Git内部原理介绍

李朋飞

2020.10.28

Git概念

- **工作区**是对项目的某个版本独立提取出来的内容。 这些从 Git 仓库的压缩数据库中提取出来的文件，放在磁盘上供你使用或修改。
- **暂存区**是一个文件，保存了下次将要提交的文件列表信息，一般在 Git 仓库目录中。 按照 Git 的术语叫做“索引”，不过一般说法还是叫“暂存区”。
- **Git仓库**目录是 Git 用来保存项目的元数据和对象数据库的地方。 这是 Git 中最重要的部分，从其它计算机克隆仓库时，复制的就是这里的数据。

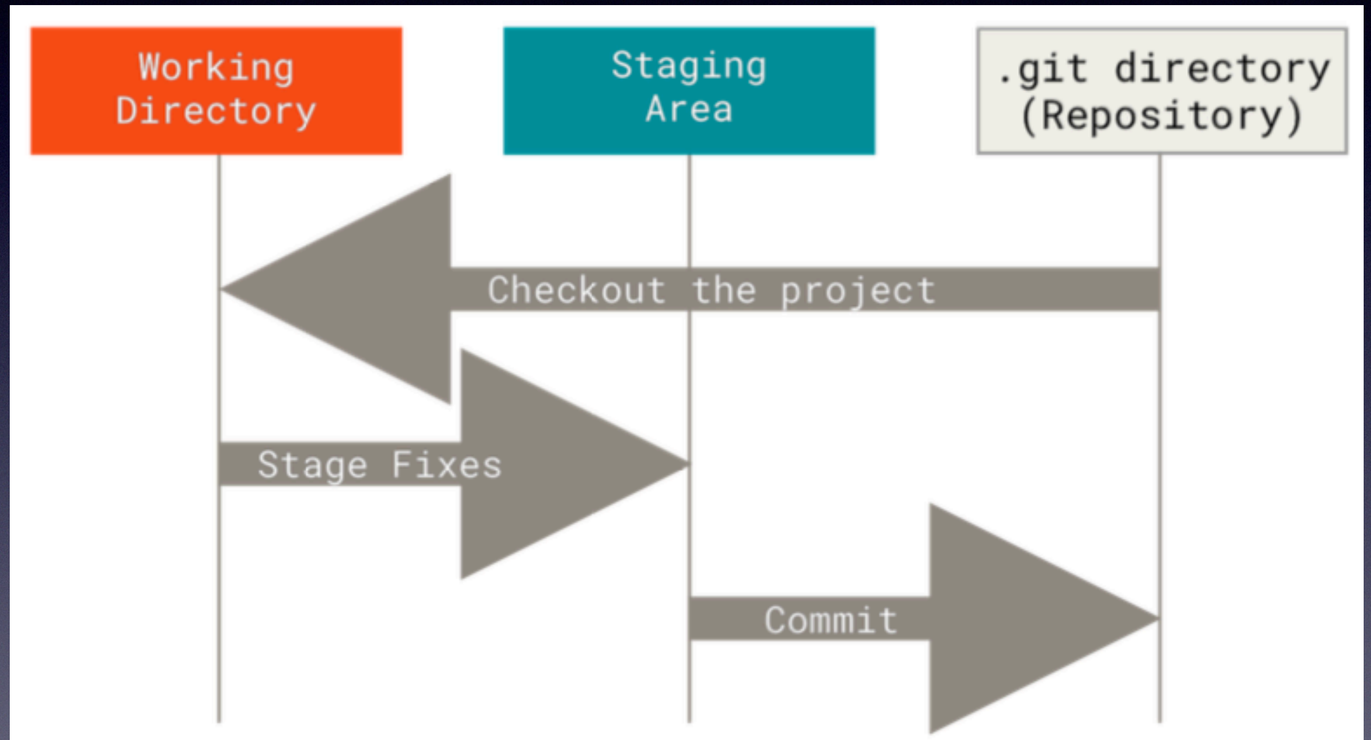


Figure 6. 工作目录、暂存区域以及 Git 仓库.

