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

Reconcile UsersǴuide

John H. Howard

TR99-14 December 1999

Abstract

Reconcile combines different versions of file directories, propagating all updates between them and making them identical but never losing updates at one site because of updates performed at another. Among its applications are: - Road warriors: synchronize files between laptop and home base This report describes the program in detail, including motivation, basic concepts, applications, and the programś interface.

MERL – A Mitsubishi Electric Research Laboratory

Technical Report TR99-14 **September 1, 1999**

Reconcile Users' Guide

John H Howard

Abstract

Reconcile combines different versions of file directories, propagating all updates between them and making them identical but never losing updates at one site because of updates performed at another. Among its applications are:

- Road warriors: synchronize files between laptop and home base
- Telecommuters: use a distant file server efficiently
- Power users: work with a local copy of your files, update the file server occasionally
- Collaborators: safely share working files with peers
- Individuals: make private backups on a removable disk

Reconcile currently runs on (and between) Windows 95, 98, and NT, and Unix (Linux, OSF1, SunOS, and IRIX64 tested, others likely.)

This report describes the program in detail, including motivation, basic concepts, applications, and the program's interface.

Revision History

1. Initial version, TR93-09, June 1993.
2. Revised and reformatted, TR98-09a, July 1994.
3. Major revisions, TR98-04, April 1998.
4. Revised (additional parameters and directives), TR98-04a, June 1998.
5. Revised (minor changes), August 10, 1998.
6. Revised (remote reconciliation), TR98-04b, September 23, 1998.
7. Revised (minor changes), TR98-04c, January 12, 1999.
8. Revised (minor changes), TR98-04d, February 11, 1999.
9. Major revisions (added remote reconciliation, revised user interface, many minor updates), TR99-14, March 18, 1999.
10. Revised (minor changes), July 9, 1999.
11. Revised (minor changes), August 26, 1999.
12. Revised (identify obsolete options, add –also), September 1, 1999.

CONTENTS

Quick Start

PLEASE READ THE REST OF THIS MANUAL! It contains valuable information about what to expect from reconcile and how to use its features.

These instructions assume that you are starting with a directory "/homes/doe" on a file server named "hercules", and you want to create and start using a local copy on c:\doe (a local disk under Windows).

Installation

√ Copy the reconcile program file to an appropriate directory, such as /usr/sbin (for Unix), c:\winnt\system32 (for Windows NT), or c:\windows (Windows 95 or 98). You need a copy on the server as well as the client machine.

First time setup

√ *Important!* Make sure your local disk has a unique, descriptive label.

√ Create a journal for the existing directory hercules:/homes/doe with the command:

```
reconcile -v –c hercules:/homes/doe
```

This may take a while, as reconcile computes digests of every new file it finds. It will be much faster if a journal already exists.

√ Create local copies of the directory and journal, for example on c:\doe:

```
mkdir c:\doe
reconcile -v hercules:/homes/doe –c c:\doe
```

Normal operation

To reconcile hercules:/homes/doe and c:\doe with each other, use the command:

```
reconcile -v hercules:/homes/doe c:\doe
```

You may wish to put this command in a batch file or on an icon, so that you can click it to run. Once set up, you may want to schedule it to run every evening automatically.

Trial Runs

To be extra careful after installing a new version, do a trial run with the -n option:

```
reconcile -n -v hercules:/homes/doe c:\doe
```

Reconcile will tell you what it would do, without actually doing it.

Ignoring selected files

To prevent reconcile from copying some sorts of files, for example core dumps or browser caches, create a file named ".reconcilerc" containing directives like:

```
#ignore *cache
#ignore core
```

Overview

Reconcile compares file directories, finds the differences, and updates the directories so that they all contain exactly the same files. It does this safely and automatically, requiring user intervention only when there are unavoidable conflicts between the versions of files stored in different locations.

Features

Safe updating is accomplished by using a journal of past updates for each directory. An older version of a file is deleted only if the journals indicate that the newer version was derived directly from the older one. If this criterion is not met, the older version is renamed, rather than being deleted. This allows you to inspect the two files and determine how to merge or otherwise resolve the conflict between them. Reconcile prints a warning message every time it runs, until the renamed file is dealt with.

A newer version of a file can safely replace an older one whenever a journal contains both of them. This can only happen if the newer version was created by a deliberate act replacing the older version. The journaling mechanism also handles deletions by noting the disappearance of files as well as the appearance of new versions. This means that if a file is deleted at any site, Reconcile can safely delete it at other sites. This is a key advantage over programs which simply compare time-stamps on files. It also avoids relying on synchronized clocks between computers.

