10.4 Git Internals - Packfiles

Packfiles

If you followed all of the instructions in the example from the previous section, you should now have a test Git repository with 11 objects — four blobs, three trees, three commits, and one 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 compresses the contents of these files with zlib, and you're not storing much, so all these files collectively take up only 925 bytes. Now you'll add some more sizable content to the repository to demonstrate an interesting feature of Git. To demonstrate, we'll add the `repo.rb` file from the Grit library — this is about a 22K source code file:

```
$ curl https://raw.githubusercontent.com/mojombo/grit/master/lib/grit/repo.rb > repo.rb
$ git checkout master
$ git add repo.rb
$ git commit -m 'added repo.rb'
[master 484a592] added repo.rb
 3 files changed, 709 insertions(+), 2 deletions(-)
 delete mode 100644 bak/test.txt
 create mode 100644 repo.rb
 rewrite test.txt (100%)
```

If you look at the resulting tree, you can see the SHA-1 value that was calculated for your new `repo.rb` blob object:

```
$ git cat-file -p master^{tree}
100644 blob fa49b077972391ad58037050f2a75f74e3671e92      new.txt
100644 blob 033b4468fa6b2a9547a70d88d1bbe8bf3f9ed0d5      repo.rb
100644 blob e3f094f522629ae358806b17daf78246c27c007b      test.txt
```

You can then use `git cat-file` to see how large that object is:

```
$ git cat-file -s 033b4468fa6b2a9547a70d88d1bbe8bf3f9ed0d5
22044
```

At this point, modify that file a little, and see what happens:

```
$ echo '# testing' >> repo.rb
$ git commit -am 'modified repo.rb a bit'
[master 2431da6] modified repo.rb a bit
 1 file changed, 1 insertion(+)
```

Check the tree created by that last commit, and you see something interesting:

```
$ git cat-file -p master^{tree}
100644 blob fa49b077972391ad58037050f2a75f74e3671e92      new.txt
100644 blob b042a60ef7dff760008df33cee372b945b6e884e      repo.rb
100644 blob e3f094f522629ae358806b17daf78246c27c007b      test.txt
```

The blob is now a different blob, which means that although you added only a single line to the end of a 400-line file, Git stored that new content as a completely new object:

```
$ git cat-file -s b042a60ef7dff760008df33cee372b945b6e884e
22054
```

You have two nearly identical 22K objects on your disk (each compressed to approximately 7K). Wouldn't it be nice if Git could store one of them in full but then the second object only as the delta between it and the first?

It turns out that it can. The initial format in which Git saves objects on disk is called a "loose" object format. However, occasionally Git packs up several of these objects into a single binary file called a "packfile" in order to save space and be more efficient. Git does this if you have too many loose objects around, if you run the `git gc` command manually, or if you push to a remote server. To see what happens, you can manually ask Git to pack up the objects by calling the `git gc` command:

```
$ git gc
Counting objects: 18, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (18/18), done.
Total 18 (delta 3), reused 0 (delta 0)
```

If you look in your `objects` directory, you'll find that most of your objects are gone, and a new pair of files has appeared:

```
$ 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
```

The objects that remain are the blobs that aren't pointed to by any commit — in this case, the "what is up, doc?" example and the "test content" example blobs you created earlier. Because you never added them to any commits, they're considered dangling and aren't packed up in your new packfile.

The other files are your new packfile and an index. The packfile is a single file containing the contents of all the objects that were removed from your filesystem. The index is a file that contains offsets into that packfile so you can quickly seek to a specific object. What is cool is that although the objects on disk before you ran the `gc` command were collectively about 15K in size, the new packfile is only 7K. You've cut your disk usage by half by packing your objects.

How does Git do this? When Git packs objects, it looks for files that are named and sized similarly, and stores just the deltas from one version of the file to the next. You can look into the packfile and see what Git did to save space. The `git verify-pack` plumbing command allows you to see what was packed up:

```
$ git verify-pack -v .git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.idx
2431da676938450a4d72e260db3bf7b0f587bbc1 commit 223 155 12
69bcdaff5328278ab1c0812ce0e07fa7d26a96d7 commit 214 152 167
80d02664cb23ed55b226516648c7ad5d0a3deb90 commit 214 145 319
43168a18b7613d1281e5560855a83eb8fde3d687 commit 213 146 464
092917823486a802e94d727c820a9024e14a1fc2 commit 214 146 610
702470739ce72005e2edff522fde85d52a65df9b commit 165 118 756
d368d0ac0678cbe6cce505be58126d3526706e54 tag    130 122 874
fe879577cb8cffcdf25441725141e310dd7d239b tree   136 136 996
d8329fc1cc938780ffdd9f94e0d364e0ea74f579 tree   36 46 1132
deef2e1b793907545e50a2ea2ddb5ba6c58c4506 tree   136 136 1178
d982c7cb2c2a972ee391a85da481fc1f9127a01d tree   6 17 1314 1 \
  deef2e1b793907545e50a2ea2ddb5ba6c58c4506
3c4e9cd789d88d8d89c1073707c3585e41b0e614 tree   8 19 1331 1 \
  deef2e1b793907545e50a2ea2ddb5ba6c58c4506
0155eb4229851634a0f03eb265b69f5a2d56f341 tree   71 76 1350
83baae61804e65cc73a7201a7252750c76066a30 blob   10 19 1426
fa49b077972391ad58037050f2a75f74e3671e92 blob   9 18 1445
b042a60ef7dff760008df33cee372b945b6e884e blob   22054 5799 1463
```

```
033b4468fa6b2a9547a70d88d1bbe8bf3f9ed0d5 blob   9 20 7262 1 \
  b042a60ef7dff760008df33cee372b945b6e884e
1f7a7a472abf3dd9643fd615f6da379c4acb3e3a blob   10 19 7282
non delta: 15 objects
chain length = 1: 3 objects
.git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.pack: ok
```

Here, the `033b4` blob, which if you remember was the first version of your `repo.rb` file, is referencing the `b042a` blob, which was the second version of the file. The third column in the output is the size of the object in the pack, so you can see that `b042a` takes up 22K of the file, but that `033b4` only takes up 9 bytes. What is also interesting is that the second version of the file is the one that is stored intact, whereas the original version is stored as a delta — this is because you're most likely to need faster access to the most recent version of the file.

The really nice thing about this is that it can be repacked at any time. Git will occasionally repack your database automatically, always trying to save more space, but you can also manually repack at any time by running `git gc` by hand.

prev | next