Wizbit, a distributed revisioning file system

Rob Taylor and Mark Doffman
Codethink Ltd

Contents

- **Background**
  - File system problems & solutions
- **Goals**
  - What we intend to solve
- **Design**
  - Git internals & implementation
  - Wizbit design
- **Blinding future**
  - How Wizbit may be (mis)used in the future

Background

What's the problem?

- **We all now have lots of devices**
  - Computers
  - Web services
  - Phones
  - Tivos
  - Tablets
  - ....

What's the problem??

- **All of these have their own file/data stores**
  - You need to regularly make decisions about what should be on what.
  - You need to transfer data between the devices
  - Things get out of sync, especially when it's something you're working on. Especially if you're working on it with other people.

What's the problem???

- Sync – FAIL!
- Collaboration – FAIL!
- Backup – FAIL! (ish..)

Doing data sharing now

- Flickr
- E-mailing file versions
- Shared drives
- Google docs
- Google file system (Wuala)
- SVN
- Git / DVCS (Distributed Revision Control Systems)

Problems

- Flickr / Google Docs and DVCS are application specific.
- E-Mailing revisions about is incredibly frustrating.
- Google file system / Shared drives do not allow off-line collaboration.

Device data syncronisation

- OpenSync
- Conduit
- Multiple commercial software packages, allwaysync, GoodSync.
- iTunes

Problems

- Limited data formats, proprietary or lots of failure cases
- Two-way sync is hard.

Backup

- Files are not automatically backed up leading to data loss.
- Many enterprise solutions.
- rsync and dirvish may provide complete solution.
- Lack of open-source GUI.

Where is my data?

- User experience should be that you have the same file store on all devices
- Perhaps different views onto the same file store
- Media player gets to see your media files
- Desktop sees all
- Flickr sees pictures
- BUT it should be the same file store!

So?

- We need a distributed file system
- Must still be fully usable when disconnected
- Must cope with conflicting changes

Conflicts

- Most existing distributed file systems don't allow conflicts – no disconnected operation possible
- Those that do resolve conflicts by a complicated (and error prone) system of rules when the file system syncs (e.g. Coda)

Conflicts, fixed..

- **We need conflict resolution to involve the user and the application.**
  - Only the applications fully understand their data formats
  - Some merge cases can only be resolved by the user telling the application what is right.
- **When to resolve a conflict should be the choice of the user.**
  - Don't force users to make decisions when they don't need to.
  - Don't get in the way of the workflow

Collaboration

- If we can solve this, we get free collaboration tools.

What does the solution look like?

- **Looks a bit like a DVCS (Distributed Revision Control System)**
  - Branching history
  - Merges on request
  - Store synchronization without changing working tree or even necessarily the working branch

What doesn't fit?

- **DVCSs have the luxury of understanding what kind of data they are storing**
  - Text files where unit of merge is a line
- **In a DVCS you merge a whole tree**
  - For a user file system the user has to be able to merge individual files and leave other conflicts unresolved.
- **User interface is not suitable, not even for DVCS ;-)**
  - (ok, maybe with Bazaar..)

Wizbit Design

Attempt 1: Git wrapper design

- Each file referenced by UUID
- Each file stored in separate git repository
- XML storage of the directory structure
- Python implementation using Popen and git commands

Attempt 1: FAIL

- **Git issues**
  - Whole tree merge was unsuitable – Separate repository per file.
  - Git repository per file was heavy
  - Directory structure is not versioned
- **Implementation issues**
  - XML storage of file information was heavy and error prone
  - Python and Popen were error prone
  - Language bindings and library impossible
  - Directory file system is the only application

Attempt 2

- **Move core Git functionality to reusable library**
  - All of Git UI not wanted.
- **Create new directory structure**
  - Uses git object and pack format, but rewrite the storage of references.
- **Build file system on-top of this**
  - File system is a particular application of a versioned collection of files.
- **Git chosen for speed and language choice**
  - 'C' library would make for easier language bindings

