**[深入理解Git的实现原理](https://www.cnblogs.com/mamingqian/p/9711975.html)**

**0、导读**

本文适合对git有过接触，但知其然不知其所以然的小伙伴，也适合想要学习git的初学者，通过这篇文章，能让大家对git有豁然开朗的感觉。在写作过程中，我力求通俗易懂，深入浅出，不堆砌概念。你能够从本文中了解以下知识：

* Git是什么
* Git能够解决哪些问题
* Git的实现原理

请注意,本文的阐述逻辑是：Git是什么——>Git要解决的根本问题是什么——>git是如何解决这些问题的。

**1、Git是什么？**

Git是一种分布式版本控制系统。

有人要问了，什么是“版本控制”？Git又为什么被冠以“分布式”的名头呢？这两个问题我们一一解答。

版本控制这个说法多少有一点抽象。事实上，版本控制这件事儿我们一直在做，只是平时不这么称呼。举一个栗子，boss让你写一个策划案，你先完成了一稿，之后又有了一些新的想法，但是并不确定新的想法是否能得到boss的认可，于是你保存了一个**初稿**，之后在初稿的基础上另存了一个文件，做了部分修改完成了一个**修改稿**。OK，这时你的策划案就有了两个版本——初稿和修改稿。如果boss对**修改稿**不满意，你可以很轻易的把**初稿**拿出来交差。

在这个简单的过程中，你已经执行了一个简单的版本控制操作——把文档保存为初稿和修改稿的过程就是版本控制。

学术点说，版本控制就是对文件**变更过程**的管理。说白了，版本控制就是要把一个文件或一些文件的各个版本按一定的方式管理起来，目的是需要用到某个版本的时候可以随时拿出来。

另一个个问题，为什么说Git是“分布式”版本控制系统呢？

这里的“分布式”是相对于“集中式”来说的。把数据集中保存在服务器节点，所有的客户节点都从服务节点获取数据的版本控制系统叫做集中式版本控制系统，比如svn就是典型的集中式版本控制系统。

与之相对，Git的数据不止保存在服务器上，同时也完整的保存在本地计算机上，所以我们称Git为分布式版本控制系统。

Git的这种特性带来许多便利，比如你可以在完全离线的情况下使用Git，随时随地提交项目更新，而且你不必为单点故障过分担心，即使服务器宕机或数据损毁，也可以用任何一个节点上的数据恢复项目，因为每一个开发节点都保存着完整的项目文件镜像。

**2、Git能够解决哪些问题？**

就像上文举的例子一样，在未接触版本控制系统之前，大多人会通过保存项目或文件的备份来达到版本控制的目的。通常你的文件或文件夹名会设置成“XXX-v1.0”、“XXX-v2.0”等。

这是一种简单的办法，但过于简单。这种方式无法详细记录版本附加信息，难以应付复杂项目或长期更新的项目，缺乏版本控制约定，对协作开发无能为力。如果你不慎使用了这种方式，那么稍稍过一段时间你就会发现连自己都不知道每个版本间的区别，版本控制形同虚设。

Git能够为我们解决版本控制方面的大多数问题，利用Git

* 我们可以为每一次变更提交版本更新并且备注更新的内容；
* 我们可以在项目的各个历史版本之间自如切换；
* 我们可以一目了然的比较出两个版本之间的差异；
* 我们可以从当前的修改中撤销一些操作；
* 我们可以自如的创建分支、合并分支；
* 我们可以和多人协作开发；
* 我们可以采取自由多样的开发模式。

诸如此类，数不胜数。然而实现这些功能的基础是**对文件变更过程的存储**。如果我们能抓住这个根本，提纲挈领的学习git，会事半功倍。

随着对Git更深入的学习，你会发现它会变得越来越简单，越来越纯粹。道家有万法归宗的说法，用在这里再合适不过。因为Git之所以有如此多炫酷的功能，根源只有一个：它很好的解决了**文件变更过程存储**这一个问题。

所以，如果问“Git能够解决哪些问题？”我们可以简单的回答：Git解决了版本控制方面的很多问题，但最核心的是**它很好的解决了版本状态存储（即文件变更过程存储）**的问题。

