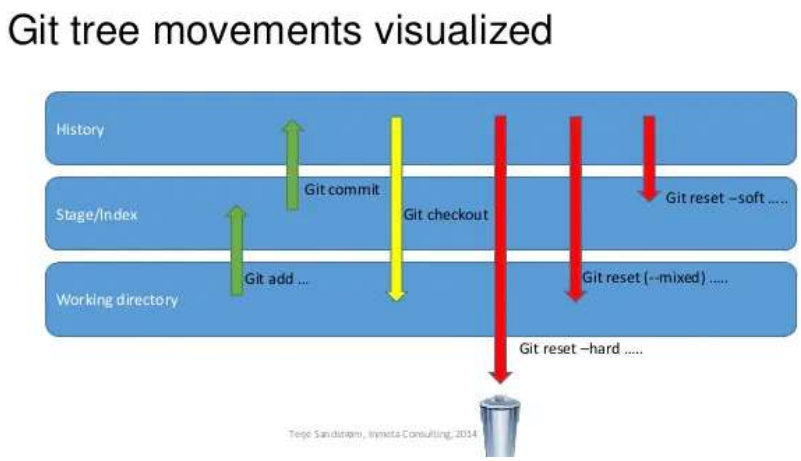
Git Reset 三种模式

carway 关注
16 2019.01.11 17:59:41 字数 2,908 阅读 327,281

有时候，我们用Git的时候有可能commit提交代码后，发现这一次commit的内容是有错误的，那么有两种处理方法：
1、修改错误内容，再次commit一次 2、使用git reset 命令撤销这一次错误的commit
第一种方法比较直接，但会多次一次commit记录。
而我个人更倾向第二种方法，错误的commit没必要保留下来。
那么今天来说一下git reset。它的一句话概括

```
git-reset - Reset current HEAD to the specified state
```

意思就是可以让HEAD这个指针指向其他的地方。例如我们有一次commit不是不是很满意，需要回到上一次的Commit里面。那么这个时候就需要通过reset，把HEAD指针指向上一次的commit的点。
它有三种模式，soft,mixed,hard，具体的使用方法下面这张图，展示的很全面了。



git各个区域和命令关系

这三个模式理解了，对于使用这个命令很有帮助。在理解这三个模式之前，需要略微知道一点Git的基本流程。正如上图，Git会有三个区域：

- Working Tree 当前的工作区域
- Index/Stage 暂存区域，和git stash命令暂存的地方不一样。使用git add xx，就可以将xx添加近Stage里面
- Repository 提交的历史，即使用git commit提交后的结果

推荐阅读
Git备忘
阅读 52
前端管理工具（git之道）
阅读 183
来自大牛总结的 Git 使用技巧，写得太好了
阅读 1,055
保姆级Git入门教程
阅读 358
很nice的git学习
阅读 240

文件存入Repository流程

以下简单敘述一下把文件存入Repository流程：

1. 刚开始 working tree 、 index 与 repository(HEAD)里面的内容都是一致的

阶段1

2. 当git管理的文件夹里面的内容出现改变后，此時 working tree 的內容就会跟 index 及 repository(HEAD)的不一致，而Git知道是哪些文件(Tracked File)被改动过，直接将文件状态设置为 modified (Unstaged files)。

阶段2

3. 当我們执行 git add 后，会将这些改变的文件内容加入 index 中 (Staged files)，所以此时 working tree跟index的内容是一致的，但他们与repository(HEAD)内容不一致。

阶段3

4. 接着执行 git commit 後，將Git索引中所有改变的文件内容提交至 Repository 中，建立出新的 commit 节点(HEAD)后， working tree 、 index 與与repository(HEAD)区域的内容 又会保持一致。

