

Git 命令手册

Author: Lawinder



2017-10-31

说明

该文档基于[《Git版本控制管理 第2版.pdf》](Git版本控制管理%20第2版.pdf)摘抄整理。

——赖玮东(Lawinder) 2017-10-31

# 基本的Git概念

## 版本库

Git版本库（Repository）只是一个简单的数据库，其中包含所有用来维护与管理项目的修订版本和历史的信息。

## 对象库

对象库是Git版本库实现的心脏。它包含你的原始数据文件和所有日志消息、作者信息、日期，以及其他用来重建项目任意版本或分支的信息。

Git放在对象库里的对象只有四种类型：块（blob）、目录树（Tree）、提交（commit）和标签（Tag）。

* 块（blob）

文件的每一个版本表示为一个块（blob）。blob是“二进制大对象”（binary large object）的缩写。一个blob保存一个文件的数据，但不包含任何关于这个文件的元数据，甚至连文件名也没有。

* 目录树（tree）

一个目录树（tree）对象代表一层目录信息。它记录blob标识符、路径名和在一个目录里所有文件的一些元数据。它也可以递归引用其他目录树或子树对象，从而建立一个包含文件和子目录的完整层次结构。

* 提交（commit）

一个提交（commit）对象保存版本库中每一次变化的元数据，包括作者、提交者、提交日期和日志消息。每一个提交对象指向一个目录树对象，这个目录树对象在一张完整的快照中捕获提交时版本库的状态。

* 标签

一个标签对象分配一个任意的且人类可读的名字给一个特定对象，通常是一个提交对象。

为了有效地利用磁盘空间和网络带宽，Git把对象压缩并存储在打包文件（pack file）里，这些文件也在对象库里。

## 索引

索引是一个临时的、动态的二进制文件，它描述整个版本库的目录结构。更具体地说，索引捕获项目在某个时刻的整体结构的一个版本。

工作原理：作为开发人员，你通过执行git命令在索引中暂存（stage）变更，变更通常是添加、删除、或者编辑某个文件或某些文件。索引会记录和保存那些变更，保障它们的安全直到你准备好提交了。还可以删除或者替换索引中的变更。因此，索引支持一个由你主导的从复杂的版本库状态到一个可推测的更好状态的逐步过渡。

（git add 即添加索引）。

## 可寻址内容名称

git对象库被组织及实现成一个内容寻址的存储系统。具体而言，对象库中的每个对象都会有一个唯一的名称，这个名称是向对象的内容应用SHA1得到的SHA1散列值。

## Git追踪内容

见[《Git版本控制管理 第2版.pdf》](file:///F:\个人学习\Git\Git版本控制管理%20第2版.pdf)的4.1.5章节

## 路径名与内容

见[《Git版本控制管理 第2版.pdf》](file:///F:\个人学习\Git\Git版本控制管理%20第2版.pdf)的4.1.6章节

## 打包文件

见[《Git版本控制管理 第2版.pdf》](file:///F:\个人学习\Git\Git版本控制管理%20第2版.pdf)的4.1.7章节

# 文件操作

## 添加文件

该操作是将文件添加到暂存区（添加索引）

git add test.c

## 删除文件

该操作是删除文件和相应的索引（版本库不再追踪该文件的改动）

git rm test.c

git 在删除前，会先检查以确保工作目录下该文件的版本与当前分支的最新版本是匹配的。这个验证会防止文件的修改意外丢失。

## 将一个文件由暂存区移到工作区（已暂存转为为暂存）

git rm –cached test.c

执行完该操作，git 会删除索引中的文件并把它保留到工作目录中。

如果工作区中的test.c已被事先修改为和暂存区不同的内容，那么执行git rm –cached test.c的时候，会提示相关错误。因此如果需要删除暂存区的，那么需要再加一个 参数’-f’，表示强制删除暂存区的test.c。

注意，由于test.c的索引已经被git移除，会在工作区中留有副本。因此修改好后，需要及时添加索引，否则commit的时候，test.c的修改是没有提交到仓库中的，仍然只在工作区中。

