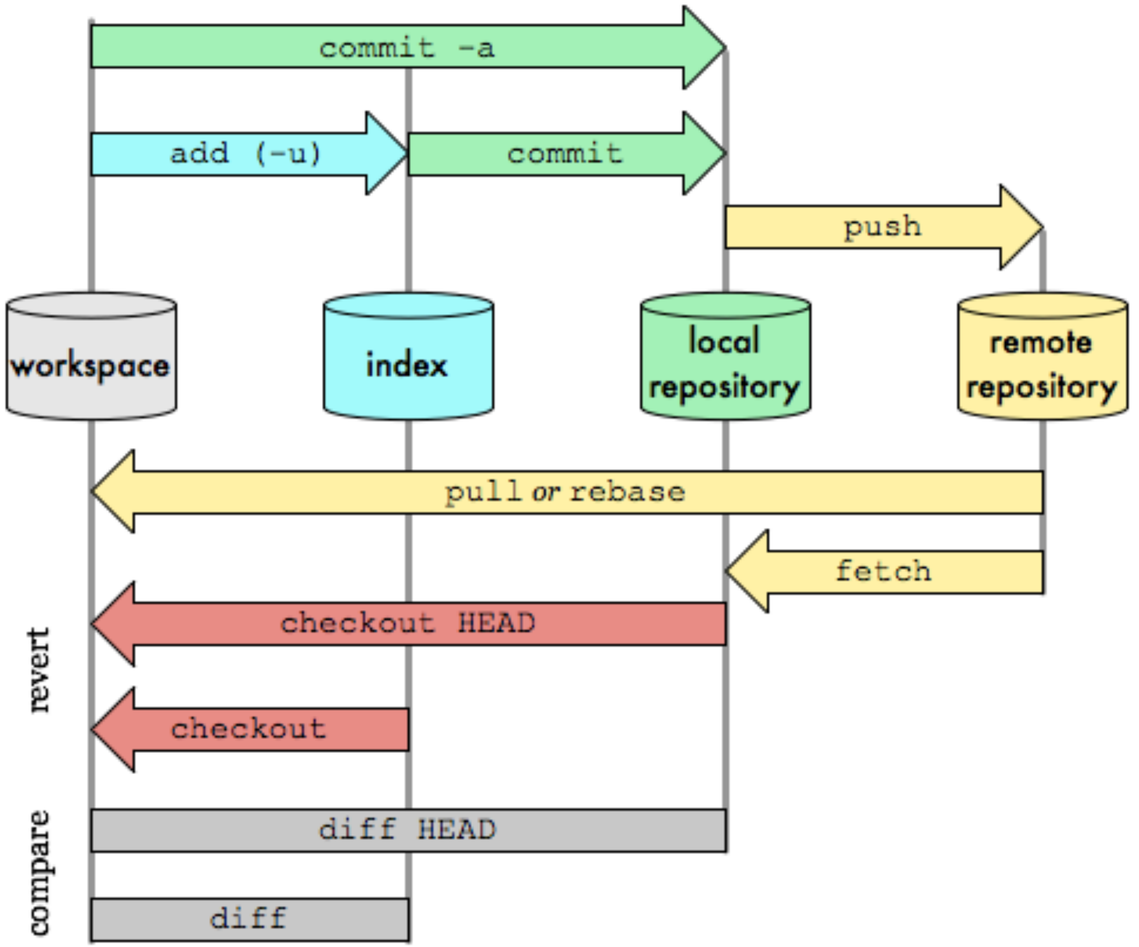
Git

刘钦

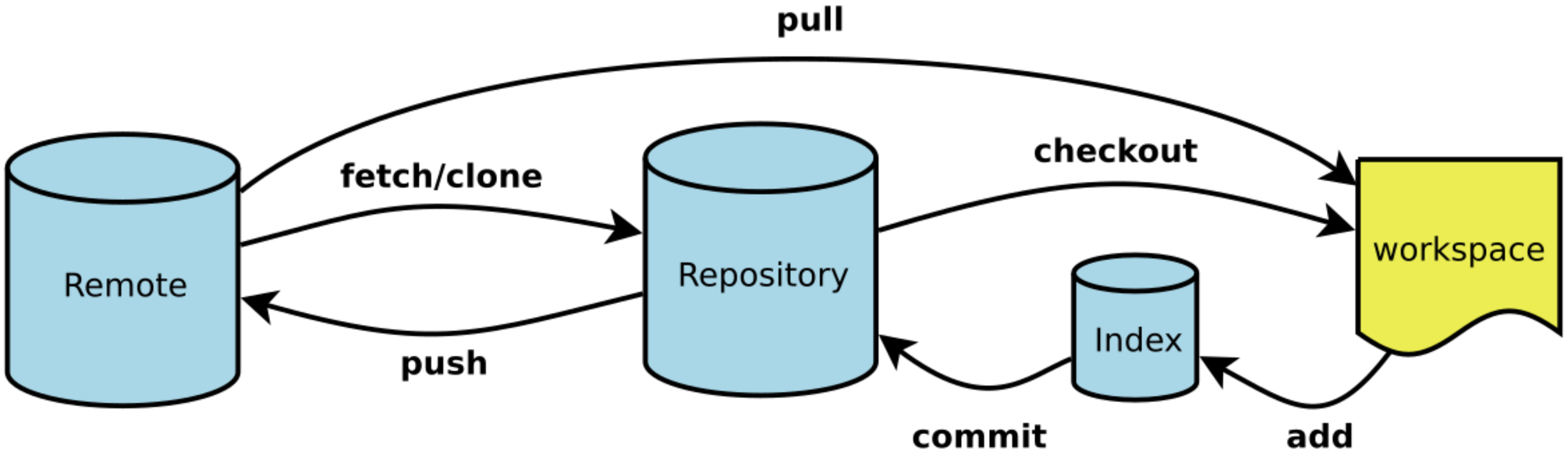
Git安装和使用

- https://git-scm.com/book/zh/v2/%E8%B5%B7%E6%AD%A5-%E5%AE%89%E8%A3%85-Git
- https://www.liaoxuefeng.com/wiki/001373951630592960 6dd18361248578c67b8067c8c017b000/001373962877033 54d8c6c01c904c7d9ff056ae23da865a000
- http://www.ruanyifeng.com/blog/2015/12/git-cheat-sheet.html
- https://learngitbranching.js.org/
- http://www.ruanyifeng.com/blog/2016/01/commit_message_change_log.html
- https://github.com/xirong/my-git/blob/master/git-workflow-tutorial.md#23-gitflow%E5%B7%A5%E4%BD%9C%E6%B5%81

工作流程


commit -a
add (-u)
commit
push
workspace
index
local repository
remote repository
pull or rebase
fetch
checkout HEAD
revert
checkout
compare
diff HEAD
diff

6个最常用的命令


pull
fetch/clone
checkout
Remote
Repository
workspace
Index
push
commit
add

下载项目