**3、Git的实现原理**

我们说到，Git很好的解决了版本状态记录的问题，在此基础上实现了版本切换、差异比较、分支管理、分布式协作等等炫酷功能。那么，这一节我们就先从最根本的讲起，看看Git是如何解决版本状态记录（即文件变更过程记录）问题的。

我们都有版本记录的经验，比如在文档撰写的关键点上保留一个备份，或在需要对文件进行修改的时候“另存”一次。这都是很好的习惯，也是版本状态记录的一种常用方式。事实上，Git采取了差不多的方式。

在我们向Git系统提交一个版本的时候，Git会把这个版本完整保存下来。这是不是和“另存”有异曲同工之妙呢？不同之处在于存储方式，在Git系统中一旦一个版本被提交，那么它就会被保存在“Git数据库”中。

**3.1 Git数据库**

我们提到了“Git数据库”，这是什么玩意儿呢？为了能够说清楚Git数据库的概念，我们暂且引入三个Git指令，通过这三个命令，我们就能一探git数据库的究竟。

* git init  用于创建一个空的git仓库，或重置一个已存在的git仓库
* git hash-object  git底层命令，用于向Git数据库中写入数据
* git cat-file  git底层命令，用于查看Git数据库中数据

首先，我们用git init新建一个空的git仓库。（希望小伙伴们可以跟着我的节奏一起来实际操作一下，会加深理解。如果有还没有安装好git工具的同学，请自行百度安装，我不会讲安装的过程。）

我用的是ubuntu系统，在terminal下执行

|  |
| --- |
| $ git init GitTest  Initialized empty Git repository in /home/mp/Workspace/GitTest/.git/ |

这一命令在当前目录下生成了一个新的文件夹-GitTest，在GitTest中，包含了一个新建的空git仓库。如果你不明白git仓库是什么，那么可以简单的理解为存放git数据的一个空间，这这个例子中，是“/home/mp/Workspace/GitTest/.git”目录。

接下来，我们看看git仓库的结构是什么样的。执行

|  |
| --- |
| $ cd GitTest  $ ls  $ find .git  .git .git/HEAD .git/config .git/objects .git/objects/info .git/objects/pack .git/refs .git/refs/heads .git/refs/tags .git/hooks .git/hooks/commit-msg.sample .git/hooks/post-update.sample .git/hooks/update.sample .git/hooks/pre-rebase.sample .git/hooks/pre-applypatch.sample .git/hooks/fsmonitor-watchman.sample .git/hooks/applypatch-msg.sample .git/hooks/pre-receive.sample .git/hooks/pre-push.sample .git/hooks/prepare-commit-msg.sample .git/hooks/pre-commit.sample .git/info .git/info/exclude .git/branches .git/description |

我们发现，GitTest目录下，除隐藏目录.git之外，并没有其他文件或文件夹。

我们通过find .git命令查看新生成的空git仓库的结构，会发现其中有一个objects文件夹，这就是git数据库的**存储位置**。

**3.1 Git数据库的写入操作**

紧接着，我们利用git底层命令git hash-object向git数据库中写入一些内容。执行命令：

|  |
| --- |
| $ echo "version 1" | git hash-object -w --stdin  83baae61804e65cc73a7201a7252750c76066a30  $ find .git/objects/ -type f .git/objects/83/baae61804e65cc73a7201a7252750c76066a30 |

"|"表示这是一条通道命令，意思是把“|”前边的命令的输出作为“|”后边命令的输入。git hash-object -w --stdin 的意思是向git数据库中写入一条数据（-w）,这条数据的内容从标准输入中读取（--stdin）。

命令执行后，会返回个长度为40位的hash值，这个hash值是将待存储的数据外加一个头部信息一起做SHA-1校验运算而得的校验和。在git数据库中，它有一个名字，叫做“键值（key）”。相应的，git数据库其实是一个简单的“键值对（key-value）”数据库。事实上，你向该数据库中插入任意类型的内容，它都会返回一个键值。通过返回的键值可以在任意时刻再次检索该内容。

