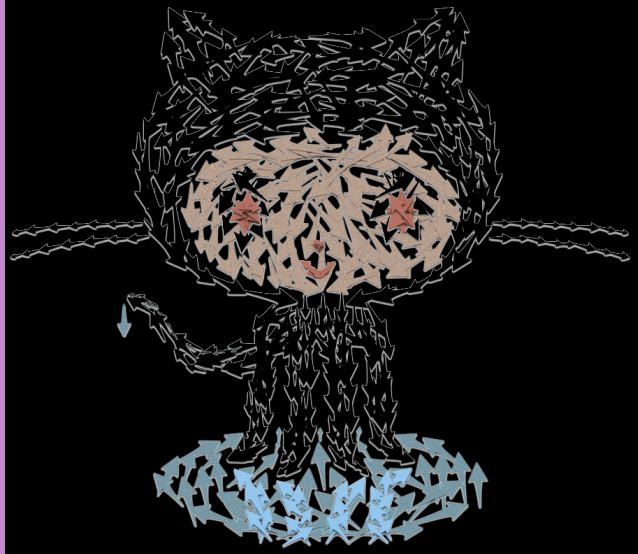
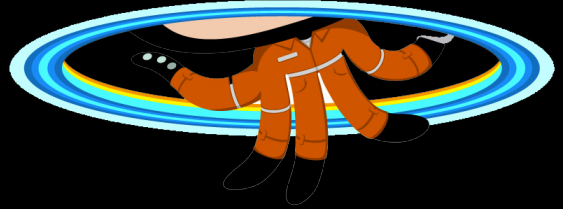




Git Internals

by Jonathan Miedel and Alvin Wang

Last Week on Git

- Midterm Review
- Cherry Picking
- Reflog

What is git internals and why?

- We will look at how Git works at a lower level.
- How Git manages your files and how it creates commits.

- These will give you a higher level of understanding of Git
- “Your ability to fix problems that arise in git will greatly increase” -- Alvin Wang

Historically Low Level

- Early in its history it had a much more complex interface
- Closer to toolkit for VCS's rather than a full fledged VCS in its own right
- git at its base is a content addressable filesystem

Content Addressable Filesystem

- Retrieve files based on content instead of location/path
- Retrieves data using Hash keys
- High-speed storage
- Great for storing files that will not change

Hashing

- Take some data and shorten it to a key
  - The key is a hopefully unique identifier
- Produce random looking key even with small changes
- Can use key to lookup data later
- Should minimize collisions
- Easy to generate
- Hard to invert

SHA-1

- Cryptographic Hash Function
  - cryptographic implies extremely hard to invert
  - Essentially means 1 way
- Most widely used SHA hash function
  - used in SSL and SSH
- Published in 1995

SHA-1 Features

- 20 byte key size
- Key size affects security
  - Being deprecated in very secure applications
  - Wikipedia says you only need 2^69 to build a collision (far less than ideal)
- Merkle Damgard Construction

Hashing Demo

http://www.sha1-online.com/

Hashing in Git

- git uses SHA-1
- git uses hashing to identify and organize blobs
  - Blobs are chunks of files with information regarding chunk size
- Also used to identify commits
  - It is hash of the entire commit object
- Used for consistency not for security
  - Consistency indirectly ensures security

Plumbing vs. Porcelain

- Commands can be grouped into two groups
- Plumbing commands are lower level
  - intended to be used programmatically
  - git cat-file, git hash-object, git update-ref
- Porcelain are higher level
  - intended to be used by normal everyday users
  - git pull, git add, git branch, git bisect