- # 下载一个项目和它的整个代码历史
- $ git clone [url]

添加文件

- # 添加指定文件到暂存区
- $ git add [file1] [file2] ...

- # 添加指定目录到暂存区，包括子目录
- $ git add [dir]

- # 添加当前目录的所有文件到暂存区
- $ git add .

代码提交

- # 提交暂存区到仓库区
- $ git commit -m [message]

- # 提交暂存区的指定文件到仓库区
- $ git commit [file1] [file2] ... -m [message]

- # 提交工作区自上次commit之后的变化，直接到仓库区
- $ git commit -a

- # 提交时显示所有diff信息
- $ git commit -v

查看

- # 显示有变更的文件
- $ git status

- # 显示当前分支的版本历史
- $ git log

- # 显示暂存区和工作区的差异
- $ git diff

撤销

- # 恢复暂存区的指定文件到工作区
- $ git checkout [file]

- # 恢复某个commit的指定文件到暂存区和工作区
- $ git checkout [commit] [file]

- # 恢复暂存区的所有文件到工作区
- $ git checkout .

- # 重置暂存区的指定文件，与上一次commit保持一致，但工作区不变
- $ git reset [file]

- # 重置暂存区与工作区，与上一次commit保持一致
- $ git reset --hard

- # 重置当前分支的指针为指定commit，同时重置暂存区，但工作区不变
- $ git reset [commit]

- # 重置当前分支的HEAD为指定commit，同时重置暂存区和工作区，与指定commit一致
- $ git reset --hard [commit]

