HOW DO (SOME) VERSION CONTROL SYSTEMS WORK?

GIT

Git is split into two components. The **porcelain** which is the version control system UI that users interact with and the **plumbing** which are the lower level primitives on top of which the **porcelain** sits and the focus of this presentation.

CONTENT ADDRESSABLE FILESYSTEM

Git at it's core is a **content addressable filesystem**.
This means that *files* are *addressed* by their *content* rather than their *name* or *path*

IN PRACTICE

Git stores its **database** at the `.git/objects` directory using a primitive called **object**

There are 4 types of **objects**

- Blobs
- Trees
- Commits
- Tags

HOW ARE THEY STORED?

```
function encode_object(ObjectType object_type, byte[] content) {
    output = make_buffer();
    ouput.write(`${object_type} ${size(content)}\0`);
    output.write(content);
    hash = SHA1_hash(output);
    compressed_output = ZLIB_deflate(output);
    return {hash, compressed_output};
}
```

WHERE ARE THEY STORED?

```
function store_object(ObjectType type, byte[] content) {
    hash, compressed_object = encode_object(type, content);
    mkdir(".git/objects/${hash[0..1]}");
    write(".git/objects/${hash[0..1]}/${hash[2..]}",
compressed_object);
}
```

BLOBS

Blobs are the equivalent of files. They simply contain the content of a file at the time of their creation.

TREES

Trees represent directories. They also function as a way to give a name to a blob object. Trees can also contain other trees. The easiest way to create a tree is to write the current **index***.

COMMITS

The commit object represents a snapshot of the repository at a given time. It contains the SHA-1 of the root tree object, the committers identity, the time the snapshot was created and a message explaining why the snapshot was created.

A commit object can also contain a reference to another commit called the **parent** which is used to implement version control features

TAGS

Tags look a lot like commits. The main difference is that they can point to any object type rather than a tree.

REFERENCES

Having to remember the SHA-1 of a commit in order to operate on it is not convenient so git can assign a human readable name to a commit. Git stores these names in a directory named `.git/refs`

THE HEAD

In order for the git **porcelain** to know on which object to operate on a special file at `.git/HEAD` contains the current active reference.

TAGS (CONT)

There are two types of tags

LIGHTWEIGHT

Which are simply **references** to a commit object

ANNOTATED

These are both a **tag object** and a **reference** pointing to that tag object.

PACKFILES

In order for git to remain performant and not waste a ton of disk space a way to deduplicate the object database is needed.
This is achieved using **packfiles**. A **packfile** is a single file holding the contents of many objects (usually objects that share names or have a lot of content in common). A change to a file is stored as a **delta** in order to save disk space. Any object not in a **packfile** is called a **loose object**.

A **packfile** is accompanied by an **index** file that contains offsets into the packfile so you can quickly seek to a specific object.

THAT'S ALL FOR NOW.

Some other topics you can research on your own:

- `.git` directory layout :: **https://git-scm.com/docs/gitrepository-layout**
- Git index :: **https://git-scm.com/docs/gitformat-index**
- Git reset :: **https://git-scm.com/book/en/v2/Git-Tools-Reset-Demystified**
- Git sparse checkouts :: **https://git-scm.com/docs/sparse-checkout**
- Git worktrees :: **https://git-scm.com/docs/git-worktree**

This presentation was a sum up of the 10th chapter of the git book: ***https://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain***