图形表示的分支合并历史
89
90     --pretty: 使用其他格式显示历史提交信息
91
91 按q退出日志页面
```

javascript >





6、git reflog查看所有命令历史痕迹(一般在版本重返未来时，找到回退操作之前commit的版本)

- 复制代码
- 1 版本4
 - 2 版本3
 - 3 版本2
 - 4 版本1
 - 5 // 当你不小心回退到版本3后又想回退到版本4的操作，则可以通过如下命令跳跃来回穿梭版本

```
6 git reflog //找到回退前的版本4的版本号commit_id
7 git reset --hard commit_id //此时回到了版本4
```

javascript >



7-1、git diff用来比较文件之间的不同

复制代码

```
1 //显示某个文件在此区(工作区或暂存区)与最后一次commit (本地版本库
  最新版本) 之后的区别
2     git diff HEAD --文件名
3
4 //显示某个文件在不同分支之间的详细差异
5     git diff branch1 branch2 具体文件路径
6
7 //显示暂存区(已add但未commit文件)和最后一次commit(HEAD)之间的
  所有不相同文件的增删改
8     git diff --cached 或 git diff --staged
9
10 //比较两个分支在最后 commit 之后, 所有有差异的文件
11     git diff <分支名1> <分支名2>
12
```

javascript >

```
D:\idea_workspace\remote-git1>git diff dev22 dev33
diff --git a/cc.txt b/cc.txt
index 09d40f1..69af2a9 100644
--- a/cc.txt
+++ b/cc.txt
@@ -4,3 +4,4 @@
+CCCC
\ No newline at end of file
```

变动前的文件

变动后的文件

变动的结束位置

变动的起始位置

变动的内容

变动前的文件cc.txt, 第4行开始, 连续3行。

变动后的文件cc.txt, 第4行开始, 连续4行。

7-2、查看暂存区中有上一次commit 过的所有文件，之后如果工作区中有文件改动，其实比较的也是工作区和暂存区的差异

git ls-files

7-3、git log -p查看本地分支与远程分支的区别

git log -p master ..origin/master //比较本地的master分支和origin/master分支的差别



8~9、在未进行git push前的所有操作，都是在“本地仓库”中执行的。我们暂且将“本地仓库”的代码还原操作叫做“撤销”。（撤销包括撤回修改和版本回退）

8-1、git reset回退版本：（直接回退到指定的某个版本，而此版本之后的版本都已经丢失）

- reset是直接回退到指定的某个版本，而此版本之后的版本都已经丢失。
- 其实所有的git reset操作都只是本地仓库的操作，哪怕版本丢失，都能通过git reflog还能找回来。
- 只有通过push后，才是保持远程和本地进行了同步。
- 不要害怕丢失版本，只要commit过，通过git reflog进行版本的历史穿梭，可跳到对应版本，此版本之前的版本都还存在(commit之后就算一个版本，只是此时远程和本地的版本不是同步一致的情况而已)

<https://blog.csdn.net/xybelieve1990/article/details/62885292>

<https://blog.csdn.net/asoar/article/details/84111841>

<https://www.cnblogs.com/aligege/p/10221174.html>

复制代码

```
1 //注意：执行回退版本命令后^被当作换行符进行换行，more? 的意思是问你下一行是否需要再输入
2 //使用Terminal操作时是window命令行(dos命令行，cmd命令行)，换行符默认是^，而不是\，所以需要转义。
3 //4种解决方案：
4 // (1) 加引号：git reset --hard "HEAD^"
5 // (2) 加一个^：git reset --hard HEAD^^
```




```
6 // (3) 换成~: git reset --hard HEAD~ 或者 git reset --hard
  HEAD~1~ 后面的数字表示回退几次提交, 默认是一次当然.
7 // (4) 可以换成git bash, powershell等就不会出现这种问题了
8
9 git reset --hard HEAD^ //本地仓库回退到最新版本的上一个版本
10
11 git reset --hard HEAD^^ //本地仓库回退到最新版本的上上个版本
12
13 git reset --hard HEAD~100 //本地仓库回退最新版本的上100个版本
  本
14
15 git reset --hard 3628164 3628164为commit id//本地仓库回
  退到某个commit的版本
16
17 //使本地分支和远程分支保持同步, 丢弃本地分支的所有更改, 保持与
  origin/master远程分支最新版本完全相同。
18 git reset --hard origin/分支名
19 git reset --hard origin/master
20
21
22
23 //本地和远程分支同步回退版本
24 git reset只是在本地仓库中回退版本(假如回退一个版本), 而远程仓库
  的版本不会变化 .即本地仓库版本比远程仓库少一个版本。即使本地reset
  了, 但如果再git pull, 那么, 远程仓库的内容又会merge到本地。
25
26 方法1: 直接在远程server的仓库目录下, 执行git reset --
  soft commit_id来回退。注意: 在远程不能使用mixed或hard参数。
27 方法2: 在本地直接把远程的分支给删除, 然后再把reset后的分支内
  容给push上去。
```



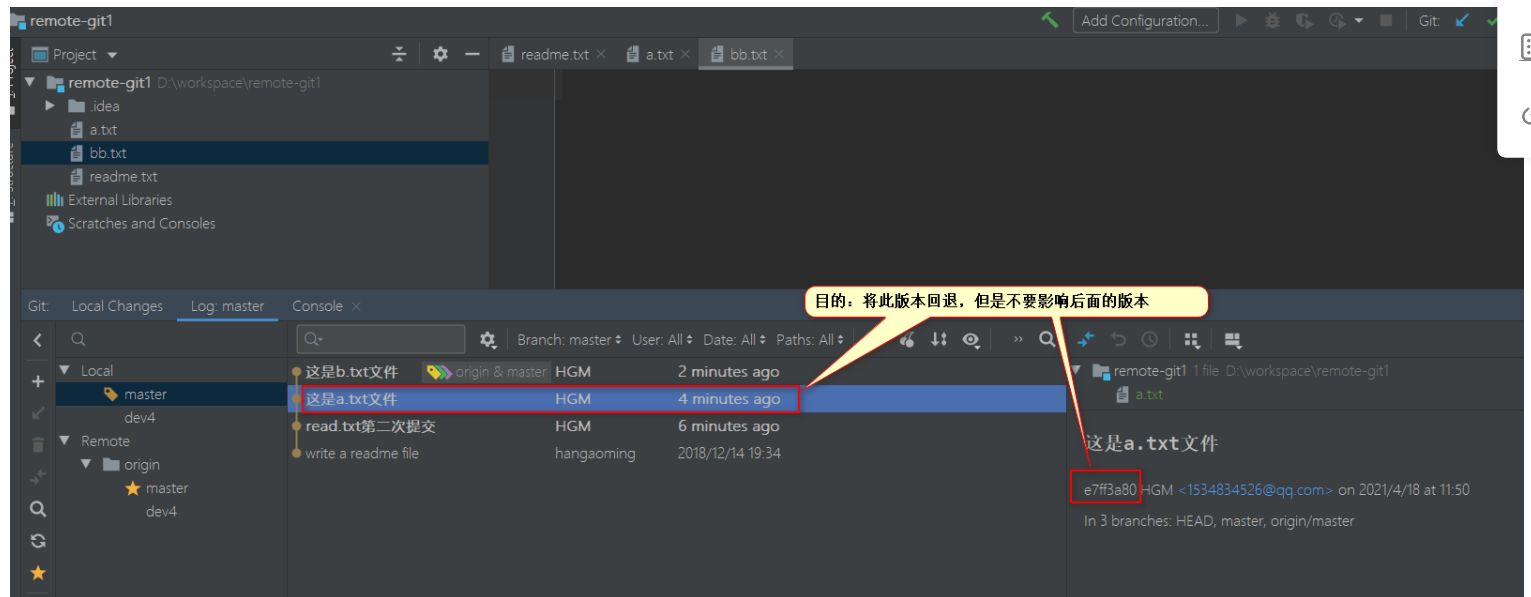
8-2、git revert回退版本(也可叫做逆转某个版本，只回退某个版本，不影响此版本的之后版本的修改)

- 执行指令后，会生成一个新的版本，新版本的修改（只会把要逆转的那个版本的修改给丢弃删除，保留了逆转版本之后提交的版本）。
- 所有的git reset操作都只是本地仓库的操作,最后才会通过push保持远程和本地同步。
- <https://blog.csdn.net/yxlshk/article/details/79944535>

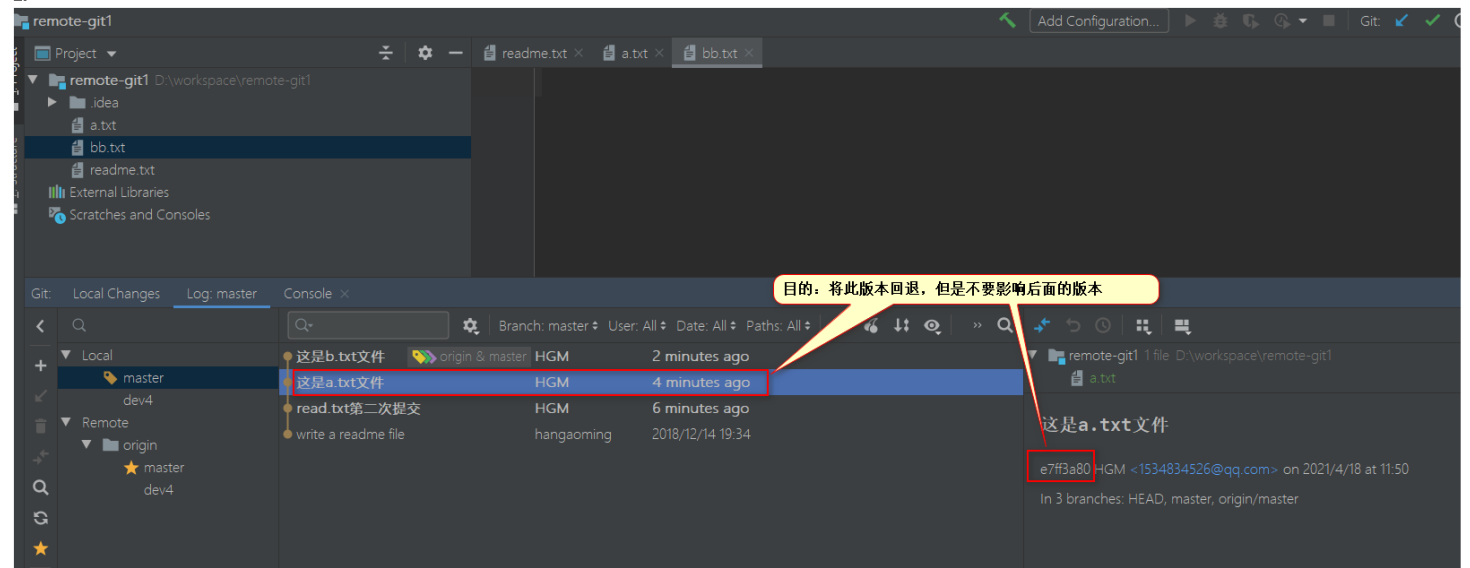
```
git revert -n 要逆转的版本号
git status //暂存区已经有一个删除的修改，就是会将要逆转的那个版本内容丢弃
git commit -m ""//将删除状态给提交到版本库
//如果此时感觉没问题就直接git push 即可，如果突然不想逆转回退了，回到逆转某个版本前的状态，执行如下命令
回到最新版本的上个版本 git reset --hard HEAD^或者git reset --hard
commit_id ( commit_id新版本的上个版本的commit_id )
```

示例：

- 回退某个版本



1.



2.



```
D:\workspace\remote-git1>
```

3.

```
D:\workspace\remote-git1>git status
On branch master
Your branch is up to date with 'origin/master'.