Git是什么

- Git是一个**内容寻址（content-addressable）的文件系统**，并在此之上提供了一个版本控制系统的界面。

底层命令与上层命令

- 底层（plumbing）
- 上层（porcelain）

底层命令与上层命令

- .git目录
- Git核心组成部分：HEAD、index、object、refs

问题：数据（代码）保存在哪里？

Git对象

- blob object 数据对象
- tree object 树对象
- commit object 提交对象
- tag object 标签对象

Object思想

Git对象 - blob object

- Git核心：键值对数据库（key-value data store）
- 使用 git hash-object 保存数据到.git/objects目录（即对象数据库）
- 使用 git cat-file 查看数据

```
$ echo 'test content' | git hash-object -w --stdin
d670460b4b4aece5915caf5c68d12f560a9fe3e4
```

```
$ find .git/objects -type f
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
```

```
$ git cat-file -p d670460b4b4aece5915caf5c68d12f560a9fe3e4
test content
```

问题：hash-object做了什么？

Git对象 - blob object

- 一个文件不通的版本
- 保存每个版本的完整数据，而非差异

问题：怎么保存文件名?

```
$ echo 'version 1' > test.txt
$ git hash-object -w test.txt
83baae61804e65cc73a7201a7252750c76066a30
```

```
$ echo 'version 2' > test.txt
$ git hash-object -w test.txt
1f7a7a472abf3dd9643fd615f6da379c4acb3e3a
```

```
$ find .git/objects -type f
.git/objects/1f/7a7a472abf3dd9643fd615f6da379c4acb3e3a
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
```

```
$ git cat-file -p 83baae61804e65cc73a7201a7252750c76066a30 > test.txt
$ cat test.txt
version 1
```

```
$ git cat-file -p 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a > test.txt
$ cat test.txt
version 2
```

Git对象 - tree object

- 解决了文件名保存从问题，并将多个文件组织到一起。
- Git 以一种类似于 UNIX 文件系统的方式存储内容，但作了些许简化。 所有内容均以树对象和数据对 象的形式存储，其中树对象对应了 UNIX 中的目录项，数据对象则大致上对应了 inodes 或文件内容。 一个树对 象包含了一条或多条树对象记录(tree entry)，每条记录含有一个指向数据对象或者子树对象的 SHA-1 指针， 以及相应的模式、类型、文件名信息。