Git Overview

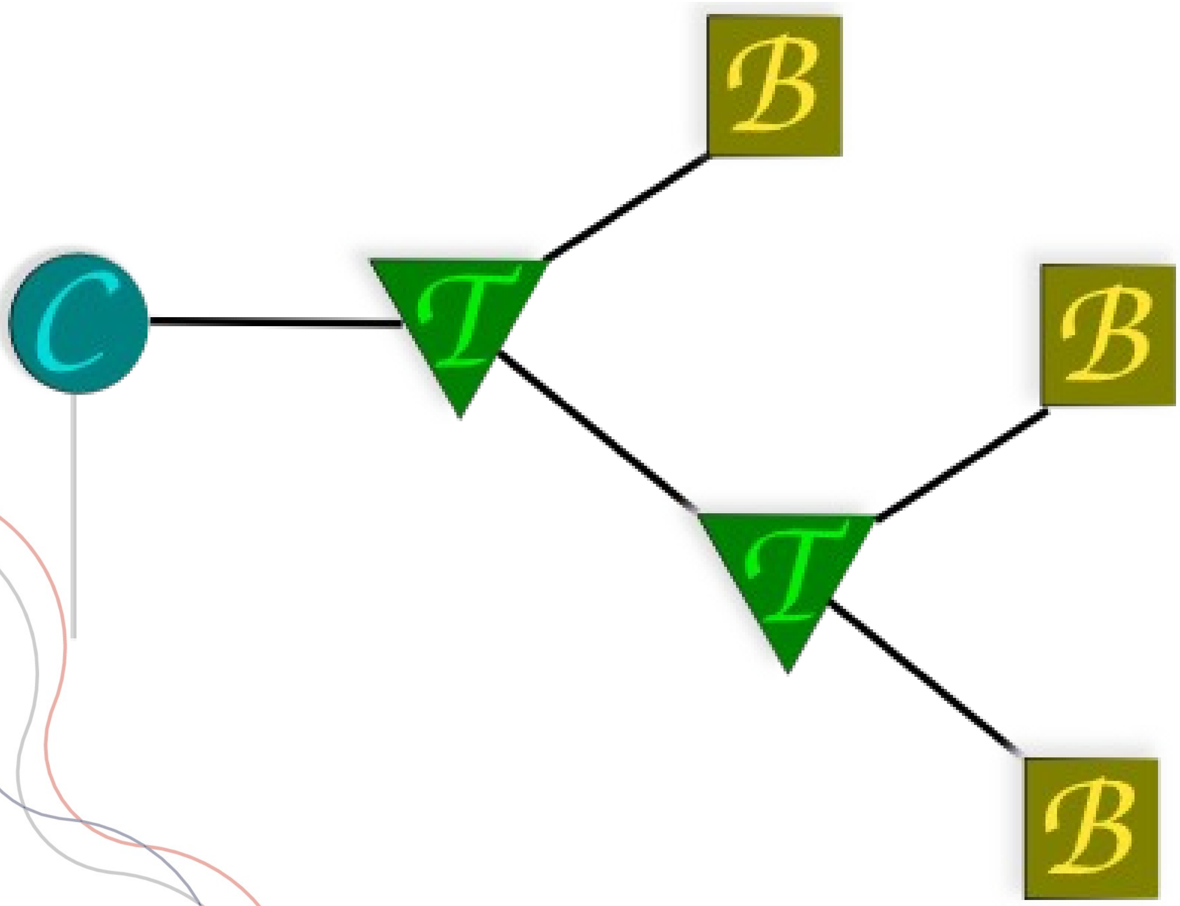
Git design

- **Git objects are files referenced by their own hash**
- **They are stored compressed**
- **They may contain data or meta-data**
  - Meta-data in this case is version history.