此时，我们再次执行find .git/objects/ -type f命令查看objects目录，会发现目录中多出了一个文件，这个文件存储在以新存入数据对应hash值的前2位命名的文件夹内，文件名为hash值的后38位。这就是git数据库的存储方式，一个文件对应一条内容，就是这么简单直接。

**3.2 Git数据库的查询操作**

我们可以通过git cat-file这个git底层命令查看数据库中某一键值对应的数据。执行

|  |
| --- |
| $ git cat-file -t  83baa  blob  $ git cat-file -p 83baa  version 1 |

其中，-t选项用于查看键值对应数据的类型，-p选项用于查看键值对应的数据内容，83bba为数据键值的简写。

由执行结果可见，所查询的键值对应的数据类型为blob，数据内容为“version 1”。blob对象我们称之为数据对象，这是git数据库能够存储的对象类型之一，后面我们还会讲到另外两种对象分别是树（tree）对象和提交（commit）对象。

截止到这里，你已经掌握了如何向git数据库里存入内容和取出内容。这很简单但是却意义非凡，因为对git数据库的操作正是git系统的核心——git的版本控制功能就是基于它的对象数据库实现的。在git数据库里，存储着纳入git版本管理的所有文件的所有版本的完整镜像。

git这么简单吗？不用怀疑，git就是这么简单，我们已经准确的抓住了它的根本要义——对象数据库。接下来我们会利用git数据库搭建起git的高楼大厦。

**3.3 使用Git跟踪文件变更**

我们明白，所谓跟踪文件变更，只不过是把文件变更过程中的各个状态完整记录下来。

我们模拟一次文件变更的过程，看看仅仅利用git的对象数据库能不能实现“跟踪文件变更”的功能。

首先，我们执行

|  |
| --- |
| $ echo "version 1" > file.txt    $ git hash-object -w file.txt  83baae61804e65cc73a7201a7252750c76066a30 |

我们把文本“version 1”写入file.txt中，并利用git hash-object -w file.txt命令将其保存入数据库中。如果你足够细心，会发现返回的hash键值和利用echo "version 1" | git hash-object -w --stdin写入数据库时是一致的，这很正常，因为我们写入的内容相同。git hash-object命令在只指定-w选项的时候，会把file.txt文件内容写入数据库。

此时，执行

|  |
| --- |
| $ find .git/objects -type f  .git/objects/83/baae61804e65cc73a7201a7252750c76066a30 |

会发现.git/objects目录中，依然只有一个文件。可见，git数据库存储文件时，只关心文件内容，与文件的名字无关。

接下来，我们修改file.txt的内容，执行

|  |
| --- |
| $ echo "version 2" > file.txt  $ git hash-object -w file.txt  1f7a7a472abf3dd9643fd615f6da379c4acb3e3a  $ find .git/objects -type f  .git/objects/1f/7a7a472abf3dd9643fd615f6da379c4acb3e3a .git/objects/83/baae61804e65cc73a7201a7252750c76066a30 |

我们发现，.git/objects下多出了一个文件，这是我们新保存进数据库的file.txt。接下来，我们执行git cat-file搞清楚这两条数据的内容分别是什么。执行

|  |
| --- |
| $git cat-file -p 83baa  version 1  $git cat-file -p 1f7a7a  version 2 |

我们发现，file.txt的变更过程被完整的记录下来了。

当前的file.txt中保存的内容是“version 2”，如果我们想把文件恢复到修改为“version 2”之前的状态，只需执行

|  |
| --- |
| $ cat file.txt version 2  $ git cat-file -p 83baa > file.txt  $ cat file.txt version 1 |

file.txt的内容成功恢复到了修改前的状态，变成了“version 1”。这其实就是版本回滚的实质。

OK，文件变更状态跟踪的道理就是这么简单。

但做到这一步还远远不算完美，至少有以下几方面的问题：

* 第一，无法记录文件名的变化；
* 第二，无法记录文件夹的变化；
* 第三，记忆每一个版本对应的hash值无聊且乏味且不可能；
* 第四，无法得知文件的变更时序；
* 第五，缺少对每一次版本变化的说明。