You are currently reverting commit e7ff3a8.
(all conflicts fixed: run "git revert --continue")
(use "git revert --skip" to skip this patch)
(use "git revert --abort" to cancel the revert operation)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:    a.txt
```

查看状态，此时已经将a.txt删除状态加入暂存区

```
D:\workspace\remote-git1>git commit -m "将e7ff3a80此版本回退(a.txt)"
[master d7d85be] 将e7ff3a80此版本回退(a.txt)
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 a.txt

D:\workspace\remote-git1>
```

将删除状态提交commit到本地版本库

4.

The screenshot shows the IntelliJ IDEA interface with the 'remote-git1' project. The 'Project' view on the left shows the file structure: 'remote-git1' (D:\workspace\remote-git1) containing 'idea', 'bb.txt', and 'readme.txt'. The 'Git' tool window at the bottom shows the 'Log: master' view. The log entries are as follows:

Commit	Author	Date	Message
d7d85be2	HGM	Moments ago	将e7ff3a80此版本回退(a.txt)
d7d85be1	HGM	10 minutes ago	这是b.txt文件
d7d85be0	HGM	13 minutes ago	这是a.txt文件
d7d85be2	HGM	14 minutes ago	read.txt第二次提交
d7d85be1	hangaoming	2018/12/14 19:34	write a readme file

The 'Local Changes' view shows 'a.txt' as deleted. The 'Console' view shows the output of the 'git commit' command. A red arrow points from the 'a.txt' file in the 'Project' view to the 'Log: master' view, highlighting the commit message '将e7ff3a80此版本回退(a.txt)'. A yellow callout box points to the commit message, stating: '可以看到新版本中，已经不包含e7ff3a80这个版本所提交的a.txt文件了'.

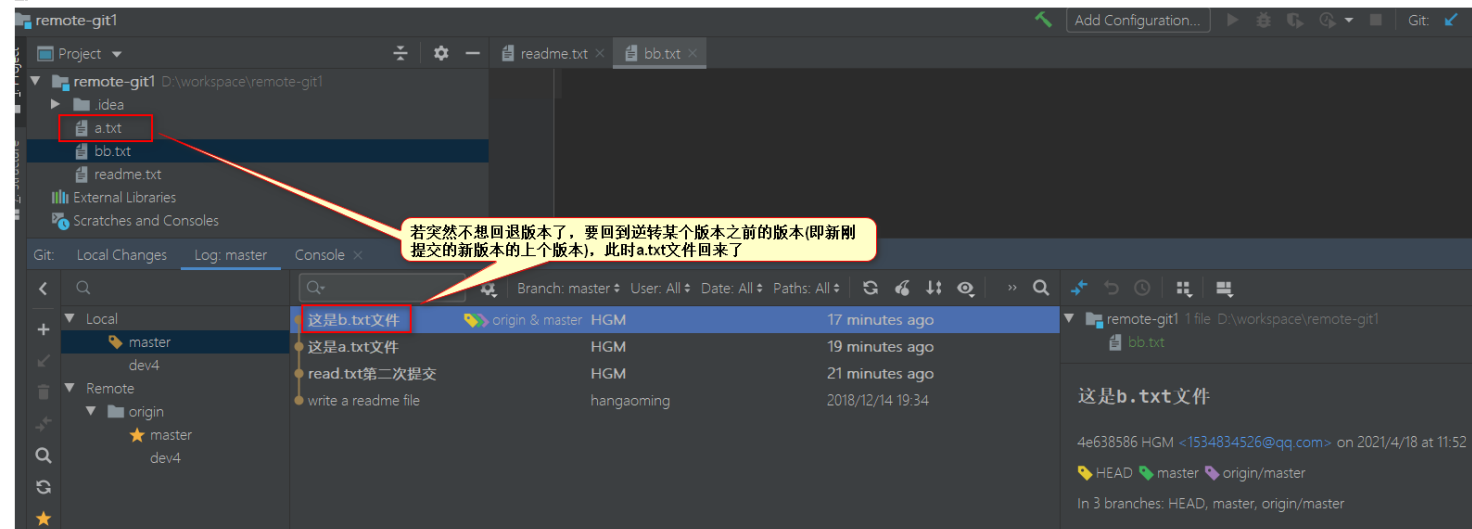
- 还原（取消刚刚上面的回退操作）

1.

```
D:\workspace\remote-git1>git reset --hard "HEAD^"  
HEAD is now at 4e63858 这是b.txt文件  
D:\workspace\remote-git1>
```

刚提交的最新版本的上个版本

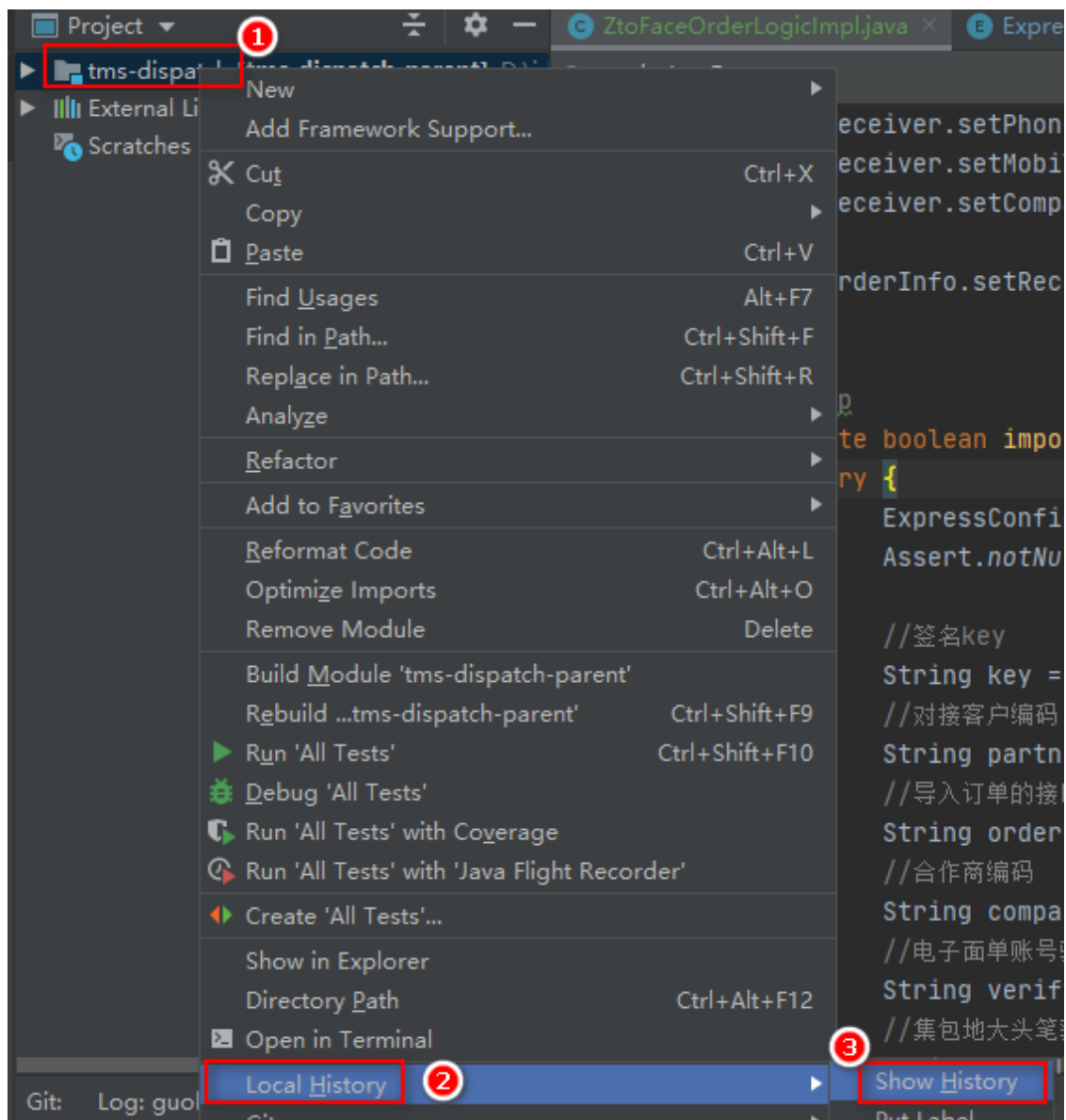
2.



9、checkout,restore,reset, revert将修改撤销(文件新增或修改，给撤销到变更前的状态)

- 在未进行git push前的所有指令操作，都是在“本地仓库”中执行的(比如 add,commit,reset, checkout --.). 我们暂且将“本地仓库”的reset代码还原操作叫做“撤销本地版本”，将“工作区”的checkout --.操作叫做“丢弃本地修改”
- 参考

- <https://www.cnblogs.com/dream397/p/13729260.html>
- <https://www.cnblogs.com/qlqwjy/p/8378851.html>
- 工作区的修改不小心给git checkout --.丢弃了，可以通过Show History找到历史的操作文件的记录，可以把修改的文件一个一个给恢复。





```
1 1、丢弃工作区的修改
2 git checkout -- a.txt    // 丢弃工作区某个文件的修改，即让这个
   文件回到最近一次git commit(还原到和版本库一致)或git add时的状
   态。
3      //解释：
4      //一种是a.txt自修改后还没有被放到暂存区，现在，撤销修改
   就回到和版本库一模一样的状态；
5      //一种是a.txt已经添加到暂存区后，又作了修改，现在，撤销修
   改就回到git add添加到暂存区后的状态。
6
7 或者
8      git checkout -- .      // 丢弃工作区全部文件的修改。对
   之前已经保存在暂存区里的代码不会有任何影响，只是工作区文件的操作。
   即只丢弃当前目录下所有工作区文件的修改，还原到和版本库一致
9
10 注意：但是新建的文件和文件夹无法清除，还必须使用：git clean -df
   清除所有新建的文件及文件夹
11
12
13
14
15 2、丢弃添加到暂存区的修改回到工作区修改（代码git add到缓存区（暂存
   区），并未commit提交）
16 //撤销当前目录及子目录下的所有添加到暂存区的修改，回到工作区的修
   改。
17 git reset HEAD .
18 等同 git reset HEAD -- .
19 等同 git restore --staged .
20
21
22 //撤销暂存区单个文件的修改回到工作区的修改
```

```
23 git restore --staged a.txt
24 等同 git reset HEAD <file>
25 等同 git reset HEAD a.txt
26 //1、撤销单个文件的暂存区的修改回到工作区修改
27 //2、这个命令仅改变暂存区，并不改变工作区，这意味着在无任何其他操作的情况下，工作区中的实际文件同该命令运行之前无任何变化
28
29
30
31 3、git commit到本地分支、但没有git push到远程
32 git log # 先得到已经提交过的commit id
33 git reset HEAD^ // 本次的commit回撤到工作区，本地分支的工作区代码保留，回到 git add 之前
34 或
35 git reset --mixed HEAD^
36
37 解释：
38 --mixed (不加时，是默认携带)
39 意思是：不删除工作空间改动代码，撤销commit，并且撤销git add . 操作
40 这个为默认参数,git reset --mixed HEAD^ 和 git reset HEAD^ 效果是一样的。
41 --soft
42 不删除工作空间改动代码，撤销commit，不撤销git add .
43 --hard
44 删除工作空间改动代码，撤销commit，撤销git add .
45 注意完成这个操作后，就恢复到了上一次的commit状态。
46
47
48 4、git push把本地分支的提交已经推送到远程仓库的分支上
49
50 1) git reset --hard 【<commit_id>或HEAD^或HEAD~1】
```




```
51         git log # 得到你已经推送过的commit id
52         git reset --hard commit_id是回到其中你想要的某个
           版本，直接删除了指定的commit之后的提交版本或之后的修改，包括工作
           区修改的内容。
53     或者
54         git reset --hard HEAD^ # 回到最新版本(自己或别人
           刚推送的)的上个版本 //即只能回撤到最新的commit_id的那一次的上个
           版本(这里的commit应该是push同步后的commit_id),这个命令不是回撤
           到刚推送的那个版本的工作区内容，而是连带刚推送的那个版本的工作区内
           容也删除，直接回撤到上个版本
55
56         最后必须git push -f origin //假如远程主机是origin,
           并且分支已经进行追踪。强制将远程分支与本地分支同步，即覆盖远程分
           支。
57
58     2) git reset HEAD^
59         推送后再执行这个命令，是回撤到最新版本的上个版本的工作区
           (推送前执行，是回撤到本地commit版本所在的工作区修改)
60
61         回撤完毕并提交到版本库后，最后必须执行 git push origin
           HEAD --force # 强制提交一次，之前错误的提交就从远程仓库删除，保
           证了远程与本地分支版本同步
62
63     3) 通过git revert是用一次新的commit来回退之前的某个commit
64         git log //得到你需要回退一次提交的commit id
65         git revert <commit_id> //回退指定的版本，此次操作
           也会作为一次提交进行保存
66         git status //发现commit_id版本的修改已经是删除状态
67         git commit -m "回退commit_id这个版本"; //执行后，本
           地分支生成一个新版本，但是本地分支已经删除commit_id版本的修改，但
           是不影响此版本之后的版本。
```