Reconcile is *automatic* in the sense that it does not require user intervention. It can be invoked from a batch file, periodically through a system scheduler agent, when you log in or out, or at other convenient moments. In its batch mode, Reconcile runs to completion, logging messages about any conflicts between versions of files or other problems it encounters. In the unusual case of a conflict that Reconcile can not resolve by itself, it renames one of the conflicting version with a distinctive name to make manual resolution easier.

Reconcile can ignore certain files, such as backup copies and object files generated by compilers. This avoids unnecessary conflicts between files which are derived from other files rather than being created or updated directly by user actions.

Finally, Reconcile offers *cross-platform compatibility.* Versions exist both for Windows and for Unix. When copying text-mode files (defined as files containing only white space and printable characters) between Windows and Unix sites, Reconcile automatically converts line endings to the form used by the target site.

Key concepts: sites, journals, and versions

A **site** is one version or copy of a set of files, organized as a sub-tree of the file system hierarchy. A typical site might be an individual's working directory on a particular computer. The basic purpose of Reconcile is to make several sites identical by safely copying individual files between them. If you use two computers, you would have a site on each of them.

One should not think of a site as being the total disk storage on any one computer. Computers contain many unrelated directories, including the operating system, installed software, personal

working files, and shared projects. File systems typically glue these together into a single hierarchy. Nevertheless, one can usually identify sub-trees of the overall hierarchy with specific individuals, projects, or products. Each such subtree is a potential site.

A site may be stored in the internal disk of a computer, a portable medium such as a floppy diskette or Zip disk, on a network file server, or on another computer accessible via the Unix "rsh" command. Sites accessible directly through the local file system are called **local sites**, even if they are actually mounted from a file server. Sites accessible via the rsh command are called **remote sites**. All files and subdirectories beneath a site's top-level directory are part of the site unless they are explicitly excluded. If the file system supports symbolic links, the links are included in the site, but the objects they refer to are not. (Reconcile preserves symbolic links for use by Unix system but does not attempt to copy them to Windows systems.)

To do its work, Reconcile creates or updates a **journal** file for each site, listing the history and status of every file at the site. A journal is actually a text file listing the successive versions of files contained in the root directory and each sub-directory of the site. A new **version** of a file is created every time the contents of the file are modified, or if the file is deleted. Thus a version describes both an action, which can be either "update" or "delete", and the contents of the file. By including deletion operations in journals Reconcile can (safely) propagate deletions to other sites, again checking for conflicts.

Journal entries for versions of files contain the action, file name, a **time-stamp** (the date and time the file was most recently modified) and a **digest**, which is a checksum of the entire file computed in such a way that it is extremely unlikely that two different files would ever have the same digest. In order to promote portability between UNIX, Windows, and Macintosh computers, Reconcile ignores differences in line endings (CR, LF, or CRLF) when computing digests of text files, and (optionally) translates line endings when copying text files between systems of different type.

Reconcile only uses the time-stamp to determine when it should compute a new digest. It does not depend on times being synchronized between computers, and is even insensitive to arbitrary changes in the clock on one computer. Since time stamps have a resolution of a second or two, it is extremely unlikely that two different versions of the same file would have the same timestamp.

It might be a good idea to include other file information, such as protection flags, in the journal, but this is currently not done in Reconcile because of the lack of consistency between Windows and UNIX in this area. Reconcile does copy protection information along with files between same systems.

Operation

Reconcile operates in several phases sequentially:

Read and update individual journals

Reconcile starts by reading the entire existing journal for each site. It brings the journal up to date by traversing the file tree, comparing the files actually present with the files listed in the journal. When they disagree it creates new journal entries reflecting creation, modification, and deletion of files at each site.

Deletions are inferred from the absence of an actual file for which a journal entry exists. New files and updates to existing files are detected by comparing hash codes. For efficiency's sake, the hash code is recomputed and compared only if the timestamp changes. This approach can be fooled if two different versions of a file have the same timestamp; a very unusual possibility. It is not fooled by the reverse case, namely that the same data has two different timestamps. In this case reconcile simply corrects one of the timestamps without actually copying the data.

Merge journals

Next, reconcile merges the journal entries for each file by finding a maximum matching subsequence of entries among all the sites. The main purpose of this is to find the last common entry for each file, that is, the most recent matching entry among all sites. Therefore the merging operation prefers matching the most recent version of each file in the journal over than matching earlier versions.

Propagate missing actions

If the last matching entry of a file is present at all sites, then they are already consistent and no action is needed.