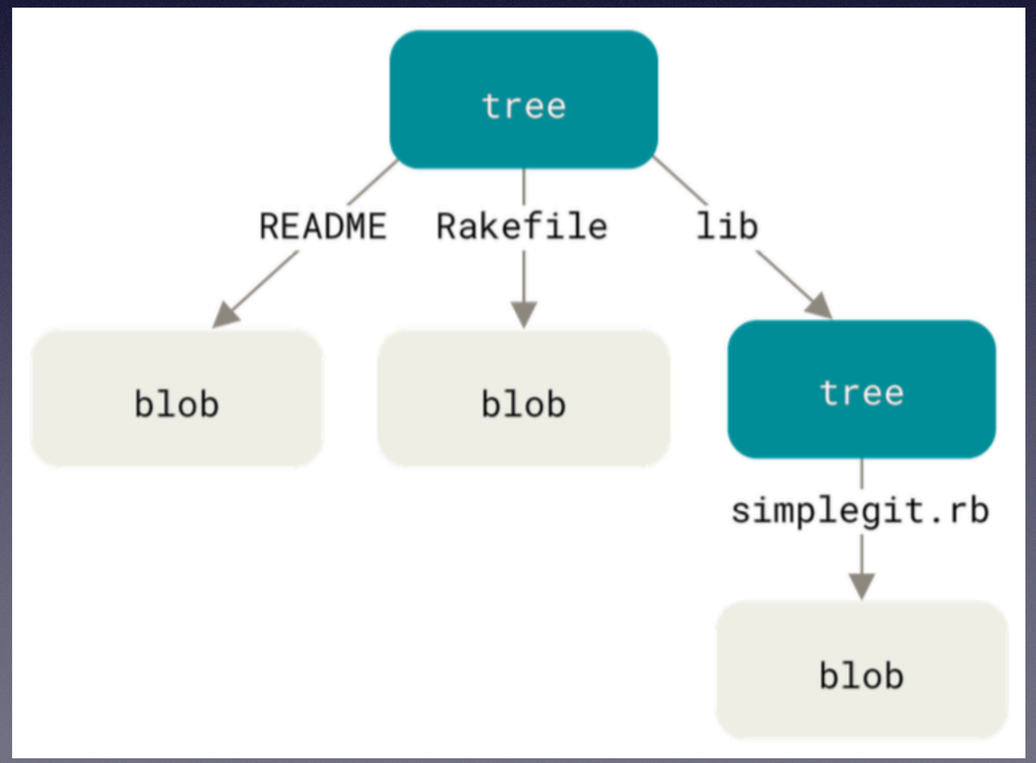
思想：借鉴了Unix文件系统的设计

Git对象 - tree object

- master^{tree} 语法表示 master 分支上最新的提交所指向的树对象
- 文件模式
  - 040000 目录
  - 100644 普通文件
  - 100755 可执行文件
  - 120000 符号链接
  - 160000 子模块
-

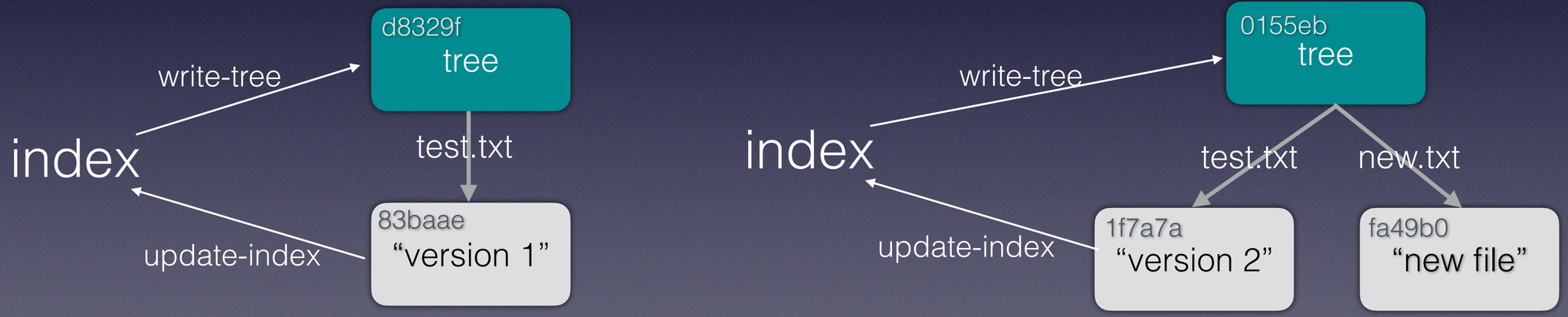
数据对象

```
$ git cat-file -p master^{tree}
100644 blob a906cb2a4a904a152e80877d4088654daad0c859      README
100644 blob 8f94139338f9404f26296befa88755fc2598c289      Rakefile
040000 tree 99f1a6d12cb4b6f19c8655fca46c3ecf317074e0      lib
```