## 重命名（移动）文件

git mv test.c testRename.c

注：执行完操作后，需commit，否则版本库中的相应文件并没有被执行对应操作。

# Branch-分支操作：

## 命名分支需遵循的规则

* 可以使用斜杠(/)创建一个分层的命名方案。但是该分支不能以斜线结尾。
* 分支名不能以减号(-)开头。
* 以斜杠分割的组件不能以点(.)开头。如feature/.new这样的分支是无效的。
* 分支名的任何地方都不能包含两个连续的点(..)。
* 此外，分支名不能包含以下内容：

——任何空格或其他空白字符

——在git中具有特殊含义的字符，包括波浪线(~)、插入符(^)、冒号(:)、问号(?)、星号(\*)、左方括号([)。

——ASCII码控制字符，即小于八进制\040的字符，或DEL字符(八进制\177)。

## 查看分支

* 查看所有分支：

git branch –a

* 查看本地分支：

git branch

* 查看远程分支：

git branch -r

* 显示分支相关信息:

git branch –av

* 查看更详细的输出

git show-branch [PatternName]

[PatternName]：带有通配符的分支名

可查看[《Git版本控制管理 第2版.pdf》](file:///F:\个人学习\Git\Git版本控制管理%20第2版.pdf)的7.6章节。

* 合并变更到不同分支

某个时刻，在编辑某个文件时，后面发现不是在想要的分支上修改，这时如果直接切换到正确的分支，git可能会报错，不让切到新分支上。但是这时候又不想在该分支上提交本修改，可使用下面的操作来将本修改合并到正确的分支上。合并完后，可能需要解决该文件中的冲突部分。

git checkout –m BranchTest

## 创建分支

* 创建命令的基本形式

git branch branchName [starting-commit]

branchName:分支的名称

[starting-commit]:某个提交的名字(推荐使用提交ID，具有唯一性)。

例如：git branch BranchName 399750a2342

* 在当前节点新建分支

git branch test

* 创建分支，并切换到该分支

git checkout –b test

## 重命名分支

* 重命名本地分支

git branch -m test test\_NewName

* 重命名远程分支

方法就是先删除远程分支，然后重命名本地分支，最后再将重命名的分支提交到远程端。

## 删除分支

* 删除本地分支：

git branch –d test

* 删除远程分支：

git push origin –delete test

或者可使用如下来删除远端分支：通过推送一个空分支到远程分支

git push origin :<branchname>

* 删除掉没有与远程分支对应的本地分支（原先在服务器上有的分支，后来被删除了，本地仓库也删除该分支）

先查看origin

$ git remote show origin

\* remote origin

Fetch URL: git@github.com:xxx/xxx.git

Push URL: git@github.com:xxx/xxx.git

HEAD branch: master

Remote branches:

master tracked

refs/remotes/origin/b1 stale (use 'git remote prune' to remove)

Local branch configured for 'git pull':

master merges with remote master

Local ref configured for 'git push':

master pushes to master (up to date)

这时候便会提示某个分支过于陈旧，使用git remote prune删除。

或者直接使用下面操作来删（在fetch之后删除掉没有与远程分支对应的本地分支）。

git fetch –p

## 将分支推送到远程分支

git push origin test

## 获取远程分支

git fetch origin test

## 获取分支上的内容到本地

git pull origin test

# Tag-标签操作

## 创建标签

git tag V1.0

该方式是将标签打在最新提交的commit上

对某个提交打标签：

git tag v1.1 833429721

对commit id为833429721的提交打标签

## 将本地tag推送到远程

git push --tags

## 获取远程的标签

git fetch origin tag <tagname>

## 删除标签

删除本地仓库标签：

git tag –d <tagname>

删除远程的标签：

git push origin –delete tag <tagname>

## 创建带有说明的标签

用-a 指定标签名，-m 指定说明文字

git tag -a v0.1 -m "version 0.1 released" 3628164

## 显示标签内容、信息

git show v1.1

# diff-差异比对

## 工作区与仓库的比对

git diff

## 暂存区与仓库的比对

git diff --cached

## 显示针对两个树状态之间差异的统计数据

git diff --stat

## 对称差

git diff commit1…commit2

## 路径限制

gir diff master~5 master Folder/Document.txt

## 搜索最近50个提交中包含“String”的变更

git diff –S”String” master~50

# config-配置文件

# log-日志查看

## 查看全部提交日志

git log

## 查看所有操作日志

git reflog

## 查看某个文件的日志

git log test.c

* 如果该文件有重命名或者移动过，可增加--follow来查看所有的日志

git log --follow test.c

## 查看每次提交所更改的文件和每个改动文件中有多少行做了改动

git log –stat（可添加文件/路径，则可查看详细的文件/文件夹日志）

## 查看所有分支记录

git log –graph --all

## 单行、缩写显示所有提交

git log --graph --pretty=oneline --abbrev-commit

## 查看一段区间的提交

git log commit1…commit2

# merge-合并

## 查看在合并中产生冲突的确切来源

git log –mrge –left-right –p (filename)

