Unison File Synchronizer

User Manual and Reference Guide

Version 2.40.128

Contents

1 Overview

Unison is a file-synchronization tool for Unix and Windows. It allows two replicas of a collection of files and directories to be stored on different hosts (or different disks on the same host), modified separately, and then brought up to date by propagating the changes in each replica to the other.

Unison shares a number of features with tools such as configuration management packages (CVS, PRCS, etc.), distributed filesystems (Coda, etc.), uni-directional mirroring utilities (rsync, etc.), and other synchronizers (Intellisync, Reconcile, etc). However, there are several points where it differs:

- Unison runs on both Windows (95, 98, NT, 2k, and XP) and Unix (OSX, Solaris, Linux, etc.) systems. Moreover, Unison works *across* platforms, allowing you to synchronize a Windows laptop with a Unix server, for example.
- Unlike a distributed filesystem, Unison is a user-level program: there is no need to modify the kernel or to have superuser privileges on either host.
- Unlike simple mirroring or backup utilities, Unison can deal with updates to both replicas of a distributed directory structure. Updates that do not conflict are propagated automatically. Conflicting updates are detected and displayed.
- Unison works between any pair of machines connected to the internet, communicating over either a direct socket link or tunneling over an encrypted `ssh` connection. It is careful with network bandwidth, and runs well over slow links such as PPP connections. Transfers of small updates to large files are optimized using a compression protocol similar to rsync.
- Unison has a clear and precise specification, described below.
- Unison is resilient to failure. It is careful to leave the replicas and its own private structures in a sensible state at all times, even in case of abnormal termination or communication failures.
- Unison is free; full source code is available under the GNU Public License.

2 Preface

2.1 People

Benjamin Pierce leads the Unison project. The current version of Unison was designed and implemented by Trevor Jim, Benjamin Pierce, and Jérôme Vouillon, with Alan Schmitt, Malo Denielou, Zhe Yang, Sylvain Gommier, and Matthieu Goulay. The Mac user interface was started by Trevor Jim and enormously improved by Ben Willmore. Our implementation of the rsync protocol was built by Norman Ramsey and Sylvain Gommier. It is based on Andrew Tridgell's thesis work and inspired by his rsync utility. The mirroring and merging functionality was implemented by Sylvain Roy, improved by Malo Denielou, and improved yet further by Stéphane Lescuyer. Jacques Garrigue contributed the original Gtk version of the user interface; the Gtk2 version was built by Stephen Tse. Sundar Balasubramaniam helped build a prototype implementation of an earlier synchronizer in Java. Insik Shin and Insup Lee contributed design ideas to this implementation. Cedric Fournet contributed to an even earlier prototype.

2.2 Mailing Lists and Bug Reporting

Mailing Lists: Moderated mailing lists are available for bug reporting, announcements of new versions, discussions among users, and discussions among developers. See

`http://www.cis.upenn.edu/~bcpierce/unison/lists.html`

for more information.

2.3 Development Status

Unison is no longer under active development as a research project. (Our research efforts are now focused on a follow-on project called Harmony, described at `http://www.cis.upenn.edu/~bcpierce/harmony`.) At this point, there is no one whose job it is to maintain Unison, fix bugs, or answer questions.

However, the original developers are all still using Unison daily. It will continue to be maintained and supported for the foreseeable future, and we will occasionally release new versions with bug fixes, small improvements, and contributed patches.

Reports of bugs affecting correctness or safety are of interest to many people and will generally get high priority. Other bug reports will be looked at as time permits. Bugs should be reported to the users list at `unison-users@yahoogroups.com`.

Feature requests are welcome, but will probably just be added to the ever-growing todo list. They should also be sent to `unison-users@yahoogroups.com`.

Patches are even more welcome. They should be sent to `unison-hackers@lists.seas.upenn.edu`. (Since safety and robustness are Unison's most important properties, patches will be held to high standards of clear design and clean coding.) If you want to contribute to Unison, start by downloading the developer tarball from the download page. For some details on how the code is organized, etc., see the file `CONTRIB`.

2.4 Copying

This file is part of Unison.

Unison is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Unison is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

The GNU Public License can be found at `http://www.gnu.org/licenses`. A copy is also included in the Unison source distribution in the file `COPYING`.