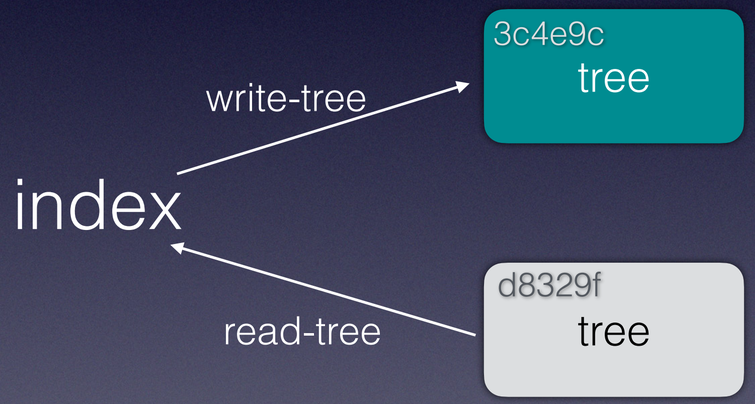


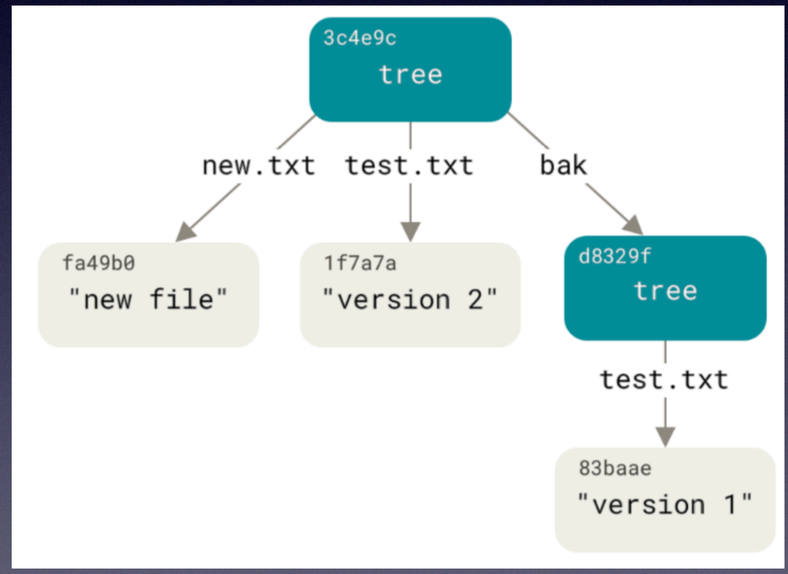
Git对象 - tree obejct

- git update-index —add 将文件加入到暂存区
- git write-tree将暂存区的内容写入一个树对象



Git对象 - tree object





Git对象 - commit object

- 跟踪不同项目的快照，记录每个项目顶层树对象的SHA-1哈希值，保存原因，保存时间等
- 调用commit-tree命令创建提交对象
- 提交对象包含一个树对象的SHA-1，以及该提交的父提交对象（如果有的话）等

```
$ echo 'first commit' | git commit-tree d8329f
fdf4fc3344e67ab068f836878b6c4951e3b15f3d
```

```
$ git cat-file -p fdf4fc3
tree d8329fc1cc938780ffdd9f94e0d364e0ea74f579
author Scott Chacon <schacon@gmail.com> 1243040974 -0700
committer Scott Chacon <schacon@gmail.com> 1243040974 -0700

first commit
```

