Git 词典

（查看帮助，命令前加 -h）

**帮助中的语法：**

[-i] 中括号里面是命令可用的参数，根据参数说明使用

参数中“|”表示或，使用时两个参数只能选一个，如：[-x | -X]

<> 角括号中的表示用户输入的内容

●查看当前版本状态（不知道干啥时就用它）

git status

git status -s //简洁模式

●查看远程仓库地址

git remote -v

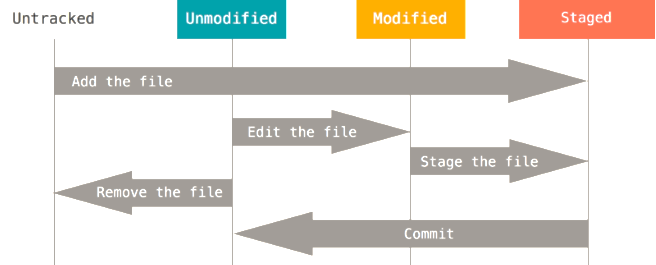
●还原本地版本为远程最新

git reset --hard //恢复到上一本地commit版本

git pull origin develop //拉取指定版本库分支覆盖本地

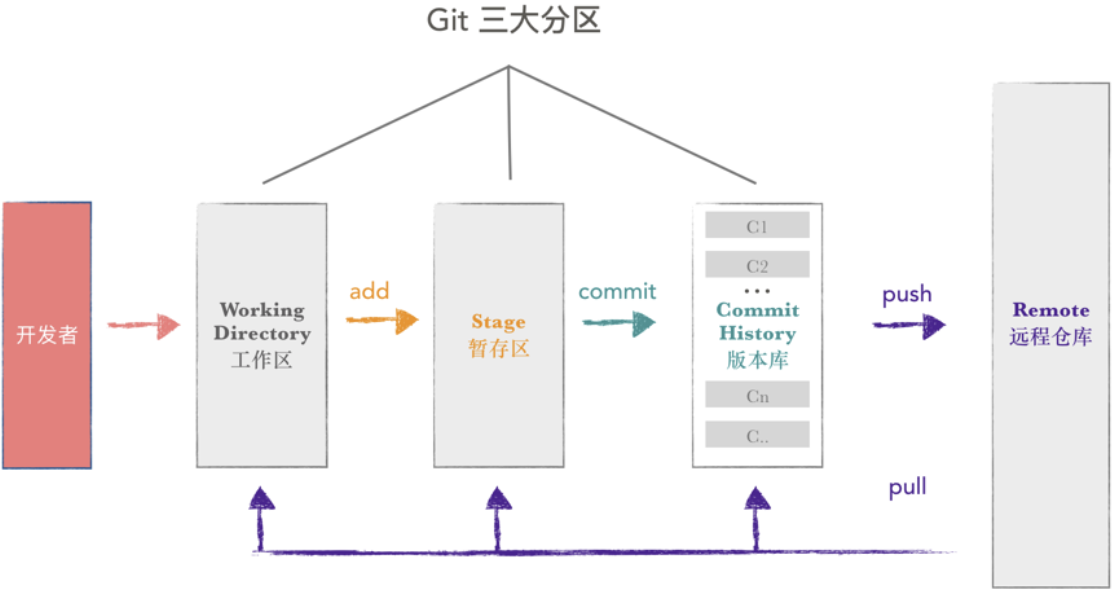
◆1下图展示了文件在本地管理中各操作的去向和状态：

未跟踪的 无修改状态 **修改状态** **暂存区**



上图修改状态下的Stage the file既是add。

◆2 Git三大分区



# ◇3 Git三棵树的思维框架

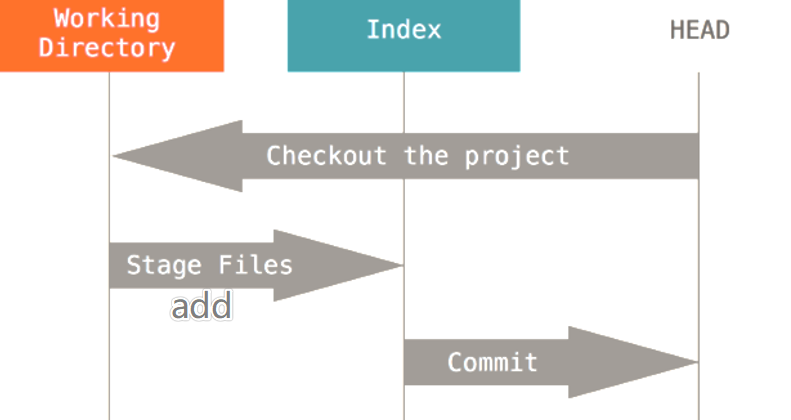
通过操作来管理三棵树，这里“树”代表文件的集合。

|  |  |
| --- | --- |
| **树** | **用途** |
| HEAD | 上次提交的快照，下次提交的父节点 |
| Index | 预期的下一次提交的快照 |
| Working Directory | 工作目录 |