2.5 Acknowledgements

Work on Unison has been supported by the National Science Foundation under grants CCR-9701826 and ITR-0113226, *Principles and Practice of Synchronization*, and by University of Pennsylvania's Institute for Research in Cognitive Science (IRCS).

3 Installation

Unison is designed to be easy to install. The following sequence of steps should get you a fully working installation in a few minutes. If you run into trouble, you may find the suggestions on the Frequently Asked Questions page[1] helpful. Pre-built binaries are available for a variety of platforms.

Unison can be used with either of two user interfaces:

1. a simple textual interface, suitable for dumb terminals (and running from scripts), and
2. a more sophisticated grapical interface, based on Gtk2 (on Linux/Windows) or the native UI framework (on OSX).

You will need to install a copy of Unison on every machine that you want to synchronize. However, you only need the version with a graphical user interface (if you want a GUI at all) on the machine where you're actually going to display the interface (the *client* machine). Other machines that you synchronize with can get along just fine with the textual version.

3.1 Downloading Unison

The Unison download site lives under `http://www.cis.upenn.edu/~bcpierce/unison`.

If a pre-built binary of Unison is available for the client machine's architecture, just download it and put it somewhere in your search path (if you're going to invoke it from the command line) or on your desktop (if you'll be click-starting it).

The executable file for the graphical version (with a name including `gtkui`) actually provides *both* interfaces: the graphical one appears by default, while the textual interface can be selected by including `-ui text` on the command line. The `textui` executable provides just the textual interface.

If you don't see a pre-built executable for your architecture, you'll need to build it yourself. See Section 3.4 [Building Unison]. There are also a small number of contributed ports to other architectures that are not maintained by us. See the Contributed Ports page[2] to check what's available.

Check to make sure that what you have downloaded is really executable. Either click-start it, or type `unison -version` at the command line.

Unison can be used in three different modes: with different directories on a single machine, with a remote machine over a direct socket connection, or with a remote machine using `ssh` for authentication and secure transfer. If you intend to use the last option, you may need to install `ssh`; see Section A [Installing Ssh].

3.2 Running Unison

Once you've got Unison installed on at least one system, read Section 4 [Tutorial] of the user manual (or type `unison -doc tutorial`) for instructions on how to get started.

3.3 Upgrading

Upgrading to a new version of Unison is as simple as throwing away the old binary and installing the new one.

Before upgrading, it is a good idea to run the *old* version one last time, to make sure all your replicas are completely synchronized. A new version of Unison will sometimes introduce a different format for the archive files used to remember information about the previous state of the replicas. In this case, the old archive will be ignored (not deleted — if you roll back to the previous version of Unison, you will find the old archives intact), which means that any differences between the replicas will show up as conflicts that need to be resolved manually.

[1] `http://www.cis.upenn.edu/~bcpierce/unison/faq.html`
[2] `http://www.cis.upenn.edu/~bcpierce/unison/download.html`

3.4 Building Unison from Scratch

If a pre-built image is not available, you will need to compile it from scratch; the sources are available from the same place as the binaries.

In principle, Unison should work on any platform to which OCaml has been ported and on which the `Unix` module is fully implemented. It has been tested on many flavors of Windows (98, NT, 2000, XP) and Unix (OS X, Solaris, Linux, FreeBSD), and on both 32- and 64-bit architectures.

3.4.1 Unix

You'll need the Objective Caml compiler (version 3.11.2 or later), which is available from `http://caml.inria.fr`. Building and installing OCaml on Unix systems is very straightforward; just follow the instructions in the distribution. You'll probably want to build the native-code compiler in addition to the bytecode compiler, as Unison runs much faster when compiled to native code, but this is not absolutely necessary. (Quick start: on many systems, the following sequence of commands will get you a working and installed compiler: first do `make world opt`, then `su` to root and do `make install`.)

You'll also need the GNU `make` utility, standard on many Unix systems. (Type `make --version` to check that you've got the GNU version.) Unison's build system is not parallelizable, so don't use make flags which cause it to start processes in parallel, e.g. -j for GNU make.