- If there is an entry present at all sites, it is definitive evidence that any subsequent entries represent intentional creation, modification, or deletion of the file. Entries after the common entry which match each other but are missing at some of the sites reflect creation, modification, or deletion operations that have occurred at only some of the sites. Any such matching subsequent entries are replicated to the other sites by copying or deleting files as required.
- If two sites has different entries after the last common entry, there is a conflict. In this case, the older versions are renamed rather than being deleted, using names of the form "filename#1.extension". The number is chosen to generate a unique filename. Presence of the unusual "#" character in the file name makes it easy to search for such files. Reconcile reminds you of the unresolved conflict every time it runs, until the renamed file is deleted. This feature can be turned off with the `#alwaysnew` directive, which reduces warnings but runs the risk of losing conflicting updates.
- For safety's sake, deleted and files and subdirectories are temporarily saved in a hidden subdirectory named ".trash" rather than actually being deleted. The trash is automatically

purged after several days, but before then you can recover files from it with standard file copy or move commands.

Purge obsolete versions

Old journal entries become obsolete when they are known at all sites. Reconcile tracks this by maintaining a cumulative list of all sites being reconciled now and other sites that have participated in past reconciliations with other known sites. It marks each version with the set of known sites which have heard about the version. Once a newer version is known at all sites where the older version is known, the older version is no longer needed and is purged from the journal.

If a known site has not participated in a reconciliation for a very long time, old versions will accumulate and the journal will slowly grow. To deal with this, Reconcile abandons known sites which have not been heard from for a month or more. (It generates warning messages for several runs before doing this.) If an abandoned site rejoins the set of known sites, its old versions of files will be treated as conflicting versions, generating false warnings, rather than being updated cleanly.

There is a special case for old deletion entries in the journal. If some, but not all, sites purge an old deletion, the others will see this as a missing journal entry the next time they reconcile and therefore restore the purged version entry. To deal with this, reconcile discards month-old deletions regardless of whether they are known at all sites.

Write updated journals

After updating, Reconcile writes out the updated journals at all sites. In order to minimize the possibility of a crash leaving a partial journal written, Reconcile actually writes the new journal with extension ".jnw", then renames it to ".jnl" once the write is complete. Reconcile also saves the previous copy of the journal by renaming it with extension ".jbk".

Set the time

PC clocks are notoriously unreliable, so reconcile provides an option to keep them approximately synchronized. By program option, a site may be designated to be a time reference. The time *at that site* is determined by creating a dummy file and reading its timestamp. If the time thus obtained differs from the local clock by more than a few seconds, reconcile attempts to reset the local clock to agree with the reference site. Note that this is at best approximate, and that setting the clock may require administrative privileges. A better solution is to install a network time client. However, this option can be useful for laptops and other only occasionally connected Windows systems.

Remote Operation

When a site is named using the syntax <hostname>:<path>, reconcile uses the rexec or rlogin protocol to open a client-server connection with a copy of itself running separately on <hostname>, which it starts as /usr/sbin/reconcile, with most of the local parameters passed

through, and the –x option added. A "server" reconcile, started with –x, expects to communicate with a client through its standard input and output streams.

The client starts by issuing a `getjournal` command. The server responds by reading the local journal, updating it from the local site directory, and sending the journal to the client. The client then does all the actual reconciliation, issuing `get`, `put`, `rename`, `delete`, and `fixtime` commands to update files as necessary. At the end it sends back the updated journal using a `putjournal` command, during which both client and server write out the updated journal to their local disks. More detail on this is to be found in the reference material.

Remote operation has several substantial advantage over direct reconciliation using a mounted disk. By running the directory scans and journal updates in parallel at each site it cuts the overall running time in half. The streaming TCP-based protocol it uses moves data more efficiently than the block-by-block transfers typical of mounted file systems, thus saving network time and permitting reconciliation over long-distance connections. Round-trip interactions are reduced to one per changed file. No file server is required, only rsh or rlogin, which are part of the standard TCP/IP protocol suite, plus a copy of reconcile installed at the server.

Remote operation requires a remote user name and password. These are taken from environment variables USER and PASSWORD respectively. If the value of PASSWORD is "*", reconcile prompts for a password. If the value of PASSWORD is empty or not given, reconcile attempts to use rsh (which requires special authorizations at the server in lieu of a password, and usually doesn't work through firewalls.)

Examples

This section describes how to use Reconcile in several common scenarios: saving to a removable disk, moving files between computers, file sharing in a network, and server-less file sharing.