* --merge :只显示跟产生冲突的文件相关的提交
* --left-right ：如果提交来自合并的“左”边则显示<（“我们的”版本，就是你开始的版本），如果提交来自合并的“右”边则显示>（“他们的”版本，就是你要合并到的版本）
* -p：显示提交信息和每个提交相关的补丁。

## 查看工作树中未合并的一组文件

git status

或者

git ls-flies –u

## 查看合并基础（当前目录下的）和你要合并入的版本之间的区别

git diff :1:filename :3:filename

## 结束解决冲突

当解决一个文件的冲突之后，执行git add(或者git rm、git update-index等)以清除它的冲突状态。

## 查看本次合并提交

git show

## 冲突解决方案搞砸，想返回到尝试解决前的原始冲突状态

git checkout -m

# 更改提交

## git reset

git reset命令会把版本库和工作目录改变为已知状态。git reset具有破坏性的，因为它可以覆盖并销毁工作目录中的修改。

有三个选项，默认为 git reset –mixed

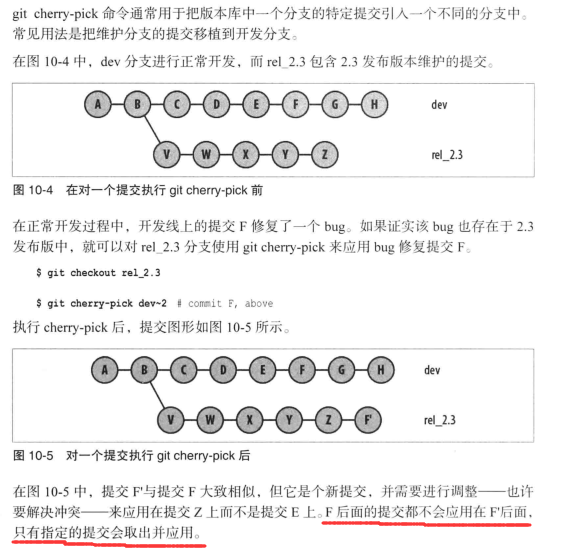
* --soft:会将HEAD引用指向给定提交，索引和工作目录的内容保持不变，该命令影响最小。
* --mixed：会将HEAD指向给定提交，索引内容也会跟着改变以符合给定提交的树结构，但是工作目录中的内容保持不变。
* --hard：索引内容也会跟着改变以符合给定提交的树结构，工作目录的内容也会随之改变以反映给定提交表示的树的状态。做的修改都将丢失，新文件将被删除。

## git cherry-pick

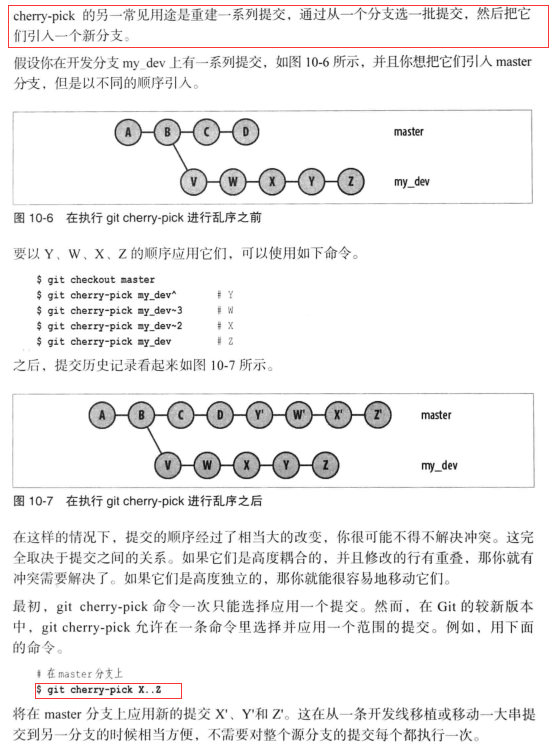
git cherry-pick提交命令会在当前分支上应用给定提交引入的变更，这将是一个新的独特提交。严格来说，使用git cherry-pick并不改变版本库中的现有历史记录，而是添加历史记录。

