MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Autonomous File System: The RECONCILE program

John H. Howard

TR93-09a December 1996

Abstract

RECONCILE combines files stored at two or more sites while avoiding the danger of losing updates at one site because of updates performed concurrently at another. This report describes the program in detail, including motivation, basic concepts, the programś external interface, and internal design.

Revision History:

1. Initial version, MN93-12, May 12, 1993.
2. Revised and reissued as TR93-09, June 8, 1993.
3. Reformatted, July 6, 1994.

Scenario

Imagine that you are writing a book using personal computers at your office and your home, carrying your files back and forth on a diskette. Your normal procedure is to copy all the working files from the diskette to the computer you are about to use, edit one or more chapters, and copy the edited files back to the diskette when you are done.

You have three different copies of your files, one stored in the office computer, one at home, and one on the diskette. Even though there are really three copies, you think of them as being different versions of the same files.

Now suppose you forget to copy the files you edited at the office and go home with an out-of-date diskette. You copy the diskette to your home computer and continue editing, not noticing that you are starting with stale information. (Perhaps you are editing a different chapter.) The next day you copy the updated files back to your office computer, losing yesterday's work!

There are some things you can do to help protect against this common error. For example, some file copying programs have an option to check dates and refuse to replace a newer version of a file with an older one. This helps considerably, but is not perfect. It does not detect the error described above, for example, since the versions of the files you edited at home in the evening do have a later date than the versions you edited yesterday at work. It also fails to handle the case of deleting obsolete files.

The *reconcile* program detects conflicting updates, so you can use it to update files safely. It will replace a file with a later version only if it is sure that the later version was derived from the one being replaced. If the file to be replaced is not an earlier version, *reconcile* will report an error so that you can resolve the conflict yourself.

In the scenario, neither the office nor the evening version of the files was derived from the other, so *reconcile* won't overwrite them. (They were both derived from the same earlier version, but not from each other.)

How does *reconcile* know when one version of a file was derived from another? The basic idea is to keep a history of past versions of files. If one history says that a file has gone through versions 1, 2, and 3 while the other has only 1 and 2, it is safe to copy version 3. If one history shows versions 1, 2, and 3 while the other shows versions 1, 2, and 4, there is a conflict since neither version 3 nor version 4 was derived from the other.

Concepts

A **file** is a body of closely related information stored in a computer. Typical examples of files would be documents you edit with a word processor, or spreadsheets, or messages. Each individual memorandum, letter, or book chapter is kept in its own file. In addition to its contents, a file has a **name** and a **time stamp**. The name identifies the file and the time stamp says when the file was created or changed. As time passes, a file with the same name will have different **versions**, distinguished by their different time stamps.

A **directory** is a collection of files. Usually the files in a directory have some loose relationship, for example that they are all part of some larger body of information like the chapters in a book, or that they were created by the same person, relate to the same topic, or are owned by the same organization. Directories also have names, and may also have time stamps although directory time stamps aren't very useful.

Most computer systems arrange files and directories in a **hierarchy** (or **tree**), which simply means that directories can contain sub-directories. An advantage of sub-directories is that they can group closely related files together. To find a file you work your way into the successive sub-directories until you reach the file you want.

A **working session** is a period of work on a single computer. During the course of a working session, your files might be in an incomplete or inconsistent state and you ordinarily don't want to make a permanent record of them or to send copies elsewhere. Usually you try to finish a day's work by cleaning up the inconsistencies before ending your working session, although occasionally a session might last several days. For the sake of discussion here, a session can be anything you choose. The important thing about it is that you don't use *reconcile* to copy files during a session, but only at the beginning and/or end.

A **site** is a specific storage location for a directory hierarchy. It is an assumption of this document that one should think of several sites as all containing versions of the same directory hierarchy. These versions might be the same or different. The basic purpose of *reconcile* is to combine hierarchies at different sites, making them all the same by safely updating versions of individual files.

One should not think of a site as being the total disk storage on any one computer. Usually a site would contain a number of unrelated hierarchies defined according the user's convenience. A personal computer, for example, might contain separate hierarchies for system software, installed applications, and one or more individuals' working files. While actual systems often glue these into a single super-hierarchy, it is easier to think of them as being separate.