Labeling your disks

It is important that you label each disk (both fixed and removable) with a unique and meaningful name. Use the label command to review and set the label of any disk before you create a new directory on it. For example, if your office computer is named "thor", you might label its C disk with the command:

```
label c: THOR-C
```

Creating a new journal

To create a new journal for existing files in c:\doe, use the command:

```
reconcile -w c:\doe
```

Making private backups on a removable disk

Suppose you keep your working files in the directory c:\doe, and want to keep a backup copy on a removable disk in drive z. Create a journal for c:\doe as described above. Also label the removable disk, create an empty directory and journal on it:

```
label z: WORKCOPY
mkdir z:\doe
reconcile -c z:\doe
```

You only need to do the above steps once.

Once the removable disk is set up, you update it by running the command

```
reconcile c:\doe z:\doe
```

Reconcile generates messages describing any actions it takes and problems it finds. You don't need to remember these messages on the spot, as any important warnings will be repeated until you do something about them. The first run may take a little while as many files are copied and the journal initialized, but subsequent runs will be much faster.

A restriction in this example is that the removable disk must be large enough to hold all the files.

Local copy from a file server

The example above used a removable disk, but could equally well use a disk shared across a network by another computer. The only change is to replace the removable disk's drive letter (z: in the example) with the corresponding name of the network drive or subdirectory you use. Thus, if the server's copy happened to be available as /homes/doe on server hercules, the reconciliation command would be:

```
reconcile c:\doe hercules:/homes/doe
```

Alternatively, if you are using a Windows file server which exports your working files as \\herc\doe$, you can use the command:

```
reconcile c:\doe \\herc\doe$
```

You could also use a mounted disk letter (h:) instead of \\herc\doe$ after mounting it on the H disk.

Since file servers are usually available overnight, you can schedule periodic automatic reconciliations, preferably at a time when nobody is likely to be using the files involved.

Although in fact there are two complete copies of the files involved, one can think of this as offering a single replicated set of files. The cost is the extra disk storage needed to store copies of all the files. The benefits are availability and performance. Availability is obvious - you don't need the network or file server to work on the local copy. Performance comes from the fact that your computer doesn't need to be in continuous communication with the file server, telling it about every file it updates and checking for updates performed elsewhere. The only overhead is the periodic reconciliation, typically late at night when the computers and network are relatively idle.

Moving files between a workstation and a laptop or home computer

Now suppose you occasionally use a laptop or other second computer, and would like use your files there. You can do this using any medium accessible to both the laptop and the workstation, for example a removable disk or a file server accessible to both. The example below uses the former, assuming a removable disk mounted on z:\doe.

It is convenient to use the same working directory name (c:\doe, in this example) on the laptop as you do on your workstation. Be sure to label the laptop's hard disk uniquely, and then create an empty directory and journal:

```
(Create the removable disk as above and move it to the laptop.)
(First time initialization)
label c: LAPTOP
mkdir c:\doe
reconcile -c c:\doe z:\doe
```

Now you're ready to go. To bring the laptop up to date:

```
(Update the removable disk on the workstation)
reconcile c:\doe z:\doe

(Move the removable disk to the laptop and update the laptop)
reconcile c:\doe z:\doe
```

Notice that you use exactly the same command on both computers, assuming that the local directory names are the same. When you're done with the laptop and want to return to the workstation, simply reverse the process:

```
(Update the removable disk on the laptop)
reconcile c:\doe z:\doe

(Move it back to the workstation, and update the workstation)
reconcile c:\doe z:\doe
```

From now on, you can carry your files back and forth at will. If you happen to forget to run reconcile at the beginning or end of a session or if you leave the disk behind, all is not lost. Reconcile will automatically find the newest version of each file you modify, regardless of where you do so. If you update some file (say oops.doc) differently on both computers, you will get the error message during reconciliation:

```
WARNING: oops#1.doc is a saved version of oops.doc
```

At this point you will probably want to look at both files to find the differences between them and to copy any valuable changes from oops#1.doc into oops.doc. Different applications have different ways to do this: for text files there is a convenient tool named diff or windiff which finds and displays a minimum set of differences. If all else fails you can simply display or print the files and compare them manually. When you are satisfied that you have recovered everything into oops.doc, delete oops#1.doc.

It is OK if you do not resolve the conflict right away. Reconcile keeps a record of unresolved conflicts in its journal file and warns you every time it runs, until you delete or rename the saved file (oops#1.doc.) It's also easy to locate such files by searching for file names containing the # character, which is unusual in normal file names.