Git对象 - commit object

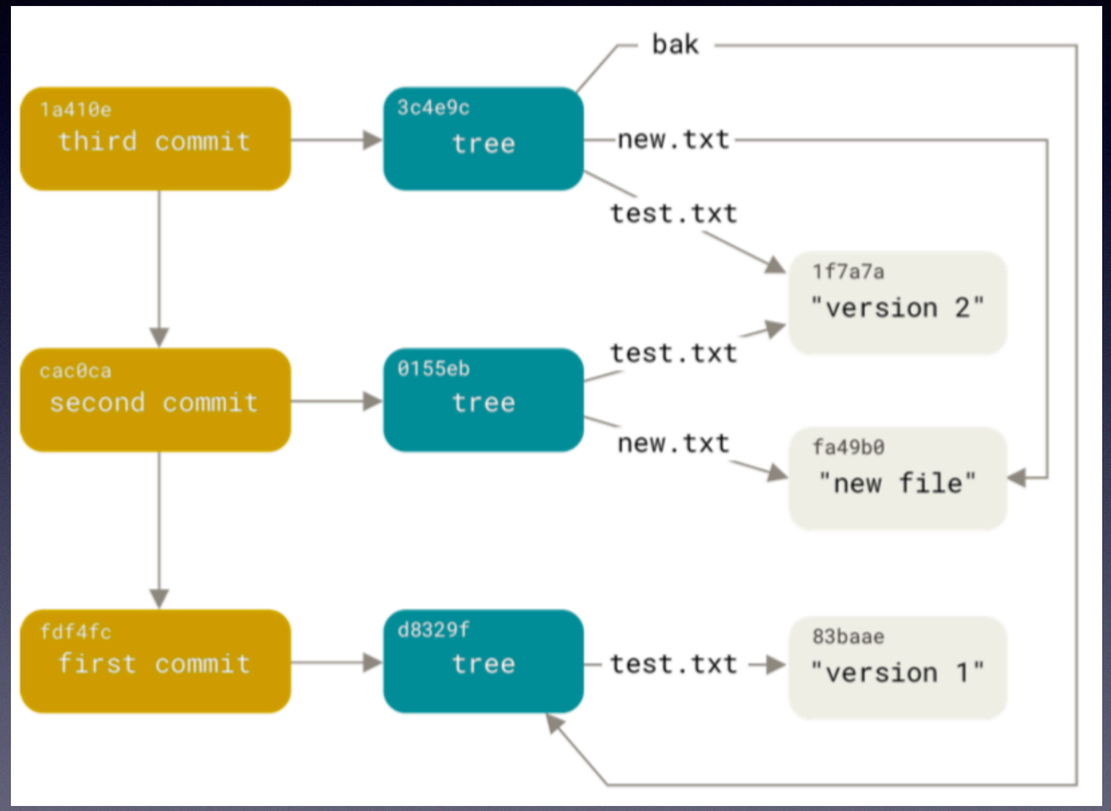
```
$ echo 'second commit' | git commit-tree 0155eb -p fdf4fc3
cac0cab538b970a37ea1e769cbbde608743bc96d
$ echo 'third commit'  | git commit-tree 3c4e9c -p cac0cab
1a410efbd13591db07496601ebc7a059dd55cfe9
```

- 未借助任何上层命令，仅凭几个底层操作，便完成了一个Git提交历史的创建
- 运行git add和git commit命令时，Git所做的工作实质就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。

```
$ git log --stat 1a410e
commit 1a410efbd13591db07496601ebc7a059dd55cfe9
Author: Scott Chacon <schacon@gmail.com>
Date:   Fri May 22 18:15:24 2009 -0700

    third commit

 bak/test.txt | 1 +
 1 file changed, 1 insertion(+)

commit cac0cab538b970a37ea1e769cbbde608743bc96d
Author: Scott Chacon <schacon@gmail.com>
Date:   Fri May 22 18:14:29 2009 -0700

    second commit

 new.txt  | 1 +
 test.txt | 2 +-
 2 files changed, 2 insertions(+), 1 deletion(-)

commit fdf4fc3344e67ab068f836878b6c4951e3b15f3d
Author: Scott Chacon <schacon@gmail.com>
Date:   Fri May 22 18:09:34 2009 -0700

    first commit

 test.txt | 1 +
 1 file changed, 1 insertion(+)
```