A site may also be nothing more than a diskette. The way we will copy files to and from the diskette in the introductory scenario will be to reconcile the diskette version with the computer (home or office). At the beginning of a working session, *reconcile* will notice newer files on the diskette and copy them to the computer. At the end of the session, *reconcile* will notice newer files on the computer and copy them to the diskette.

A **journal** is a history of file versions. To do its work, *reconcile* creates a journal for each site, merges them to look for missing versions, and either updates by copying more recent non-conflicting versions, or else reports errors if there are conflicts.

As with database journals, the journals used by *reconcile* contain not only names and time stamps but also actions. For *reconcile* these are very simple: either "update" or "delete", which can be inferred from the fact that a previously present file has disappeared. Including deletion operations in journals means that *reconcile* can (safely) propagate deletions to other sites, again checking for conflicts.

There are actually two kinds of journals: **internal** and **external**. An internal journal is stored as a special file within the directory it describes. In a hierarchy, each directory has

its own internal journal. An external journal contains the same information, but has been extracted into a separate file, and stored somewhere else. Although *reconcile* can use both kinds, it can only update to or from internal journals. External journals are sources of information about necessary updates, but the actual files and directories involved are not directly accessible.

Using reconcile

The simplest and standard way to use *reconcile* is to apply it to several directly accessible sites such as mounted disks or diskettes. For example the command

```
reconcile . a:/
```

would reconcile the current working directory (named "." in most systems) with the diskette in drive A. The order of the two directories doesn't matter. In the introductory scenario, you would do this when you begin using either your office or your home computer, and again at the end. So long as you never forget to do this, all updating will be automatic. You can even delete obsolete files without having them "come back" at the other computer.

Suppose you do happen to forget to reconcile at the beginning or end of a session, and you then update some file (say "oops"). The next time you reconcile with the two conflicting versions of the file, you will get the error message:

```
SAVE OLD:  a:/oops in oops1
WARNING: oops1 is a saved version of oops
```

At this point you might use a tool such as *diff* to find and display the differences between the two versions, then edit `a:/oops` to recover any changes in `a:/oops1`. When you are satisified that you have recovered everything, delete `a:/oops1` in order to suppress the warning message on later reconciliations.

Building journals

Reconcile builds its journals by comparing the actual directories with the previous versions of its own journals each time it is run. This means that it makes sense to run *reconcile* even for a single site:

```
reconcile .
```

updates the internal journal of the current working directory. If you make several successive versions of a file, *reconcile* will only see the last one since the last time it was run. This can actually be an advantage, since the other versions are of no particular significance as long as they are not transmitted to any other site. You can choose how often you want to run *reconcile*. Even if you forget to run it at the end of a working session, you will not lose anything permanently. The cost of forgetting a reconciliation will be an increased probability of conflicting updates, needing manual intervention at a later time.

Some other applications

In addition to the introductory scenario, here are some other applications for *reconcile*.

Suppose you are jointly writing a research paper with a colleague. You store the various sections of the paper in a directory to which each of you has access. You ordinarily communicate directly to avoid conflicting updates, but sometimes you forget. You can handle this with *reconcile*. Make a private copy of the entire directory for each of you. Let's say the directories are named ~tom/paper, ~dick/paper, and ~common/paper, and that you are Tom.

Before beginning a working session, you perform the command

```
reconcile ~tom/paper ~common/paper
```

at this point there might be conflicts; if there are you may need to give Dick a call to resolve them. Having done so if necessary, you are sure that your working version of the paper is in agreement with the shared version. During the course of your work various sections might be temporarily wrong, or inconsistent with each other, but since this is just your working copy and not the common version, you need not feel concerned. Eventually you're happy with what you've done; you've proofread it and it appears to be consistent and free of errors. You check it back in with exactly the same command as above.

Reconcile Command Syntax

The syntax of the *reconcile* command is

```
reconcile [options] [[-mode] (directory | file)] ...
```

If no directories or files are provided, the current working directory (".") is used.