远程同步

- # 显示所有远程仓库
- $ git remote -v

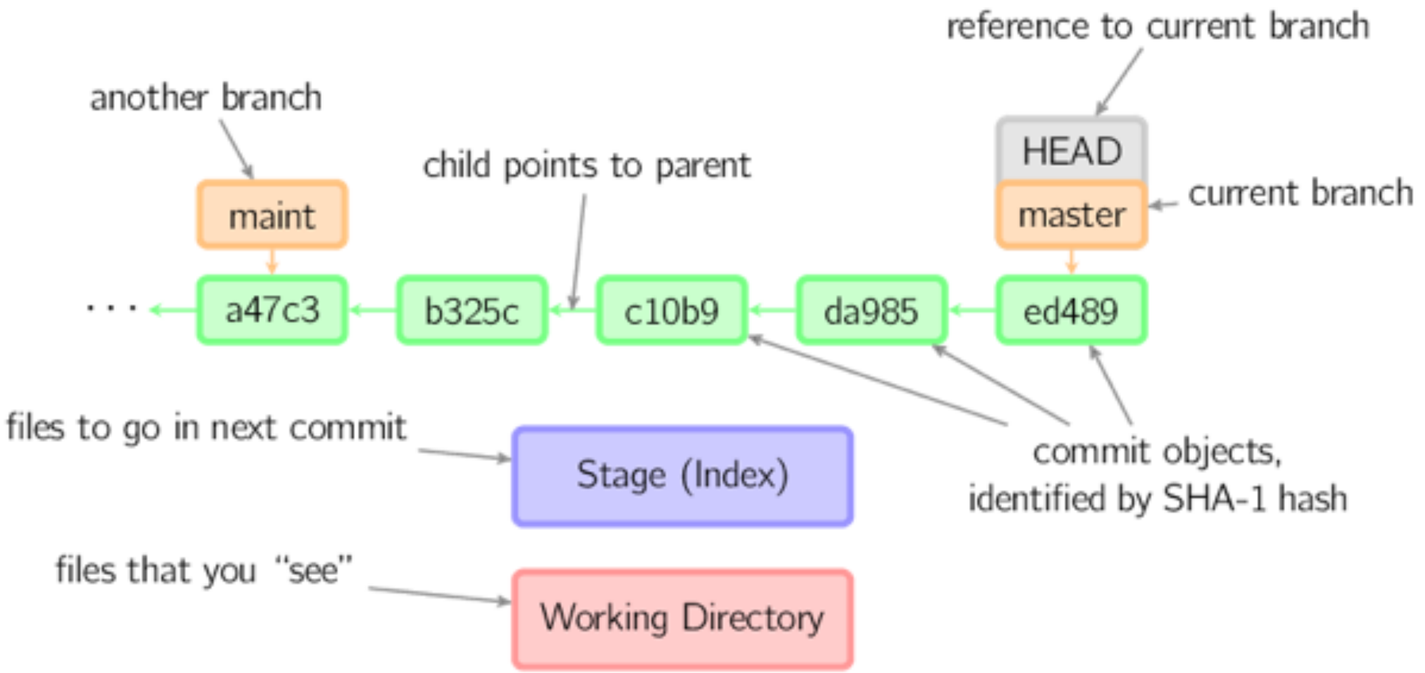
- # 显示某个远程仓库的信息
- $ git remote show [remote]

- # 下载远程仓库的所有变动
- $ git fetch [remote]

- # 取回远程仓库的变化，并与本地分支合并
- $ git pull [remote] [branch]

- # 上传本地指定分支到远程仓库
- $ git push [remote] [branch]

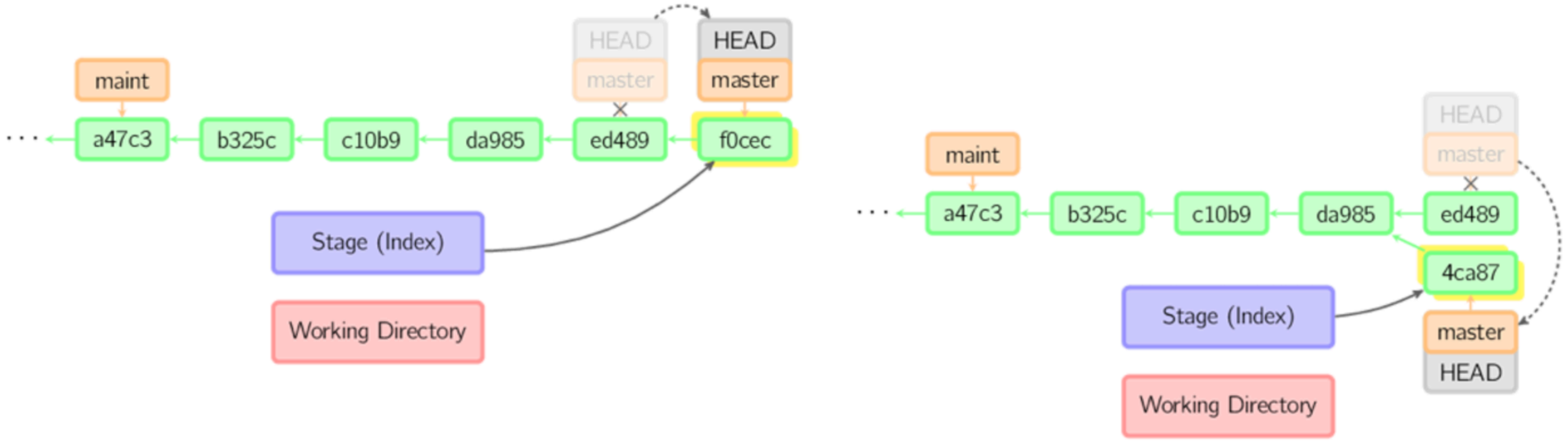
图例


reference to current branch
another branch
HEAD
child points to parent
maint
master
current branch
. . .
a47c3
b325c
c10b9
da985
ed489
files to go in next commit
Stage (Index)
commit objects,
identified by SHA-1 hash
files that you "see"
Working Directory