```
68      git reset --hard HEAD^ //若突然不想回退某个版本了，
      需回到原来的状态，即回到新版本的上个版本
69      git push //若没问题，执行git push进行与远程同步即可
70  4) git revert 和 git reset的区别
71      - git revert是用一次新的commit来回退之前的某次
      commit，此次新提交之前的commit都会被保留； //安全的
72      - git reset是回到某次提交，提交及之前的commit都会被保
      留，但是此commit id之后的修改都会被删除。但是可以通过git reflog
      进行版本穿梭跳转，找回版本。
73      3~4的详细操作请参见8-1,8-2的讲解
74
75  但是新建的文件和文件夹无法清除，还必须使用：git clean -df清除所
      有新建的文件及文件夹
```

javascript >

删除的参考文章：<https://www.jianshu.com/p/c3ff8f0da85e>

10-1、rm删除文件(前提只是一个工作区文件)

直接 \$ rm xxx.后缀 即可

```
Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent (master-senders)
$ rm hs_err_pid18068.log

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent (master-senders)
$ rm hs_err_pid27140.log

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent (master-senders)
$ rm hs_err_pid9644.log

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent (master-senders)
$ git status
On branch master-senders
Your branch is up to date with 'origin/master-senders'.

nothing to commit, working tree clean

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent (master-senders)
$ |
```

10-2、git rm和rm删除文件(前提是一个添加到暂存区的工作区文件)

- git rm-1版本(`git rm --cached xxx.java` 强制删除暂存区修改, 并恢复工作区修改, `rm` 再删除工作区文件。)

2021/3/25 11:58:03

删除：删除已经提交到暂存区的文件

```
Admin@PS2019LVAKDGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git rm bbb.java
error: the following file has changes staged in the index:
    tms-dispatch/src/main/java/com/scmsafe/tms/dispatch/bbb.java
(use --cached to keep the file, or -f to force removal)

Admin@PS2019LVAKDGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git rm --cached bbb.java
rm 'tms-dispatch/src/main/java/com/scmsafe/tms/dispatch/bbb.java'

Admin@PS2019LVAKDGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    bbb.java

nothing added to commit but untracked files present (use "git add" to track)

Admin@PS2019LVAKDGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ rm bbb.java

Admin@PS2019LVAKDGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$
Admin@PS2019LVAKDGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

nothing to commit, working tree clean

Admin@PS2019LVAKDGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$
```

强制删除暂存区的修改，保留工作区修改

删除工作区文件

- git rm-2版本(`git rm --f xxx.java` 强制删除暂存区修改，并且同时恢复工作区修改,同时删除工作区的文件。)

```
Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git rm ccc.java
error: the following file has changes staged in the index:
    tms-dispatch/src/main/java/com/scmsafe/tms/dispatch/ccc.java
(use --cached to keep the file, or -f to force removal)

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git rm -f ccc.java
rm 'tms-dispatch/src/main/java/com/scmsafe/tms/dispatch/ccc.java'

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

nothing to commit, working tree clean

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$
```

- rm版本-1(这个是rm最终的流程版本)

```
Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (master-senders)
$ git checkout -- aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (master-senders)
$ git status
On branch master-senders
Your branch is up to date with 'origin/master-senders'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (master-senders)
$ git restore --staged aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (master-senders)
$ git status
On branch master-senders
Your branch is up to date with 'origin/master-senders'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    aaa.java

nothing added to commit but untracked files present (use "git add" to track)

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (master-senders)
$ rm aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (master-senders)
$ git status
On branch master-senders
Your branch is up to date with 'origin/master-senders'.

nothing to commit, working tree clean

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (master-senders)
$ |
```

- rm版本-2
 - rm删除了工作区文件，但是暂存区的修改还在保留着

- 执行git reset HEAD xxx.java , 删除暂存区的修改,并恢复到工作区的修改, 并删除了工作区的文件

```
Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ rm aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   aaa.java

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git reset HEAD aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

nothing to commit, working tree clean
```

- rm版本-3

- rm 删除工作区文件
- 使用git restore xxx.java 撤销刚刚工作区文件的删除操作, 工作区文件还原
- 使用git restore --staged xxx.java 撤销暂存区修改到工作区修改
- 删除工作区文件

```
Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ rm aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   aaa.java

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git restore aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git restore --staged aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ rm aaa.java

Admin@PS2019LVAKD0GP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

nothing to commit, working tree clean
```

- rm版本-4


```
Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ rm aaa.java

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   aaa.java

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    aaa.java

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git restore --staged aaa.java

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

nothing to commit, working tree clean

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
h (xiaoyaoyao-prod)
$ |
```

10-3、git rm和rm删除文件(前提是一个已经提交到本地版本库的工作区文件)

- 注意：如果还没提交的版本库，误删除了，则找不回来，这个只是还原到和最近一次提交的版本一致。
- 删除本地工作区文件后，别忘了提交覆盖

复制代码



```
1 方式1:
2  //rm先删除工作区文件，后续执行其它命令再进行添加，提交(覆盖删除版本库中的文件)
3 打开git bash窗口(git的运行环境)
4  rm BasePackage.java
5  git add
6  git commit -m "xxx"
7  git push
8
9  //若是误操作删除了工作区的文件，可以进行撤销刚刚删除工作区文件的操作，还原成和版本库一致。
10 //其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。
11 //小提示：如果一个文件已经被提交到版本库，永远不要担心误删，但是你能只能还原到版本库最新版本，但是会丢失最近一次提交版本后的修改。
12 git checkout -- . // (这个.是工作区文件所在目录)
13
14
15 =====
16
17
18 方式2:
19 //git rm删除工作区文件
20 //同rm BasePackage.java不同，此命令会多执行一步：先将工作区文件删除后，再将删除状态添加到暂存区，即自动执行了git add。即叫做：将删除状态提交到暂存区。
21 git rm BasePackage.java
22 git commit -m "xxx"
23 git push
24
25 //如何恢复工作区文件呢？ 两步
```

```
26 git reset head BasePackage.java 把暂存区的删除状态恢复到工作
   区的修改状态
27 git checkout -- test.txt 撤销工作区的修改，还原到和版本库一致
28 git status
```

javascript >

- rm版的删除及误删恢复工作区文件。图解
 - rm删除工作区文件后(则此文件在工作区是删除状态)，使用git add，git commit，git push覆盖才算真正删除。
 - git checkout -- .或git checkout 文件名.后缀名，是撤销工作区文件的删除状态。（还原）

1.

```
MINGW64:/d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/
tms/dispatch (xiaoyaoyao-prod)
$ rm BasePackage.java

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/
tms/dispatch (xiaoyaoyao-prod)
$ git checkout -- .

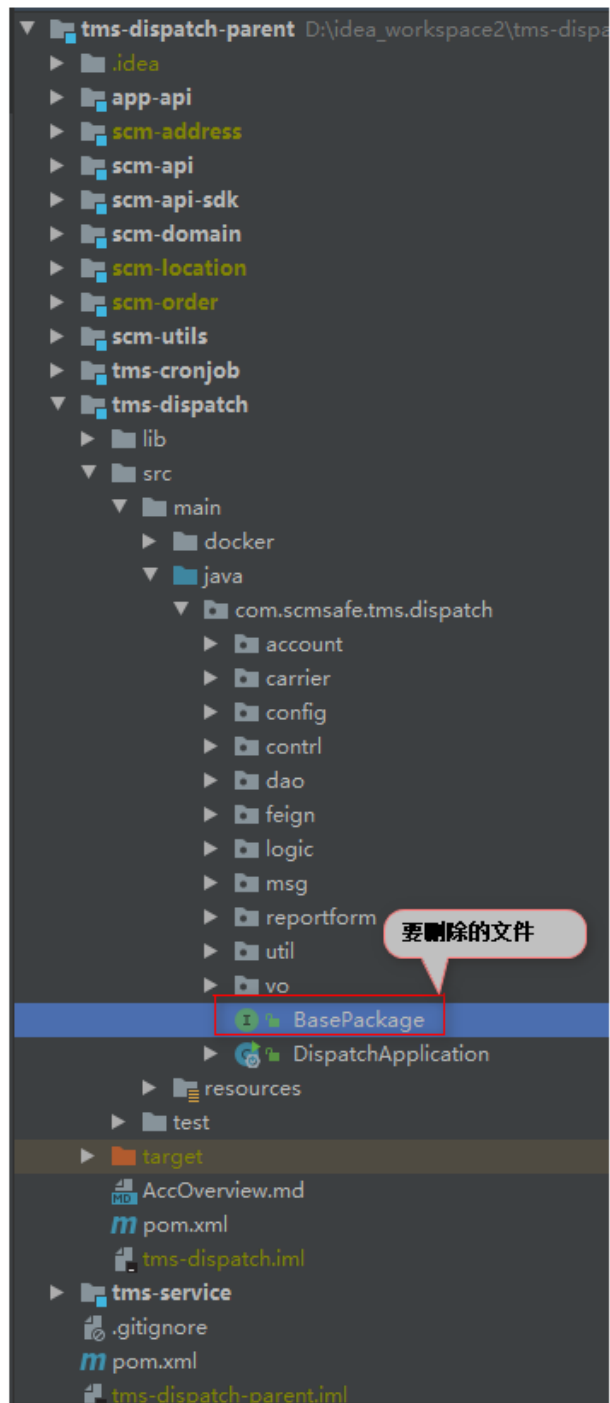
Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispat
ch (xiaoyaoyao-prod)
$ git checkout -- BasePackage.java