**工作流程：**

经典的git工作流程是通过操纵这三个区域来以更加连续的状态记录项目快照的。

工作目录 索引 指针



**工作目录**

也叫**工作区**，可以把工作目录当做**沙盒**，在修改提交到暂存区并记录到历史之前可以随意更改。

**索引**

是你**预期的下一次提交**，Git的暂存区，通过git commit将它们转换为树来作新的提交。（通过幕后命令ls-files查看索引）

git ls-files -s

**HEAD**

是当前分支引用的指针，它总是指向该分支的最后一次提交。可以理解它是**该分支上的最后一次提交**的快照。（通过底层命令cat-file和ls-tree显示HEAD快照的目录列表）

git cat-file -p HEAD

tree daf6913ca6b55a86a13d63567781abb50b035c1b

parent 20b8b2c6503208d52ec0e15e7c8c9e2082296f71

author vic.zeng <vic.zeng@idreamsky.com> 1655633403 +0800

committer vic.zeng <vic.zeng@idreamsky.com> 1655633403 +0800

git ls-tree -r HEAD

100644 blob 2b90b7cdf14d86c313e13ceb9f5719b472d96dc3 "git\name.smmx"

100644 blob de395e44b5fe8a8c5437978173dc9fadfd7ce7df "\name.docx"