问题不少，但都是简单的小问题，我们一一解决。

**3.4 利用树对象（tree object）解决文件名保存和文件组织问题**

Git利用树对象（tree object）解决文件名保存的问题，树对象也能够将多个文件组织在一起。

Git通过树（tree）对象将数据（blob）对象组织起来，这很类似于一种文件系统——blob对象对应文件内容，tree对象对应文件的目录和节点。一个树（tree）对象包含一条或多条记录，每条记录含有一个指向blob对象或tree对象的SHA-1指针，以及相应的模式、类型、文件名。

有了树对象，我们就可以将文件系统任何时间点的状态保存在git数据库中，这是不是很激动人心呢？你的一个复杂的项目可能包含成百上千个文件和文件目录，有了树对象，这一切都不是问题。

**创建树对象**

通常，Git根据某一时刻**暂存区**所表示的状态创建并记录一个对应的树对象，如此重复便可以依次记录一系列的树对象。Git的暂存区是一个文件——.git/index。下面，我们通过创建树对象的过程来认识暂存区和树对象。

为了创建一个树对象，我们需要通过暂存一些文件来创建一个**暂存区**。为此我们引入两个命令：

* git update-index     git底层命令，用于创建暂存区
* git ls-files --stage    git底层命令，用于查看暂存区内容
* git write-tree            git底层命令，用于将暂存区内容写入一个树对象

OK，万事俱备，**我们将file.txt的第一个版本放入暂存区**，执行

|  |
| --- |
| $ find .git/index find: ‘.git/index’: No such file or directory $ git update-index --add file.txt $ find .git/index .git/index $ cat .git/index DIRC[���$�;�[���$�;�A���� ���a�Ne�s� rRu                vjfile.txt�݀3%A��,I­� �` $ find .git/objects/ -type f .git/objects/1f/7a7a472abf3dd9643fd615f6da379c4acb3e3a .git/objects/83/baae61804e65cc73a7201a7252750c76066a30 $ git ls-files --stage 100644 83baae61804e65cc73a7201a7252750c76066a30 0    file.txt $ git write-tree 391a4e90ba882dbc9ea93855103f6b1fa6791cf6 $ find .git/objects/ -type f .git/objects/39/1a4e90ba882dbc9ea93855103f6b1fa6791cf6 .git/objects/1f/7a7a472abf3dd9643fd615f6da379c4acb3e3a .git/objects/83/baae61804e65cc73a7201a7252750c76066a30 |

 分析执行结果：

首先，我们注意.git/index文件的变化。在添加file.txt到暂存区前，index文件并不存在，这说明暂存区还没有创建。添加file.txt到暂存区的同时，index文件被创建。

其次，我们看git数据库的变化。我们发现在执行git update-index 之后，git数据库并没有改变，依然是只有两条数据。在执行git write-tree之后，git数据库中多出了一条新的记录，键值为391a4e90ba882dbc9ea93855103f6b1fa6791cf6。

我们执行git cat-file来查看一下多出来的这条记录是什么内容。执行

|  |
| --- |
| $ git cat-file -t 391a4e tree $ git cat-file -p 391a4e 100644 blob 83baae61804e65cc73a7201a7252750c76066a30    file.txt |

