In a git repository, where do your files live?

Hello! I was talking to a friend about how git works today, and we got onto the topic – where does git store your files? We know that it's in your `.git` directory, but where exactly in there are all the versions of your old files?

For example, this blog is in a git repository, and it contains a file called `content/post/2019-06-28-brag-doc.markdown`. Where is that in my `.git` folder? And where are the old versions of that file? Let's investigate by writing some very short Python programs.

git stores files in `.git/objects`

Every previous version of every file in your repository is in `.git/objects`. For example, for this blog, `.git/objects` contains 2700 files.

```
$ find .git/objects/ -type f | wc -l
2761
```

note: `.git/objects` actually has more information than "every previous version of every file in your repository", but we're not going to get into that just yet

Here's a very short Python program (find-git-object.py) that finds out where any given file is stored in `.git/objects`.

```
import hashlib
import sys


def object_path(content):
    header = f"blob {len(content)}\0"
    data = header.encode() + content
    digest = hashlib.sha1(data).hexdigest()
    return f".git/objects/{digest[:2]}/{digest[2:]}"


with open(sys.argv[1], "rb") as f:
    print(object_path(f.read()))
```

What this does is:

- read the contents of the file
- calculate a header (`blob 16673\0`) and combine it with the contents
- calculate the sha1 sum (`e33121a9af82dd99d6d706d037204251d41d54` in this case)
- translate that sha1 sum into a path (`.git/objects/e3/3121a9af82dd99d6d706d037204251d41d54`)

We can run it like this:

```
$ python3 find-git-object.py content/post/2019-06-28-brag-doc.markdown
.git/objects/8a/e33121a9af82dd99d6d706d037204251d41d54
```

jargon: "content addressed storage"

The term for this storage strategy (where the filename of an object in the database is the same as the hash of the file's contents) is "content addressed storage".

One neat thing about content addressed storage is that if I have two files (or 50 files!) with the exact same contents, that doesn't take up any extra space in Git's database – if the hash of the contents is `aabbbbbbbbbbbbbbbbbbbbbbbb`, they'll both be stored in `.git/objects/aa/bbbbbbbbbbbbbbbbbbbb`.

how are those objects encoded?

If I try to look at this file in `.git/objects`, it gets a bit weird:

```
$ cat .git/objects/8a/e33121a9af82dd99d6d706d037204251d41d54
x^A<8D><9B>}s<E3>Ƒ<C6><EF>o|<8A>^Q<9D><EC>ju<92><E8><DD>\<9C><9C>*<89>j<FD>^...
```

What's going on? Let's run `file` on it:

```
$ file .git/objects/8a/e33121a9af82dd99d6d706d037204251d41d54
.git/objects/8a/e33121a9af82dd99d6d706d037204251d41d54: zlib compressed data
```

It's just compressed! We can write another little Python program called `decompress.py` that uses the `zlib` module to decompress the data:

```
import zlib
import sys

with open(sys.argv[1], "rb") as f:
    content = f.read()
    print(zlib.decompress(content).decode())
```

Now let's decompress it:

```
$ python3 decompress.py .git/objects/8a/e33121a9af82dd99d6d706d037204251d41d54
blob 16673---
```

```
title: "Get your work recognized: write a brag document"
date: 2019-06-28T18:46:02Z
url: /blog/brag-documents/
categories: []
---
... the entire blog post ...
```

So this data is encoded in a pretty simple way: there's this `blob 16673\0` thing, and then the full contents of the file.

there aren't any diffs

One thing that surprised me here is the first time I learned it: there aren't any diffs here! That file is the 9th version of that blog post, but the version git stores in the `.git/objects` is the whole file, not the diff from the previous version.

Git actually sometimes also does store files as diffs (when you run `git gc` it can combine multiple different files into a "packfile" for efficiency), but I have never needed to think about that in my life so we're not going to get into it. Aditya Mukerjee has a great post called Unpacking Git packfiles about how the format works.

what about older versions of the blog post?

Now you might be wondering – if there are 8 previous versions of that blog post (before I fixed some typos), where are they in the `.git/objects` directory? How do we find them?

First, let's find every commit where that file changed with `git log`:

```
$ git log --oneline  content/post/2019-06-28-brag-doc.markdown
c6d4db2d
423cd76a
7e91d7d0
f105905a
b6d23643
998a46dd
67a26b04
d9999f17
026c0f52
72442b67
```

Now let's pick a previous commit, let's say `026c0f52`. Commits are also stored in `.git/objects`, and we can try to look at it there. But the commit isn't there! `ls .git/objects/02/6c*` doesn't have any results! You know how we mentioned "sometimes git packs objects to save space but we don't need to worry about it?". I guess now is the time that we need to worry about it.