Commit

commit

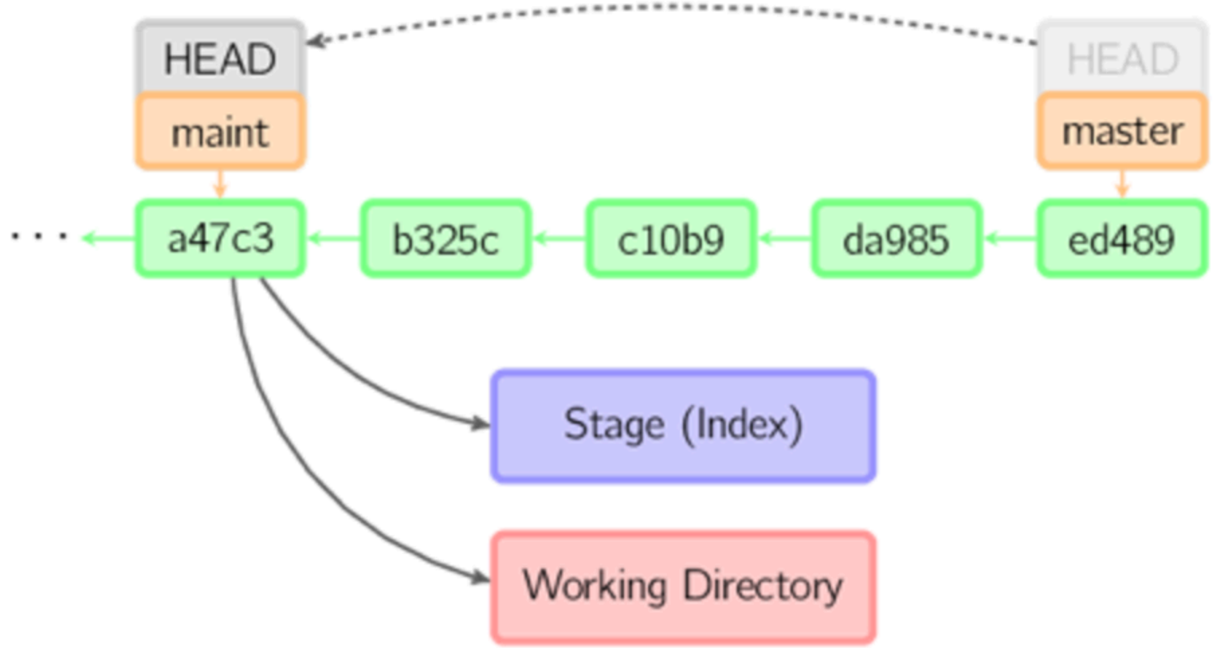
commit把暂存区的内容存入到本地仓库，并使得当前分支的HEAD向后移动一个提交点。如果对最后一次commit不满意，可以使用`git commit --amend`来进行撤销，修改之后再提交。如图所示的，ed489被4ca87取代，但是git log里看不到ed489的影子，这也正是amend的本意：原地修改，让上一次提交不露痕迹。



Checkout

checkout

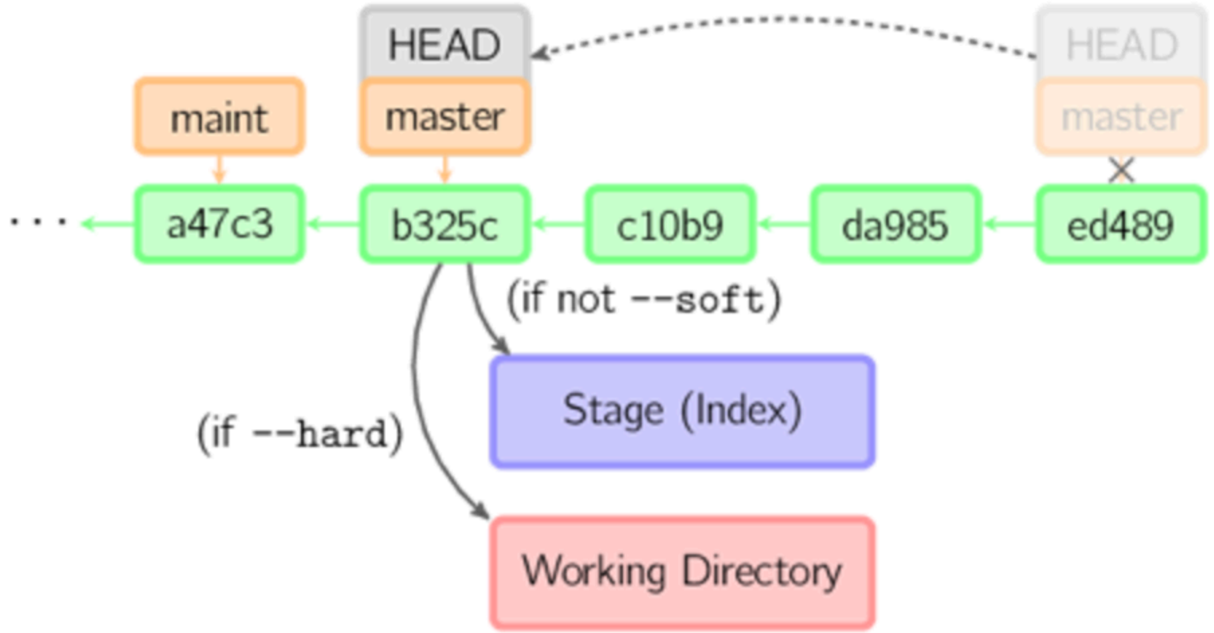
checkout用来检出并切换分支。checkout成功后，HEAD会指向被检出分支的最后一次提交点。对应的，工作目录、暂存区也都会与当前的分支进行匹配。下图是执行`git checkout maint`后的结果：