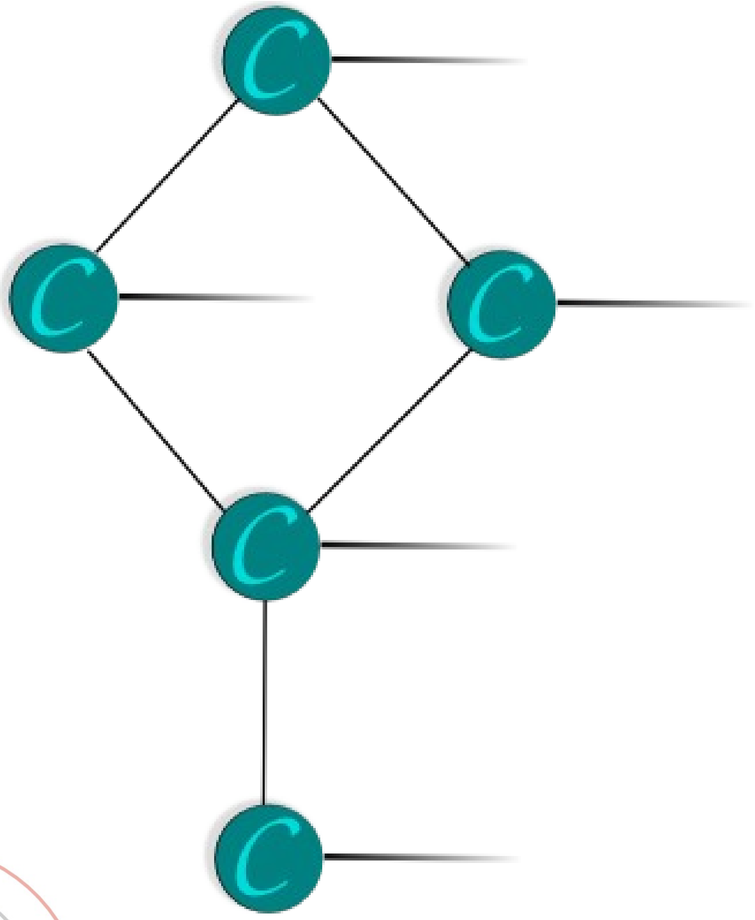
Git object types

- **Blob**
  - File data objects
- **Tree**
  - Store directory data a list of references to blob or tree objects.
- **Commit**
  - Represent a commit including a commit message, time, references to parent commits, and a single tree object.
- **Tag**
  - Used to implement Git tags, we won't worry about these.

Git tree – Single commit



Git commit tree



Git refs

- “Entry point” required for commit tree
- Hash reference used as this entry point – The most recent commit

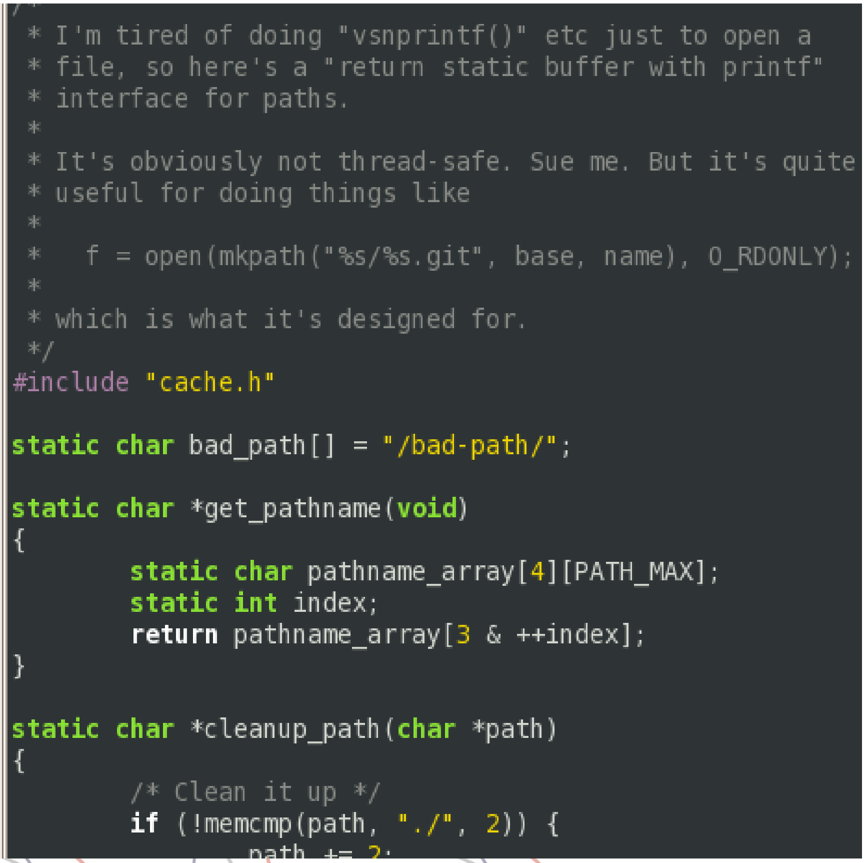
Git design overview

- All objects referenced by their hash.
- A commit object uniquely references and entire directory tree.
- Commit tree used for revision history.
- Files store branch references
- Special file stores merge reference

