Lecture #32

ıse Study in System and Data-Structure Design

ributed version-control system, apparently the most ıese currently.

, it stores snapshots (*versions*) of the files and directory a project, keeping track of their relationships, authors, g messages.

ıted, in that there can be many copies of a given repository, ing independent development, with machinery to transmit versions between repositories.

ı is extremely fast (as these things go).

A Little History

/ Linus Torvalds and others in the Linux community when r of their previous, propietary VCS (Bitkeeper) withdrew sion.

ıentation effort seems to have taken about 2–3 months, he 2.6.12 Linux kernel release in June, 2005.

ame, according to Wikipedia,

ds has quipped about the name Git, which is British ang meaning "unpleasant person". Torvalds said: "I'm ical bastard, and I name all my projects after myself. ux', now 'git'." The man page describes Git as "the tent tracker."

a collection of basic primitives (now called "plumbing") scripted to provide desired functionality.

-level commands ("porcelain") built on top of these to venient user interface.

Major User-Level Features (I)

is of a graph of versions or snapshots (called *commits*) e project.

ructure reflects ancestory: which versions came from

contains

ry tree of files (like a Unix directory).

on about who committed and when.

ge.

o commit (or commits, if there was a merge) from which it was derived.

Conceptual Structure

l components consist of four types of *object:*

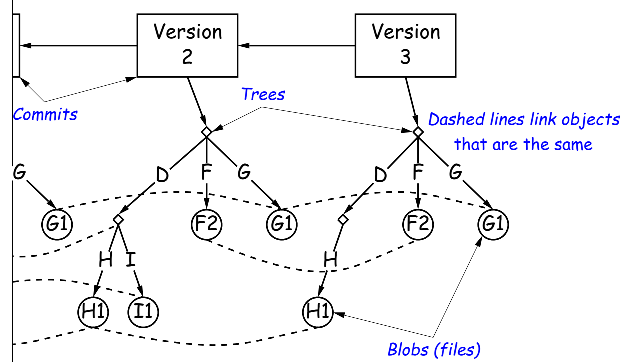
ically hold contents of files.

ectory structures of files.

Contain references to trees and additional information r, date, log message).

ferences to commits or other objects, with additional on, intended to identify releases, other important versions, useful information. (Won't mention further today).

Commits, Trees, Files



Major User-Level Features (II)

has a name that uniquely identifies it to all versions.

can transmit collections of versions to each other.

a commit from repository A to repository B requires nsmission of those objects (files or directory trees) not yet have (allowing speedy updating of repositories).

maintain named *branches*, which are simply identifiers commits that are updated to keep track of the most its in various lines of development.

s are essentially named pointers to particular commits. branches in that they are not usually changed.

The Pointer Problem

it are files. How should we represent pointers between

ble to *transmit* objects from one repository to another nt contents. How do you transmit the pointers?

transfer those objects that are missing in the target How do we know which those are?

counter in each repository to give each object there a But how can that work consistently for two independent ?

How A Broken Idea Can Work

to use a hash function that is so unlikely to have a we can ignore that possibility.

ic Hash Functions have relevant property.

ion, f, is designed to withstand cryptoanalytic attacks. , should have

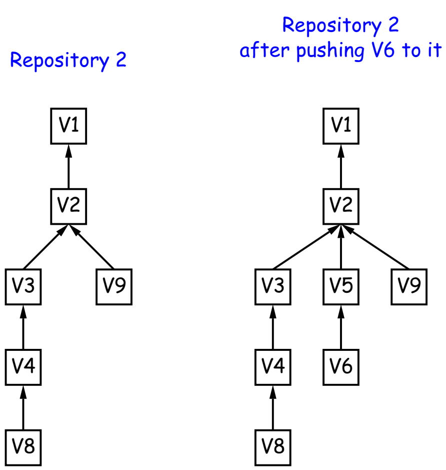
resistance: given $h = f(m)$, should be computationally to find such a message m.

e-image resistance: given message m_1, should be infeasible $\neq m_1$ such that $f(m_1) = f(m_2)$.

esistance: should be difficult to find *any* two messages uch that $f(m_1) = f(m_2)$.

roperties, scheme of using hash of contents as name is likely to fail, even when system is used maliciously.

rsion Histories in Two Repositories



Internals

ository is contained in a directory.

ay either be *bare* (just a collection of objects and r may be included as part of a working directory.

he repository is stored in various *objects* corresponding ther "leaf" content), trees, and commits.

e, data in files is *compressed*.

ge-collect the objects from time to time to save additional

ontent-Addressable File System

e way of naming objects that is universal.

ames, then, as pointers.

Which objects don't you have?" problem in an obvious

, what is invariant about an object, regardless of repository, *ts*.

the contents as the name for obvious reasons.

hash of the contents as the address.

t doesn't work!

: Use it anyway!!

Low-Level Blob Management

out the hashcode that Git uses for the blob containing
ng.java with the command

```
h-object something.java
```

ells you that the file would have hash code

```
0d159f1550b0b5e102f7e06867cc44782
```

ally `git add` this file, its compressed contents will be
: file

```
bjects/19/2a0ca0d159f1550b0b5e102f7e06867cc44782
```

ook at them (uncompressed) with

```
t-file -p 192a0ca0d159f1550b0b5e102f7e06867cc44782
```


SHA1

41 (Secure Hash Function 1).

ınd with this using the `hashlib` module in Python3.

mes in Git are therefore 160-bit hash codes of contents,

commit in the shared CS61B repository could be fetched
vith

```
ckout 3b30599cc43f4616eb626f8fa4fb2d0610d97963
```