Reset

reset

reset命令把当前分支指向另一个位置，并且相应的变动工作目录和索引。如下图，执行`git reset HEAD~3`后，当前分支相当于回滚了3个提交点，由ed489回到了b325c：

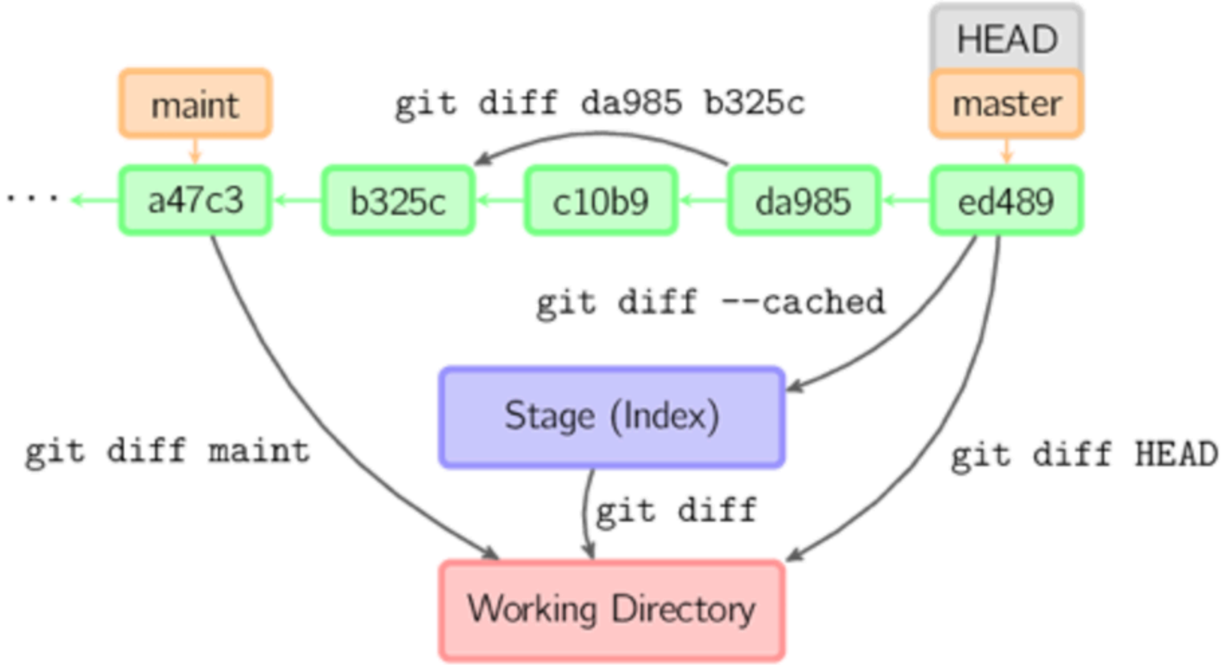


reset有3种常用的模式：
–soft，只改变提交点，暂存区和工作目录的内容都不改变
–mixed，改变提交点，同时改变暂存区的内容。这是默认的回滚方式
–hard，暂存区、工作目录的内容都会被修改到与提交点完全一致的状态