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/
tms/dispatch (xiaoyaoyao-prod)
$ |
```

2.

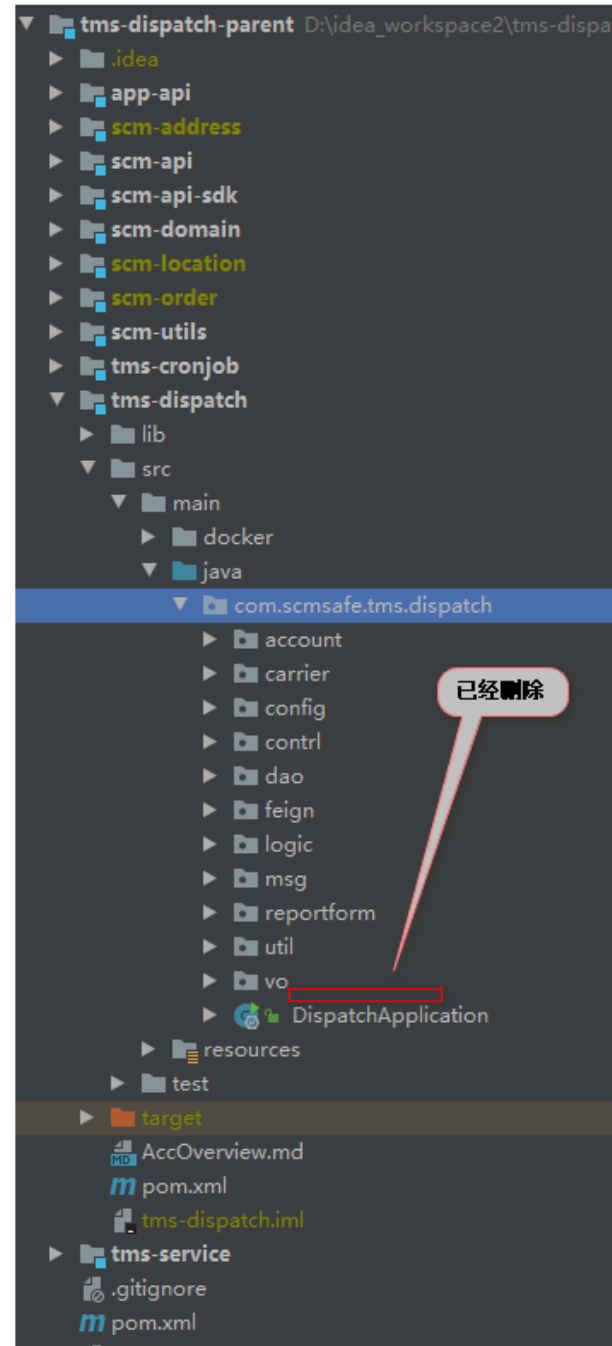
```
Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispat
ch (xiaoyaoyao-prod)
$ git checkout -- BasePackage.java
```

3.

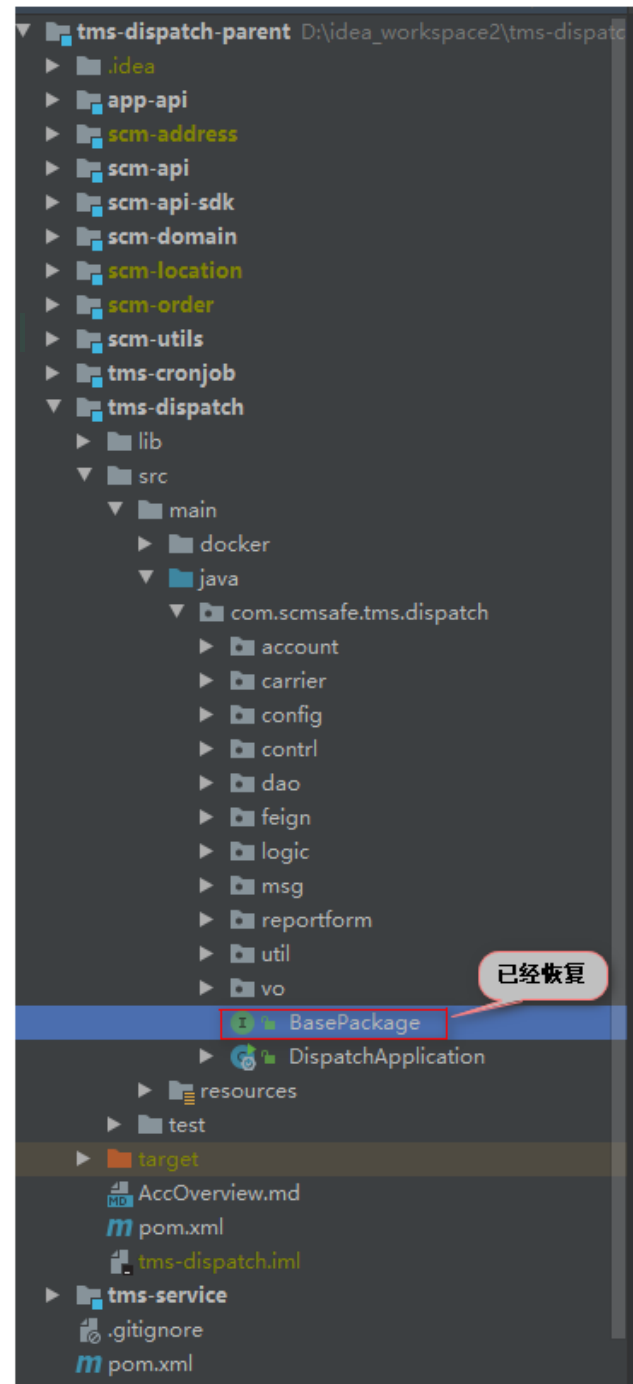


要删除的文件

4.



5.





- git rm版的删除工作区文件及恢复工作区文件。图解。
 - git rm删除工作区文件后，会将删除状态添加到暂存区，后面git commit,git push才算真正删除。
 - git reset HEAD 文件名.后缀名，撤销暂存区的删除状态到工作区的删除状态(因为工作区文件已经删除)。
 - git checkout --.或git checkout 文件名.后缀名，丢弃工作区的删除状态，还原到和版本库一致(工作区文件显示)。

1.

```
Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
ch (xiaoyaoyao-prod)
$ git rm BasePackage.java
rm 'tms-dispatch/src/main/java/com/scmsafe/tms/dispatch/BasePackage.java'

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
ch (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    BasePackage.java

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
ch (xiaoyaoyao-prod)
```

2.

```
Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
ch (xiaoyaoyao-prod)
$ git reset HEAD BasePackage.java
Unstaged changes after reset:
D    tms-dispatch/src/main/java/com/scmsafe/tms/dispatch/BasePackage.java

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
ch (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.
```

```

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    BasePackage.java

no changes added to commit (use "git add" and/or "git commit -a")

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
ch (xiaoyaoyao-prod)
$

```

3.

```

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
ch (xiaoyaoyao-prod)
$ git checkout -- BasePackage.java

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
ch (xiaoyaoyao-prod)
$ git status
On branch xiaoyaoyao-prod
Your branch is up to date with 'origin/xiaoyaoyao-prod'.

nothing to commit, working tree clean

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace2/tms-dispatch-parent/tms-dispatch/src/main/java/com/scmsafe/tms/dispatch
ch (xiaoyaoyao-prod)
$

```

10-4、git rm删除文件(前提是一个已经推送到远程版本库的工作区文件)

删除操作(本地库删除并且推送到远程库):

```

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ git rm aa.txt
rm 'aa.txt'

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ git status
On branch dev22
Your branch is up to date with 'origin/dev'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    aa.txt

Untracked files:

```



```
(use "git add <file>..." to include in what will be committed)
.idea/

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ git commit -m "删除aa.txt文件"
[dev22 27f9d3f] 删除aa.txt文件
1 file changed, 7 deletions(-)
delete mode 100644 aa.txt

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ git branch -vv
* dev22 27f9d3f [origin/dev: ahead 1] 删除aa.txt文件
dev33 8e58fe5 [origin/dev33] cc.txt
dev4 b5f93a3 [origin/dev4] bb.txt
master 5b07600 [origin/master] 第五次提交

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ git status
On branch dev22
Your branch is ahead of 'origin/dev' by 1 commit.
(use "git push" to publish your local commits)

Untracked files:
(use "git add <file>..." to include in what will be committed)
.idea/

nothing added to commit but untracked files present (use "git add" to track)

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ git push
fatal: The upstream branch of your current branch does not match
the name of your current branch. To push to the upstream branch
on the remote, use

    git push origin HEAD:dev

To push to the branch of the same name on the remote, use

    git push origin HEAD

To choose either option permanently, see push.default in 'git help config'.

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ git push -u origin dev22:dev
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 6 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 241 bytes | 241.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:1534834526/remote-git1.git
40ca4d3..27f9d3f dev22 -> dev
```

```
40ca4d3.27f9d3f dev22 => dev
Branch 'dev22' set up to track remote branch 'dev' from 'origin'.

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ |
```

恢复工作区文件操作(本地库回滚到上个版本，然后把回滚后的版本提交)

```
Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ git log --pretty=oneline
27f9d3f5e5be811e9151272b685ca010dc831f33 (HEAD -> dev22, origin/dev) 删除aa.txt文件
40ca4d3dbf8446e6cde42d59521b47bcd56363f5 aa.txt提交2
d8c28a391ed8c5f7bc10ed4ec5007273858867b0 read.txt提交
87648d84786a97fa135f446773ca0fbee054da55 ee.txt提交
229e849a32669134b396da4c0a99bab1e429b4eb dd.txt第一次提交
7924c9d92a1c9ce3f454f2c05e54f53e0d1c1dc7 ee.txt第一次
5b07600cc57ba2ace404525a6ba33dbc67557e11 (origin/master, origin/HEAD, master) 第五次提交
6d1572109fe6032e1e549779da2fd0edbf807bb9 git第4次提
b01db6f64d906e5f82f436af9ea4a03afc2a3e38 第三次提交aa.txt
5f330b3a06d94d55d4d96df88e1c5da405a8ef71 aa.txt修改
2baff0996f5a3762e4f18d4d799bdc85776b0170 Merge branch 'dev'
c7003e96c94a6e24a33d1b22fa2e81a28933ae35 ee.txt
8b2f93ee5f8c00a0b6297d0856fcc2306ef803d2 dd.txt
e1d28193661fe697f8f1df7ecc646be4649fca1c Merge branch 'dev'
38be65cbe368462299d0cc0c58292134239257c4 bb.txt
9ee0eb892ba81a2e3d634153356459489f8d9cad Update cc.txt
4370750c8bde0b7198f3c511a110c6042c04c0ae aa.txt he
bd50281470574e087e4c0754694c9544332d6c56 Create cc.txt
afafcba2cf9154a9cca1e9338d71528e1ab02f86 aaV1.0
b1761fef4c3c8580bfa83a7c4c51211c58df329 V6.0
75dba820a22870fc3018bc6e27f742c61675cceb V4.0
7fbbd3da6211fb2d803b43a403fd51828ab478fc V4.0
80f274fc2bb9551988894e9eef86203f7bc5a68c V3.0
0b99619daae6f4420758b46f7ef39b19669ea1f4 V2.0
2c4bf3c33106258d2535b0a6e236ccd95d00f3ee write a readme file

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ git reset --hard 40ca4d3 本地回退到上一个版本
HEAD is now at 40ca4d3 aa.txt提交2

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ git branch -vv
* dev22 40ca4d3 [origin/dev: behind 1] aa.txt提交2
dev33 8e58fe5 [origin/dev33] cc.txt
dev4 b5f93a3 [origin/dev4] bb.txt
master 5b07600 [origin/master] 第五次提交

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$ git status
On branch dev22
本地回退版本后，本地库会落后远程库一个版本
```

on branch dev22
Your branch is behind 'origin/dev' by 1 commit, and can be fast-forwarded.
(use "git pull" to update your local branch)

Untracked files:
(use "git add <file>..." to include in what will be committed)
.idea/

nothing added to commit but untracked files present (use "git add" to track)

目的是：使用本地库的低版本覆盖远程库的多一个的那个版本。但是git拒绝，除非强制推送

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)

\$ git push -u origin dev22:dev

To github.com:1534834526/remote-git1.git

! [rejected] dev22 -> dev (non-fast-forward)

error: failed to push some refs to 'github.com:1534834526/remote-git1.git'

hint: Updates were rejected because a pushed branch tip is behind its remote

hint: counterpart. Check out this branch and integrate the remote changes

hint: (e.g. 'git pull ...') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)

\$ git push -h 查看帮助

usage: git push [<options>] [<repository> [<refspec>...]]

-v, --verbose	be more verbose
-q, --quiet	be more quiet
--repo <repository>	repository
--all	push all refs
--mirror	mirror all refs
-d, --delete	delete refs
--tags	push tags (can't be used with --all or --mirror)
-n, --dry-run	dry run
--porcelain	machine-readable output
-f, --force	force updates
--force-with-lease[=<refname>:<expect>]	require old value of ref to be at this value
--recurse-submodules (check on-demand no)	control recursive pushing of submodules
--thin	use thin pack
--receive-pack <receive-pack>	receive pack program
--exec <receive-pack>	receive pack program
-u, --set-upstream	set upstream for git pull/status
--progress	force progress reporting
--prune	prune locally removed refs
--no-verify	bypass pre-push hook
--follow-tags	push missing but relevant tags
--signed[=(yes no if-asked)]	GPG sign the push
--atomic	request atomic transaction on remote side
-o, --push-option <server-specific>	option to transmit
-4, --ipv4	use IPv4 addresses only
-6, --ipv6	use IPv6 addresses only

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)

\$ git push -f -u origin dev22:dev 强制推送覆盖

```
total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:1534834526/remote-git1.git
+ 27f9d3f...40ca4d3 dev22 -> dev (forced update)
Branch 'dev22' set up to track remote branch 'dev' from 'origin'.