由执行结果可见，git数据库中新增加的记录是一个tree对象，该tree对象指向一个blob对象，hash键值为

|  |
| --- |
| 83baae61804e65cc73a7201a7252750c76066a30 |

这一个blob对象是之前我们添加进数据库的。

**以上我们添加了一个已经存在在git数据库中的文件到暂存区，如果我们新建一个未曾保存到git数据库的文件存入暂存区，进而保存为tree对象，会有什么不同吗？**我们试试看。执行

|  |
| --- |
| $ echo "new file" > new $ git update-index --add new $ find .git/objects/ -type f .git/objects/39/1a4e90ba882dbc9ea93855103f6b1fa6791cf6 #tree对象 .git/objects/1f/7a7a472abf3dd9643fd615f6da379c4acb3e3a #blob对象 .git/objects/83/baae61804e65cc73a7201a7252750c76066a30 #blob对象 .git/objects/fa/49b077972391ad58037050f2a75f74e3671e92 #新增的blob对象 $ git write-tree 228e49bb0bf19df94b49c3474f5d4ee55a371fbe #新生成的tree对象键值 $ find .git/objects/ -type f .git/objects/39/1a4e90ba882dbc9ea93855103f6b1fa6791cf6 .git/objects/1f/7a7a472abf3dd9643fd615f6da379c4acb3e3a .git/objects/83/baae61804e65cc73a7201a7252750c76066a30 .git/objects/fa/49b077972391ad58037050f2a75f74e3671e92 .git/objects/22/8e49bb0bf19df94b49c3474f5d4ee55a371fbe #新增的tree对象  $ git ls-files --stage 100644 83baae61804e65cc73a7201a7252750c76066a30 0    file.txt 100644 fa49b077972391ad58037050f2a75f74e3671e92 0    new |

由执行结果我们可以看到，这一次执行update-index之后，和上次不同，git数据库发生了变化，新增加了一条hash键值为“fa49b0”的数据；暂存区中也多出了文件new的信息。

这说明两个问题：

* 如果添加git数据库中尚未存储的数据到暂存区，则在执行update-index的时候，会同时把该数据保存到git数据库。
* 添加文件进入暂存区的操作是追加操作，**之前已经加入暂存区的文件依然存在——很多人会有误区，认为变更提交之后，暂存区就清空了。**

此时，我们查看新添加的树对象，执行

|  |
| --- |
| $ git cat-file -p 228e49 100644 blob 83baae61804e65cc73a7201a7252750c76066a30    file.txt 100644 blob fa49b077972391ad58037050f2a75f74e3671e92    new |

此次write-tree写入数据库的是tree对象包含了两个文件。

**更进一步，我们是否能将一个子文件夹保存到树对象呢？**尝试一下，执行

|  |
| --- |
| $ mkdir new\_dir $ git update-index --add new\_dir error: new\_dir: is a directory - add files inside instead fatal: Unable to process path new\_dir |

**我们发现，无法将一个新建的空文件夹添加到暂存区。**错误提示告诉我们，**应该将文件将文件夹中的文件加入到暂存区（add files inside instead）。**

OK，接下来，我们在新建的文件夹下写入一个文件，再尝试将这一文件加入暂存区。执行

|  |
| --- |
| $ echo "file in new dir" > new\_dir/new $ git update-index --add new\_dir/new $ git ls-files --stage  100644 83baae61804e65cc73a7201a7252750c76066a30 0    file.txt 100644 fa49b077972391ad58037050f2a75f74e3671e92 0    new 100644 138c554a661371c9c40ae62dfb5d51b48b9b3f6b 0    new\_dir/new  $ git write-tree 06564b76e0fcf9f3600fd055265cf2d4c45847a8  $ git cat-file -p 06564b 100644 blob 83baae61804e65cc73a7201a7252750c76066a30    file.txt 100644 blob fa49b077972391ad58037050f2a75f74e3671e92    new 040000 tree 8d6bc0bdbba1d28caf4ee66a125169500080e206    new\_dir |

从执行结果可见，文件夹new\_dir对应一个tree对象。

**至此，在git数据库中，我们可以完整的记录文件的状态、文件夹的状态；并且可以把多个文件或文件夹组织在一起，记录他们的变更过程。我们离一个完善的版本控制系统似乎已经不远了，而这一切实现起来又是如此简单——我们只是通过几个命令操作git数据库就完成了这些功能。**

接下来，我们只要把数据库中各个版本的时序关系记录下来，再把对每一个版本更新的注释记录下来，不就完成了一个逻辑简单、功能强大、操作灵活的版本控制系统吗？

那么，如何记录版本的时序关系，如何记录版本的更新注释呢？这就要引入另一个git数据对象——提交对象（commit object）。

**3.5 利用提交对象（commit object）记录版本间的时序关系和版本注释**

commit对象能够帮你记录什么时间，由什么人，因为什么原因提交了一个新的版本，这个新的版本的父版本又是谁。

git提供了底层命令commit-tree来创建提交对象（commit object），我们需要为这个命令指定一个被提交的树对象的hash键值，以及该提交对象的父提交对象（如果是第一次提交，不需要指定父对象）。