So let's take care of that.

let's unpack some objects

So we need to unpack the objects from the pack files. I looked it up on Stack Overflow and apparently you can do it like this:

```
$ mv .git/objects/pack/pack-adeb3c14576443e593a3161e7e1b202faba73f54.pack .
$ git unpack-objects < pack-adeb3c14576443e593a3161e7e1b202faba73f54.pack
```

This is weird repository surgery so it's a bit alarming but I can always just clone the repository from Github again if I mess it up, so I wasn't too worried.

After unpacking all the object files, we end up with way more objects: about 20000 instead of about 2700. Neat.

```
find .git/objects/ -type f | wc -l
20138
```

back to looking at a commit

Now we can go back to looking at our commit `026c0f52`. You know how we said that not everything in `.git/objects` is a file? Some of them are commits! And to figure out where the old version of our post `content/post/2019-06-28-brag-doc.markdown` is stored, we need to dig pretty deep into this commit.

The first step is to look at the commit in `.git/objects`.

commit step 1: look at the commit

The commit `026c0f52` is now in `.git/objects/02/6c0f5208c5ea10608afc9252c4a56c1ac1d7e4` after doing some unpacking and we can look at it like this:

```
$ python3 decompress.py .git/objects/02/6c0f5208c5ea10608afc9252c4a56c1ac1d7e4
commit 211tree 01832a9109ab738dac78ee4e95024c74b9b71c27
parent 72442b67590ae1fcbfe05883a351d822454e3826
author Julia Evans <julia@jvns.ca> 1561998673 -0400
committer Julia Evans <julia@jvns.ca> 1561998673 -0400

brag doc
```

We can also get same information with `git cat-file -p 026c0f52`, which does the same thing but does a better job of formatting the data. (the `-p` option means "format it nicely please")

commit step 2: look at the tree

This commit has a **tree**. What's that? Well let's take a look. The tree's ID is `01832a9109ab738dac78ee4e95024c74b9b71c27`, and we can use our `decompress.py` script from earlier to look at that git object. (though I had to remove the `.decode()` to get the script to not crash)

```
$ python3 decompress.py .git/objects/01/832a9109ab738dac78ee4e95024c74b9b71c27
b'tree 396\x00100644 .gitignore\x00\xc3\xf7`$8\x9b\x8dO\x19/\x18\xb7}|\xc7\xce\x8e:h\xad100644 README.m
```

This is formatted in kind of an unreadable way. The main display issue here is that the commit hashes (`\xc3\xf7$8\x9b\x8dO\x19/\x18\xb7}|\xc7\xce\`…) are raw bytes instead of being encoded in hexadecimal. So we see `\xc3\xf7$8\x9b\x8d` instead of `c3f76024389b8d`. Let's switch over to using `git cat-file -p` which formats the data in a friendlier way, because I don't feel like writing a parser for that.

```
$ git cat-file -p 01832a9109ab738dac78ee4e95024c74b9b71c27
100644 blob c3f76024389b8d4f192f18b77d7cc7ce8e3a68ad    .gitignore
100644 blob 7ebaecb311a05e1ca9a43f1eb90f1c6647960bc1    README.md
100644 blob 0f21dc9bf1a73afc89634bac586271384e24b2c9    Rakefile
100644 blob 00b9d54abd71119737d33ee5d29d81ebdcea5a37    config.yaml
040000 tree 61ad34108a327a163cdd66fa1a86342dcef4518e    content <-- this is where we're going next
040000 tree 6d8543e9eeba67748ded7b5f88b781016200db6f    layouts
100644 blob 22a321a88157293c81e4ddcfef4844c6c698c26f    mystery.rb
040000 tree 8157dc84a37fca4cb13e1257f37a7dd35cfe391e    scripts
040000 tree 84fe9c4cb9cef83e78e90a7fbf33a9a799d7be60    static
040000 tree 34fd3aa2625ba784bced4a95db6154806ae1d9ee    themes
```

This is showing us all of the files I had in the root directory of the repository as of that commit. Looks like I accidentally committed some file called `mystery.rb` at some point which I later removed.

Our file is in the `content` directory, so let's look at that tree: `61ad34108a327a163cdd66fa1a86342dcef4518e`

commit step 3: yet another tree

```
$ git cat-file -p 61ad34108a327a163cdd66fa1a86342dcef4518e

040000 tree 1168078878f9d500ea4e7462a9cd29cbdf4f9a56    about
100644 blob e06d03f28d58982a5b8282a61c4d3cd5ca793005    newsletter.markdown
040000 tree 1f94b8103ca9b6714614614ed79254feb1d9676c    post <-- where we're going next!
100644 blob 2d7d22581e64ef9077455d834d18c209a8f05302    profiler-project.markdown
040000 tree 06bd3cee1ed46cf403d9d5a201232af5697527bb    projects
040000 tree 65e9357973f0cc60bedaa511489a9c2eeab73c29    talks
040000 tree 8a9d561d536b955209def58f5255fc7fe9523efd    zines
```

Still not done…

commit step 4: one more tree….