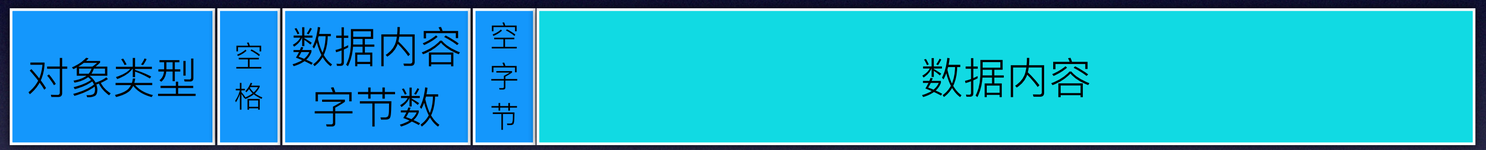
Git对象 - commit object

```
$ find .git/objects -type f
.git/objects/01/55eb4229851634a0f03eb265b69f5a2d56f341 # tree 2
.git/objects/1a/410efbd13591db07496601ebc7a059dd55cfe9 # commit 3
.git/objects/1f/7a7a472abf3dd9643fd615f6da379c4acb3e3a # test.txt v2
.git/objects/3c/4e9cd789d88d8d89c1073707c3585e41b0e614 # tree 3
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30 # test.txt v1
.git/objects/ca/c0cab538b970a37ea1e769cbbde608743bc96d # commit 2
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4 # 'test content'
.git/objects/d8/329fc1cc938780ffdd9f94e0d364e0ea74f579 # tree 1
.git/objects/fa/49b077972391ad58037050f2a75f74e3671e92 # new.txt
.git/objects/fd/f4fc3344e67ab068f836878b6c4951e3b15f3d # commit 1
```