使用场景：

* 一

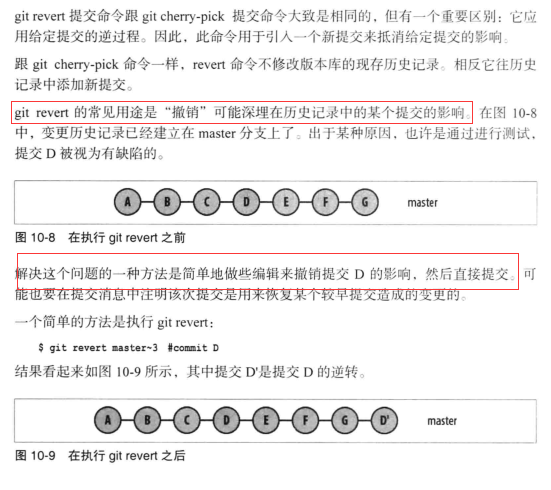


* 二



## git revert

该命令作用是应用给定提交的逆过程。比如提交了某个版本，现在需撤销该提交，则使用git revert指令可以达到这个效果。

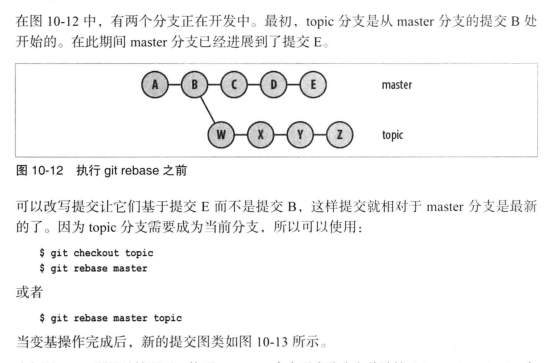


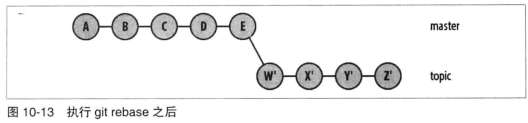
## git commit –amend

该命令最常用的是在刚做出一个提交后修改录入错误；实际上，这不是唯一的用途，然而对于任何提交，这条命令可以修改版本库中的任何文件，事实上可以作为新提交的一部分添加或删除文件（修改相应文件，使用命令更新索引，如git add/git rm，然后发出git commit –amend命令），便可以替换先去的提交。

## git rebase

git rebase命令是用来改变一串提交以什么为基础的。此命令至少需要提交将迁往的分支名。默认情况下，不在目标分支中的当前分支提交会变基。





# 查找提交

## 使用git bisect

使用git bisect可从一个确定的稳定的版本到一个出错的版本间，找出具体在哪个版本出现的修改问题，从而修复该问题。

详细介绍可参考[《Git版本控制管理 第2版.pdf》](file:///F:\个人学习\Git\Git版本控制管理%20第2版.pdf)的6.4.1章。

## 使用git blame

如果你要查看文件的每个部分是谁修改的, 那么 git blame 就是不二选择. 只要运行'git blame [filename]', 你就会得到整个文件的每一行的详细修改信息;你也可以用"-L"参数在命令(blame)中指定开始和结束行:‘git blame -L 160,+10 sha1\_file.c’

## 使用Pickaxe

带有-S选项的git log命令称为pickaxe，这种称为一种暴力考古。

git blame命令告诉你文件的当前状态，git log –Sstring则根据给定的string沿着文件的差异历史搜索。通过搜索修订版本间的实际差异，这条命令可以找到那些执行改变（增加或删除）的提交。