我们尝试将之前创建的树对象提交为commit 对象，执行

|  |
| --- |
| $ git write-tree  cb0fbcc484a3376b3e70958a05be0299e57ab495  $ git commit-tree cb0fbcc -m "first commit"  7020a97c0e792f340e00e1bb8edcbafcc4dfb60f  $ git cat-file 7020a97  tree cb0fbcc484a3376b3e70958a05be0299e57ab495 author john <john@163.com> 1537961478 +0800 committer john <john@163.com> 1537961478 +0800  first commit |

在git commit-tree命令中，-m选项用于指定本次提交的注释。

我们可以很清楚的看到，一个提交对象包含着所提交版本的树对象hash键值，author和commiter，以及修改和提交的时间，最后是本次提交的注释。

其中committer和author是通过git config命令设置的。

接下来，修改某个文件，重新创建一个树对象，并将这一树对象提交，作为项目的第二个提交版本。执行

|  |
| --- |
| $ echo "new version" > file.txt  $ git update-index file.txt  $ git write-tree 848e967643b947124acacc3a2d6c5a13c549231c $ git commit-tree 848e96 -p 7020a97 -m "second commit" e838c8678ef789df84c2666495663060c90975d7 $ git cat-file -p e838c tree 848e967643b947124acacc3a2d6c5a13c549231c parent 7020a97c0e792f340e00e1bb8edcbafcc4dfb60f author john <john@163.com> 1537962442 +0800 committer john <john@163.com> 1537962442 +0800  second commit |

我们可以按照上述步骤，再提交第三个版本。

|  |
| --- |
| $ echo "another version" > file.txt $ git update-index file.txt $ git write-tree 92867fcc5e0f78c195c43d1de25aa78974fa8103 $ git commit-tree 92867 -p e838c -m "third commit" 491404fa6e6f95eb14683c3c06d10ddc5f8e883f $ git cat-file -p 49140 tree 92867fcc5e0f78c195c43d1de25aa78974fa8103 parent e838c8678ef789df84c2666495663060c90975d7 author john <john@163.com> 1537963274 +0800 committer john <john@163.com> 1537963274 +0800    third commit |

提交完三个版本，我们通过git log 查看最近一个提交对象的提交记录

|  |
| --- |
| $ git log 49140 commit 491404fa6e6f95eb14683c3c06d10ddc5f8e883f Author: john <john@163.com> Date:   Wed Sep 26 20:01:14 2018 +0800      third commit  commit e838c8678ef789df84c2666495663060c90975d7 Author: john <john@163.com> Date:   Wed Sep 26 19:47:22 2018 +0800      second commit  commit 7020a97c0e792f340e00e1bb8edcbafcc4dfb60f Author: john <john@163.com> Date:   Wed Sep 26 19:31:18 2018 +0800      first commit |

太神奇了： 就在刚才，我们围绕git数据库，仅凭几个底层数据库操作便完成了一个 Git 提交历史的创建。到此为止，我们已经完全掌握了git的内在逻辑。

接触过git的小伙伴会发现，以上我们用到的这些指令在使用git过程中是用不到的。这是为什么呢？因为git对以上这些指令进行了封装，给用户提供了更便捷的操作命令，如add，commit等。

**每次我们运行 git add 和 git commit 命令时， Git 所做的实质工作是将被改写的文件保存为数据对象，更新暂存区，记录树对象，最后创建一个指明了顶层树对象和父提交的提交对象。 这三种主要的 Git 对象——数据对象、树对象、提交对象——最初均以单独文件的形式保存在 .git/objects 目录下。**

然而，小问题依然存在，截止目前为止，我们对版本和数据对象的操作都是基于hash键值的，这些毫无直观含义的字符串让人很头疼，不会有人愿意一直急着最新提交对应的hash键值的。git不会允许这样的问题存在的，它通过引入“引用（references）”来解决这一问题。

**3.6 Git的引用**

Git的引用（references）保存在.git/refs目录下。git的引用类似于一个指针，它指向的是某一个hash键值。

创建一个引用实在再简单不过。我们只需把一个git对象的hash键值保存在以引用的名字命名的文件中即可。

执行

|  |
| --- |
| $ echo "491404fa6e6f95eb14683c3c06d10ddc5f8e883f" > .git/refs/heads/master $ cat .git/refs/heads/master  491404fa6e6f95eb14683c3c06d10ddc5f8e883f |