Git对象 - 对象存储

构造对象



计算SHA-1

进行zlib压缩

落盘

- 数据对象的内容几乎可以是任何东西
- 提交对象和树对象的内容却有各自固定的格式

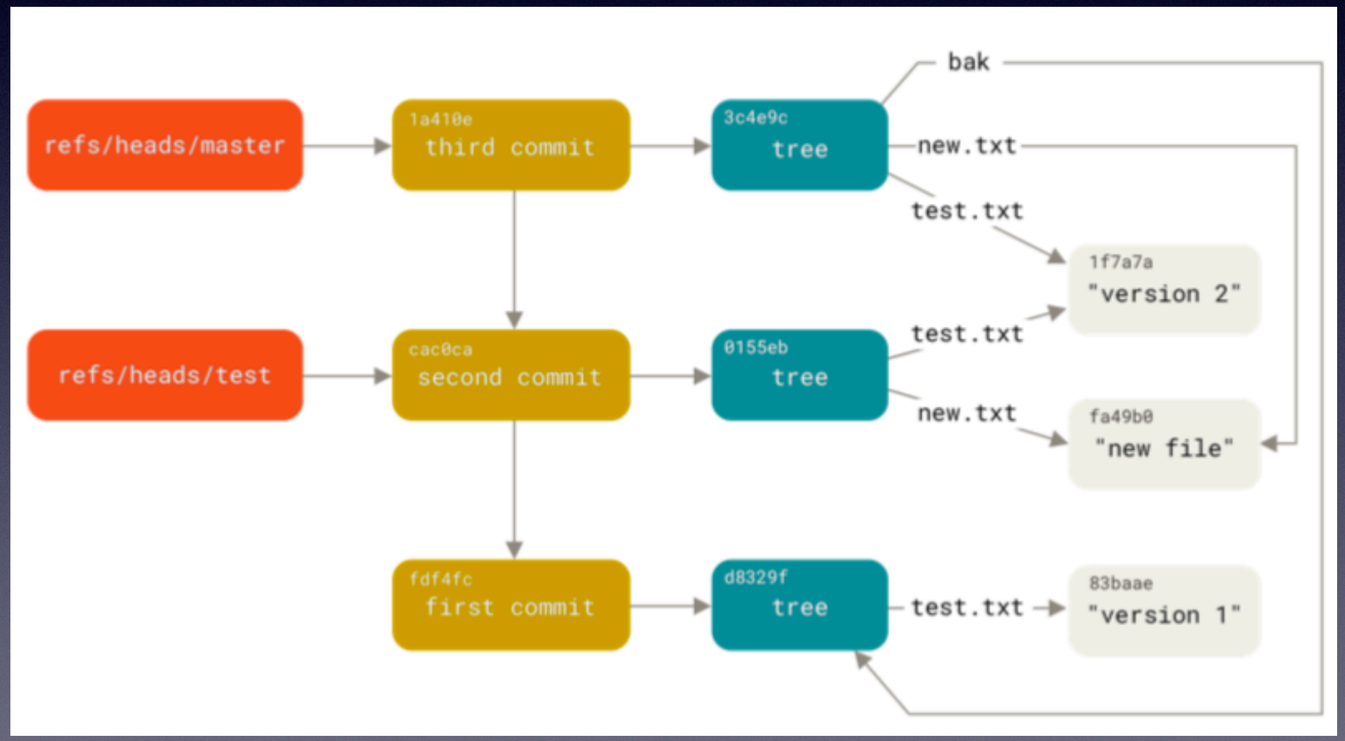
Git引用

- 引用（references，简写为refs）
- 使用名字记录历史，例如：创建一个引用用来帮助记忆最新提交所在的位置（提交对象）
- 引用放置在.git/refs目录下
- git update-ref refs/heads/test cac0ca
- 不要直接编辑引用文件

```
$ find .git/refs
.git/refs
.git/refs/heads
.git/refs/heads/master
.git/refs/heads/dev
.git/refs/tags
.git/refs/tags/V1.0.0
.git/refs/remotes
.git/refs/remotes/origin
.git/refs/remotes/origin/HEAD
.git/refs/remotes/origin/master
.git/refs/remotes/origin/dev
.git/refs/remotes/upstream
.git/refs/remotes/upstream/master
.git/refs/remotes/upstream/dev
```

Git引用

- Git 分支的本质:一个指向某一系列提交之首的指针或引用。



问题：如何知道当前工作分支是哪个?

Git引用 - HEAD引用

- HEAD 文件通常是一个符号引用(symbolic reference)，指向目前所在的分支。
- 在某些罕见的情况下，HEAD 文件可能会包含一个 git 对象的 SHA-1 值。 当你在检出一个标签、提交或远程 分支，让你的仓库变成 “分离 HEAD”状态时，就会出现这种情况。
- 使用git checkout <branch>切换分支，HEAD文件会随之改变，指向目标分支。
- git symbolic-ref HEAD refs/heads/test
- 当执行git commit时，该命令会创建一个提交对象，并用HEAD文件中那个引用所指向的SHA-1值设置 其父提交字段。

Git引用 - 标签引用

- 标签对象（tag object），一般指向提交对象（提交对象指向树对象）
- 像是一个永不移动的 分支引用——永远指向同一个提交对象，只不过给这个提交对象加上一个更友好的名字罢了。
- Git中存在两种类型的标签：**附注标签**和**轻量标签**。

Git引用 - 标签引用

-
```
$ git update-ref refs/tags/v1.0 cac0cab538b970a37ea1e769cbbde608743bc96d
```

- 这就是轻量标签的全部内容——一个固定的引用。然而，一个附注标签则更复杂一些。 若要创建一个附注标 签，Git 会创建一个标签对象，并记录一个引用来指向该标签对象，而不是直接指向提交对象。 可以通过创建一 个附注标签来验证这个过程(使用 -a 选项):

-
```
$ git tag -a v1.1 1a410efbd13591db07496601ebc7a059dd55cfe9 -m 'test tag'
```

```
$ cat .git/refs/tags/v1.1
9585191f37f7b0fb9444f35a9bf50de191beadc2
```

```
$ git cat-file -p 9585191f37f7b0fb9444f35a9bf50de191beadc2
object 1a410efbd13591db07496601ebc7a059dd55cfe9
type commit
tag v1.1
tagger Scott Chacon <schacon@gmail.com> Sat May 23 16:48:58 2009 -0700

test tag
```