Admin@PS2019LVAKDOGP MINGW64 /d/idea_workspace/remote-git1 (dev22)
$
```

11-1、远程分支有，本地分支还没有

<https://www.cnblogs.com/ruiyang-/p/10764711.html>

复制代码

```
1 //切换分支
2 git checkout 分支名
3 //若上述未切换成功(没有此分支)，则需要同步下远程仓库中的分支到本地
4
5 //git fetch origin只是手动指定了要fetch的远程仓库,拉取远程仓库
  的新分支。
6 git fetch origin
7
8 //根据已有的远程分支master，在本地创建指定名字的新分支
9 git fetch origin 远程分支名 : 新的本地分支名
10 git fetch origin master:master1==git fetch origin
   :master1 //省略远程分支名
11
12
13
14 //设定当前分支为远程服务器的shenghai-0930
15 //在这种情况下，不会在本地创建分支
16 //这个命令可以用来测试远程主机的远程分支shenghai-0930是否存在，
   如果存在，返回0，如果不存在，返回128，抛出一个异常
17 git fetch origin shenghai-0930
18
19
```

```
20 //使用远程shenghai-0930分支在本地创建shenghai-0930(但不会切换到该分支),
21 //如果本地不存在shenghai-0930分支, 则会自动创建一个新的shenghai-0930分支,
22 //如果本地存在shenghai-0930分支, 并且是`fast forward`, 则自动合并两个分支, 否则, 会阻止以上操作.
23 git fetch origin shenghai-0930:shenghai-0930
24
25
26
```

javascript >

11-2、git fetch远程分支有, 还能看到(其实已经删除)

「//当你的远程分支已删除时, 本地git branch-a 查看时, 发现那些删除的分支还在, 想删除, 则执行如下命令更新远程分支列表

```
git fetch origin --prune
or
git remote prune origin
or
git remote update origin --prune
```

」

12、git pull拉取远程仓库中此分支内容到本地分支(更新)

```
1 git pull <远程主机名> <远程分支名>:<本地分支名>
2
3 //如果当前分支与远程分支存在追踪关系，git pull就可以省略远程分支
  名。
4 git pull origin
5
6 //如果当前分支只有一个追踪分支，连远程主机名都可以省略。
7 git pull
8
9 //如果合并需要采用rebase模式（替换掉merge方式），可以使用--
  rebase选项。可参考
  https://blog.csdn.net/yao_hou/article/details/108178717
  查看rebase和merge区别rebase没有新的节点，merge出现新的节点
10 git pull --rebase <远程主机名> <远程分支名>:<本地分支名>
11
12
13 //如果远程主机删除了某个分支，默认情况下，git pull 不会在拉取远
  程分支的时候，删除对应的本地分支。这是为了防止，由于其他人操作了远
  程主机，导致git pull不知不觉删除了本地分支。但是，你可以改变这个
  行为，加上参数 -p 就会在本地自动删除远程已经删除的分支。
14 git pull -p
15 或
16 git branch -d 本地分支名 //手动删除指定的本地分支
```

复制代码



javascript >

13、Git中参数的解释

```
1 -d
2 --delete: 删除
```

复制代码

```
3
4 -D
5 --delete --force的快捷键
6
7 -f
8 --force: 强制
9
10 -m
11 --move: 移动或重命名
12
13 -M
14 --move --force的快捷键
15
16 -r
17 --remote: 远程
18
19 -a
20 --all: 所有
21
22 -u
23 --upstream: 本地分支与远程分支的流通道
24
```

javascript >

14、git fetch命令

```
1 //1、将某个远程主机的更新全部取回本地
2 $ git fetch <远程主机名>
3
4
```

复制代码

```
5 //2、如果只想取回特定分支的更新，可以指定分支名
6 $ git fetch <远程主机名> <分支名>
7
8 //3、示例
9 最常见的命令如取回origin主机的master 分支：
10 $ git fetch origin master
11 取回更新后，会返回一个FETCH_HEAD，指的是某个branch在服务器上的
    最新状态，我们可以在本地通过它查看刚取回的更新信息：
12 $ git log -p FETCH_HEAD
13
14 //4、强制将默认当前远程主机的更新，全部取回本地
15 git fetch --prune
16
```

javascript >

15、git pull命令

复制代码

```
1 //1、将远程主机的某个分支的更新取回到本地，并与本地指定的分支合
    并，完整格式可表示为：
2 $ git pull <远程主机名> <远程分支名>:<本地分支名>
3
4 //2、如果远程分支是与当前本地分支合并，则冒号后面的部分可以省略
5 $ git pull origin master
6
7
8 //3、从远程主机origin的master分支强制拉取最新内容
9 git fetch origin master
10
11 //将拉取下来的最新内容(FETCH_HEAD：假如最新内容是dev的内容)合并
    到当前所在的分支中(假如当前在master分支)
```




16、git remote

- <https://zhuanlan.zhihu.com/p/115462947>

复制代码

```
1 //查看和当前本地仓库关联的所有远程仓库的简写和地址，如果该本地库是
  通过克隆下来的，其远程仓库的默认名为origin，如果手动关联的远程仓库，
  则远程仓库的名字是可以自定义的。如果还没有关联远程仓库，则没有
  推送权限，看不到远程仓库地址。
2 git remote
3
4 //查看所有远程仓库的名称简写、抓取(fetch)、推送(push)的URL地址
5 git remote -v
6
7 //查看指定名remoteName的远程仓库信息
8 git remote show [remoteName]
9
10 //查看名为origin的远程仓库信息
11 git remote show origin
12
13 //移除指定名remoteName的远程仓库
14 git remote rm [remoteName]
15
16 //移除名为origin的远程仓库
17 git remote rm origin
18
19 //将名为 oldRemoteName 的远程仓库重命名为 newRemoteName
20 git remote rename [oldRemoteName] [newRemoteName]
21 //将名称为 origin 的远程仓库重命名为 demo
```

```
22 git remote rename origin demo
23
24
25 //向本地代码库关联指定地址的远程仓库，同时指定远程仓库的简称
26 git remote add [remoteName] [URL]
27 //向本地代码库关联指定地址的远程仓库，其仓库命名为 origin
28 git remote add origin git@github.com:Guanghua-
  Zhu/GitDemo.git
29
30 //添加远程仓库后，可直接通过下述命令显式地将本地分支和远程分支关
  联，这样日后的推送/拉取操作时可以省略远程分支名称
31 git branch --set-upstream-to=
  [remoteName]/[remoteBranchName] [localBranchName]
32
33 //亦可以在第一次推送分支时，通过添加 -u 参数以将本地当前分支和远
  程仓库的指定分支关联起来，在日后的推送/拉取操作时可以省略远程分支
  名称
34 git push -u [remoteName] [remoteBranchName]
35
36
37 //使用下述命令，将远程仓库克隆到本地，其远程仓库简称，默认为
  origin，通过克隆方式进行关联时，会自动将本地仓库master分支和远程
  仓库的master分支关联起来
38 git clone [URL]
39 git clone git@github.com:Guanghua-Zhu/GitDemo.git
40
41 //可以使用下述两种方式从远程仓库拉取别人的更新和提交，分支名可省
  略。其中，fetch命令将更新拉取到本地后，需要手动合并；而pull则会尝
  试进行自动合并
42 git fetch [remoteName] [branchName] //从指定名remoteName
  的远程仓库的branchName分支拉取更新
```



```
43 git pull [remoteName] [branchName] //从指定名remoteName
    的远程仓库的branchName分支拉取更新
44
45 //当拉取没有共同commit的远程仓库时，会出现 fatal: refusing to
    merge unrelated histories 错误，致使拉取失败。该情况一般会在
    用git remote add命令添加远程仓库的情况下出现，只需添加 --
    allow-unrelated-histories 参数即可
46 git pull --rebase origin master --allow-unrelated-
    histories
47
48 //使用push命令向远程仓库推送你本地的更新和提交。如前所述，在使用
    git remote add 建立远程仓库关联时，第一次推送时，在push 和 远
    程仓库名之间添加 -u 参数，将本地当前分支 和 远程仓库分支关联起来
49 git push [remoteName] [branchName] //向指定名remoteName
    的远程仓库的branchName分支推送更新
```

java >

```
D:\idea_workspace2\tms-dispatch-parent>git remote
origin

D:\idea_workspace2\tms-dispatch-parent>git remote -v
origin http://gitlab.scmsafe.com/scmsafe-pro/tms-dispatch.git (fetch)
origin http://gitlab.scmsafe.com/scmsafe-pro/tms-dispatch.git (push)
```

17、untracked files未跟踪的文件

- 情况1: git没有把需要提交的文件加载进来，但是把需要提交的文件都列出来了，只需要用git add XXX(文件名) 把需要提交的文件添加到暂存区，然后git commit -m "xx", 再 git push -u origin master重新提交就可以了。

- 情况2：目测这些文件是你的程序在运行过程中生成的，可以直接把它们删除掉，最好的方案是把生成的临时内容放在某些特定的目录内，然后把这个目录添加到.gitignore文件中

```
D:\idea_workspace2\tms-dispatch-parent\tms-dispatch>git status
On branch master-senders
Your branch is up to date with 'origin/master-senders'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  ../hs_err_pid18068.log
  ../hs_err_pid9644.log

nothing added to commit but untracked files present (use "git add" to track)

D:\idea_workspace2\tms-dispatch-parent\tms-dispatch>
```

未跟踪的文件：工作区的文件，还没有添加到暂存区。

18、查看Git官方帮助文档

git branch --help

19、找回git add过但是已经不存在的修改

https://blog.csdn.net/qq_31608451/article/details/78342365

<https://segmentfault.com/q/1010000016122016>

第一步：

git fsck --lost-found //可以在本地项目文件中路径为.git/lost-found/other中找到它们。
这里面包含了所有的没有被commit的文件，甚至可能还包括你每次git add的版本



第二步：

```
find .git/objects -type f | xargs ls -lt | sed 60q //查看最近被你add到本地仓库暂存区的60个文件
```

注：

- 没有git add过的文件修改是无法找回的，所以至少git add过的文件修改才能找回。
- 没有 commit 无法通过 git reflog 找回，git reflog 是以 commit 为单位的

20、git cherry-pick操作：将分支dev的某次或多次提交(版本号feff5bf)合并到master分支

<https://blog.csdn.net/u010126792/article/details/84563420>

git bash命令行的操作

语法：

git cherry-pick 选项 commit_id

cherry-pick后面的选项

--quit,--continue,--abort是git cherry-pick发生冲突后需要做的操作

--quit 退出当前的chery-pick序列，中断这次cherry-pick

--continue 继续当前的chery-pick序列，继续cherry-pick。首先需要解决冲突，通过git add .将文件标记为已解决,再继续执行git cherry-pick --continue。

--abort 取消当前的chery-pick序列，恢复当前分支，当前分支恢复到cherry-pick前的状态，没有改变

-n, --no-commit 不自动提交，默认是合并后自动提交

-e, --edit 编辑提交信息

cherry-pick会自动提交（如果没有冲突），-n可以取消自动提交，之后需要自己add, commit

示例：