Diff

diff

我们在commit、merge、rebase、打patch之前，通常都需要看看这次提交都干了些什么，于是diff命令就派上用场了：



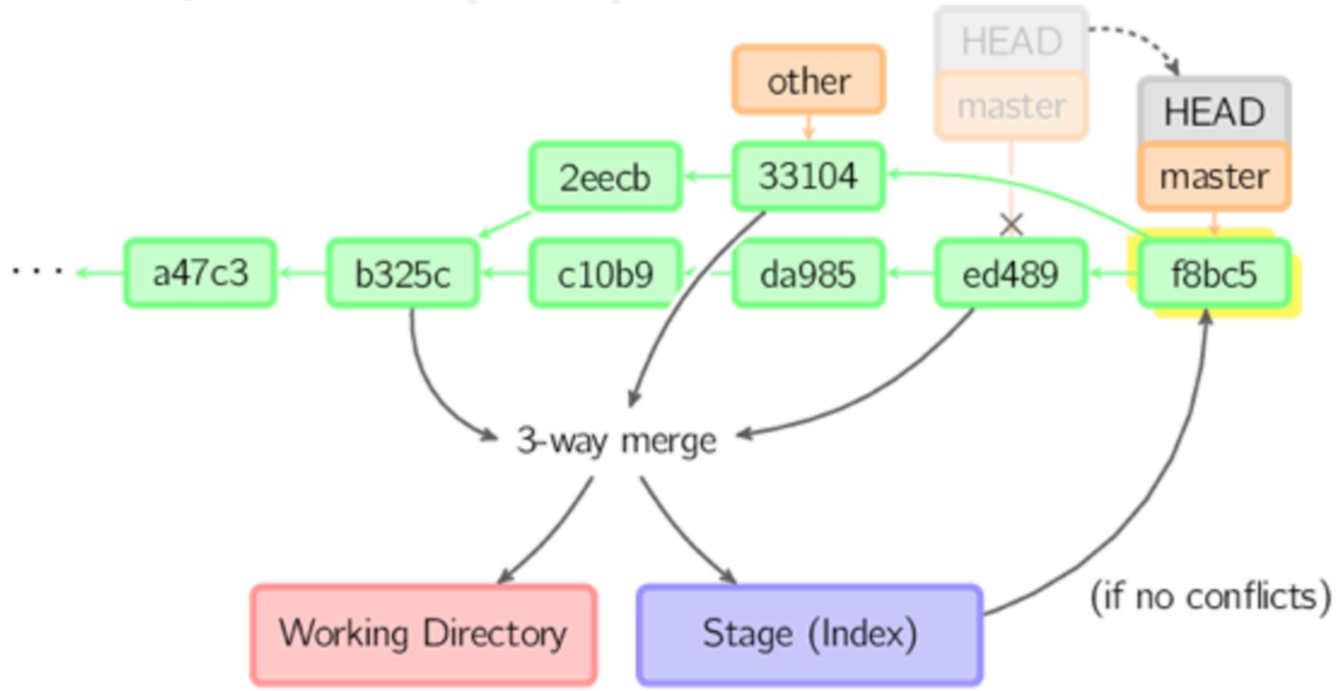
来比较下上图中5种不同的diff方式：

比较不同的提交点之间的异同，用`git diff 提交点1 提交点2`
比较当前分支与其他分支的异同，用`git diff 其他分支名称`
在当前分支内部进行比较，比较最新提交点与当前工作目录，用`git diff HEAD`
在当前分支内部进行比较，比较最新提交点与暂存区的内容，用`git diff --cached`
在当前分支内部进行比较，比较暂存区与当前工作目录，用`git diff`

Merge

merge

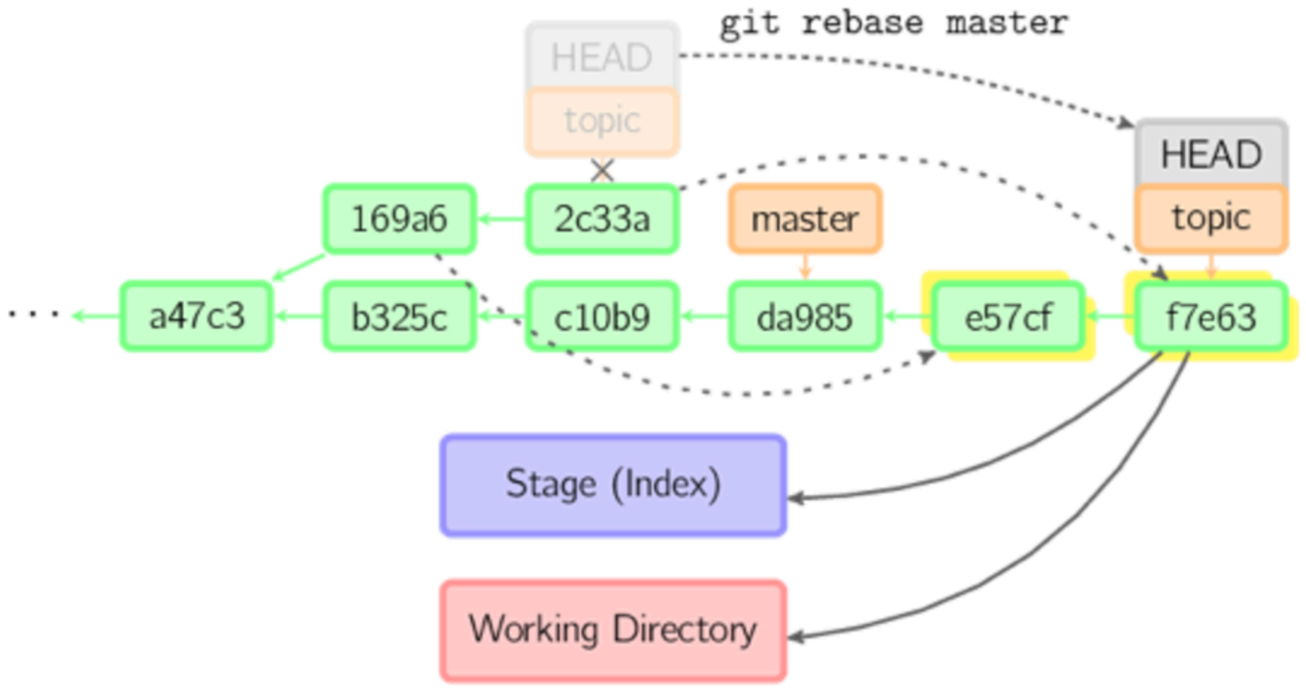
merge命令把不同的分支合并起来。如下图，HEAD处于master分支的ed489提交点上，other分支处于33104提交点上，项目负责人看了下觉得other分支的代码写的不错，于是想把代码合并到master分支，因此直接执行 `git merge other`，如果没有发生冲突，other就成功合并到master分支了。