下面我们以图形方式来观察一下三棵树的工作流程：

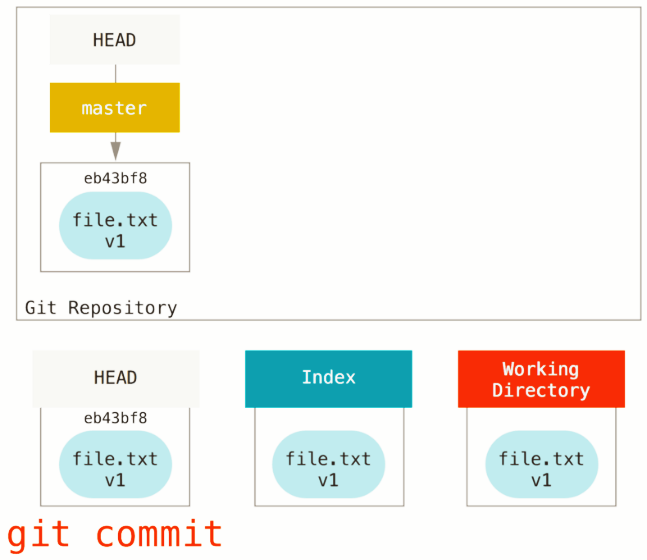




此时只有工作目录有内容，我们要提交这个文件，用git add来获取工作目录中的内容复制到索引中。



接着运行git commit取得索引中的内容并保存为一个永久的快照，创建一个指向该快照的提交对象，最后更新当前分支指向本次提交。



此时运行git status会发现没有任何改动，因为三棵树完全相同。

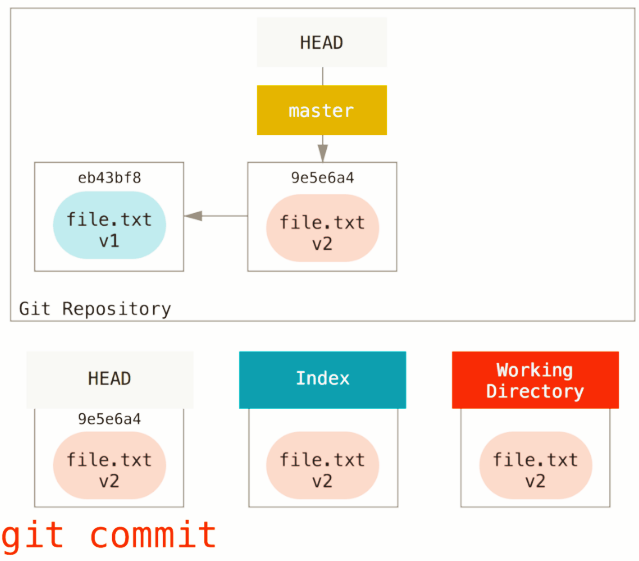
现在对文件进行修改然后提交它，修改后的文件为v2版本，并标记为红色，



接着我们运行git add来将它暂存到索引中，



现在预期的下一次提交与上次提交不同，同样运行git commit来完成提交，

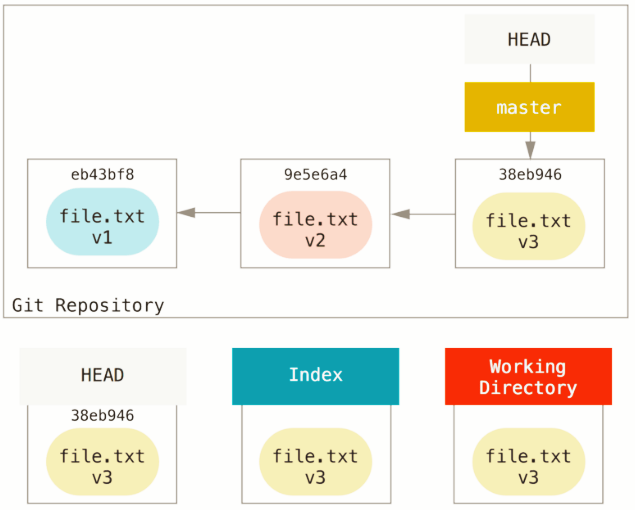


切换分支或克隆的过程也类似，当检出一个分支时会修改**HEAD**指向新的分支引用，将**索引**填充为该次提交的快照，然后将**索引**的内容复制到**工作目录**。