就这样，我们便成功的建立了一个指向最新一个提交的引用，引用名为master

在此之前我们查看提交记录需要执行 git log 491404，现在只需执行git log master。

|  |
| --- |
| $ git log 491404 commit 491404fa6e6f95eb14683c3c06d10ddc5f8e883f (HEAD -> master) Author: john <john@163.com> Date: Wed Sep 26 20:01:14 2018 +0800    third commit    commit e838c8678ef789df84c2666495663060c90975d7 Author: john <john@163.com> Date: Wed Sep 26 19:47:22 2018 +0800    second commit    commit 7020a97c0e792f340e00e1bb8edcbafcc4dfb60f Author: john <john@163.com> Date: Wed Sep 26 19:31:18 2018 +0800    first commit $ git log master commit 491404fa6e6f95eb14683c3c06d10ddc5f8e883f (HEAD -> master) Author: john <john@163.com> Date: Wed Sep 26 20:01:14 2018 +0800    third commit    commit e838c8678ef789df84c2666495663060c90975d7 Author: john <john@163.com> Date: Wed Sep 26 19:47:22 2018 +0800    second commit    commit 7020a97c0e792f340e00e1bb8edcbafcc4dfb60f Author: john <john@163.com> Date: Wed Sep 26 19:31:18 2018 +0800    first commit |

结果完全相同。

Git并不提倡直接编辑引用文件，它提供了一个底层命令update-ref来创建或修改引用文件。

echo "491404fa6e6f95eb14683c3c06d10ddc5f8e883f" > .git/refs/heads/master 命令可以简单的写作：

|  |
| --- |
| $ git update-ref refs/heads/master 49140 |

**这基本就是 Git 分支的本质：一个指向某一系列提交之首的指针或引用。**

**4. Git基本原理总结**

Git的核心是它的对象数据库，其中保存着git的对象，其中最重要的是blob、tree和commit对象，blob对象实现了对文件内容的记录，tree对象实现了对文件名、文件目录结构的记录，commit对象实现了对版本提交时间、版本作者、版本序列、版本说明等附加信息的记录。这三类对象，完美实现了git的基础功能：对版本状态的记录。

Git引用是指向git对象hash键值的类似指针的文件。通过Git引用，我们可以更加方便的定位到某一版本的提交。Git分支、tags等功能都是基于Git引用实现的。

作者：马鸣谦

邮箱：[1612557569@qq.com](mailto:1612557569@qq.com) (如有和文章相关的技术问题，欢迎通过邮件探讨，见信必回)

声明：原创文章，转载请注明出处，勿用于商业用途，侵权必究！

分类: [Git系列](https://www.cnblogs.com/mamingqian/category/1315641.html)

标签: [git](https://www.cnblogs.com/mamingqian/tag/git/), [git实现原理](https://www.cnblogs.com/mamingqian/tag/git%E5%AE%9E%E7%8E%B0%E5%8E%9F%E7%90%86/)

[**好文要顶**](javascript:void(0);) [**关注我**](javascript:void(0);) [**收藏该文**](javascript:void(0);) **[https://common.cnblogs.com/images/icon_weibo_24.png](javascript:void(0);)** **[https://common.cnblogs.com/images/wechat.png](javascript:void(0);)**

[https://pic.cnblogs.com/face/1496761/20180927094410.png](http://home.cnblogs.com/u/mamingqian/)

[马鸣谦](http://home.cnblogs.com/u/mamingqian/)  
[关注 - 2](http://home.cnblogs.com/u/mamingqian/followees)  
[粉丝 - 17](http://home.cnblogs.com/u/mamingqian/followers)

[+加关注](javascript:void(0);)

16

0

[»](https://www.cnblogs.com/mamingqian/p/9867697.html)下一篇：[python爬虫实战：利用scrapy，短短50行代码下载整站短视频](https://www.cnblogs.com/mamingqian/p/9867697.html" \o "发布于2018-10-28 22:31)

posted @ 2018-09-27 11:00 [马鸣谦](https://www.cnblogs.com/mamingqian/) 阅读(1875) 评论(13) [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=9711975) [收藏](https://www.cnblogs.com/mamingqian/p/9711975.html)