推荐阅读

Git备忘
阅读 52

前端管理工具（git之道）
阅读 183

来自大牛总结的 Git 使用技巧，写得太好了
阅读 1,055

保姆级Git入门教程
阅读 358

很nice的git学习
阅读 240

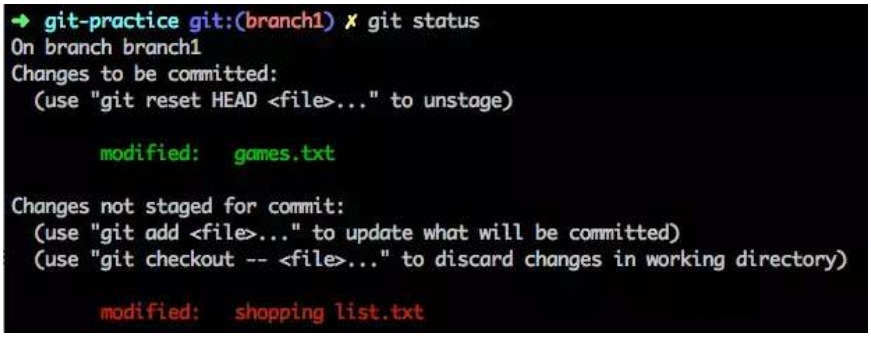
实战演示

reset --hard：重置stage区和工作目录:

reset --hard 会在重置 HEAD 和branch的同时，重置stage区和工作目录里的内容。当你在 reset 后面加了 --hard 参数时，你的stage区和工作目录里的内容会被完全重置为和HEAD的新位置相同的内容。换句话说，就是你的没有commit的修改会被全部擦掉。

例如你在上次 commit 之后又对文件做了一些改动：把修改后的ganmes.txt文件add到stage区，修改后的shopping list.txt保留在**工作目录**

```
git status
```



最初状态

然后，你执行了reset并附上了--hard参数：

```
git reset --hard HEAD^
```

你的 HEAD **和当前** branch 切到上一条commit 的同时，你工作目录里的新改动和已经add到stage区的新改动也一起全都消失了：

```
git status
```

reset --hard head^之后

可以看到，在 reset --hard 后，所有的改动都被擦掉了。

reset --soft：保留工作目录，并把重置 HEAD 所带来的新的差异放进暂存区

reset --soft 会在重置 HEAD 和 branch 时，保留工作目录和暂存区中的内容，并把重置 HEAD

推荐阅读

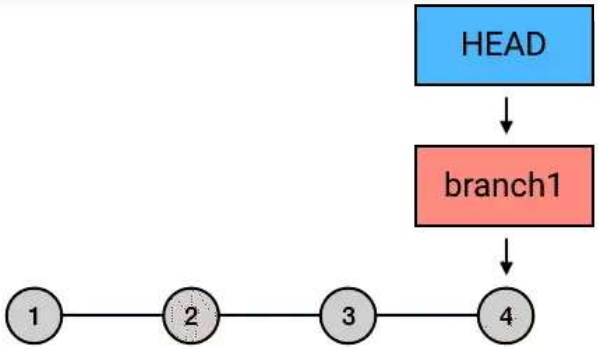
Git备忘
阅读 52

前端管理工具（git之道）
阅读 183

来自大牛总结的 Git 使用技巧，写得太好了
阅读 1,055

保姆级Git入门教程
阅读 358

很nice的git学习
阅读 240

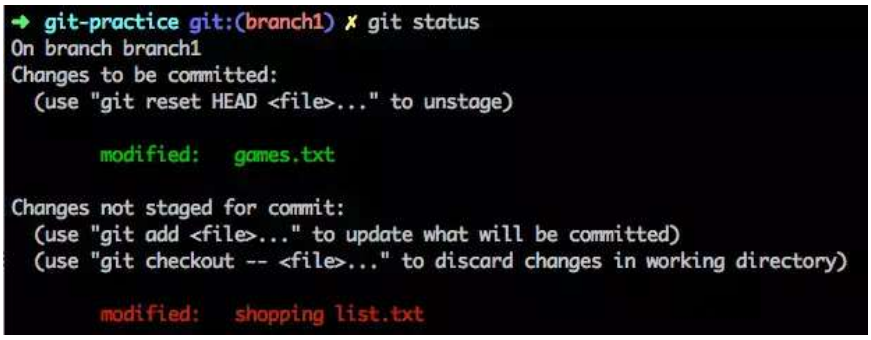


由于 HEAD 从 4 移动到了 3，而且在 reset 的过程中工作目录和暂存区的内容没有被清理掉，所以 4 中的改动在 reset 后就也成了工作目录新增的「工作目录和 HEAD 的差异」。这就是上面一段中所说的「重置 HEAD 所带来的差异」。