.git Overview

- HEAD
- FETCH_HEAD
- ORIG_HEAD
- config
- description
- hooks/
- index
- info/
- logs
- objects/
- packedrefs/
- refs/

git ls-files

- `git ls-files`
- shows the index in human readable format

How does git stores files and commits

Lets look at the folder

Git Objects

- Git stores pretty much everything in objects
- Objects consist of a type, a size and content
  - types:
    - blob - chunks of binary data
    - tree - similar to directory, references other trees and blobs
    - commit - pointer to a single tree
    - tag - special marking on a commit
- `find .git/objects/` to look at all the objects
  - first two letters become the folder name and the remaining 38 characters are the filename of the object

git show

- `git show -s --pretty=raw <commit hash>`
- allows you to look at detailed commit information

git ls-tree

- git ls-tree <object>
- Displays the tree of the object
  - displays mode type hash path
- Only works on tree objects

git cat-file

git cat-file <object hash>

will show contents of the file

Objects Demo

git show and git ls-tree

Object Types: part 1

Blob Object

5b1d3..

blob	size

```
#ifndef REVISION_H
#define REVISION_H

#include "parse-options.h"

#define SEEN            (1u<<0)
#define UNINTERESTING   (1u
#define TREESAME  (1u<<2)
```

Tree Object

c36d4..

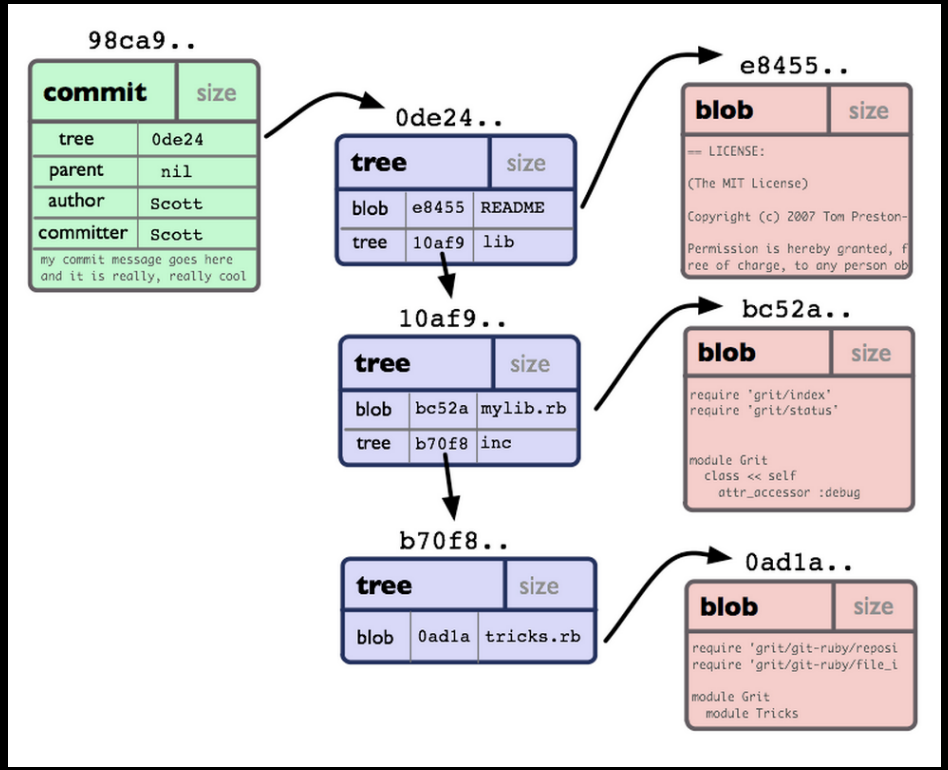
tree		size
blob	5b1d3	README
tree	03e78	lib
tree	cdc8b	test
blob	cba0a	test.rb
blob	911e7	xdiff

Object Types: part 2

Commit Object

Tag Object

Git Object Illustration



http://git-scm.com/book/en/Git-Internals-Git-Objects

Ruby implementation of git file storage

```ruby
def put_raw_object(content, type)
  size = content.length.to_s      <-- Size is one of the 3 components of an object

  header = "#{type} #{size}\0" # type(space)size(null byte)  <-- type is the object type
  store = header + content

  sha1 = Digest::SHA1.hexdigest(store)  <-- hash the header+content
  path = @git_dir + '/' + sha1[0...2] + '/' + sha1[2..40]  <-- creates the path by taking the first two characters as the folder and the last 38 as the file name

  if !File.exists?(path)
    content = Zlib::Deflate.deflate(store)  <-- if file does not exist, compress it using ZLib

    FileUtils.mkdir_p(@directory+'/'+sha1[0...2])
    File.open(path, 'w') do |f|
      f.write content   <-- Write compressed content
    end
  end
  return sha1   <-- return SHA-1 hash
end
```

