Luxor

A Performant and User-Friendly File Versioning System

John Wang
6.033 Computer Systems Engineering
Design Project 1 Proposal

Overview

File versioning is an integral feature of computer systems because it allows the user to seamlessly save and restore file history. Current file versioning systems, however, either incur tremendous speed and memory costs or require advanced technical knowledge. In 2011, over 94% of people risked losing their computer data, in part, from the lack of automatic and user-friendly file versioning systems.[1] Tools such as Git and Subversion provide powerful frameworks for saving and replaying work, but are difficult to use and have not caught on outside the developer community. Luxor is a file versioning solution for casual computer users which provides automatic saving and fast access to previous file versions. Luxor focuses on usability while retaining the performance and memory of a less straightforward system.

Design Description

In order to accomplish the dual goals of usability and performance, Luxor adopts an abstraction model which hides the complexity of the versioning system from the user. This section describes the tree of abstraction, starting with high level concepts, like user-facing commands, and ending with the implementation details of the versioning system.

User Commands and Interface

Luxor shall primarily run on UNIX-based operating systems, though its modular design allows it

[1] https://secure.backblaze.com/press/Backblaze_press_June_Backup_2011.pdf

to be adapted to other systems. Luxor shall come as a built-in package on Linux distributions, so that users without significant technical knowledge do not have to worry about installation.

Luxor provides two modes, reverted mode and the active filesystem. The ***luxor revert*** command begins reverted mode, where one can browse previous files. In reverted mode, all non-excluded files are returned to their states at a particular date and time. For example, ***luxor revert '9/15/1995 5:34pm EST'*** returns all non-excluded files to how they looked on September 15, 1995 at 5:34pm EST.

To reduce complexity, no files in reverted mode can be changed. Instead, Luxor provides a method for copying files from reverted mode to the active filesystem. In this way, users cannot modify file history, which leaves the results of ***luxor revert*** consistent. Figure 1 shows an example of the copying mechanism, where a user enters reverted mode and copies a file from version 1 into the active filesystem using ***luxor clone***.

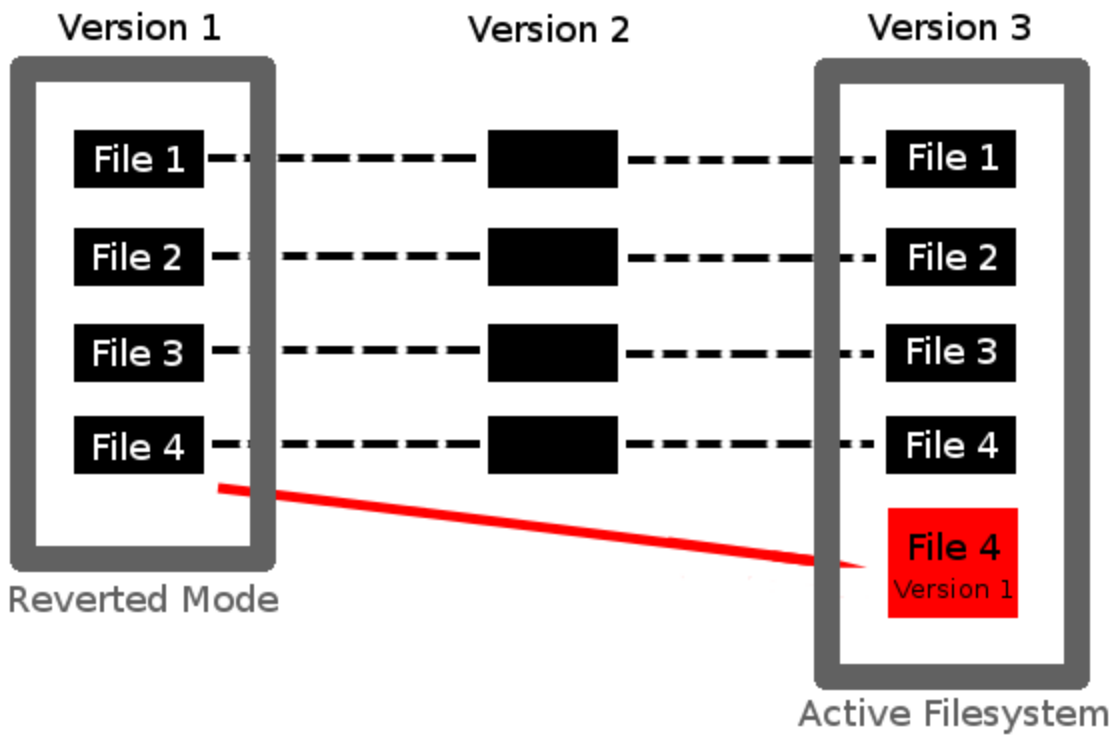

Figure 1. Copying a file from reverted mode into the active filesystem with ***luxor clone***.

Other means of rewriting historical files can be quite convoluted. Branching, for example, starts a new version of history in parallel with the original history. This process, however, leads to a profileration of different histories, which makes it difficult for a user keep track of each branch. Copying files to the active filesystem allows Luxor users to easily obtain previous history without other complications.

Versioning Abstraction

Luxor internally contains entities called versions, which represent snapshots in time. The version for a particular date allows one to view all the files as they were on that date. Luxor implements the **luxor revert** command by storing versions along with their associated dates in the vDb (version database) and querying for a specific date. One can search over all history by scanning the entire database.

```
=== Version
45

=== Previous Version
44

=== Files Added
d8d4ca2a38a0c7cc0c69cbc49b3b905690103c49
  /home/john/added_file.txt
4dceef4f895818a99b21da34ae3c2246774ed4b3
  /home/john/added_directory/another_added_file.txt

=== Files Deleted
9912ab7c92824e08284333a87af82c4a0fc4d227
  /home/john/deleted_file.txt
deadf610221c89db0d1c40b4fb6fee1eec5176a9
  /home/john/deleted_directory/

=== Diff

 == /home/john/modified_file.txt
 - Some old text
 - that I removed
 + Some new text
 + that I added

 == /home/john/some_code.rb
 - def broken_function
 -   raise ArgumentError
 - end
 +
 + def do_nothing_function
 + end
```

Figure 2. Example of a version file which shows the files added, deleted, and modified in that particular snapshot, along with the previous version.

Verions are implemented as a difference from a previous version snapshot. The version file, shown in figure 2, records all information of a version by pointing to the previous version and

recording changes that have occurred since that version. The version file includes checksums for each file that was added or deleted. These checksums serve as keys in the fDb (file database) for storing the original files.

To revert the filesystem to a particular date, Luxor starts at the closest full filesystem copy before that date and rebuilds the filesystem by using the differences from each sequential version file. Periodically, Luxor does a full filesystem copy so that rebuilding versions takes less time.

Automatic Saving

Luxor works in the background of the operating system. The user never needs to explicitly save, because Luxor uses a UNIX filesystem hook to create a new version whenever a file gets modified. Since version files only store differences, they are inexpensive to build and store. Luxor can therefore handle the load of many modifications happening in a short time span.

Conclusion

Luxor provides a user-friendly way to track filesystem history. Built on version files, Luxor can create new filesystem snapshots quickly. The speed of Luxor allows it to automatically take snapshots whenever files are saved, reducing work and complexity for the user. Reverting the filesystem is easy because of Luxor's simple commands. Luxor provides the casual computer user with a means of recording file history and the potential to significantly improve productivity.