此模式下会保留 working tree**工作目录**的内容，不会改变到目前所有的git管理的文件夹的内容；也会
保留 index**暂存区**的内容，让 index **暂存区**与 working tree 工作目录的内容是一致的。就只有 repository 中的内容的更变需要与 reset 目标节点一致，因此原始节点与reset节点之间的差异变更集合会存在与index暂存区中(Staged files)，所以我们可以直接执行 git commit 将 index**暂存区**中的内容提交至 repository 中。当我们想合并「当前节点」与「reset目标节点」之间不具太大意义的 commit 记录(可能是阶段性地频繁提交)時，可以考虑使用 Soft Reset 来让 commit 演进线图较为清晰点。

所以在同样的情况下，还是老样子：把修改后的ganmes.txt文件add到stage**区**，修改后的shopping list.txt保留在**工作目录**

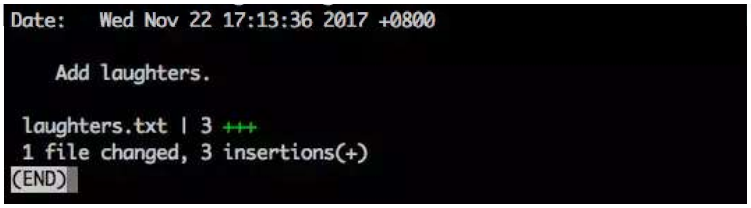
```
git status
```



最初状态

假设此时当前 commit 的改动内容是新增了 laughters.txt 文件：

推荐阅读
Git备忘
阅读 52
前端管理工具（git之道）
阅读 183
来自大牛总结的 Git 使用技巧，写得太好了
阅读 1,055
保姆级Git入门教程
阅读 358
很nice的git学习
阅读 240



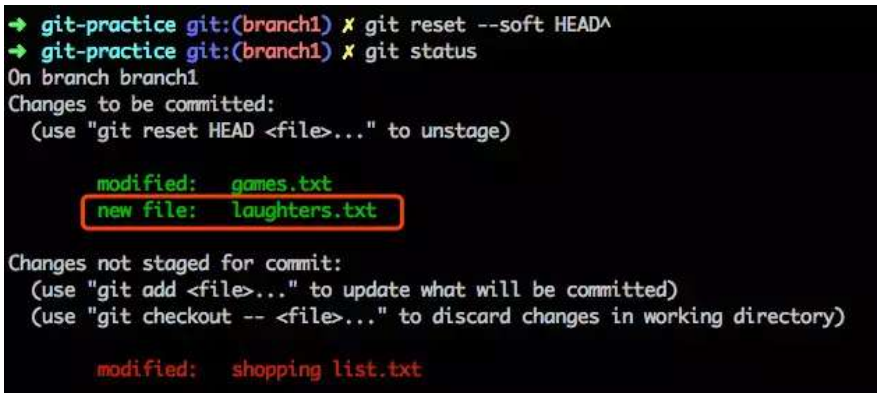
git show --stat

如果这时你执行：

```
git reset --soft HEAD^
```

那么除了 HEAD 和它所指向的 branch1 被移动到 HEAD^ 之外，原先 HEAD 处 commit 的改动（也就是那个 laughters.txt 文件）也会被放进暂存区：

```
git status
```



使用git reset --soft HEAD^后

这就是--soft 和 --hard 的区别：--hard 会清空工作目录和暂存区的改动,*而 --soft**则会保留工作目录的内容，并把因为保留工作目录内容所带来的新的文件差异放进暂存区。**

reset **不加参数**(mixed)：保留工作目录，并清空暂存区

reset 如果不加参数，那么默认使用 --mixed 参数。它的行为是：保留工作目录，并且清空暂存区。也就是说，工作目录的修改、暂存区的内容以及由 reset 所导致的新的文件差异，都会被放进工作目录。简而言之，就是「把所有差异都混合（mixed）放在工作目录中」。

还以同样的情况为例：

```
git status
```

推荐阅读

Git备忘
阅读 52

前端管理工具（git之道）
阅读 183

来自大牛总结的 Git 使用技巧，写得太好了
阅读 1,055

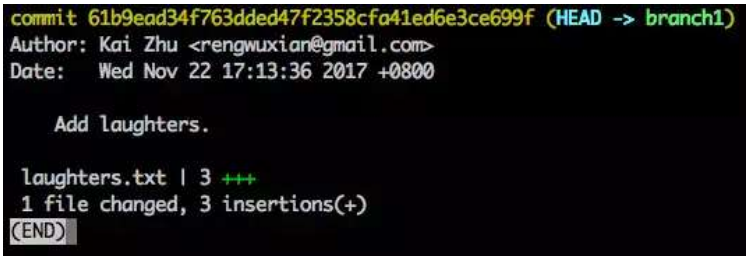
保姆级Git入门教程
阅读 358

很nice的git学习
阅读 240

最初状态

修改了 的games.txt **和** shopping list.txt，**并把** games.txt **放进了暂存区。**

```
git show --stat
```