`directory` names a directory containing an internal journal and files

`file` names an external journal describing some remote site
("-" refers to an external journal on standard input or output)

`-mode` one or more of the following letters:
- `r` read the journal but don't write it
- `w` write the journal but don't read it
- `o` do not update files, only the journal

The `[options]` are:

`-q` work quietly, suppressing messages about actions taken

`-n` do not update any files (OK to update journals)

`-h` print a description of the command syntax, don't do anything else.

`-a sitename` Abandon named site. Use this to forget a site that is no longer in use. This allows the program to discard obsolete journal entries needed only for reconciliations with old sites. Sites are automatically abandoned after two months, with warnings being printed after one month.

Environment:

In addition to command line parameters, *reconcile* gets a name for the computer system being used from the environment variable `$HOST`, using "`UNKNOWN`" if it is undefined. This name is needed only as a substitute for the disk volume label used to identify sites.

Internal Documentation

This section gives the details how the program works. It will be of interest primarily to individuals desiring a deeper understanding of the program's behavior and to programmers interesting in maintaining or modifying the program. See also the appendixes, which give supporting information.

Overall Sequence of Events

Reconcile performs its processing in the following general steps:

1. Parse parameters, building a list of sites to be reconciled. If no sites are given, use "." (the current working directory) as the only site.

2. Read the old journal file for each site.

3. For each internal journal, update the journal by examining the files currently present at that site.

4. Update the list of known sites and their most recent reconciliation times. Discard events known from these times to be obsolete at all known sites.

5. Perform the actual reconciliation, detecting conflicts and replicating files when there is no conflict, and warning the user when a conflict exists.

6. For each site, write out an updated journal file.

These steps are described in more detail below.

Parameter parsing (step 1)

This is a straightforward process of examining the parameters sequentially. It is performed by procedure `main` in module `reconcile`.

Reading and writing journals (steps 2 and 5)

Journals read by procedures `readjournal` and `readentries` in module `journal`, and written by procedures `writejournal` and `writeentries`. The journal file format is editable text, described in Appendix A.

Updating from the actual directory (step 3)

Journals are brought up to date with reality by reading the actual directory and inserting journal entries accordingly. This work is done by procedure `readdirectory` in module `journal`. Current directory entries are considered one at a time. For each one found, a new journal entry for the site, time, and filename is created if none exists, and the new or existing entry is marked as having been confirmed.

After the entire directory has been read, a pass is made through the journal looking for unconfirmed entries. An unconfirmed entry indicates that a file once existed but no longer

does, that is, that it has been deleted. For each such unconfirmed entry, a new journal entry is created with a deletion action and the current time as its time stamp.

Reconciliation (step 5)

The actual business of reconciliation is performed on the internal representation of the journals by procedure `reconcile` in module `journal`.

Reconciliation is performed only for the "current" entry, that is, the most recent entry for the file. Previous entries refer to out-of-date versions of the file. The goal is to make the entry (and file) present at every site. For each site at which the entry is *not* present, first check to see if there is a conflict (see below). If there is none, copy the file from any site at which the current entry exists. Copying the file may actually be deleting it if the current event is a deletion event, or may involve creating or deleting a directory or symbolic link.

ASCII text files vary slightly in format between DOS and UNIX systems. The copying procedure deals with this by inspecting the beginning of each file it copies. If the file looks like a text file (it contains only standard printing and control characters organized into lines no longer than 128 characters) then the program automatically adjusts the text format for the target system type.

Detecting inconsistencies

A conflict exists for a site if the site does not have the current version of a file, but does have some previous version, and the current version is not derived from the previous one, as defined in the next paragraph. Existence of derivation shows that the current version came from the previous one by a connected series of user actions. This implies that it is safe to replace the prior version with the current one, since doing so is equivalent to replaying the sequence of actions in the derivation. Lack of a derivation implies that the replacement may be unsafe and therefore should not be done automatically.

A derivation is a sequence of steps that convert the older version to the newer one. Each step should be directly reflected in the journal of some site, as a successive pair of entries, one following the other, with no other entries for the file in between at that site. (There might be intervening entries at other sites, which would mean conflicts elsewhere but not for the derivation being considered.) This sequentiality at a particular site implies that the later of the two successive versions was created directly by editing or modifying the earlier of the two. A derivation is like an audit trail; it records all the steps taken to convert the older version to the newer one.