Rebase

rebase

rebase又称为衍合，是合并的另外一种选择。merge把两个分支合并到一起进行提交，无论从它们公共的父节点开始(如上图，other分支与 master分支公共的父节点b325c)，被合并的分支(other分支)发生过多少次提交，合并都只会在当前的分支上产生一次提交日志，如上图的 f8bc5。所以merge产生的提交日志不是线性的，万一某天需要回滚，就只能把merge整体回滚。而rebase可以理解为verbosely merge，完全重演下图分支topic的演化过程到master分支上。如下图：

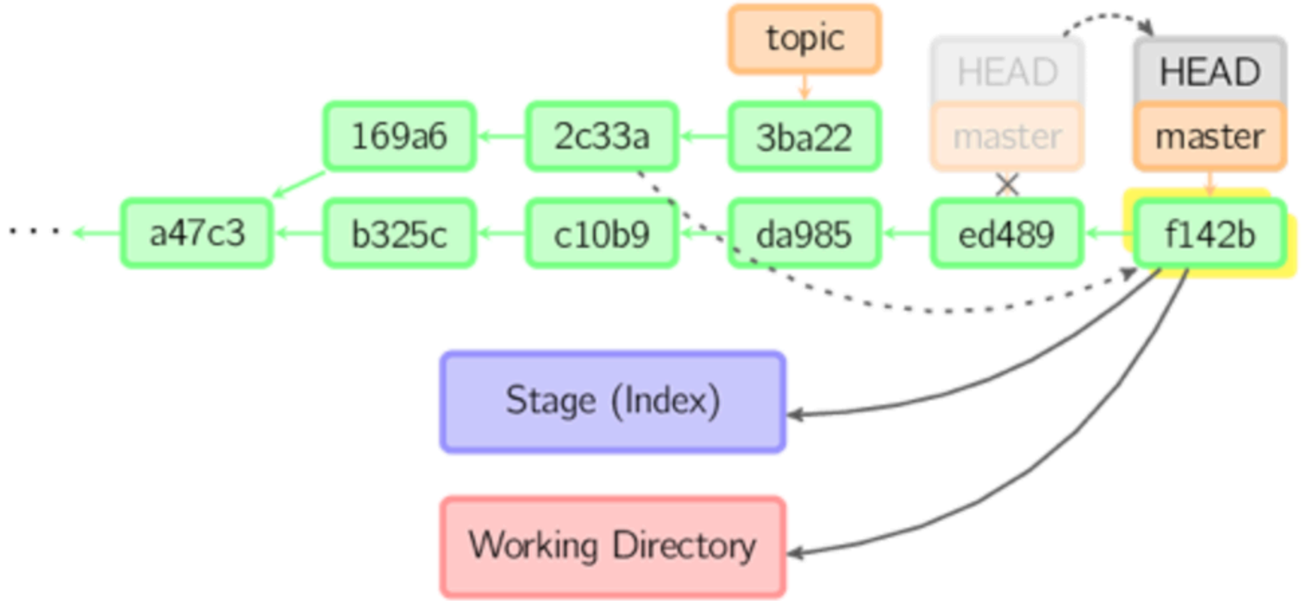


在开始阶段，我们处于topic分支上，执行`git rebase master`，那么169a6和2c33a上发生的事情都在master分支上重演一遍，分别对应于master分支上的e57cf和f7e63，最后checkout切换回到topic分支。这一点与merge是一样的，合并前后所处的分支并没有改变。`git rebase master`，通俗的解释就是topic分支想站在master的肩膀上继续下去。