git show --stat

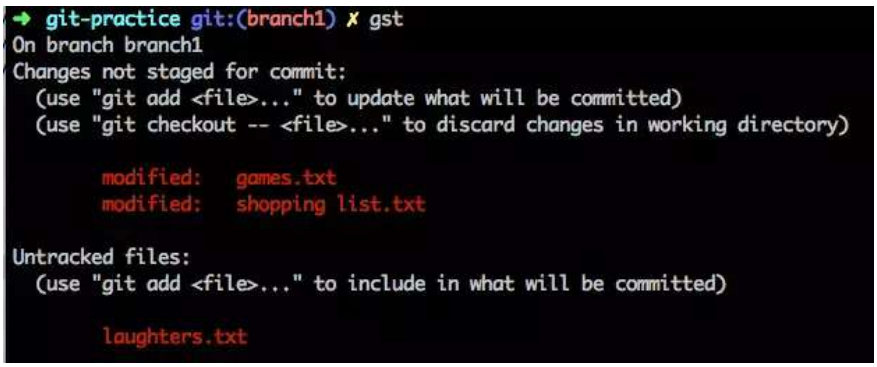
最新的 commit **中新增了** laughters.txt **文件。**

这时如果你执行**无参数**的reset或者带--mixed参数：

```
git reset HEAD^
git reset --mixed HEAD^
```

工作目录的内容和 --soft 一样会被保留，但和 --soft 的区别在于，它会把暂存区清空,并把原节点和reset节点的差异的文件放在工作目录，总而言之就是，工作目录的修改、暂存区的内容以及由 reset 所导致的新的文件差异，都会被放进工作目录

```
git status
```



git reset HEAD^之后

总结

reset **的本质：移动** HEAD **以及它所指向的** branch

实质上，reset 这个指令虽然可以用来撤销 commit ，但它的实质行为并不是撤销，而是移动 HEAD ，并且「捎带」上 HEAD 所指向的 branch（如果有的话）。也就是说，reset 这个指令的行为其实和它的字面意思 "reset"（重置）十分相符：它是用来重置 HEAD 以及它所指向的 branch 的位置的。

推荐阅读

Git备忘
阅读 52

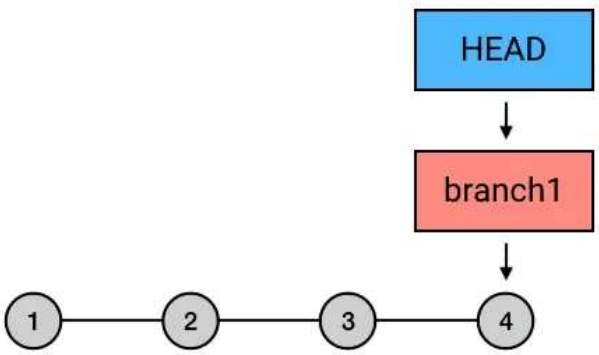
前端管理工具（git之道）
阅读 183

来自大牛总结的 Git 使用技巧，写得太好了
阅读 1,055

保姆级Git入门教程
阅读 358

很nice的git学习
阅读 240


自动立体仓库

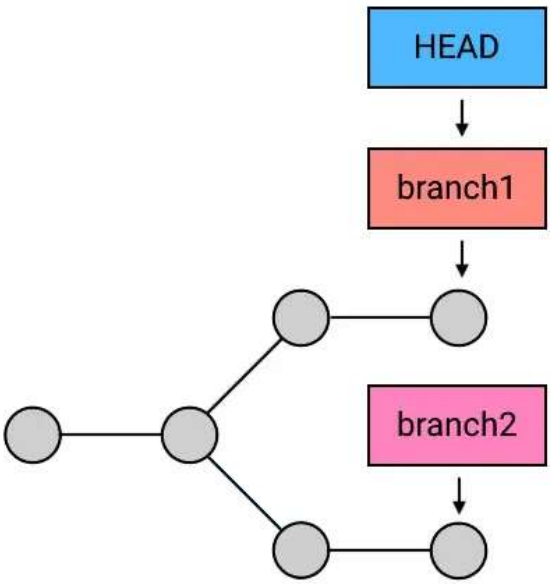


git reset

Git 的历史只能往回看，不能向未来看，所以把 HEAD 和 branch 往回移动，就能起到撤回 commit 的效果。

所以同理，reset --hard 不仅可以撤销提交，还可以用来把 HEAD 和 branch 移动到其他的任何地方。

```
git reset --hard branch2
```