The file we're looking for is in the `post/` directory, so there's one more tree:

```
$ git cat-file -p 1f94b8103ca9b6714614614ed79254feb1d9676c
.... MANY MANY lines omitted ...
100644 blob 170da7b0e607c4fd6fb4e921d76307397ab89c1e    2019-02-17-organizing-this-blog-into-categories
100644 blob 7d4f27e9804e3dc80ab3a3912b4f1c890c4d2432    2019-03-15-new-zine--bite-size-networking-.mark
100644 blob 0d1b9fbc7896e47da6166e9386347f9ff58856aa    2019-03-26-what-are-monoidal-categories.markdow
100644 blob d6949755c3dadbc6fcbdd20cc0d919809d754e56    2019-06-23-a-few-debugging-resources.markdown
100644 blob 3105bdd067f7db16436d2ea85463755c8a772046    2019-06-28-brag-doc.markdown <-- found it!!!!!
```

Here the `2019-06-28-brag-doc.markdown` is the last file listed because it was the most recent blog post when it was published.

commit step 5: we made it!

Finally we have found the object file where a previous version of my blog post lives! Hooray! It has the hash `3105bdd067f7db16436d2ea85463755c8a772046`, so it's in `git/objects/31/05bdd067f7db16436d2ea85463755c8a772046`.

We can look at it with `decompress.py`

```
$ python3 decompress.py .git/objects/31/05bdd067f7db16436d2ea85463755c8a772046 | head
blob 15924---
title: "Get your work recognized: write a brag document"
date: 2019-06-28T18:46:02Z
url: /blog/brag-documents/
categories: []
---
... rest of the contents of the file here ...
```

This is the old version of the post! If I ran `git checkout 026c0f52 content/post/2019-06-28-brag-doc.markdown` or `git restore --source 026c0f52 content/post/2019-06-28-brag-doc.markdown`, that's what I'd get.

this tree traversal is how `git log` works

This whole process we just went through (find the commit, go through the various directory trees, search for the filename we wanted) seems kind of long and complicated but this is actually what's happening behind the scenes when we run `git log content/post/2019-06-28-brag-doc.markdown`. It needs to go through every single commit in your history, check the version (for example `3105bdd067f7db16436d2ea85463755c8a772046` in this case) of `content/post/2019-06-28-brag-doc.markdown`, and see if it changed from the previous commit.

That's why `git log FILENAME` is a little slow sometimes – I have 3000 commits in this repository and it needs to do a bunch of work for every single commit to figure out if the file changed in that commit or not.

how many previous versions of files do I have?

Right now I have 1530 files tracked in my blog repository:

```
$ git ls-files | wc -l
1530
```

But how many historical files are there? We can list everything in `.git/objects` to see how many object files there are:

```
$ find .git/objects/ -type f | grep -v pack | awk -F/ '{print $3 $4}' | wc -l
20135
```

Not all of these represent previous versions of files though – as we saw before, lots of them are commits and directory trees. But we can write another little Python script called `find-blobs.py` that goes through all of the objects and checks if it starts with `blob` or not:

```
import zlib
import sys

for line in sys.stdin:
    line = line.strip()
    filename = f".git/objects/{line[0:2]}/{line[2:]}"
    with open(filename, "rb") as f:
        contents = zlib.decompress(f.read())
        if contents.startswith(b"blob"):
            print(line)
```

```
$ find .git/objects/ -type f | grep -v pack | awk -F/ '{print $3 $4}' | python3 find-blobs.py | wc -l
6713
```

So it looks like there are `6713 - 1530 = 5183` old versions of files lying around in my git repository that git is keeping around for me in case I ever want to get them back. How nice!

that's all!

Here's the gist with all the code for this post. There's not very much.

I thought I already knew how git worked, but I'd never really thought about pack files before so this was a fun exploration. I also don't spend too much time thinking about how much work `git log` is actually doing when I ask it to track the history of a file, so that was fun to dig into.

As a funny postscript: as soon as I committed this blog post, git got mad about how many objects I had in my repository (I guess 20,000 is too many!) and ran `git gc` to compress them all into packfiles. So now my `.git/objects` directory is very small:

```
$ find .git/objects/ -type f | wc -l
14
```