- 另外要注意的是，标签对象并非必须 指向某个提交对象;你可以对任意类型的 Git 对象打标签。 例如，在 Git 源码中，项目维护者将他们的 GPG 公 钥添加为一个数据对象，然后对这个对象打了一个标签。

Git引用 - 远程引用

- 远程引用（remote reference）
- 如果添加了一个远程版本库并对其执行过 推送操作，Git 会记录下最近一次推送操作时每一个分支所对应的值，并保存在 refs/remotes 目录下。
- 远程引用和分支(位于 refs/heads 目录下的引用)之间最主要的区别在于，远程引用是只读的。 虽然可以 git checkout到某个远程引用，但是Git并不会将HEAD引用指向该远程引用。因此，你永远不能通过 commit 命令来更新远程引用。Git 将这些远程引用作为记录远程服务器上各分支最后已知位置状态的书签来管理。

```
$ git remote add origin git@github.com:schacon/simplegit-progit.git
$ git push origin master
Counting objects: 11, done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 716 bytes, done.
Total 7 (delta 2), reused 4 (delta 1)
To git@github.com:schacon/simplegit-progit.git
  a11bef0..ca82a6d  master -> master
```

```
$ cat .git/refs/remotes/origin/master
ca82a6dff817ec66f44342007202690a93763949
```

包文件

- Git 最初向磁盘中存储对象时所使用的格式被称为“**松散(loose)**”对象格式。
- Git 会时不时地将多个这些对象打包成一个称为“**包文件(packfile)**”的二进制文件，以节省空间和提高效率。
- 悬空 (dangling)
- git verify-pack -v .git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.idx

```
$ find .git/objects -type f
.git/objects/bd/9dbf5aae1a3862dd1526723246b20206e5fc37
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
.git/objects/info/packs
.git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.idx
.git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.pack
```

引用规范

- 运行’git remote add origin https://github.com/schacon/simplegit-progit’会在你仓库中的 .git/config 文件中添加一个小节， 并在其中指定远程版本库的名称(origin)、URL 和一个用于获取操作的 引用规范(refspec):
-
```
[remote "origin"]
    url = https://github.com/schacon/simplegit-progit
    fetch = +refs/heads/*:refs/remotes/origin/*
```
- 引用规范的格式由一个可选的 + 号和紧随其后的 <src>:<dst> 组成， 其中 <src> 是一个模式(pattern)， 代表远程版本库中的引用; <dst> 是本地跟踪的远程引用的位置。 + 号告诉 Git 即使在不能快进的情况下也要 (强制)更新引用。
- 默认情况下，引用规范由git remote add origin命令自动生成，Git获取服务器中refs/heads/下面的 所有引用，并将它写入到本地的 refs/remotes/origin/ 中。

传输协议

- 哑协议：下载
- 智能协议：下载+上传

总结

- 现在，你应该相当了解 Git 在背后都做了些什么工作，并且在一定程度上也知道了 Git 是如何实现的。 本章讨论 了很多底层命令，这些命令比我们在本书其余部分学到的高层命令来得更原始，也更简洁。 从底层了解 Git 的工 作原理有助于更好地理解 Git 在内部是如何运作的，也方便你能够针对特定的工作流写出自己的工具和脚本。

- 作为一套内容寻址文件系统，Git 不仅仅是一个版本控制系统，它同时是一个非常强大且易用的工具。 我们希望 你可以借助新学到的 Git 内部原理相关知识来实现出自己的应用，并且以更高级、更得心应手的方式来驾驭 Git。

Git的创造者和主要维护者

Linus 林纳斯

Junio C Hamano 滨野 纯

https://github.com/gitster

参考

- Pro Git（Git入门）
- 关于Git的主要维护者滨野纯的访谈
- Git 15 周年：当年的分道扬镳，成就了今天的开源传奇