File sharing without a server

Both Windows and Unix allow client computers to act as servers, offering other computers access to their local disks. This can be done using a variety of interconnection techniques including networks, modems, and even printer cables. With any such connection, the client can perform a reconciliation against the server. This is convenient for a portable computer. It even works across a dial-in connection provided that the amount of updated information is reasonable. In the general case one can imagine a very loosely coupled network of portable computers which occasionally meet and exchange information without ever needing a central repository.

Backing up files by one-way reconciliations

Reconcile provides several modes to support "one-way" reconciliations, which simply replicate changes from one site to a second. The `-o` mode prevents a source site from being modified. The `-c` mode makes a site be a copy target only, never as the source of a version or other information (but allows re-use of existing information at the site if it matches.) Thus, a strict incremental one-way reconciliation from a primary to a backup site would be done using a command like:

```
reconcile –o primary –c backup
```

The above command will leave a journal in the root directory of the primary site. There is a site mode to place the journal somewhere else, for example something like:

```
reconcile –o –j=/.reconciliations/primary.jnl primary –c backup
```

Command Line Reference

Syntax

```
reconcile [options] [[modes] siteID] ...
reconcile [options] –x=workingdir [modes] siteID]
reconcile {options and sites} –also {options and sites} …
```

Options (these apply to the entire command):

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

`-n` do not update anything; just say what would happen

`-autocreate` automatically create journals (-w flag) for site directories lacking them

`-autocreate=Windows|Unix|Mac` set the system type in autocreated journals

`-ignoredeletions` do not propagate deletions

`-nochown` do not attempt to update owner and group ID's (Unix only)

`-p` pause and wait for dialog input if any errors encountered

`-p=`*`n`* pause for *n* (decimal) seconds before exiting (Windows only, n=10 by default)

`-q` quiet – suppress updating messages

`-v` verbose – display progress messages

`-k=n` keep deleted and replaced files in .trash for n (default 3) days before purging them

`-k` suppress keeping old files in .trash; just delete old files and rename old directories

`-d="`*`directive`*`"` apply *directive* throughout unless overridden

`-l=`*`tracefile`* log actions performed to *tracefile*

`-t=`*`tracefile`* trace details about actions performed to *tracefile*

Note: tracefile may be specified differently for each remote site and all local sites

`-x=`*`workingdir`* identifies this instance of Reconcile as a remote server, responding to commands issued over the remote connection, and optionally sets the working directory for the remote command. "`=`*`workingdir`*" may be omitted, in which case the working directory at the remote site defaults to the directory containing the journal file. A server should be provided with exactly one site. This option is generated automatically by your local machine when you specify a remote site, so you seldom use it explicitly.

`-tag=#` insert # (or other string) followed by digits in file names when renaming due to a version conflict. "#" is the default tag. The tag string should not end with a digit.

Modes (these apply only to the single site they precede on the command line):

`-a` abandon this site, that is, remove references to it in other sites' journals.

`-c` use this site as a target only, re-using existing files if they match, but never copying from this site to another one. –c will create a journal if none exists.

`-c=Windows|Unix|Mac` set the system type of the site.

`-cc` re-use the journal but don't recompute digests. This assumes that the site is maintained *only* by reconcile, so existing files at this site have not changed. This is *not recommended*, as there are too many ways for the journal and supposedly unchanged directory to become inconsistent.

`-w` write a new journal, ignoring the old one entirely.

`-w=Windows|Unix|Mac` set the system type of the site in its new journal.

`-r` read-only journal.

`-o` read-only files.

`-s` use this site as a time reference

`-j=`*`journal`* explicitly sets the journal file name

`-name=`*`sitename`* explicitly sets the site name

Note that a new mode is established for each different site.

Site identification (siteID):

- **Local** sites are usually identified in the program parameters by giving the path to the top-level directory "/path/name". Usually the directory contains the journal file "/path/name/name.jnl" . In this case the name used to identify the site is "label:/path/name" where label is the Windows disk label or Unix host name. Adding such a label insures that the actual site names are unique on each computer or disk.

  The following alternative ways to identify a site in the program parameters are accepted:

  1. The name used to identify the site can be specified explicitly using the mode "`-name=`*`sitename`*".
  2. The journal journal file name can be provided explicitly with the `-j` mode. The journal file name may be either relative to the working directory, or absolute.
  3. The parameter is "/path/name", which is a directory as usual, but the journal file exists in parallel with rather than inside of the directory, i.e. "/path/name.jnl" exists and "/path/name/name.jnl" does not. This form is accepted but not recommended.
  4. The parameter is a file "/path/name.ext", which is the journal file, and the corresponding directory is "/path/name" obtained by stripping off the extension (usually ".jnl"). This form is accepted but not recommended.
  5. For network-mounted disks, reconcile applies the above rules relative to the path name at the server rather than at the client, in order that site names remain consistent regardless of whether or how they are mounted.
  6. In the special case of the root directory of a disk, the disk label (for Windows) or host name (for Unix) is used in place of the empty string for the root directory name.

