Repository Working Group	J. Kunze
	S. Abrams
	E. Hetzner
	D. Loy
	California Digital Library
	June 23, 2009

Reverse Directory Deltas (ReDD)

Abstract

ReDD is a "reverse delta" scheme that permits the reconstruction of an earlier version of a set of files starting from a given full set of files. In a chain of ReDD version deltas, access to prior version *N-M* starting from version *N* requires the sequential reconstruction of versions *N-1*, *N-2*, ..., *N-M*, so the later the version the faster the access. Because later versions don't depend on earlier versions, the old part of a chain broken at an arbitrary point can be discarded without adjustment to the remainder of the chain. Any ReDD version can also be represented with a full set of files rather than with deltas, which may be useful for caching recently reconstructed versions or for establishing points of departure for runs of deltas. Typically, full file sets occur in the latest version at the head of a chain and in any versions that serve as checkpoints or cover interior modifications of the original version chain.

1. The multiple versions problem

Keeping multiple versions of a set of digital files can be costly if each version is stored in full. If differences between versions tend to be small, however, it can make sense to compress a chain of versions by storing a full set of files once and recording only the changes, or "deltas", thereafter. For instance, with multiple versions of a set of 80 image and text files, one of which holds metadata that changes 5 times a year, keeping just the changed files over the course of a year requires maintaining only 85 files instead of 400 files.

The trouble is access. The file set in a compressed version is no longer ready-to-go, but must first be reconstructed according to the deltas. Making reconstruction simple and understandable might sacrifice some storage savings, but that may be worthwhile in digital library settings where reliability, transparency, and platform-independence are also important. An especially simple approach is to use file-level deltas as opposed to subfile-level deltas (e.g., Unix `diff` and `patch`). For file-level deltas, reconstructing a set of files that comprises a generic digital object can be thought of as a sequence of file deletion instructions followed by a sequence of file addition instructions.

2. ReDD home directory

ReDD (Reverse Directory Deltas) is a scheme for representing the differences between a given full set of files and a previous version of that set. Differences are called "reverse deltas" because they revert a version, *N*, to a previous version, *N-1*. The advantage of this inversion is that the most recent version is always ready-to-go and reconstruction costs are incurred only for what are typically uncommon access patterns (to older versions).

A *ReDD home* is a directory that holds ReDD deltas. It represents a compressed version using essentially two reserved names. In the example below, deltas specify the deletion of two files and the addition of one file in order to construct the previous version (lines of file content are shown in parentheses beneath the file name).

```
<redd_home>
 |   0=redd_0.1        # declaration that this is a ReDD directory
 |   delete.txt        # delete these in creating previous version
 |    (data/27613-h/images/q172.png)     # to delete from current
 |    (data/27613-h/images/q172.txt)     # to delete from current
 |
 \--- add/             # add these, with paths, in creating previous
       |   admin/conformance.xml         # to add to current
```

Here, the special file named `0=redd_0.1` (a name-as-text tag file **[Namaste]**) identifies the type of this directory to be "ReDD 0.1". The implementor may choose any name for the ReDD home directory.

Version differences are expressed via the text file, `delete.txt`, which lists filesystem paths to be deleted, and the directory, `add/`, which contains files to be added (with paths intact). Applying the changes to a full copy of the current version constructs the previous version.

3. Version chains

The application of deltas requires a fully instantiated version to start with, in particular, at the head of a chain of version deltas. The same holds inside the chain, where the immediately preceding version must be fully instantiated, and this can imply reconstruction of all preceding versions back to the head of the chain. Here's ReDD in the context of a "digital flat" **[Dflat]**; this example shows the current version and three previous versions.

```
<dflat_home>           # Best to read this example from the bottom up.
 |   0=dflat_0.7                    # I am a "digital flat"
 |--- v007/                         # tail (oldest) in version chain
```

```
|     \--- redd/
|           |   0=redd_0.1
|           |   delete.txt
|           |    (data/thumbnail.png)    # delete in 8->7 (add in 7->8)
|           |    (enrichment/meta.xml)  # replace in 8->7
|           \--- add/
|                 \--- enrichment/
|                       |   meta.xml
|--- v008/
|     \--- redd/
|           |   0=redd_0.1
|           |   delete.txt
|           |    (enrichment/meta.xml)  # replace in 9->8
|           \--- add/
|                 \--- enrichment/
|                       |   meta.xml
|--- v009/
|     \--- redd/
|           |   0=redd_0.1
|           |   delete.txt
|           |    (enrichment/meta.xml)  # replace in going from 10->9
|           \--- add/
|                 \--- enrichment/
|                       |   meta.xml
\--- v010/                        # head (latest) in version chain
      \--- full/
            |--- data/ ...
            \--- enrichment/ ...
```

ReDD has no concept of version numbers or chains. It is Dflat in this example that is used to represent a chain of four versions with version 10 at its head. Examining this particular chain, one may observe a common versioning pattern in which the full set of files is more stable than its metadata. To reconstruct a version does require the presence of a fully instantiated version, but ReDD does not specify where it resides.

To create version 9, for example, the `enrichment/meta.xml` file is replaced (first deleted, then an older file re-added). The same metadata replacement occurs in constructing versions 8 and 7. It is in going to version 7, however, that a file needs to be deleted. This is an image file present in version 8 but absent in version 7. It was originally added in going from 7 to 8, so it has to be deleted going from 8 to 7.

4. Applying ReDD differences

If construction of the previous version is desired, first one needs a full copy of the current version. Starting with the current version, the `delete.txt` file is a list of files and directories that need to be deleted to create the previous version. Entries are listed one per line, with a directory entry distinguished from a file entry by a terminal '/' character (useful since files and directories are often removed with different commands). It is likely an error if, before deletion, any file or directory listed in `delete.txt` does not exist.

After all entries in `delete.txt` have been deleted, construction of the previous version will be complete after all items in `add/` been added. It can be a little confusing, because the ReDD 'delete' and 'add' operations that create the previous version are precisely the inverse of those by which the current version originally evolved from the earlier version.

Here's a summary of the two-step process to go from a copy of the fully instantiated set of files in the current version to a fully instantiated set of files in the previous version.

1. If `delete.txt` is present, delete all file and directory entries found in this file. It is an error if, before deletion, any file or directory listed in this file does not exist.
2. Finally, if the `add/` directory is present, bring in copies of all files and directories found under it.

When done, the previous version (the target of this version reconstruction) should now be complete.

5. References

[Dflat]	"**Dflat: Simple File-Based Object Storage**," April 2009 (**PDF**).
[Namaste]	Kunze, J., "**Directory Description with Namaste Tags**," April 2009 (**HTML**).

Authors' Addresses

John A. Kunze
California Digital Library
415 20th St, 4th Floor
Oakland, CA 94612
US
Fax: +1 510-893-5212
Email: jak@ucop.edu

Stephen Abrams
California Digital Library
415 20th St, 4th Floor
Oakland, CA 94612
US
Fax: +1 510-893-5212
Email: stephen.abrams@ucop.edu

Erik Hetzner
California Digital Library

415 20th St, 4th Floor
Oakland, CA 94612
US
Fax: +1 510-893-5212
Email: **erik.hetzner@ucop.edu**

David Loy
California Digital Library
415 20th St, 4th Floor
Oakland, CA 94612
US
Fax: +1 510-893-5212
Email: **david.loy@ucop.edu**