Once you've got OCaml installed, grab a copy of the Unison sources, unzip and untar them, change to the new `unison` directory, and type "`make UISTYLE=text`." The result should be an executable file called `unison`. Type `./unison` to make sure the program is executable. You should get back a usage message.

If you want to build the graphical user interface, you will need to install two additional things:

- The Gtk2 libraries. These areavailable from `http://www.gtk.org` and are standard on many Unix installations.
- The `lablgtk2` OCaml library. Grab the developers' tarball from

  `http://wwwfun.kurims.kyoto-u.ac.jp/soft/olabl/lablgtk.html`,

  untar it, and follow the instructions to build and install it.

  (Quick start: `make configure`, then `make`, then `make opt`, then `su` and `make install`.)

Now build unison. If your search paths are set up correctly, simply typing `make` again should build a `unison` executable with a Gtk2 graphical interface. (In previous releases of Unison, it was necessary to add `UISTYLE=gtk2` to the 'make' command above. This requirement has been removed: the makefile should detect automatically when lablgtk2 is present and set this flag automatically.)

Put the `unison` executable somewhere in your search path, either by adding the Unison directory to your PATH variable or by copying the executable to some standard directory where executables are stored.

3.4.2 Mac OS X

To build the text-only user interface, follow the instructions above for building on Unix systems. You should do this first, even if you are also planning on building the GUI, just to make sure it works.

To build the basic GUI version, you'll first need to download and install the XCode developer tools from Apple. Once this is done, just type `make UISTYLE=macnew` in the `src` directory, and if things go well you should get an application that you can move from `uimacnew/build/Default/Unison.app` to wherever you want it.

There is also an experimental GUI with a somewhat smoother look and feel. To compile this one (once you've got the basic one working), proceed as follows:

1. Go to the `uimacnew09` directory and double-click the file `BWToolkit.ibplugin`.
2. Go back up to the `src` directory and type `make UISTYLE=macnew09`.
3. You should get an application built for you at `uimacnew09/build/Default/Unison.app`.

3.4.3 Windows

Although the binary distribution should work on any version of Windows, some people may want to build Unison from scratch on those systems too.

Bytecode version: The simpler but slower compilation option to build a Unison executable is to build a bytecode version. You need first install Windows version of the OCaml compiler (version 3.07 or later, available from `http://caml.inria.fr`). Then grab a copy of Unison sources and type

```
make NATIVE=false
```

to compile the bytecode. The result should be an executable file called `unison.exe`.

Native version: Building a more efficient, native version of Unison on Windows requires a little more work. See the file `INSTALL.win32` in the source code distribution.

3.4.4 Installation Options

The `Makefile` in the distribution includes several switches that can be used to control how Unison is built. Here are the most useful ones:

- Building with `NATIVE=true` uses the native-code OCaml compiler, yielding an executable that will run quite a bit faster. We use this for building distribution versions.
- Building with `make DEBUGGING=true` generates debugging symbols.
- Building with `make STATIC=true` generates a (mostly) statically linked executable. We use this for building distribution versions, for portability.

4 Tutorial

4.1 Preliminaries

Unison can be used with either of two user interfaces:

1. a straightforward textual interface and
2. a more sophisticated graphical interface

The textual interface is more convenient for running from scripts and works on dumb terminals; the graphical interface is better for most interactive use. For this tutorial, you can use either. If you are running Unison from the command line, just typing `unison` will select either the text or the graphical interface, depending on which has been selected as default when the executable you are running was built. You can force the text interface even if graphical is the default by adding `-ui text`. The other command-line arguments to both versions are identical.

The graphical version can also be run directly by clicking on its icon, but this may require a little set-up (see Section 6.20 [Click-starting Unison]). For this tutorial, we assume that you're starting it from the command line.

Unison can synchronize files and directories on a single machine, or between two machines on a network. (The same program runs on both machines; the only difference is which one is responsible for displaying the user interface.) If you're only interested in a single-machine setup, then let's call that machine the *client*. If you're synchronizing two machines, let's call them *client* and *server*.

4.2 Local Usage

Let's get the client machine set up first and see how to synchronize two directories on a single machine.