  Under Windows, the path may begin with either a drive letter (single letter followed by a colon) or a UNC (Universal Naming Convention) reference to a host system, such as "//herc/…". UNC names are treated a locally mounted file systems.

- **Remote** sites are identified in the form "hostname:/ path" where hostname identifies a remote system which accepts the rexec/rlogin protocol and "/path" is a local site name on that host, conforming to the above rules. The hostname must be at least two letters long to avoid confusion with Windows disk letters.

  Anything after the first dot in the hostname is removed for the sake of naming the site; this allows you to give a fully qualified host name for the sake of contacting the remote host without accidentally introducing a different site name for what is in fact the same site.

  If `-j` is used to identify a journal for a remote site, the journal file is a local file of the remote system. For remote sites only, the remote tracefile can be identified using the `-t=` or `-l=` mode before the site specification, then overridden with subsequent `-t=` or `-l=` modes to specify other remote logs. The last `-t=` or `-l=` specification in the command line also names the local tracefile.

Remote sites must be accessible via the rexec or rlogin protocols (as used by the Unix rsh and rlogin commands.) These protocols require a user name and sometimes a password, taken from the environment variables USER and PASSWORD respectively. If the given password is "*" then reconcile prompts interactively for a user name and password.

Given no password or an empty password, reconcile attempts to use the privileged "rexec" protocol, for which you must be running as root on the local machine. (Windows users are always running as root for this purpose.) This requires careful remote system configuration setup. See the remote system's manual pages for the rsh (rexec) and rlogin commands for details, and contact your local system administrator to verify that rexec is allowed by local security policies.

Case is significant for the purpose of naming files or directories under Unix, but is not significant in comparing site names. "/" may be used rather than "\" as a path separator, even under Windows. Internally, reconcile always uses "/" for path separators, translating them to "\" just in time to perform local file system operations.

Regardless of whether a local of global site name is provided, Reconcile identifies sites uniquely using the remote site name syntax, inserting the host name, disk label, or mount point at the beginning of the local directory name as appropriate. The goal of this replacement is to get a consistent unique name for the site regardless of whether it is accessed directly, through a mount point, or remotely. Therefore *it is very important that you label each disk (both fixed and removable) with a unique and meaningful name.*

Multiple reconciliations (-also parameter on command line):

Several distinct reconciliations can be performed sequentially in one command by separating their parameters with the keyword "-also". All options and modes are reset for each reconciliation, just as if each one were done on a separate command line.

Obsolete modes and parameters:

The following option and parameter flags are accepted but may be removed some time in the future and should be avoided:

`-debugname=`*`name.ext`*	generate extra trace output for this file
`-p`*`n`*	equivalent to `–p=`*`n`* (*`n`* is a number)
`-znolocks`	do not lock open files
`-zsbin`	use an alternative ("sized binary") representation for bulk file transfer during remote reconciliations.
`-i`	interactive problem repair after end of reconciliation run
`-j` *`name`* or `-j`*`name`*	equivalent to `–j=`*`name`*
`-l` *`name`* or `-j`*`name`*	equivalent to `–l=`*`name`*
`-t` *`name`* or `-j`*`name`*	equivalent to `–t=`*`name`*
`-c` *`directive`* or `-c`*`directive`*	equivalent to `–c=`*`directive`*

File Formats

Files used by reconcile

The following files may be used by reconcile.

`<site>.jnl` - journal of the contents of directory hierarchy <site>

`<site>.jnw` - new the journal being written (exists temporarily only)

`<site>.jbk` - previous copy of <site>.jnl

`.reconcilerc` - special directives applying to selected subdirectories

`<filename>#nnn[.<ext>]` - saved conflicting version of file <filename>[.<ext>]

`reconcile.log` - default name for trace of actions performed, if not overridden by -l or -t option. These files are created in the current working directory. The trace file contains the various update and error messages generated during reconciliation. If the -t option is used, it also contains detailed information about why reconcile performed the actions.

`Reconcile.oldlog` – saved previous copy of reconcile.log.

`.trash/` – temporary holding subdirectory for files and subdirectories which have been deleted or replaced in the last few days.

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 version of each file.