```
git checkout master
```



`git cherry-pick feff5bf` //feff5bf是dev分支的版本，而且dev本身是来源于master分支

注意：

理想情况是，不出现冲突，直接合并成功，这时候你直接push master分支到远程就ok。

不理想情况是，出现冲突，这时候，你需要手动解决冲突(参见13冲突解决方式)，然后在commit，然后再push。

有个重要的限制每次只能合并一个commit，如果有多个稍麻烦，需一个版本一个版本的合并，注意解决冲突。

`git cherry-pick`其实也可以将dev部分提交合并到master

`git cherry-pick <commit-id1> <commit-id2> <commit-id3>`

中间用空格隔开，commit-id1，commit-id2代表是dev分支的部分提交，一起合并到master主分支。

`git cherry-pick`时的非冲突的问题

nothing to commit,working directory clean

The previouscherry-pick is now empty, possibly due to conflict resolution.

If you wish to commitit anyway, use:

`git commit --allow-empty`

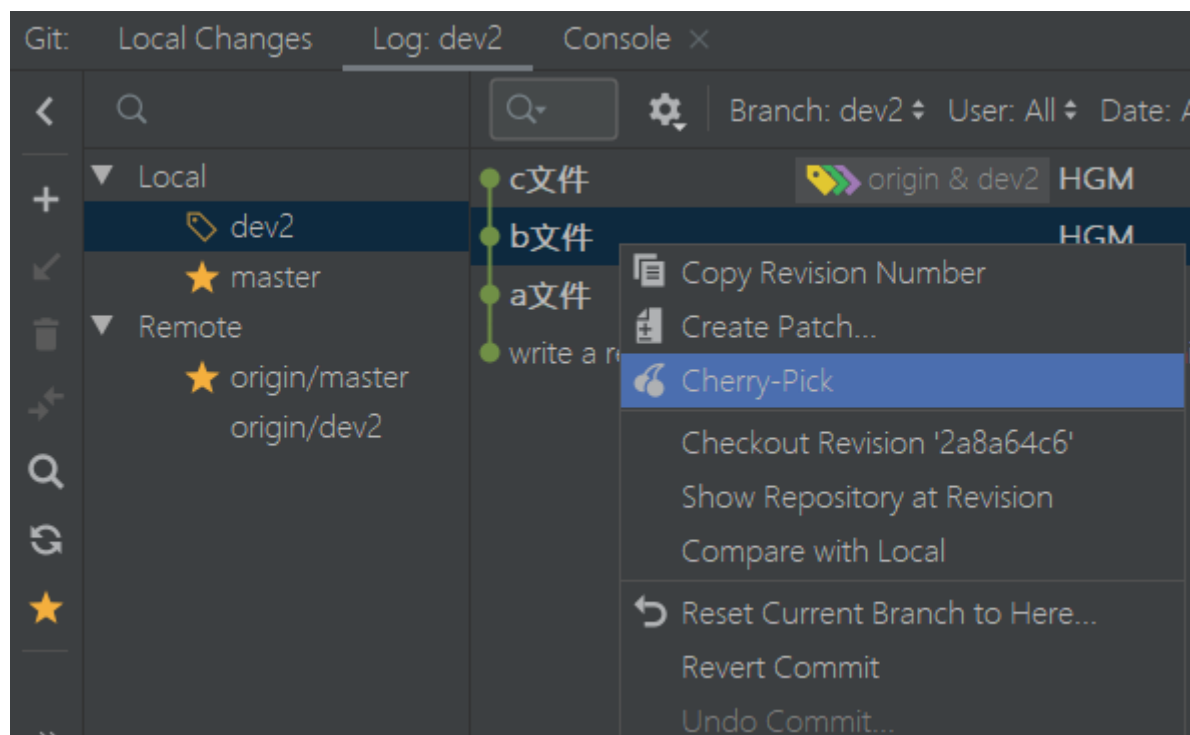
Otherwise, please use'git reset'

意思是：没有冲突输出，只是提示如果当前分支要提交，需要做一次空提交，因为这次cherry-pick操作的版本可能已经在当前分支上提交过了。

https://blog.csdn.net/qq_41345773/article/details/105717259?utm_term=git%E5%90%88%E5%B9%B6%E9%83%A8%E5%88%86%E6%8F%90%E4%B

[A%A4&utm_medium=distribute.pc_aggpage_search_result.none-task-blog-2~all~sobaiduweb~default-1-105717259&spm=3001.4430](#)

IDEA中的Terminal命令行的cherry-pick操作：



21、git push -f本地分支回撤版本后，强制推送覆盖到远程分支（直接推会报错）

回撤后，本地分支会落后远程分支几个版本（不同步），想推送到远程，只能强制推送

- git push -f origin 远程分支名
- -f : force强制

```
D:\idea_workspace3\tms-dispatch>git status
On branch guoke-prod-import
Your branch is behind 'origin/guoke-prod-import' by 2 commits, and can be fast-forwarded.
(use "git pull" to update your local branch)

nothing to commit, working tree clean

D:\idea_workspace3\tms-dispatch>git push
To http://gitlab.scmsafe.com/scmsafe-pro/tms-dispatch.git
! [rejected] guoke-prod-import -> guoke-prod-import (non-fast-forward)
error: failed to push some refs to 'http://gitlab.scmsafe.com/scmsafe-pro/tms-dispatch.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

D:\idea_workspace3\tms-dispatch>git push -f origin guoke-prod-import
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for guoke-prod-import, visit:
remote: http://gitlab.scmsafe.com/scmsafe-pro/tms-dispatch/merge\_requests/new?merge\_request%5B
remote:
To http://gitlab.scmsafe.com/scmsafe-pro/tms-dispatch.git
+ c06e30ce9...86f5080d1 guoke-prod-import -> guoke-prod-import (forced update)

D:\idea_workspace3\tms-dispatch>
```

22、git rebase操作(交互式变基，重新设置基线，为你的当前分支重新设置开始点)



rebase简单整理.pdf

621 KB

git rebase ≈ git merge

<https://blog.csdn.net/nrsc272420199/article/details/85555911>

<https://studygolang.com/articles/31027>

<https://www.cnblogs.com/jmcui/p/9723272.html>

<https://zhuanlan.zhihu.com/p/145037478>

<https://git-scm.com/book/zh/v2/Git-%E5%88%86%E6%94%AF-%E5%8F%98%E5%9F%BA>

https://blog.csdn.net/not_say/article/details/82180129 edit和fixup使用

目的：通过交互式变基(vim命令)，修改r, e, 合并s, f, 删除d历史提交

- 合并本地分支的多个提交(可以控制哪些提交需要合并)为一个提交。
- 删除或修改本地分支的某次或几次提交。
- 合并master分支(这里是线上分支)中的提交到自己(这里是dev分支)：如hotfix分支的提交合并到master后，dev分支少于master分支版本，目的就是將master分支的版本合并到dev分支中，这样dev相当于是从master最新版本进行开发，git结构树处不是显示merge，而是显示多了一些master分支新的提交版本
- rebase 是一个危险的命令，因为它改变了历史。如果你和同事共用一个开发分支，如果你本地变基了3个提交为1个提交，在你的同事开发分支中存在这3个提交，只是内容相同，commit_id不同，如果你把变基提交强行推送到远程仓库后，你的同事在本地执行git pull 的时候会导致变基版本和他自己的3个版本发生融合，且都出现在了历史提交中，会导致你的变基行为无效。如果你自己开发使用别人不拉取开发，则没关系。
- 如果有同事共同使用一个分支开发，告诉你的同事使用git pull --rebase 而不是git pull 来拉取远程分支。
- 操作的提交范围：
 - git rebase -i commit_id 将以commit_id版本为基准线，对其之后的某些提交进行操作
 - git rebase -i HEAD~3 可操作最近的3个连续版本





- 如果在rebase的过程中遇到了冲突，需要手工解决，然后使用git rebase --continue完成rebase 操作。
- git rebase -i后的通过vim交互式命令中的参数使用
 - pick：保留该commit（缩写:p）
 - reword：保留该commit，但我需要修改该commit的注释（缩写:r）
 - edit：保留该commit, 但我要停下来修改该提交(不仅仅修改注释)（缩写:e）
 - squash：只改动某个提交的话，默认将该commit和前一个commit合并，如果合并两个以上，则每个提交前都使用squash（缩写:s）
 - fixup：将该commit和前一个commit合并，但我不要保留该提交的注释信息（缩写:f），该提交的修改相当于该提交的修改合并到前一个commit中了
 - exec：执行shell命令（缩写:x）
 - drop：我要丢弃该commit（缩写:d）
 - label：用名称标记当前HEAD(缩写:l)
 - reset：将HEAD重置为标签(缩写:t)
 - merge：创建一个合并分支并使用原版分支的commit的注释(缩写:m)
- 如果vim中途或最后突然不想变基，退出编辑模式后，命令行执行git rebase --abort撤销rebase操作。
- 如果已经git rebase --continue之后突然想撤回rebase操作，则不能执行git rebase --abort撤回，而是先使用git reflog找到rebase之前的commit_id，然后git reset --hard commit_id。
- 如果确认rebase操作后，出现冲突或异常，可以接着执行git rebase --edit-todo再次进入编辑模式，进行修改操作。
- 最后再执行 git rebase --continue确认完成rebase操作。
- git push -f 远程覆盖更新
- **注意：不要对线上分支的提交记录进行变基！除非拉取时使用git pull --rebase。但是一般不推荐线上分支变基。**

(1) 合并示例步骤图：

1.

```
D:\idea_workspace\remote-git1>git branch
dev
dev4
* master

D:\idea_workspace\remote-git1>
```

2.

3.

```
D:\idea_workspace\remote-git1>t using the original merge commit's
D:\idea_workspace\remote-git1>ine, if no original merge commit was
pick 52fbfb7 C~YC~XC.txtC~V~G件 ← 这是e.txt文件
pick 29d36dd C~YC~XC.txtC~V~G件 ← 这是d.txt文件
pick e30f29e C~YC~XC.txtC~V~G件 ← 这是c.txt文件

# Rebase 4e63858..e30f29e onto 4e63858 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .      create a merge commit using the original merge commit's
# .      message (or the oneline, if no original merge commit was
# .      specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
D:/idea_workspace/remote-git1/.git/rebase-merge/git-rebase-todo [unix] (13:47 19/04/2021)
```

4.

```
D:\idea_workspace\remote-git1>git rebase -i 4e638586
```

5.

```
Terminal: Local x +
pick 52fbfb7 C:\YC\X\Q.txt~V~G件
squash 29d36dd C:\YC\X\Q.txt~V~G件
pick e30f29e C:\YC\X\Q.txt~V~G件