**评论列表**

[#1楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4077723) 2018-09-27 11:08 [Two\_for\_One](https://www.cnblogs.com/hjyBlogger/" \t "_blank) 

兄弟，看你发了这个，想问下git中已经被overwrite的代码文件如何不在status中显示？  
只能设忽略文件吗？

[支持(0)反对(0)](javascript:void(0);)

[#2楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4077771) 2018-09-27 11:47 [博客园团队](https://www.cnblogs.com/cmt/) 

您好，文中引用的外站图片都无法显示

[支持(1)反对(0)](javascript:void(0);)

[#3楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4077891) 2018-09-27 13:07 [又见嗍风起](http://home.cnblogs.com/u/1479008/) 

学习学习

[支持(0)反对(0)](javascript:void(0);)

[#4楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4078201)[楼主] 2018-09-27 17:15 [马鸣谦](https://www.cnblogs.com/mamingqian/) 

[@](https://www.cnblogs.com/mamingqian/p/9711975.html#4077771) 博客园团队  
[引用](https://www.cnblogs.com/mamingqian/p/9711975.html#4077771)您好，文中引用的外站图片都无法显示  
  
感谢提醒，刚刚开始用博客园，还有点生疏。文章的草稿是在线上笔记中编辑的，编辑完直接复制进来了，因为我本机有线上笔记的登录token，所以自己看的时候图片正常，没有发现这一问题。稍后我会一一修改的。再次感谢提醒。

[支持(0)反对(0)](javascript:void(0);)

[#5楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4078213)[楼主] 2018-09-27 17:20 [马鸣谦](https://www.cnblogs.com/mamingqian/) 

[@](https://www.cnblogs.com/mamingqian/p/9711975.html#4077723) Two\_for\_One  
方便的话可以把命令执行后返回的结果发出来，或者把问题描述的更详细一些，大家一起参详参详。

[支持(0)反对(0)](javascript:void(0);)

[#6楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4078436) 2018-09-27 22:23 [以尹天下](https://www.cnblogs.com/ileson/) 

难得一见的深入文章。支持一下。

[支持(0)反对(0)](javascript:void(0);)

[#7楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4078437) 2018-09-27 22:25 [以尹天下](https://www.cnblogs.com/ileson/) 

我只是想知道，你是从何渠道获取的这些？看源码？

[支持(0)反对(0)](javascript:void(0);)

[#8楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4078557)[楼主] 2018-09-28 08:32 [马鸣谦](https://www.cnblogs.com/mamingqian/) 

[@](https://www.cnblogs.com/mamingqian/p/9711975.html#4078437) 以尹天下  
看官方文档

[支持(0)反对(0)](javascript:void(0);)

[#9楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4078596) 2018-09-28 09:07 [ChinaSpirit](https://www.cnblogs.com/chinaspirit/" \t "_blank) 

mark

[支持(0)反对(0)](javascript:void(0);)

[#10楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4078600) 2018-09-28 09:09 [optimusprimep](http://home.cnblogs.com/u/988382/" \t "_blank) 

支持！

[支持(0)反对(0)](javascript:void(0);)

[#11楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4078717) 2018-09-28 10:15 [此时](https://www.cnblogs.com/cishi/) 

学到了，并留言，让楼主知道，楼主写的文档作用还是很大的。

[支持(0)反对(0)](javascript:void(0);)

[#12楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4079174) 2018-09-28 17:05 [0o一泓秋水o0](https://www.cnblogs.com/yihongqiushui/) 

mark

[支持(0)反对(0)](javascript:void(0);)

[#13楼](https://www.cnblogs.com/mamingqian/p/9711975.html#4079387) 2018-09-28 22:39 [以尹天下](https://www.cnblogs.com/ileson/) 

[@](https://www.cnblogs.com/mamingqian/p/9711975.html#4078557) 马鸣谦  
谢谢回复，正在研究中。感谢指路。