Header: `Journal of <label>:/<root> (<systype>) - <programID>`

Is the first line of a journal. `<label>` is the disk label (under Windows) or host system name (under Unix), /`<root>` is the fully qualified name of the root directory of the site, `<systype>` is the system type (DOS or UNIX at present), and <`programID`> gives the name and version of the Reconcile program that wrote the journal.

Version: `+ <timestamp> <name><t> <link> ?mask dt=<digest> <remarks>`

Identifies a particular version of a particular file.

<timestamp> is the date and time the action occurred (“yyyy-mm-dd hh:mm:ssZ”, where Z indicates Coordinated Universal Time)

<name> is the file's name, followed directly by the type <t> (no intervening white space) The name may be surrounded by quotation marks if it contains white space or potentially ambiguous characters.

<t> is a character indicating the file’s type:

(nothing) for an ordinary file
: for an ordinary file (used when the file name ends with : / @ or ?)
/ for a subdirectory

@ for a symbolic link
? for unknown type and missing files in the get/put protocol described later

<link>, present only for symbolic links, gives the symbolic link reference.

?mask is a bit-mask indicating which other sites do *not* know about this version yet. The individual mask bits are defined in the known site ($) lines.

dt=<digest> gives a hash code, or digest, of the contents of the file, used to determine if files are identical even though they might have different dates. "dt" means the file looks like a text file, "db" is used for binary files.

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

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

and deleted directories by the special remark:

```
!deleted <original name>
```

Deletion: `- <timestamp> <name><t> ?mask`

Identifies a deleted file, no longer present at the site.

Missing version: `> <timestamp> <name><t> <link> ?mask dt=<digest>`

Identifies a file for which another site has a more recent version, but which could not be updated locally due either to the –o mode or to some problem in the copy operation.

Missing deletion: `< <timestamp> <name><t> ?mask`

Identifies a file for which another site has a more recent deletion, which can not be carried out locally due to the –o mode or some problem.

Known site: `$ <timestamp> <sitename> ?mask`

Identifies some other "known site" associated with this site. The timestamp gives the date and time of the most recent reconciliation for the known site, and the hexadecimal mask defines the mask bit used for this site in version lines.

Directive: `#verb <pattern>`

Directives (described in detail below) may be included directly in journals. If they do appear they are preserved, propagated between journals, and work the same way as they would if they appeared in .reconcilerc. A drawback of placing directives directly into journals is that there is no convenient way of removing them other than by manually editing them out of all sites' journals at the same time.

Subdirectory: `Journal of label:/<root>/<subdirectory>`

After all the entries for a given directory have been listed, the journal file contains sub-journals for each of the sub-directories listed. Each of these consists of a new set of directives and version lines, preceded by a header line identifying the site and full name of the subdirectory path:

End of journal: `End of journal`

The entire journal file is ended by a line containing the literal string “End of journal”.

Directives and .reconcilerc files

Any directory or subdirectory may contain a file named “.reconcilerc”, containing directives which modify processing of that directory and its subdirectories. Comment lines starting with an asterisk are also accepted. Directives apply to matching files in the directory in which they appear and also all subdirectories of that directory.

Directives may also be edited into a journal file, in which case they apply to the current directory or subdirectory defined by the most recent header line. Top level directives can also be provided by the –d option.

Patterns, indicated by <pattern>, are templates for file names, matched according to the following rules:

- Path separators (“/” and “\”) separate component levels in hierarchical file names.
- Asterisks ("*") are wild cards, matching an arbitrary sequences of zero or more characters in a single file name component, not including path separators. Thus, to match all .mbx files within a mail subdirectory, use “mail/*.mbx”.
- Patterns must match the entire file name components at each hierarchical level, so to ignore all files whose names contain the letters “foo” you would use *foo*.
- If several patterns match a file name, the deepest match in the hierarchy is used; within a single level the last match is used. This allows more specific patterns to override more general ones.
- Patterns or parts of patterns may be enclosed in double quotes; a pair of double quote characters within a quoted pattern represents one double quote character. Double quotes are used to include white space and other unconventional characters in file names; they do not override wild cards or other special character meanings. There is no way to escape a wild card or path separator.
- A pattern may be followed by a space and the word “onelevel”, in which case it applies only to the current directory, not to any subdirectories.

Reconcile accepts the following directives:

#ignore <pattern>

causes matching files to be ignored during reconciliation. Reconcile creates the following implicit directives at the beginning:

```
#ignore *.jnl
#ignore *.jbk
#ignore reconcile.log
#ignore reconcile.oldlog
#ignore .trash
```

These can be overridden by #normal directives if needed. Note that the .reconcilerc file itself is *not* automatically ignored but is propagated normally. If you specify a log file name other than "reconcile.log" using the –l or –t options, the corresponding .log and .oldlog files are ignored.