Example The Commit Object

- All parent object ids
- Author name, email and date
- Committer name and email and the commit time
- Hash of the above

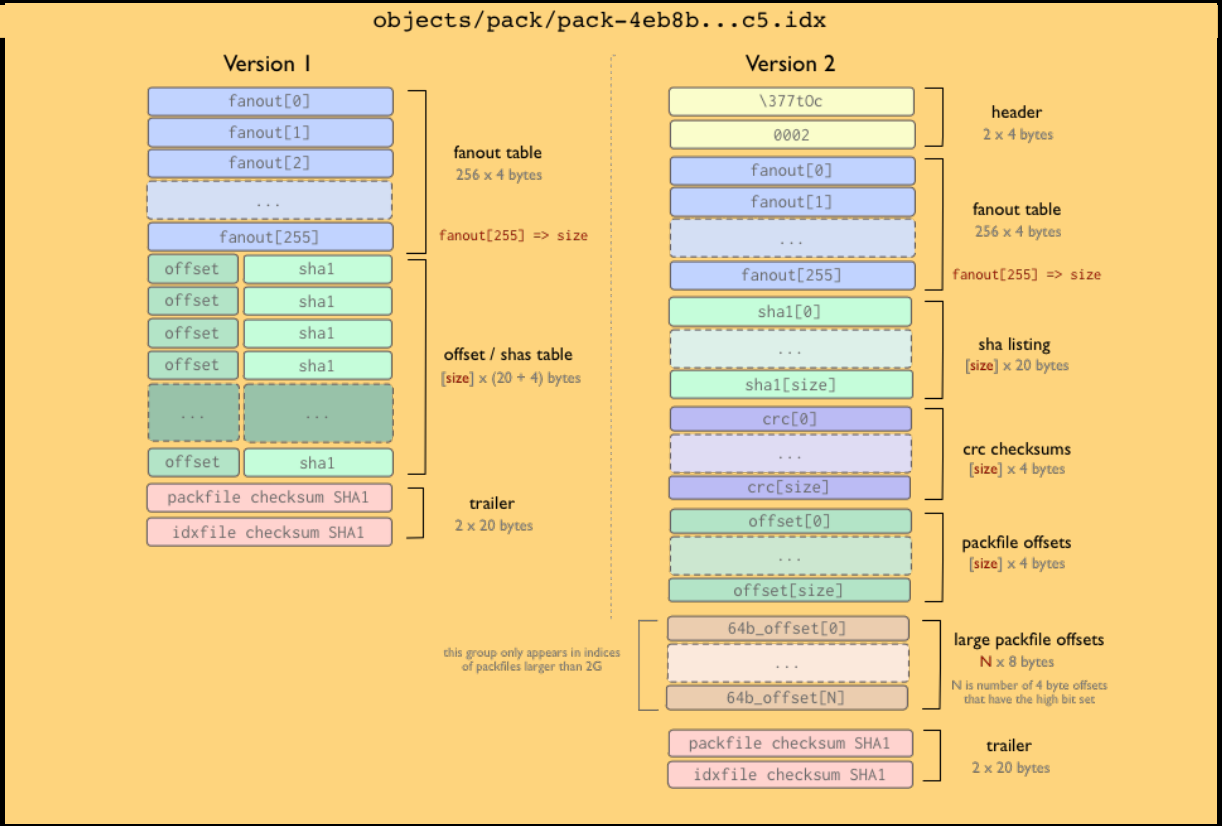
Cool Plumbing Commands

- plumbing command:
  - `git hash-object`
    - takes your data gives you back the hash of it
    - -w stores it into .git as an object
  - git cat-file <hash>
    - takes hash and outputs original information