Follow the instructions in Section 3 [Installation] to either download or build an executable version of Unison, and install it somewhere on your search path. (If you just want to use the textual user interface, download the appropriate textui binary. If you just want to the graphical interface—or if you will use both interfaces [the gtkui binary actually has both compiled in]—then download the gtkui binary.)

Create a small test directory `a.tmp` containing a couple of files and/or subdirectories, e.g.,

```
mkdir a.tmp
touch a.tmp/a a.tmp/b
mkdir a.tmp/d
touch a.tmp/d/f
```

Copy this directory to b.tmp:

```
cp -r a.tmp b.tmp
```

Now try synchronizing `a.tmp` and `b.tmp`. (Since they are identical, synchronizing them won't propagate any changes, but Unison will remember the current state of both directories so that it will be able to tell next time what has changed.) Type:

```
unison a.tmp b.tmp
```

(You may need to add `-ui text`, depending how your unison binary was built.)

Textual Interface:

You should see a message notifying you that all the files are actually equal and then get returned to the command line.

Graphical Interface:

You should get a big empty window with a message at the bottom notifying you that all files are identical. Choose the Exit item from the File menu to get back to the command line.

Next, make some changes in a.tmp and/or b.tmp. For example:

```
rm a.tmp/a
echo "Hello" > a.tmp/b
echo "Hello" > b.tmp/b
date > b.tmp/c
echo "Hi there" > a.tmp/d/h
echo "Hello there" > b.tmp/d/h
```

Run Unison again:

```
unison a.tmp b.tmp
```

This time, the user interface will display only the files that have changed. If a file has been modified in just one replica, then it will be displayed with an arrow indicating the direction that the change needs to be propagated. For example,

```
<---  new file   c  [f]
```

indicates that the file `c` has been modified only in the second replica, and that the default action is therefore to propagate the new version to the first replica. To **f**ollow Unison's recommendation, press the “f” at the prompt.

If both replicas are modified and their contents are different, then the changes are in conflict: `<-?->` is displayed to indicate that Unison needs guidance on which replica should override the other.

```
new file  <-?->  new file   d/h  []
```

By default, neither version will be propagated and both replicas will remain as they are.

If both replicas have been modified but their new contents are the same (as with the file `b`), then no propagation is necessary and nothing is shown. Unison simply notes that the file is up to date.

These display conventions are used by both versions of the user interface. The only difference lies in the way in which Unison's default actions are either accepted or overridden by the user.

Textual Interface:

The status of each modified file is displayed, in turn. When the copies of a file in the two replicas are not identical, the user interface will ask for instructions as to how to propagate the change. If some default action is indicated (by an arrow), you can simply press Return to go on to the next changed file. If you want to do something different with this file, press “`<`” or “`>`” to force the change to be propagated from right to left or from left to right, or else press “`/`” to skip this file and leave both replicas alone. When it reaches the end of the list of modified files, Unison will ask you one more time whether it should proceed with the updates that have been selected.

When Unison stops to wait for input from the user, pressing “`?`” will always give a list of possible responses and their meanings.

Graphical Interface:

The main window shows all the files that have been modified in either `a.tmp` or `b.tmp`. To override a default action (or to select an action in the case when there is no default), first select the file, either by clicking on its name or by using the up- and down-arrow keys. Then press either the left-arrow or “`<`” key (to cause the version in b.tmp to propagate to a.tmp) or the right-arrow or “`>`” key (which makes the a.tmp version override b.tmp).

Every keyboard command can also be invoked from the menus at the top of the user interface. (Conversely, each menu item is annotated with its keyboard equivalent, if it has one.)

When you are satisfied with the directions for the propagation of changes as shown in the main window, click the “Go” button to set them in motion. A check sign will be displayed next to each filename when the file has been dealt with.

4.3 Remote Usage

Next, we'll get Unison set up to synchronize replicas on two different machines.

Follow the instructions in the Installation section to download or build an executable version of Unison on the server machine, and install it somewhere on your search path. (It doesn't matter whether you install the textual or graphical version, since the copy of Unison on the server doesn't need to display any user interface at all.)

It is important that the version of Unison installed on the server machine is the same as the version of Unison on the client machine. But some flexibility on the version of Unison at the client side can be achieved by using the `-addversionno` option; see Section 6.4 [Preferences].