git reset --hard branch2

reset三种模式区别和使用场景

区别：

推荐阅读

Git备忘
阅读 52

前端管理工具（git之道）
阅读 183

来自大牛总结的 Git 使用技巧，写得太好了
阅读 1,055

保姆级Git入门教程
阅读 358

很nice的git学习
阅读 240

2. --soft：重置位置的同时，保留working Tree**工作目录**和index**暂存区**的内容，只让repository中的内容和 reset 目标节点保持一致，因此原节点和reset节点之间的【差异变更集】会放入index**暂存区**中(Staged files)。所以效果看起来就是工作目录的内容不变，暂存区原有的内容也不变，只是原节点和Reset节点之间的所有差异都会放到暂存区中。

3. --mixed **（默认）**：重置位置的同时，只保留Working Tree**工作目录**的内容，但会将 Index **暂存区** 和 Repository 中的内容更改和reset目标节点一致，因此原节点和Reset节点之间的【差异变更集】会放入Working Tree**工作目录**中。所以效果看起来就是原节点和Reset节点之间的所有差异都会放到工作目录中。

使用场景:

1. --hard：(1) **要放弃目前本地的所有改变时**，即去掉所有add到暂存区的文件和工作区的文件，可以执行 git reset -hard HEAD 来强制恢复git管理的文件夹的内容及状态；(2) **真的想抛弃目标节点后的所有**commit（可能觉得目标节点到原节点之间的commit提交都是错了，之前所有的commit有问题）。

2. --soft：原节点和reset节点之间的【差异变更集】会放入index**暂存区**中(Staged files)，所以假如我们之前工作目录没有改过任何文件，也没add到暂存区，那么使用reset --soft后，我们可以直接执行 git commit 将 index暂存区中的内容提交至 repository 中。为什么要这样呢？这样做的使用场景是：假如我们想合并「当前节点」与「reset目标节点」之间不具太大意义的 commit 记录(可能是阶段性地频繁提交,就是开发一个功能的时候，改或者增加一个文件的时候就commit，这样做导致一个完整的功能可能会好多个commit点，这时假如你需要把这些commit整合成一个commit的时候)时，可以考虑使用reset --soft来让 commit 演进线图较为清晰。总而言之，**可以使用**--soft**合并**commit**节点。**

3. --mixed **（默认）**：(1)使用完reset --mixed后，我們可以直接执行 git add 将這些改变果的文件内容加入 index **暂存区**中，再执行 git commit 将 Index**暂存区** 中的内容提交至 Repository中，这样一样可以达到合并commit节点的效果（与上面--soft合并commit节点差不多，只是多了git add添加到暂存区的操作）；(2)移除所有Index暂存区中准备要提交的文件(Staged files)，我们可以执行 git reset HEAD 来 Unstage 所有已列入 Index**暂存区** 的待提交的文件。(有时候发现add错文件到暂存区，就可以使用命令)。(3)commit提交某些错误代码，或者没有必要的文件也被commit上去，不想再修改错误再commit（因为会留下一个错误commit点），可以回退到正确的commit点上，然后所有原节点和reset节点之间差异会返回工作目录，假如有个没必要的文件的话就可以直接删除了，再commit上去就OK了。

假如手贱，又想回退撤销的版本呢?

请看另外一篇文章:TODO

参考文章:

https://dotblogs.com.tw/wasichris/2016/04/29/225157
https://www.domon.cn/2018/09/06/Git-reset-used-in-coding/
https://juejin.im/book/5a124b29f265da431d3c472e/section/5a14529bf265da43310d7351(掘金小册)

241人点赞 > 日记本 ···

推荐阅读
Git备忘
阅读 52
前端管理工具（git之道）
阅读 183
来自大牛总结的 Git 使用技巧，写得太好了
阅读 1,055
保姆级Git入门教程
阅读 358
很nice的git学习
阅读 240

写下你的评论... 评论23 赞241