Existence of a derivation is determined by procedure `connected` of module `journal`, which follows the direct steps backward from the current version, skipping unrelated events, until it reaches the prior version.

Discarding obsolete events (step 4)

Without some way of discarding obsolete events, the journal files would grow indefinitely. An event is obsolete at a given site if there is a more recent event for the same file at the

same site. If an event is obsolete at all sites, it can be discarded because it will never cause an inconsistency.

In order to track this, journals contain a list of "known sites", each marked with time of the most recent reconciliation involving the known site. This list is propagated and updated as reconciliations occur. The known site times indicate when information came *from* the known sites. In addition, each known site reflects these time stamps back by generating a list of acknowledgments. An acknowledgment gives the name of the known site, the name of a site known to it, and the time stamp of that source site. Again, these acknowledgments are propagated and updated appropriately. Originating sites may use the time stamps of the acknowledgments to determine when events are obsolete at other sites. Any event that happened before the oldest acknowledgment time must have been propagated to all known sites, so its predecessors can be discarded without causing conflicts.

Finally, there is a potential problem about lost sites. If a site fails to produce acknowledgments, it will cause events to accumulate indefinitely. This could easily happen if the site is no longer used. To avoid this, the program issues warnings about any sites last heard from more than a month ago, and removes them from the known site list after two months. There is also a command line option to discard a known site immediately.

Appendix A: Journal file format

Journal files are standard text files that can be observed (and even changed, at your own risk) with any text editor. The files contain a header line, several lines identifying known sites, and then one line for each journal entry proper.

Header line:

```
Journal of <sitename> (<systype>) - <programname>
```

where <sitename> is the fully qualified name of the file hierarchy, <systype> is the system type (DOS or UNIX at present), and <programname> gives the name and version of the reconcile program that wrote the journal.

Entry line:

The entry lines each contain a fixed set of fields, separated by tabs, in the format:

```
<verb> <date> <time> <name><type> <remarks>
```

<verb> is one of a limited set of symbols denoting possible actions:

+	create or update, making a new version
-	delete
>	some other site has a more recent version
<	some other site has a more recent deletion
*	some other site has a more recent version, but it conflicts with the previous version at this site.

<date> is the date the action occurred (yy/mm/dd format)

<time> is the time the action occurred (hh:mm:ss format)

<name> is the file's name, followed directly by the type (no intervening tab)

<type> is a single character:
(nothing) for an ordinary file
/ for a subdirectory
@ for a symbolic link

<remarks> are arbitrary comments. Saved files are indicated by the special remark:

```
!was <original name>
```

Special handling:

Lines in the journal of the form:

```
#ignore <pattern>
```

cause matching files to be ignored during reconciliation. The pattern is a file name in which wild card ("*") characters match arbitrary sequences of zero or more characters. #ignore applies to the directory in which it appears and all of its subdirectories.

Lines in the journal of the form:

```
#normal <pattern>
```

override directives such as `#ignore` inherited from parent directories, restoring normal handling to files with matching names. Within a single directory, the first matching directive is used.

Known site and acknowledgment:

The known site lines give the date and time of the most recent reconciliation for each known site. Each line is of the form:

```
$ <date> <time> <sitename>
```

Acknowledgments immediately follow the known site line for the acknowledging site, and are of the form

```
. <date> <time> <sitename>
```

where the acknowledging site (named by the immediately preceding known site line) is simply reflecting a reconciliation time back to its source.

Appendix B: Program Modules

The program is built out of the following source modules, each of which is represented by a C++ source file (<name>.cpp) and a corresponding header file (<name>.h).

entry	Defines individual journal entries.
filesys	Performs local file system input/output operations.
journal	Defines journals, performs most of the actual work.
knownsit	Defines "known sites", handles acknowledgments obsolete event times.
myalloc	Performs storage allocation and checks for memory leaks.
parse	Supports text parsing operations in parameter and journal file processing.
reconcil	Main program, parameter analysis, and user message generation
site	Defines sites.
timestmp	Defines internal and external formats for date and time information.
filetype	Handles filename-based special handling rules such as #ignore.