◆3 END

# ◇4 重置的作用 reset

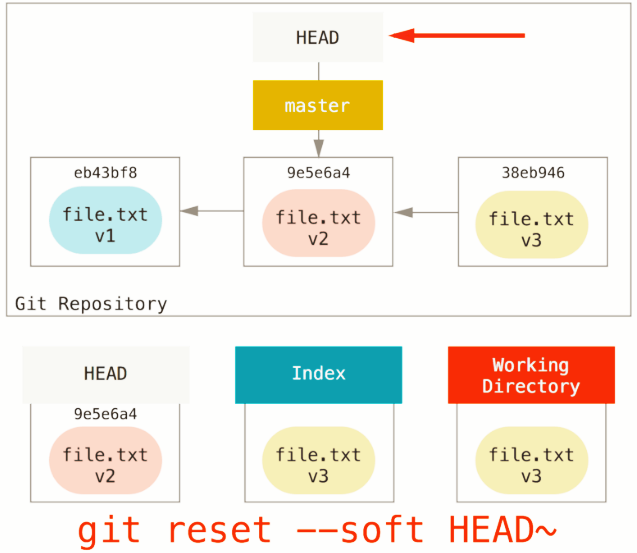
假设我们再次修改file.txt文件并第三次提交它，现在历史看起来是这样的：



reset用简单的方式直接操纵三棵树，以下三个基本操作（移动HEAD、更新索引--mixed、更新工作目录--hard）

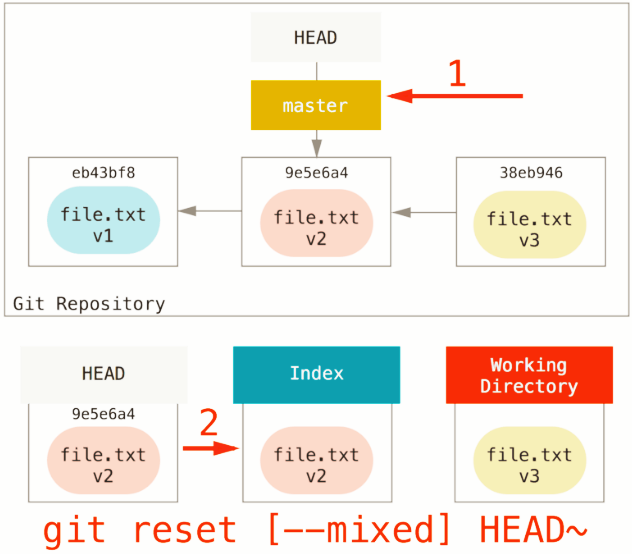
第一步：移动HEAD

reset做的第一件事是移动HEAD的指向，这与改变HEAD不同（checkout做的），运行git reset 9e5e6a4将会使master指向9e5e6a4索引，它**本质上是撤销了上一次commit命令**，当你reset回HEAD~（HEAD的父结点）时，就是把该分支移动会上一位置，而不会改变索引和工作目录，现在可以更新索引并再次git commit来完成git commit --amend要做的事情了（修改最后一次提交）。

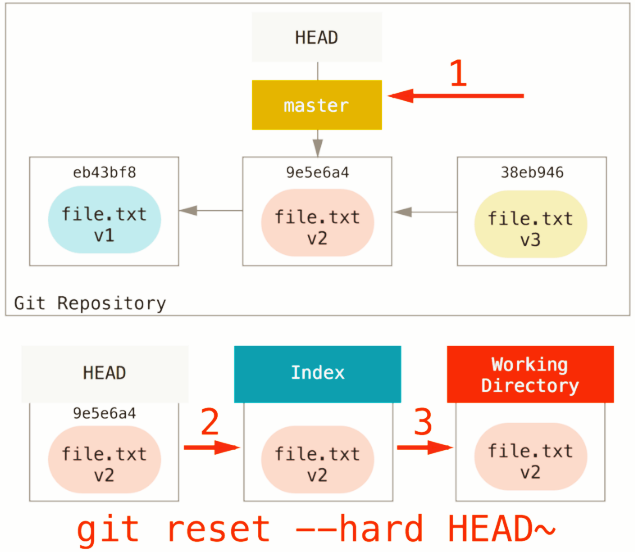


第二步：更新索引（--mixed）

现在运行status的话会看到新的HEAD以绿色标出它和索引的区别。接下来reset会用HEAD指向的当前快照的内容来更新索引。



第三步：更新工作目录（--hard）



将索引更新到工作目录，必须注意，--hard标记是reset命令唯一危险的用法，强制覆盖工作区后三棵树将保持一致，如果git数据库中有v3版本，可以通过reflog来找回。