Cherry-pick

cherry-pick

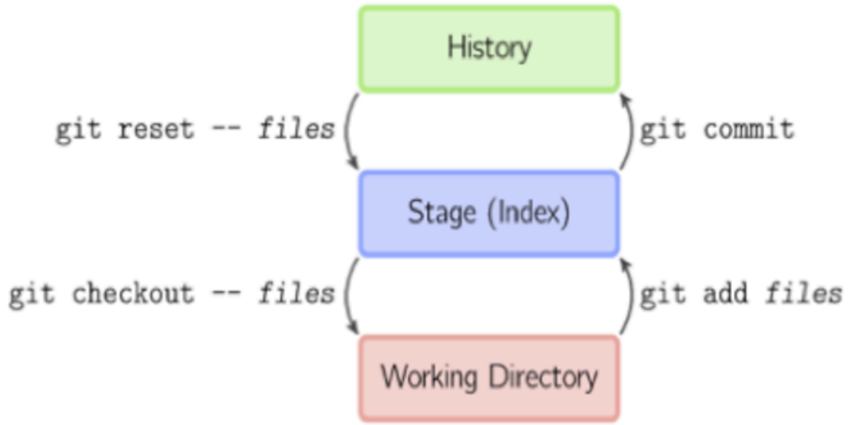
cherry-pick命令复制一个提交点所做的工作，把它完整的应用到当前分支的某个提交点上。rebase可以认为是自动化的线性的cherry-pick。
例如执行`git cherry-pick 2c33a`:

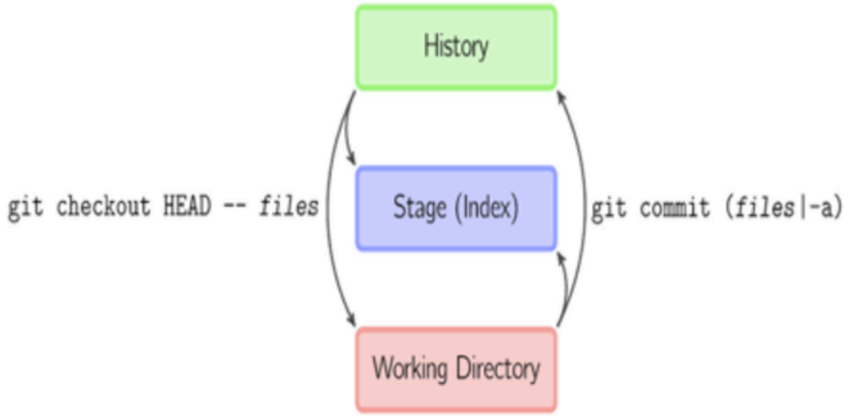


正反过程对比

正反过程对比

理解了上面最晦涩的几个命令，我们来从正反两个方向对比下版本在本地的3个阶段之间是如何转化的。如下图(history就是本地仓库)：





如果觉得从本地工作目录到本地历史库每次都要经过index暂存区过渡不方便，可以采用图形右边的方式，把两步合并为一步。

分支

- 分支的命名限定在以下范围（master | develop（缩写dev） | release | feature/xxx | hotfix/xxx | fix/xxx ） xxx为任意内容
  - Eg. 1.feature/酒店信息管理　2.hotfix/订单流程更改
- master：master永远是线上代码，最稳定的分支，存放的是随时可供在生产环境中部署的代码，当开发活动告一段落，产生了一份新的可供部署的代码。（Prod环境）
- develop（dev）：保存当前最新开发成果的分支。通常这个分支上的代码也是可进行每日夜间发布的代码，只对开发负责人开放develop权限。
- feature: 功能特性分支，每个功能特性一个 feature/ 分支，开发完成自测通过后合并入 develop 分支。可以从 master 或者develop 中拉出来。
- hotfix: 紧急bug分支修复分支。修复上线后，可以直接合并入master。
- release: 内部测试发布的分支（Test环境）
- fix: 修改master之外的分支的bug

开发流程

- 从master上创建develop分支
- 每一个需求/变更都单独从develop上创建一条feature分支；
- 用户在这个feature分支上进行Coding活动；
- 代码达到发布准入条件后上提交Codereview，Codereview通过后代码自动合并到develop分支；
- 待所有计划发布的变更分支代码都合并到develop后，系统再 rebase release分支 代码到 develop 分支，然后自行构建，打包，部署等动作。
- 应用发布成功后会基于release分支的发布版本打一个“当前线上版本Tag”基线；
- 应用发布成功后会把release分支的发布版本合并回master；

Commit Message

- "<type>(<scope>): <subject>"
- 每次commit时，添加的描述信息需要有一个特定的前缀，限定在以下范围(feat|doc|test|docs|chore|refactor|fix|style| perf): xxx　注意冒号后面有一个空格

- feat - 新功能（feature）
- docs（doc） - 文档（documentation）
- style - 格式（不影响代码运行的变动）
- refactor - 重构（即不是新增功能，也不是修改bug的代码变动）
- fix - 修补bug
- test - 增加测试
- chore - 构建过程或辅助工具的变动
- perf - 提升性能

- Eg. 1.feat: 查看酒店信息功能完成 2.fix: 浏览酒店bug修复
- * scope　SEEC系统暂未支持