# Rebase 4e63858..e30f29e onto 4e63858 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# X, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# . create a merge commit using the original merge commit's
# . message (or the oneline, if no original merge commit was
# . specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
D:/idea_workspace/remote-git1/.git/rebase-merge/git-rebase-todo[+] [unix] (14:03 19/04/2021)
:wq
```

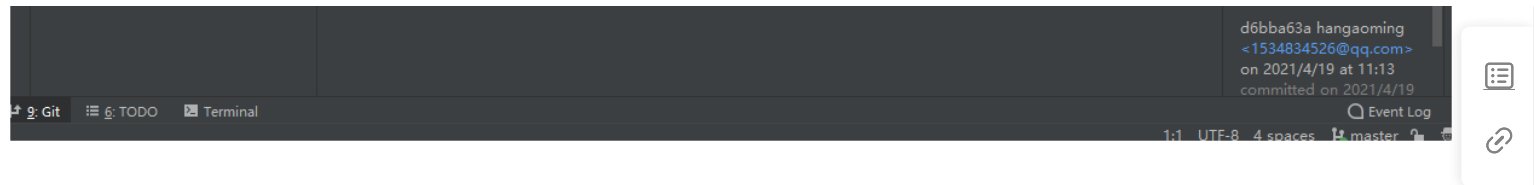
6.

Log: master

Branch	Commit	Author	Date	Hash	Files
Local	master	hangaoming	4 minutes ago	79a64d18	remote-git1 2 files
Local	dev	hangaoming	5 minutes ago	d6bba63a	c.txt, d.txt
Local	dev4	HGM	Yesterday 11:52	4e638586	
Remote	origin/master	HGM	Yesterday 11:50	e7f3e80	
Remote	read.txt第二次提交	HGM	Yesterday 11:48	a4c6e188	
Remote	write a readme file	hangaoming	2018/12/14 19:34	2c4ef3c3	

c.txt版本和d.txt版本进行合并，生成一个新的版本，内容和合并前是一样的

fix:c.txt d.txt



(2) 修改提交的注释示例

- `git rebase -i HEAD~n` (可操作最近的n个连续版本)
- 进入界面后, 摁a或者i进入编辑模式, 将需要修改注释的提交的那一行的 " pick " 改为 " edit ", 然后摁ESC退出编辑模式, 再输入:wq!来保存退出
- 退出编辑模式回到命令行后, 此时再执行`git commit --amend`, 进入界面后, 摁a或者i进入编辑模式, 将第一行中的注释(注意不是下面的带有edit或者pick字眼的注释)修改为正确的注释, 然后摁ESC退出编辑模式, 再输入:wq!来保存退出
- 退出编辑模式回到命令行后, 此时再执行`git rebase --continue`确认rebase操作, 完成本地分支的修改注释操作
- `git push -f` 强制推送到远程, 覆盖更新

(3) 删除提交的示例

- `git rebase -i HEAD~n` (可操作最近的n个连续版本)
- 进入界面后, 摁a或者i进入编辑模式, 将需要删除的提交的那一行的 " pick " 改为 " drop ", 然后摁ESC退出编辑模式, 再输入:wq!来保存退出编辑模式到命令行。
- 如果不想回撤rebase版本, `git reflog` 找到rebase之前的commit_id, `git reset --hard commit_id`
- `git push -f` 强制推送本地到远程, 覆盖更新

23、git patch创建, 应用补丁

自己总结见26

其它文章：

<https://jerryjin.blog.csdn.net/article/details/110794648>

[git apply、git am打补丁、diff 和 patch - 简书\(jianshu.com\)](#)



24、git tag打标签

<https://blog.csdn.net/lovesummerforever/article/details/51768361>

<https://git-scm.com/book/zh/v2/Git-%E5%9F%BA%E7%A1%80-%E6%89%93%E6%A0%87%E7%AD%BE>

- 作用：Git 可以给仓库历史中的**某一个提交打上标签**以示重要，比较有代表性的是人们会使用这个功能来标记发布结点（v1.0、v2.0 等等），如果以后想回来使用此发布结点，可以根据此标签名检出到一个新分支中，最新版本就是标签名对应的版本，可以将此分支（此版本号及之前的版本都会存在）重新打包即可，下面学习列出已有的标签、如何创建和删除新的标签、检出标签以及不同类型的标签分别是什么。
- 注意：打标签的操作发生在我们commit修改到本地仓库之后。
- 标签分类介绍：
 - Git 支持两种标签：轻量标签（lightweight）与附注标签（annotated）
 - 轻量标签：它只是某个特定提交的引用，即此标签只包含特定提交；
 - 附注标签：存储在 Git 数据库中的一个完整对象，即此标签包含当前提交及之前的都会保留，它们是可以被校验的，其中包含打标签者的名字及电子邮件地址、打标签的日期时间，附注信息（打标签时的备注信息），标签信息（包括某个提交版本号，提交的作者，提交的日期，提交的内容）。
- 列出已有标签（可带上可选的 -l 选项 --list）
 - 查看本地所有标签
 - git tag [-l或--list]
 - 查看本地特定标签
 - git show tag



- `git show v2.0`
 - 查看远程所有标签
 - `git ls-remote --tags origin`
- 创建标签
 - 轻量标签（不需要使用 `-a`、`-s` 或 `-m` 选项，只需要提供标签名字）
 - 创建轻量标签v1.0：`git tag v1.0` //不写commit_id默认只针对当前提交打标签进行记录。
 - 查看此轻量标签：`git show v1.0` //命令只会显示出提交信息，不会看到额外的打标签的信息。
 - 附注标签（指定 `-a`，`-m`选项可以增加标签的附注信息）
 - 创建附注标签v1.0：`git tag -a v1.0 -m "附注信息"` //不写commit_id默认针对当前提交打标签进行记录，并且可以添加标签的备注信息。
 - 创建特定版本附注标签v1.0：`git tag -a v1.0 commit_id -m "附注信息"` //针对某个提交commit_id打标签进行记录，一般用于后期某个提交忘记打标签，对过去的提交进行补标签。并且可以添加标签的备注信息。
- 推送标签：
 - 默认情况下，`git push` 命令并不会传送标签到远程仓库服务器上。在创建完标签后你必须显式地推送标签到共享服务器上。这个过程就像共享远程分支一样——你可以运行 `git push origin <tagname>`
 - `git push origin v1.0`
 - 如果想要一次性推送很多标签，也可以使用带有 `--tags` 选项的 `git push` 命令。这将会把所有不在远程仓库服务器上的标签全部传送到那里。使用 `git push <remote> --tags` 推送标签并不会区分轻量标签和附注标签，没有简单的选项能够让你只选择推送一种标签。
 - `git push origin --tags`
 - 查看此附注标签
 - `git show v1.0`



- 删除标签
 - 要删除掉你本地仓库上的标签
 - `git tag -d <tagname>`
 - `git tag -d v1.0`
 - 基于上述操作后，同步删除远程仓库上的标签。使用本地空标签覆盖远程标签
 - `git push <remote> :refs/tags/<tagname>`
 - `git push origin :refs/tags/v1.0` 或 `git push origin --delete v1.0`
- 切换标签
 - 切换到指定的标签
 - `git checkout [tag/branch/commit]` // 切换到指定tag/branch/commit都是此命令
- 检出标签
 - 先切换到某个标签
 - `git checkout [tag/branch/commit]`
 - 根据当前标签，创建并切换到新分支，将标签对应的版本检出到一个新分支中，此标签对应版本及之前版本都在
 - `git checkout -b <branchName> <tagName>`
 - `git checkout -b feature1 v1.0` //例：创建并切换到本地分支feature1，并且本地分支的最新版本是标签v1.0的那个提交的版本，此提交及以前的版本都会保留。
 - 因为标签本身指向的就是一个 commit，所以和根据commit_id切换分支是一个道理。

25、git commit --amend修改提交的注释

- 还未推送



- `git commit --amend -m "更正的注释" //使用一次新的commit，替代上一次提交，如果代码没有任何新变化，则用来改写上一次commit的提交信息。通俗说法：此次提交，除了新修改的注释，还包括文件新的修改，若文件没有变动，只包括注释的修改。`
- 已推送
 - `git rebase -i HEAD~n`（可操作最近的n个连续版本）
 - 进入界面后，摁a或者i进入编辑模式，将需要修改注释的提交的那一行的“pick”改为“edit”，然后摁ESC退出编辑模式，再输入:wq!来保存退出
 - 退出编辑模式回到命令行后，此时再执行`git commit --amend`，进入界面后，摁a或者i进入编辑模式，将第一行中的注释(注意不是下面的带有edit或者pick字眼的注释)修改为正确的注释，然后摁ESC退出编辑模式，再输入:wq!来保存退出
 - 退出编辑模式回到命令行后，此时再执行`git rebase --continue`确认rebase操作，完成本地分支的修改注释操作
 - `git push -f` 强制推送到远程，覆盖更新

26、git format-patch打补丁文件，git am应用补丁文件

[git am_左山艾艾的博客-CSDN博客](#)

- 将dev分支部分提交合并到master分支
- 在dev分支将多个提交内容分别生成对应的补丁文件到本地仓库中或指定自定义位置（又名叫补丁文件），然后切换分支到master分支，将一个或多个补丁文件同时合并到master分支。

`$ git reflog`

a6de5cc HEAD@{0}: checkout: moving from wf_dev to master

303aaef HEAD@{1}: checkout: moving from master to wf_dev

a6de5cc HEAD@{2}: pull origin master: Merge made by the 'recursive' strategy.

602559e HEAD@{3}: commit: optimize 教师信息 添加“身份证号”字段，只实现新增、修改、显示



359b9b4 HEAD@{4}: commit: optimize 学生用户 “个人信息维护” 页面显示个人照片

ab29f7b HEAD@{5}: commit: refix_IDSP-1731 过程性评价 不显示 “是否有效” 设置为无效的必修课/自选课选课活动

6881b6e HEAD@{6}: checkout: moving from wf_dev to master

...

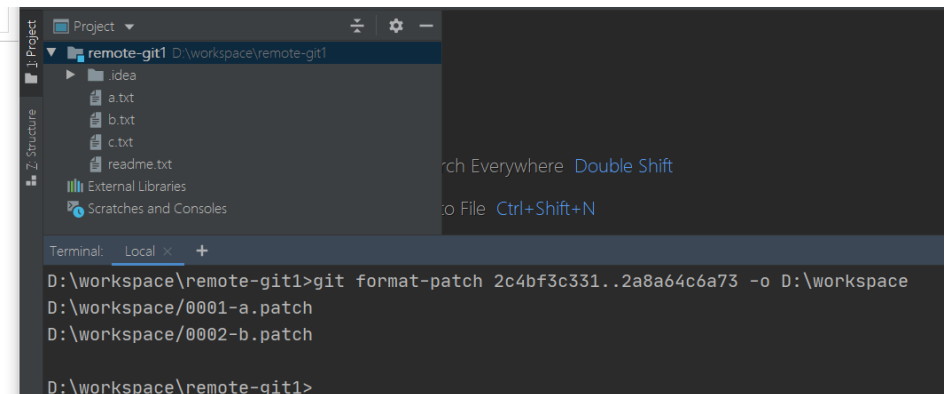
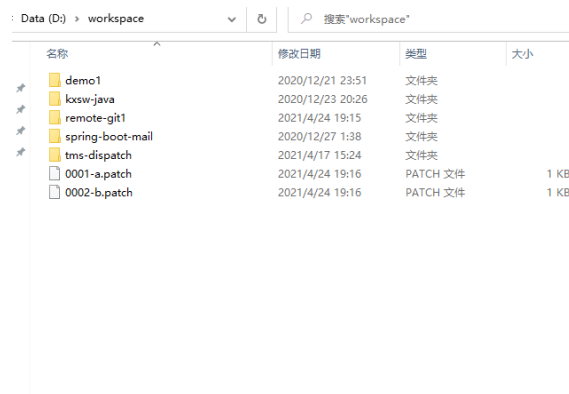
// 提交 6881b6e 之后（不包含此提交）到提交 602559e（包含此提交）的改动

语法：打补丁

- 将dev分支部分提交打成补丁文件到本地。
- 补丁文件生成到的目录，不写生成到本地仓库，-o可以自定义位置

（1）将某个连续的提交(前进)生成补丁文件，-o指定自定义位置

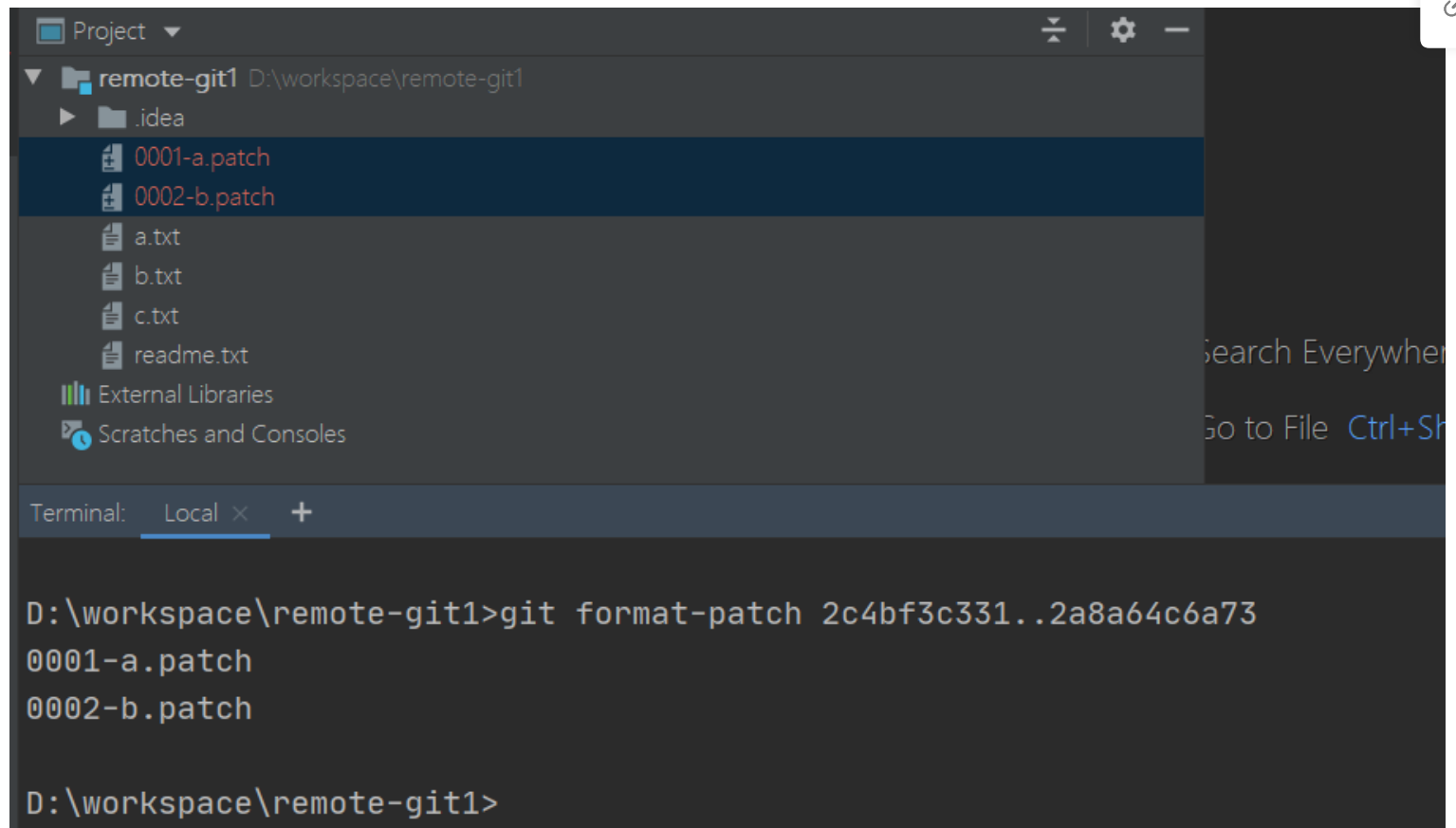
- git format-patch commit_id1..commit_id2 -o 指定生成路径
- git format-patch commit_id1..commit_id2 -o D:/workspace（不包含commit_id1,包commit_id2）



■

（2）将某个连续的提交(前进)生成补丁文件默认是生成到本地仓库的位置下

git format-patch commit_id1..commit_id2 (不包commit_id1,包
commit_id2)



The screenshot shows an IDE interface. The top part is a file explorer for a project named 'remote-git1' located at 'D:\workspace\remote-git1'. It shows a folder named '.idea' and several files: '0001-a.patch', '0002-b.patch', 'a.txt', 'b.txt', 'c.txt', and 'readme.txt'. Below the file explorer is a terminal window with the title 'Terminal: Local x +'. The terminal shows the command 'git format-patch 2c4bf3c331..2a8a64c6a73' being executed, which results in two patch files being generated: '0001-a.patch' and '0002-b.patch'. The terminal prompt is 'D:\workspace\remote-git1>'.

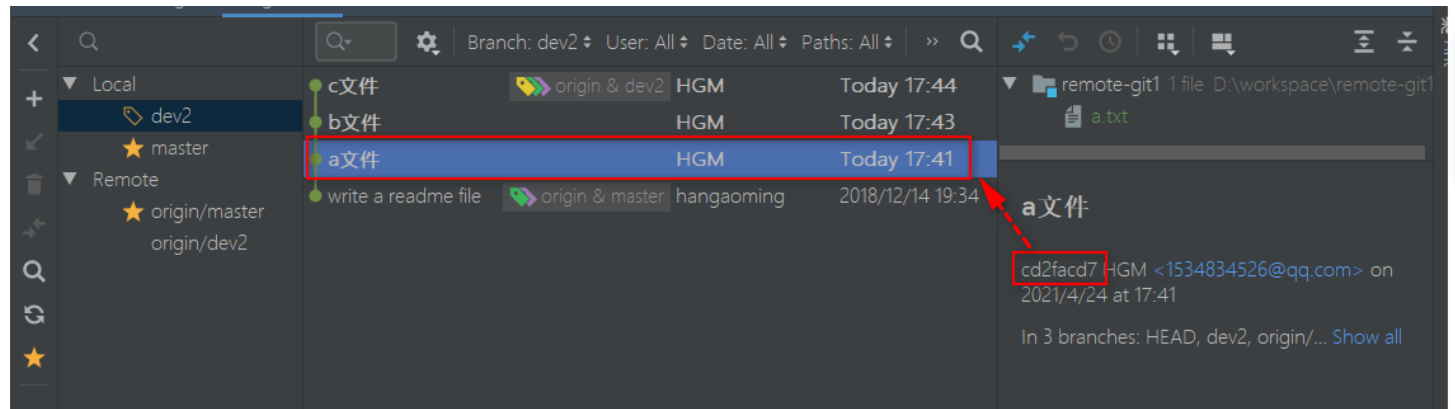
(3) 将某次commit之后的提交 (不包含该commit_id1)

- git format-patch commit_id1

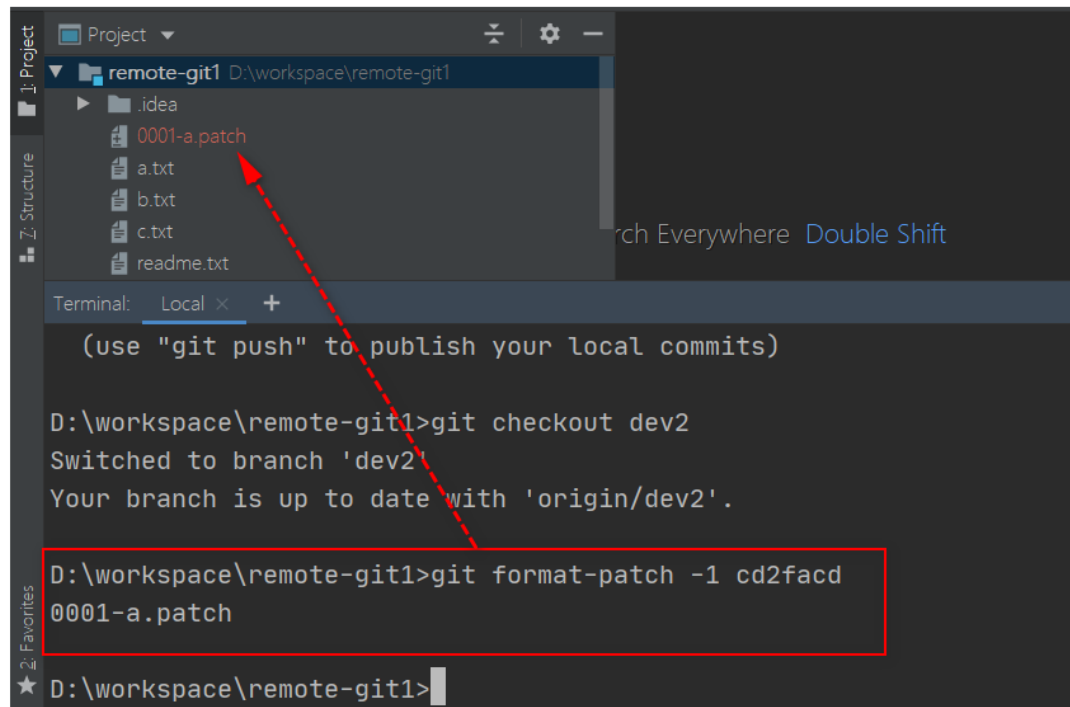
(4) 将某次提交(含)之前的n次提交(后退), -n指patch数

- git format-patch -n commit_id (-1 commit_id则只打当前的commit_id对应的提交为patch文件)

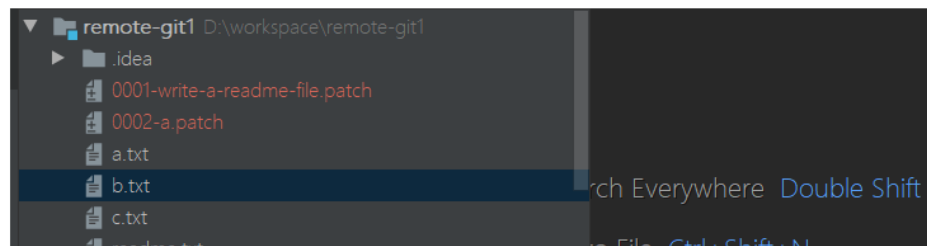
1.



2.



3.