Git implementation

- Written in 'C' with some bash scripts.
- Optimized to execute in short live process as a command line application.
- An 'evolved' code base that is not obviously modular.
- Mostly not reentrant

Git thread safety



```
/*
 * I'm tired of doing "vsnprintf()" etc just to open a
 * file, so here's a "return static buffer with printf"
 * interface for paths.
 *
 * It's obviously not thread-safe. Sue me. But it's quite
 * useful for doing things like
 *
 *   f = open(mkpath("%s/%s.git", base, name), O_RDONLY);
 *
 * which is what it's designed for.
 */
#include "cache.h"

static char bad_path[] = "/bad-path/";

static char *get_pathname(void)
{
        static char pathname_array[4][PATH_MAX];
        static int index;
        return pathname_array[3 & ++index];
}

static char *cleanup_path(char *path)
{
        /* Clean it up */
        if (!memcmp(path, "./", 2)) {
                path += 2;
```

- **Thread safety not considered in Git.**
- **Good decision for a command line application.**
- **Concurrency provided by larger git design and file locking.**

Git memory management

- Memory cleared by OS – process exits!
- Frequent memory allocations (Objects) stored in ever-growing cache
- Code-base does not consider long-lived processes

libgit.a suitability

None

(oops!)

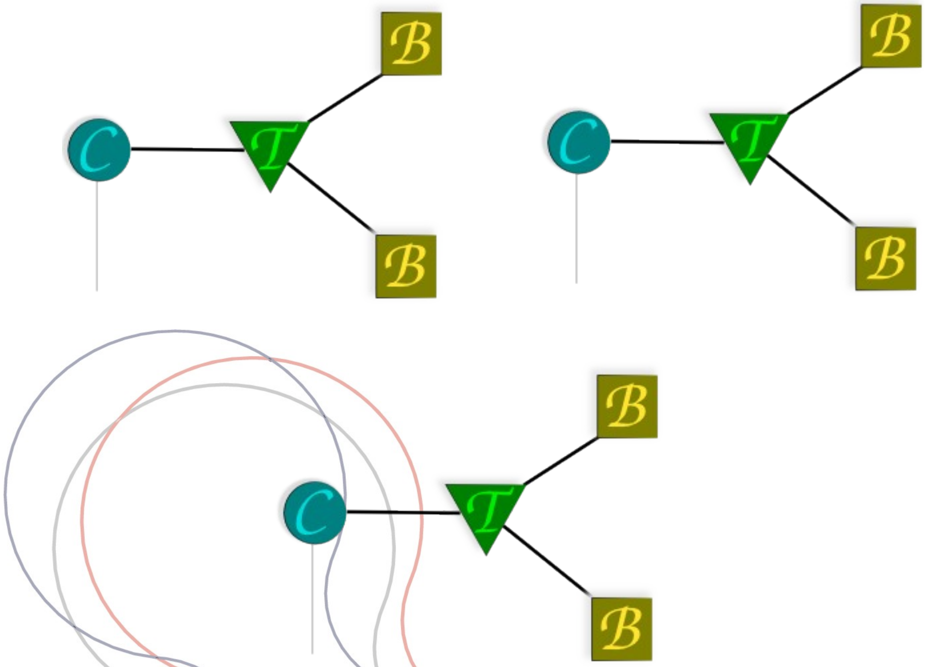
So.... libgitcore

- Reentrant, object-like rewrite of git code functionality (objects, blobs, trees, commits, tags
- Using git code but heavy reworking needed
- Usage is one git object-loader and writer per thread.
- Git code is horrible..

Wizbit design overview

- **Files referenced by UUID**
  - Common feature of distributed file systems
  - Ensures that files created on different computers do not have the same reference
- **Commit tree per file**
  - Each file can be merged separately from the rest of the repository

Wizbit data structures



- **File tree is a standard format**
- **Reduced to two blobs**
  - /data
  - /meta-data

Wizbit directory structure

- Commit refs no longer stored in files
- Git configuration no longer exists
- Object store is the same as git

Refs in Wizbit - Requirements

- **A single head reference per file**
  - The 'work branch'
- **Multiple other references**
  - Usually unmerged tips from other Wizbit instances
- **Multiple references need to be updated within a single transaction**
  - To provide posix semantics, we'll need to update multiple refs atomically.