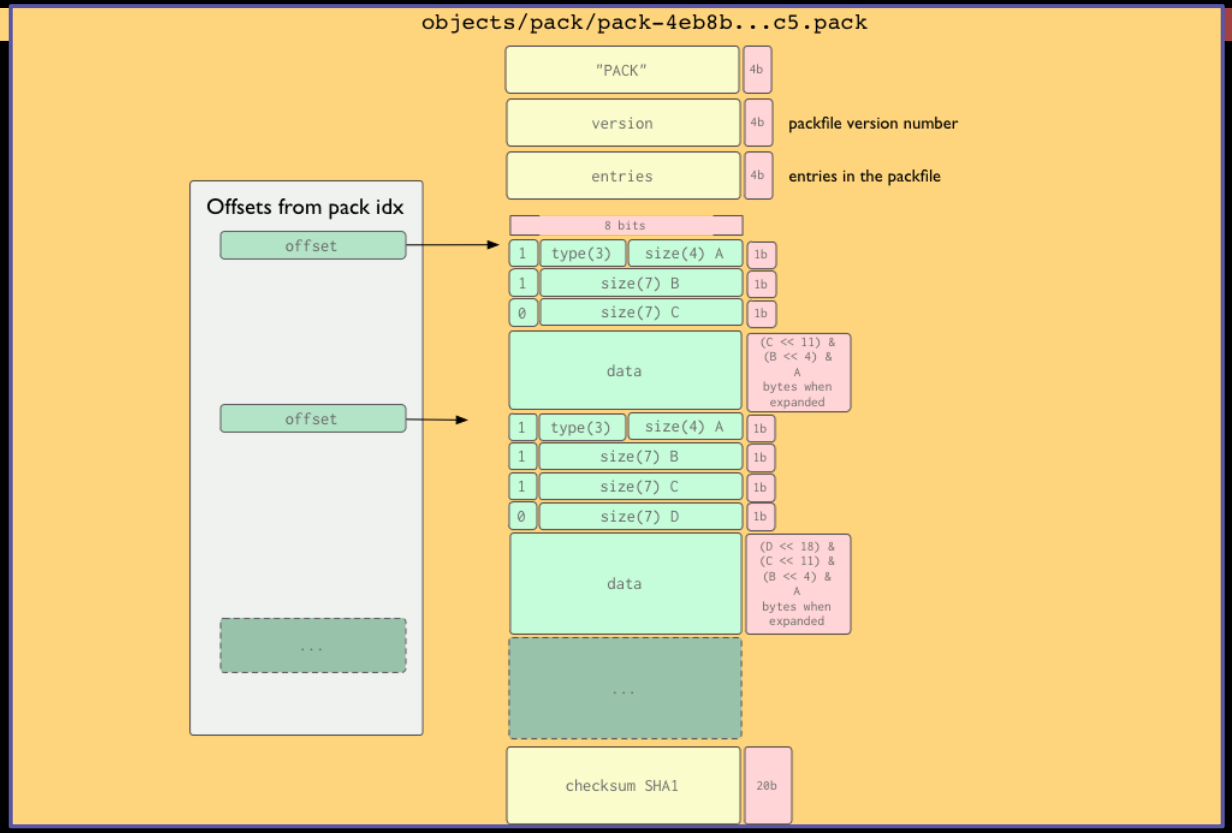
Loose objects + packed objects

- both types are compressed
- loose objects are compressed blobs
- easier and faster to access
- `git gc` - packs the files into packs
- packing algorithm analyzes loose objects to figure out deltas to prevent storing duplicate data

The Packfile Index (idx)



The Packfile



High level Description

- Starts with header
  - Version information, entry number
- List of compressed objects
- Ends in checksum

Compressed Objects

- Header
  - Size, Type
- Data
  - for non-delta objects
    - simply data
  - for delta objects it is the base object
    - base object
    - deltas needed to reconstruct

Packfile Demo

HW 5

- Review this week’s slides
- Short quiz (counts as a HW grade) at the beginning of the next class to reinforce some of these complex topics

Next Week in Git Stuco

- Git Internals (Part 2)