```
Terminal: Local x +
D:\workspace\remote-git1>git format-patch -1 cd2facd
0001-a.patch

D:\workspace\remote-git1>git format-patch -2 cd2facd
0001-write-a-readme-file.patch
0002-a.patch

D:\workspace\remote-git1>
```

(5) 某次提交以后的所有提交 (前进)

git format-patch -s commit_id

(6) 将最近的几个连续提交生成补丁文件到本地仓库

- git format-patch HEAD^^ 或 git format-patch -2 //最近的2个提交

(7) 当前分支和master分支比较, 超前的提交

- git format-patch -M master

(8) 生成从根到commit_id1提交的所有patch

- git format-patch --root commit_id1

示例:

```
$ git format-patch 6881b6e..602559e -o C://patch
```

```
C://patch/0001-refix_IDSP-1731.patch
```

```
C://patch/0002-optimize.patch
```

```
C://patch/0003-optimize.patch
```

语法: 应用补丁

- 先检查patch文件的情况



- git apply --stat xxx.patch
- 检查patch文件能否应用成功
 - git apply --check xxx.patch
- 在分支上(master)应用patch补丁文件
 - git am --signoff < D:/workspace/*.patch (使用-s或--signoff选项, 可以将打patch文件的人的信息给加上)
- 将此路径下的所有patch文件按照先后顺序, 打到master分支上
 - git am D://workspace/*.patch

```
HGM@LAPTOP-FVEP6CM2 MINGW64 /d/workspace/remote-git1 (master)
$ git am d://workspace/*.patch
Applying: a文件
Applying: b文件

HGM@LAPTOP-FVEP6CM2 MINGW64 /d/workspace/remote-git1 (master)
$
```

示例:

```
$ git am --signoff < C:\\patch\\0001-refix_IDSP-1731.patch
```

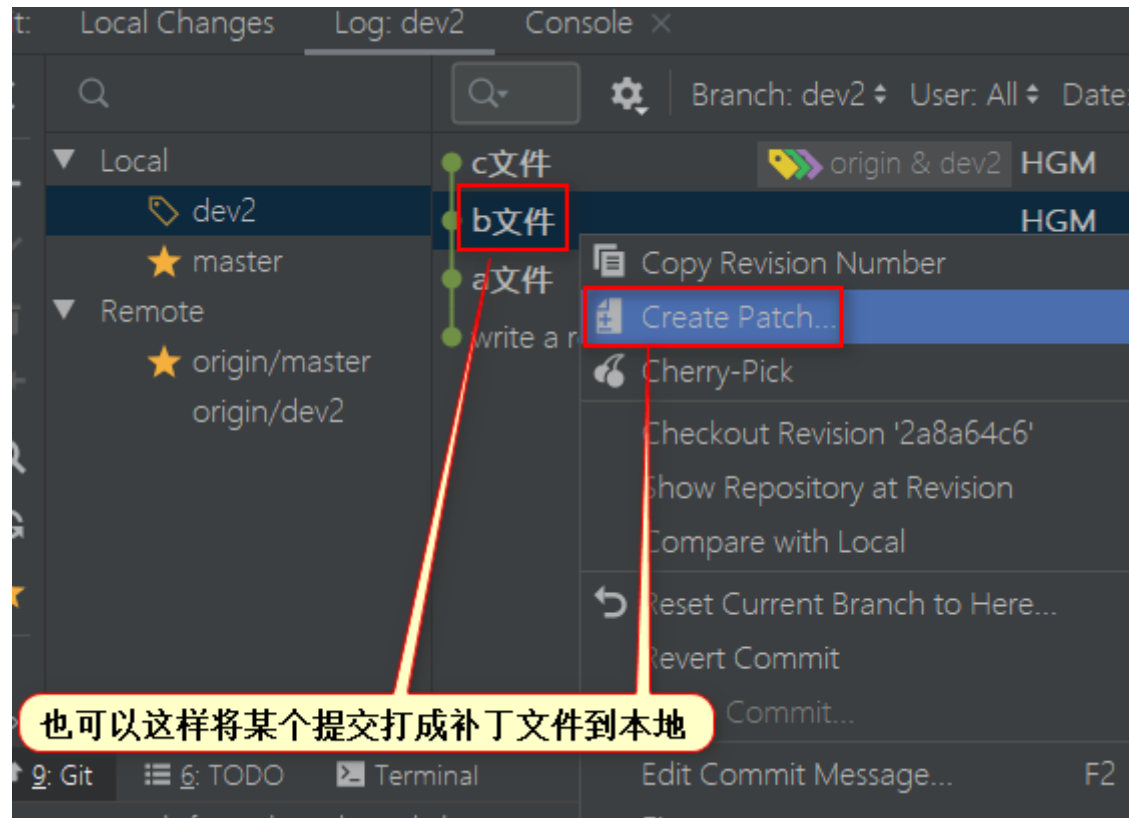
Applying: refix_IDSP-1731 过程性评价 不显示 “ 是否有效 ” 设置为无效的必修课/自选课选课活动

应用补丁失败的方案:

- 当git am失败时, 用以将已经在am过程中应用的补丁patch废弃掉(比如有三个patch, 打到第三个patch时有冲突, 那么这条命令会把打上的前两个patch丢掉, 返回没有打patch的状态)
 - git am --abort (PATCH_NAME 可以指定丢弃哪一个补丁, 不写全丢弃)
- 当git am失败后,想继续应用补丁
 - git status 看到am过程暂时中止了, 但是还处于am对话中

- `git apply --reject xxx.patch` 进行强制应用补丁（若这步无法强制进行合并，则手动解决冲突）
- 解决冲突后，`git add`到暂存区
- `git am --continue` 继续进行am过程
- 然后`git am --resolved` 重新am应用补丁，然后就会显示:Applying: XXXXX.

语法：IDEA中直接打补丁



27、`git branch --contains`查看某个提交属于哪个分支

- `git branch --contains commit_id`

28、git show 查看某个提交的修改内容

- git show commitId

29、IDEA中替换git bash命令行的操作

- Checkout Revision '2a8a64c6'是切换到这个版本（为了HEAD和分支，直接操作版本的操作，见本笔记的1部分）