Wizbit references - Solution!

- Use SQLite for the win, avoiding lots of clever file system locking code.

Difference of two repositories

- Calculating the differences between repositories is vital for syncing
- Checking the head reference of every file is not optimal in the common case of similar repositories
- Git / SVN / Everything else do not have this problem as a reference applies to an entire tree

Difference of two repositories

- Use a branching tree of UUIDs.
- Compare SHA1s at each tree depth
- Quickly find out which refs have changed

Building a file system

- **Options include**
  - Fuse
  - GVFS
- **Choice not important to us – probably do both**
- **Didn't think GVFS had any advantages**

File system structure

- **How do we build a file system on-top of the UUID structure?**
  - Store directories as versioned files in the Wizbit repository
  - Store symlinks as versioned files in the Wizbit repository
  - Store files as versioned files in the Wizbit repository
- **Extremely similar to local file system**
- **UUID could be considered as the inode**

Wizbit design implications

- **Destructively deleting is difficult**
  - Real deleting would require removing commit history.
  - Commit tree modification is nearly impossible on published or shared repositories.
  - For the moment freeing up space will not be supported.
  - Don't use wizbit for your “sensitive data”
- **Checkout of directory revisions is difficult**
  - In the versioned file system directories only point to file uuid rather than file versions, making checking out a directory out at a certain revision non-trivial.

Related Tech

- **Versioned file systems**
  - Andrew FS, CODA, Elephant FS
  - User space, slower but perhaps more flexible
  - Synchronization between repositories built in
- **Log based file systems**
  - User space implementation and commit on close make wizbit unsuitable for the forensics applications of Log based file systems
  - Possibly faster file writing.
  - Possibly better ratio of data to commit data.

Usefulness

- **Not a whole-file system solution**
  - Lack of delete
  - Temporary files to be generally avoided
- **Not a revision control solution**
  - Git / Mercurial / Bazarr much better choices as they are designed to version whole directories + Years of UI development.
- **Intended for personal files, accessed and modified on multiple devices by many people**

Usefulness

- Intended for creating a single file system across multiple devices

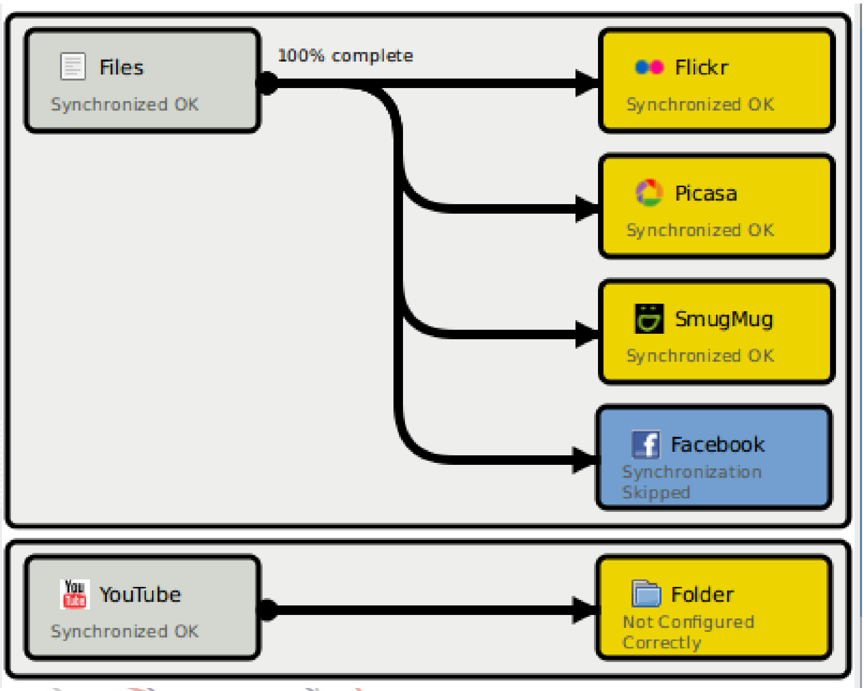
User interface – Local repository

- **File access**
  - Handled by user space file system
- **Committing**
  - When to commit is difficult. Best bet – Commit on close.
- **Version history**
  - Reuse ideas from graphical VCS.
- **Revision checkout**
- **File merging**
  - Application specific and hard. Provide some basics such as “Use most recent file”.