不过要注意，如果某个提交添加和删除了相同数量的含关键词的行，它不会显示出来。该提交必须有添加和删除数量上的变化才能技术。

# 储藏和引用日志

在日常开发周期中，当要经常中毒、修复bug、处理来自同事或者经理的请求，导致弄乱了你正在进行中的工作时，那么使用储藏（stash）就可以来解决这种困扰。储藏可以捕获你的工作进度，允许你保存工作进度并且当你方便时，再回到该进度。

git stash的储存和恢复实现了储藏状态栈，这就允许你在中断工作流的情况下再次中断。

## 储存进度

git stash save (“save message”)

## 恢复工作进度

该命令将在当前工作目录和索引中还原最近一次save操作的内容，所谓的还原，不仅仅是覆盖和替换文件，还会去除储藏的内容并合并这些变更到当前状态。

只能在一个干净的工作目录中使用git stash pop命令。

git stash pop

# .gitignore文件

可以通过在最顶层目录中新建.gitignore文件来忽略工程中不想建立索引的文件。

详细介绍可参考[《Git版本控制管理 第2版.pdf》](file:///F:\个人学习\Git\Git版本控制管理%20第2版.pdf)的5.8章。

# 远程版本库

## 版本库概念

一个Git版本库要么是一个裸(bare)版本库，要么是一个开发（非裸）（development, nonbare）版本库。

开发版本库用于常规的日常开发。它保存当前分支的概念，并在工作目录中提供检出当前分支的副本。

一个裸版本库没有工作目录，并且不应该用于正常开发。裸版本库作为协作开发的权威焦点。其他开发人员从裸版本库中克隆（clone）和抓取（fetch），并推送（push）更新。

## 使用远程版本库示例

### 创建权威版本库（可被其他人员检出的中心仓库）

首先，将一个版本库转换为权威版本库。假设中心仓库的存放目录为~/Depot，版本库存放路径为~/Repo\_Test。

cd ~/Depot

git clone –bare ~/Repo\_Test my\_ Repo\_Test.git

最后一个参数给版本库赋了一个新的名字my\_ Repo\_Test，按照惯例，裸版本库名有个.git后缀(非必需)。

### 将版本库和远程版本库建立连接

cd ~/Repo\_Test

git remote add origin ~/Depot

这样，就建立了版本库和远端版本库的单向连接。

### 从远端获取更新（更新本地数据）

git remote update

该命令会导致在这个本地的版本库中的每个remote都被更新，会从每个remote指定的版本库中检查并抓取新提交。

可限制只从一个remote获取更新：

git remote update remote\_name

此外，当最初添加远程版本库时，使用-f选项来对远程版本库执行fetch

git remote add –f origin ~/Depot

### 推送变更

将当前最新的提交从master分支推送到origin远程版本库

git push origin master

## 远程版本库配置

### 使用git remote

添加一个名为name的新远程版本库到仓库中新创建的父版本库

git remote add <name> <url>

提取关于name远程版本库的所有信息

git remote show origin

抓取远程版本库中的所有可用更新到本地版本库中

git remote update

删除本地版本库中那些陈旧的（相对于实际远程版本库）远程追踪分支。

git remote prune

更加同步的一个指令，获取更新的同时，也会删除旧的追踪分支

git remote update --prune

## 使用追踪分支

### 创建追踪分支

使用git clone克隆完一个版本库；使用git remote update 来获取远程版本库的所有可用更新到本地；然后选择一个需要追踪的远端分支，切换过去新分支，git checkout -- track <remote\_branch\_name>;如果想创建一个和远端分支不同名的追踪分支，则这样使用命令：git checkout -b <my\_branch\_name> --track <remote\_branch\_name>

将本地特性分支和上游的远程追踪分支相关联。

git remote add upstreamRepo <url>

git branch –set-upstream my\_branch upstreamRepo/branch

### 添加和删除远程分支

使用refspec的语法。

[+]源:目标

#### 添加分支

推送仅有源引用的refspec，即可以在远程版本库中创建新分支：

git push origin my\_branch

只有一个源的推送是源和目标引用使用同名的简写，也就是说git push origin my\_branch:New\_branch是将推送分支my\_branch到上游版本库，并创建或扩展名为New\_branch的分支。

#### 删除分支

同理，推送仅有目标引用的refspec导致目标从远程版本库中删除,（

为了指明引用是目标，冒号分隔符必须显示指定,如下。

git push origin :target\_branch

或者可以使用如下等价语法

git push origin --delete target\_branch

# 其他

## 查看origin所对应的远程地址内容

git remote -v