# Git引用

引用（references，简写为refs）

# A

add

添加文件到暂存区，若添加未跟踪的文件也可以说开始跟踪一个文件（add命令后可以加一个文件、多个文件、文件夹，文件可以用“\*”通配符替代）（◆1）

**用法：**git add [<-参数>] [--] <路径名>…

-n 操作预演

-v 详细模式

-x 包含未跟踪的和gitignor排除的文件

-X 只对gitignor收录的文件

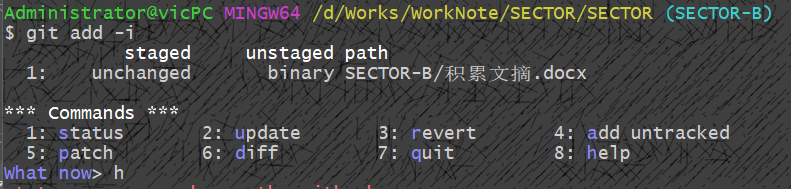
-d 含整个文件夹

git add \*.c //跟踪所有后缀c的文件

git add file1 file2 //跟踪多个文件

git add . //添加包含所有变更和新增

[-i]交互式操作



# B

branch

分支操作

# C

clone

克隆远程仓库到本地

git clone <url> <新文件夹(选填)>

commit

（◆1、2、3）

checkout

（◆3）

clean

对没有被跟踪tracked或被.gitignor收录的文件进行删除（未被add或commit的本地文件）

**用法：**git clean [-d] [-x | -X] [--] <多路径>…

-n 列出要清除的文件（在操作前最好加-n参数，确认文件无误时再去掉-n执行真正删除操作）

-f 强制操作

-x 除了未跟踪的也包含gitignor排除的文件

-X 只对gitignor收录的文件

-d 含整个文件夹

git clean -n //列出将要删除的未跟踪文件（untracked）

git clean -Xdn //列出gitignor排除的文件

# D

diff

对比各区中的文件修改差异（◆2）

git diff //工作区vs暂存区

git diff head //工作区vs版本库

git diff --cached //暂存区vs版本库

# F

fetch

从仓库下载对象和引用

fast-forwards

# L

log

查看提交历史

# M

merge

# R

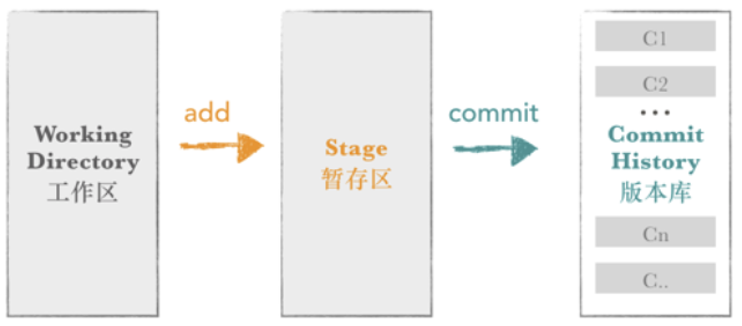
reset（◇4）

回滚文件，移动HEAD指向索引的位置，不改变索引和工作目录（◆2、4重置的作用）

git reset --soft //暂存区->工作区（移动HEAD）

git reset --mixed //版本库->暂存区（更新索引）

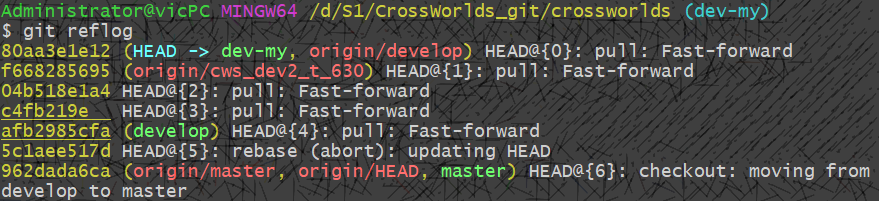
git reset --hard //版本库->暂存区->工作区（更新工作区）



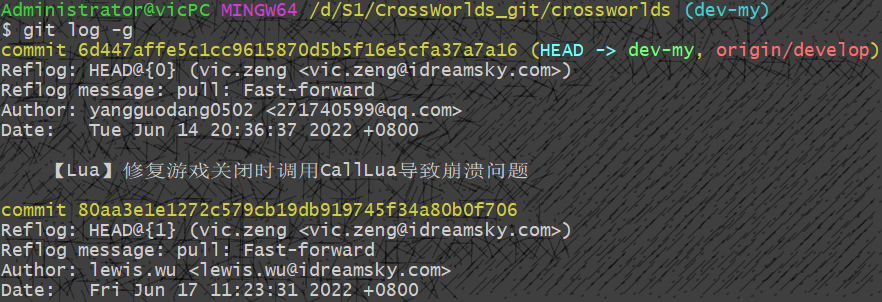
reflog

可以在任何时候执行git reflog命令来了解你曾经做过什么

git reflog



为了使信息更加详细，我们可以使用如下命令，以标准日志格式输出引用日志



git log -g

remove

（◆1）

revert

restore

rm

Git提交简书

正常工作流程