User interface - Syncing

- **Relating devices**
  - Where are we syncing from / to?
- **Sync schedules**
  - When do we sync between devices?
- **Authorization**
  - How do we obtain permission to sync to a device?

Syncing - Conduit



- Aims to integrate into desktop
- Plug and play for your device syncing
- Automatically syncs when something changes
- Already has code to handle authorization against web services

The blinding future!

Meta-data formats

- **In the core design, every file has both data and meta-data.**
- **Not yet decided what form meta-data should take: RDF/XML? Tuples?**
- **Common meta-data:**
  - User visible file name
  - mtime/ctime (atime??!)
  - Document info

Tracker / XESAM

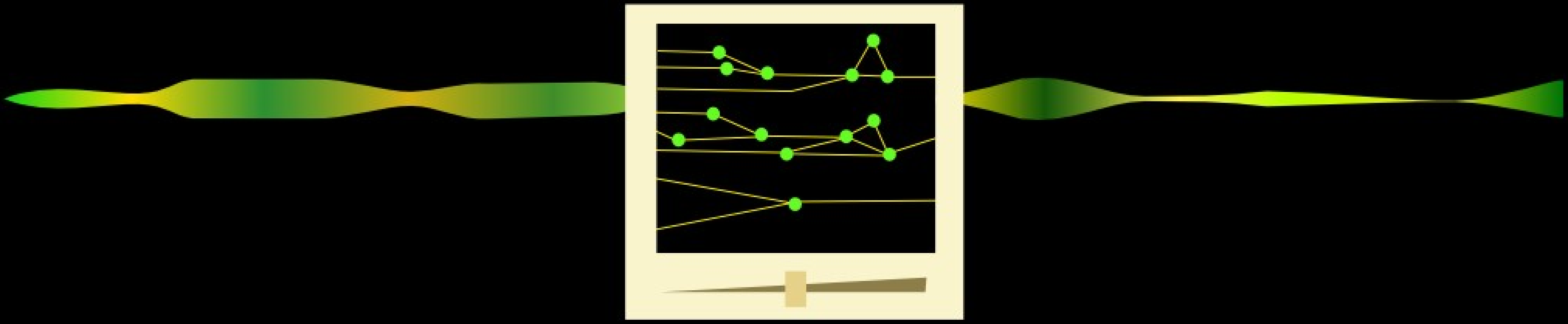
- Initially can use Tracker to create the meta-data
- If Wizbit successfully adopted, applications can create the meta-data themselves and commit along with the data.

