CHAPTER 10:

Git Internals

First, if it isn’t yet clear, Git is fundamentally a **content-addressable file system** with a VCS user interface written on top of it.

We will cover following Git related concepts in this chapter…

First: Content-addressable file system

Second: Transport mechanisms

Third: Repository maintenance

**First: Content-Addressable File System**

**Plumbing and Porcelain Commands:**

**Plumbing Commands:** Git has bunch of verbs which do low-level work and were designed to be chained together UNIX style or called from scripts. Also this gives us inner workings of Git. Many of these commands aren’t meant to be used manually on the command line, but rather to be used as building blocks for new tools and custom scripts.

**Porcelain Commands:** These are more user friendly commands.

Default .git directory contains following folder structure

Hooks/, refs/, info/, objects/ (All are directories) and config, description and HEAD (all are files)

Considering above directory structure HEAD, index (yet to be created) files, objects and refs directories are core part of Git.

The **objects** directory stores all the content for your database, the **refs** directory stores pointers into commit objects in that data (branches), the **HEAD** file points to the branch you currently have checked out, and the **index** file is where Git stores your staging area information.

**Git Objects:**

Git is a content-addressable file system. Great. What does that mean? **It means that at the core of Git is a simple key-value data store. You can insert any kind of content into it, and it will give you back a key that you can use to retrieve the content again at any time**. To demonstrate, you can use the plumbing command **hash-object**, which takes some data, stores it in your .git directory, and gives you back the key the data is stored as.

Eg. echo 'test content' | git hash-object -w --stdin

This is how Git stores the content initially – **as a single file per piece of content**, named with the SHA-1 checksum of the content and its header.

You can add content to Git and pull it back out again. You can also do this with content in files. For example, you can do some simple version control on a file.

First, create a new file and save its contents in your database:

Eg. echo ‘version 1’ > test.txt | git hash-object –w test.txt

Then, write some new content to the file, and save it again:

Eg. echo ‘version 2’ > test.txt | git hash-object –w test.txt

**Blob Object type:**

You aren’t storing the filename in your system – just the content. This object type is called a **blob.**

**Tree Objects:**

Tree, which solves the problem of storing the filename and also allows you to store a group of files together. Git stores content in a manner similar to a UNIX filesystem, but a bit simplified. All the content is stored as tree and blob objects, with trees corresponding to UNIX directory entries and blobs corresponding more or less to inodes or file contents. A single tree object contains one or more tree entries, each of which contains a SHA-1 pointer to a blob or subtree with its associated mode, type, and filename.