Packfiles

我们再来看一下 test Git 仓库。目前为止，有 11 个对象 ── 4 个 blob，3 个 tree，3 个 commit 以及一个 tag：

$ 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/95/85191f37f7b0fb9444f35a9bf50de191beadc2 # tag

.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 用 zlib 压缩文件内容，因此这些文件并没有占用太多空间，所有文件加起来总共仅用了 925 字节。接下去你会添加一些大文件以演示 Git 的一个很有意思的功能。将你之前用到过的 Grit 库中的 repo.rb 文件加进去 ── 这个源代码文件大小约为 12K：

$ curl http://github.com/mojombo/grit/raw/master/lib/grit/repo.rb > repo.rb

$ git add repo.rb

$ git commit -m 'added repo.rb'

[master 484a592] added repo.rb

3 files changed, 459 insertions(+), 2 deletions(-)

delete mode 100644 bak/test.txt

create mode 100644 repo.rb

rewrite test.txt (100%)

如果查看一下生成的 tree，可以看到 repo.rb 文件的 blob 对象的 SHA-1 值：

$ git cat-file -p master^{tree}

100644 blob fa49b077972391ad58037050f2a75f74e3671e92 new.txt

100644 blob 9bc1dc421dcd51b4ac296e3e5b6e2a99cf44391e repo.rb

100644 blob e3f094f522629ae358806b17daf78246c27c007b test.txt

然后可以用 git cat-file 命令查看这个对象有多大：

$ du -b .git/objects/9b/c1dc421dcd51b4ac296e3e5b6e2a99cf44391e

4102 .git/objects/9b/c1dc421dcd51b4ac296e3e5b6e2a99cf44391e

稍微修改一下些文件，看会发生些什么：

$ echo '# testing' >> repo.rb

$ git commit -am 'modified repo a bit'

[master ab1afef] modified repo a bit

1 files changed, 1 insertions(+), 0 deletions(-)

查看这个 commit 生成的 tree，可以看到一些有趣的东西：

$ git cat-file -p master^{tree}

100644 blob fa49b077972391ad58037050f2a75f74e3671e92 new.txt

100644 blob 05408d195263d853f09dca71d55116663690c27c repo.rb

100644 blob e3f094f522629ae358806b17daf78246c27c007b test.txt

blob 对象与之前的已经不同了。这说明虽然只是往一个 400 行的文件最后加入了一行内容，Git 却用一个全新的对象来保存新的文件内容：

$ du -b .git/objects/05/408d195263d853f09dca71d55116663690c27c

4109 .git/objects/05/408d195263d853f09dca71d55116663690c27c

你的磁盘上有了两个几乎完全相同的 12K 的对象。如果 Git 只完整保存其中一个，并保存另一个对象的差异内容，岂不更好？

事实上 Git 可以那样做。Git 往磁盘保存对象时默认使用的格式叫松散对象 (loose object) 格式。Git 时不时地将这些对象打包至一个叫 packfile 的二进制文件以节省空间并提高效率。当仓库中有太多的松散对象，或是手工调用 git gc 命令，或推送至远程服务器时，Git 都会这样做。手工调用 git gc 命令让 Git 将库中对象打包并看会发生些什么：

$ git gc

Counting objects: 17, done.

Delta compression using 2 threads.

Compressing objects: 100% (13/13), done.

Writing objects: 100% (17/17), done.

Total 17 (delta 1), reused 10 (delta 0)

查看一下 objects 目录，会发现大部分对象都不在了，与此同时出现了两个新文件：

$ find .git/objects -type f

.git/objects/71/08f7ecb345ee9d0084193f147cdad4d2998293

.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4

.git/objects/info/packs

.git/objects/pack/pack-7a16e4488ae40c7d2bc56ea2bd43e25212a66c45.idx

.git/objects/pack/pack-7a16e4488ae40c7d2bc56ea2bd43e25212a66c45.pack

仍保留着的几个对象是未被任何 commit 引用的 blob ── 在此例中是你之前创建的 "what is up, doc?" 和 "test content" 这两个示例 blob。你从没将他们添加至任何 commit，所以 Git 认为它们是 "悬空" 的，不会将它们打包进 packfile 。

剩下的文件是新创建的 packfile 以及一个索引。packfile 文件包含了刚才从文件系统中移除的所有对象。索引文件包含了 packfile 的偏移信息，这样就可以快速定位任意一个指定对象。有意思的是运行 gc 命令前磁盘上的对象大小约为 12K ，而这个新生成的 packfile 仅为 6K 大小。通过打包对象减少了一半磁盘使用空间。

Git 是如何做到这点的？Git 打包对象时，会查找命名及尺寸相近的文件，并只保存文件不同版本之间的差异内容。可以查看一下 packfile ，观察它是如何节省空间的。git verify-pack 命令用于显示已打包的内容：

$ git verify-pack -v \

.git/objects/pack/pack-7a16e4488ae40c7d2bc56ea2bd43e25212a66c45.idx

0155eb4229851634a0f03eb265b69f5a2d56f341 tree 71 76 5400

05408d195263d853f09dca71d55116663690c27c blob 12908 3478 874

09f01cea547666f58d6a8d809583841a7c6f0130 tree 106 107 5086

1a410efbd13591db07496601ebc7a059dd55cfe9 commit 225 151 322

1f7a7a472abf3dd9643fd615f6da379c4acb3e3a blob 10 19 5381

3c4e9cd789d88d8d89c1073707c3585e41b0e614 tree 101 105 5211

484a59275031909e19aadb7c92262719cfcdf19a commit 226 153 169

83baae61804e65cc73a7201a7252750c76066a30 blob 10 19 5362

9585191f37f7b0fb9444f35a9bf50de191beadc2 tag 136 127 5476

9bc1dc421dcd51b4ac296e3e5b6e2a99cf44391e blob 7 18 5193 1 \

05408d195263d853f09dca71d55116663690c27c

ab1afef80fac8e34258ff41fc1b867c702daa24b commit 232 157 12

cac0cab538b970a37ea1e769cbbde608743bc96d commit 226 154 473

d8329fc1cc938780ffdd9f94e0d364e0ea74f579 tree 36 46 5316

e3f094f522629ae358806b17daf78246c27c007b blob 1486 734 4352

f8f51d7d8a1760462eca26eebafde32087499533 tree 106 107 749

fa49b077972391ad58037050f2a75f74e3671e92 blob 9 18 856

fdf4fc3344e67ab068f836878b6c4951e3b15f3d commit 177 122 627

chain length = 1: 1 object

pack-7a16e4488ae40c7d2bc56ea2bd43e25212a66c45.pack: ok

如果你还记得的话, 9bc1d 这个 blob 是 repo.rb 文件的第一个版本，这个 blob 引用了 05408 这个 blob，即该文件的第二个版本。命令输出内容的第三列显示的是对象大小，可以看到 05408 占用了 12K 空间，而 9bc1d 仅为 7 字节。非常有趣的是第二个版本才是完整保存文件内容的对象，而第一个版本是以差异方式保存的 ── 这是因为大部分情况下需要快速访问文件的最新版本。

最妙的是可以随时进行重新打包。Git 自动定期对仓库进行重新打包以节省空间。当然也可以手工运行 git gc 命令来这么做。