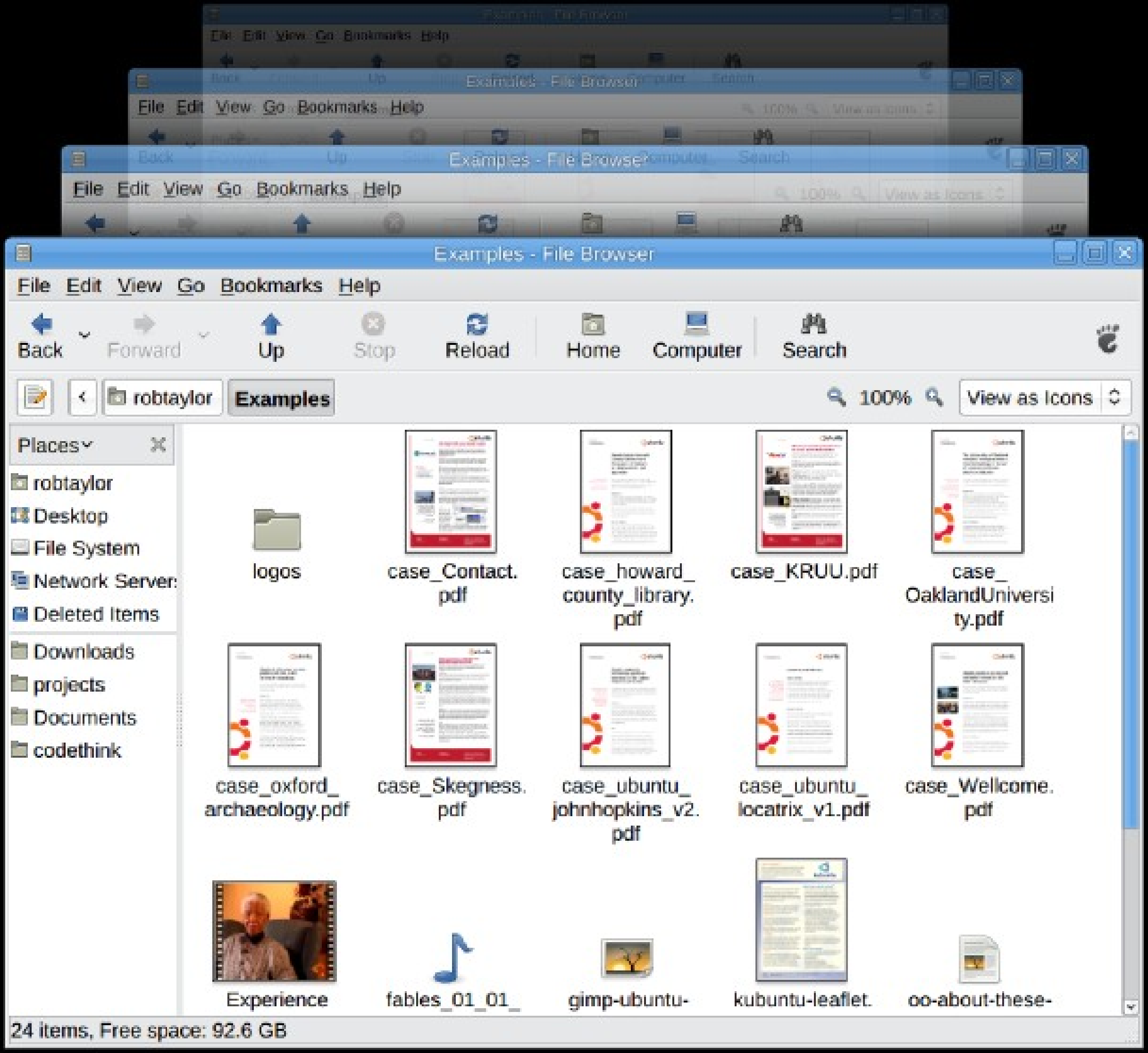
Contextual browsing

- **Given all this and a suitably rich ontology, we can start imagining new ways of reaching our data:**
  - Time line: “The work I was doing last week”
  - Context: “Documents on RDF”
  - Virtual folders with complex stored search terms
  - “Related files”

Metadata

- **Metadata should move with the file system**
  - Shouldn't need to be regenerated for each store
  - What to do about inter-file relations?





Examples - File Browser
File Edit View Go Bookmarks Help
Back
Forward
Up
Stop
Reload
Home
Computer
Search
robtaylor
Examples
100%
View as Icons
Places
robtaylor
Desktop
File System
Network Server:
Deleted Items
Downloads
projects
Documents
codethink
logos
case_Contact.pdf
case_howard_county_library.pdf
case_KRUU.pdf
case_OaklandUniversity.pdf
case_oxford_archaeology.pdf
case_Skegness.pdf
case_ubuntu_johnhopkins_v2.pdf
case_ubuntu_locatrix_v1.pdf
case_Wellcome.pdf
Experience
fables_01_01_
gimp-ubuntu-
kubuntu-leaflet.
oo-about-these-
24 items, Free space: 92.6 GB

Timeline view

- **UI Mockup by Karl Lattimer**
  - Top line shows activity over time – thicker/hotter means more activity
  - Dragging magnifier scrolls time line forwards/backwards
  - Magnifier scale gives detail level
    - Files changing
    - How files changed

More ideas welcome!

Thanks!

- **http://wizbit.org**
  - (not much there yet)
- **http://git.codethink.co.uk**
  - libgitcore
  - wizbit