#alwaysnew <pattern>

tells reconcile always to copy the newer version of a file, without preserving the older version even in the case of a detected conflict. *Use with caution! You can lose conflicting updates this way.*

#deletedirs <pattern>

tells reconcile always to propagate directory deletions, rather than renaming the deleted directory and generating a warning message.

#binary <pattern>

disable automatic line-end translation in files matching the pattern. The alternative directive #notranslate is retained for compatibility with previous versions of the program but may be dropped in the future.

#text <pattern>

forces automatic line-end translation in files matching the pattern.

#normal <pattern>

overrides all the above directives, restoring normal handling to files with matching names. Note again that directives from subdirectories override directives from parent directories in a file hierarchy; within a given directory the last matching directive overrides any previous ones, and directives from the .reconcilerc file override directives stored directly in the journal. The top level has the lowest priority and directives from the command line override the implicit #ignore directives listed above.

#mapunc <prefix>[$] <replacement>

maps the file names used to identify sites mounted from file servers into a standard form. This is primarily used to connect the "UNC" names used by Windows file with names used by the Unix "rcp" command and by remote reconciliation. The <prefix> is the beginning of a UNC file name, typically "//server/sharename", and the <replacement> replaces it in directory or journal names used to identify sites. The optional $ at the end of the prefix matches a $ at the end of the sharename rather than at the end of the server name. For example, `//herc/smith$` is mapped to `hercules:/homes/smith` in the example below. Note that forward slashes should be used throughout.

If no explicit #mapunc directives are given, reconcile automatically inserts the following default mappings, which are appropriate for the MERL file server configuration:

```
#mapunc //herc hercules:/homes
#mapunc //herc$ hercules:/homes
#mapunc //herc/homes hercules:/homes
```

```
#mapunc //herc/projects hercules:/projects
#mapunc //herc/collagen hercules:/projects/collagen
```

If #mapunc directives are used they must be appear before they are needed, i.e. before any use of a mounted share or explicit UNC file name. Practically speaking this means they must appear at the beginning of the command line. UNC mappings are only needed in Windows, and are not used at all for local disks, nor when the server name is given explicitly to indicate remote reconciliation. Order is important; put the more general mappings before the more specific ones since the directive matching mechanism searches for a match from last to first.

Remote commands and get/put protocol

If Reconcile is started with the –x option, it functions as a server under the control of a client copy of Reconcile, communicating through the standard input/output streams established by the rsh/rexec protocol. There must be exactly one site in the command line with the `-x` option, and the remote commands all apply to this one site.

Remote commands are formatted as text lines sent from the client to the server:

```
getjournal
putjournal
get <name><t> [was <timestamp>]
put <name><t> [was <timestamp>]
rename <name><t> [was <timestamp>] <newname>
move <name><t> [was <timestamp>] <newname>
delete <name><t> [was <timestamp>]
fixtime <name> <timestamp>
```

In all the above, the syntax of <name>, <t>, and <timestamp> is the same as in journal files. The square brackets indicate optional data. When provided, the current version of the named file must exist and have the given timestamp. When <t> is the character ?, the file being replaced in a put command must not currently exist.

The confirmation for most commands is a line containing of:

```
ACK <verb>
```

However, the getjournal and get commands respond positively with the contents of the journal or the object being gotten, respectively, instead of an ACK. The negative response consists of zero or more error messages followed by

```
NAK <verb>
```

The putjournal command is followed immediately by the contents of the journal. It is allowable to mix other commands in with the putjournal stream. The put command is immediately followed by the contents of the object being put. In both get and put commands, the object is described by a header line, which is one of:

```
File <name><t> <timestamp> mode=xxx uid=n gid=n <enc>
Dir <name><t> <timestamp> mode=xxx uid=n gid=n
Link <name><t> <timestamp> mode=xxx uid=n gid=n <linkname>
```

where <name> redundantly names the file, <t> gives its type, <timestamp> is its timestamp (most recently modified time), the mode, uid, and gid are Unix file protection bits in octal, user ID, and group ID (omitted if zero) respectively, and <enc> gives the encoding for the subsequent file contents. For subdirectories and links the header line is the entire response.

For files, the header is followed by the file, encoded as described by <enc>. At present the preferred encoding is "base64", and the file is represented by the MIME base64 encoding of binary data into printable data. Other encodings are possible in the future.

The end of an encoded file is indicated by a line containing

```
End of file <name>
```