Now there is a decision to be made. Unison provides two methods for communicating between the client and the server:

- *Remote shell method*: To use this method, you must have some way of invoking remote commands on the server from the client's command line, using a facility such as `ssh`. This method is more convenient (since there is no need to manually start a "unison server" process on the server) and also more secure (especially if you use `ssh`).
- *Socket method*: This method requires only that you can get TCP packets from the client to the server and back. A draconian firewall can prevent this, but otherwise it should work anywhere.

Decide which of these you want to try, and continue with Section 4.4 [Remote Shell Method] or Section 4.5 [Socket Method], as appropriate.

4.4 Remote Shell Method

The standard remote shell facility on Unix systems is `ssh`, which provides the same functionality as the older `rsh` but much better security. Ssh is available from `http://www.openssh.org`. See section A.2 for installation instructions for the Windows version.

Running `ssh` requires some coordination between the client and server machines to establish that the client is allowed to invoke commands on the server; please refer to the `ssh` documentation for information on how to set this up. The examples in this section use `ssh`, but you can substitute `rsh` for `ssh` if you wish.

First, test that we can invoke Unison on the server from the client. Typing

```
ssh remotehostname unison -version
```

should print the same version information as running

```
unison -version
```

locally on the client. If remote execution fails, then either something is wrong with your ssh setup (e.g., "permission denied") or else the search path that's being used when executing commands on the server doesn't contain the `unison` executable (e.g., "command not found").

Create a test directory `a.tmp` in your home directory on the client machine.

Test that the local unison client can start and connect to the remote server. Type

```
unison -testServer a.tmp ssh://remotehostname/a.tmp
```

Now cd to your home directory and type:

```
unison a.tmp ssh://remotehostname/a.tmp
```

The result should be that the entire directory `a.tmp` is propagated from the client to your home directory on the server.

After finishing the first synchronization, change a few files and try synchronizing again. You should see similar results as in the local case.

If your user name on the server is not the same as on the client, you need to specify it on the command line:

```
unison a.tmp ssh://username@remotehostname/a.tmp
```

Notes:

- If you want to put `a.tmp` some place other than your home directory on the remote host, you can give an absolute path for it by adding an extra slash between `remotehostname` and the beginning of the path:

  ```
  unison a.tmp ssh://remotehostname//absolute/path/to/a.tmp
  ```

- You can give an explicit path for the `unison` executable on the server by using the command-line option `-servercmd /full/path/name/of/unison` or adding `servercmd=/full/path/name/of/unison` to your profile (see Section 6.5 [Profile]). Similarly, you can specify a explicit path for the `ssh` program using the `-sshcmd` option. Extra arguments can be passed to `ssh` by setting the `-sshargs` preference.

4.5 Socket Method

> **Warning:** The socket method is insecure: not only are the texts of your changes transmitted over the network in unprotected form, it is also possible for anyone in the world to connect to the server process and read out the contents of your filesystem! (Of course, to do this they must understand the protocol that Unison uses to communicate between client and server, but all they need for this is a copy of the Unison sources.) The socket method is provided only for expert users with specific needs; everyone else should use the `ssh` method.

To run Unison over a socket connection, you must start a Unison daemon process on the server. This process runs continuously, waiting for connections over a given socket from client machines running Unison and processing their requests in turn.

To start the daemon, type

```
unison -socket NNNN
```

on the server machine, where `NNNN` is the socket number that the daemon should listen on for connections from clients. (`NNNN` can be any large number that is not being used by some other program; if `NNNN` is already in use, Unison will exit with an error message.) Note that paths specified by the client will be interpreted relative to the directory in which you start the server process; this behavior is different from the ssh case, where the path is relative to your home directory on the server.

Create a test directory `a.tmp` in your home directory on the client machine. Now type:

unison a.tmp socket://*remotehostname*:NNNN/a.tmp

The result should be that the entire directory `a.tmp` is propagated from the client to the server (`a.tmp` will be created on the server in the directory that the server was started from). After finishing the first synchronization, change a few files and try synchronizing again. You should see similar results as in the local case.

Since the socket method is not used by many people, its functionality is rather limited. For example, the server can only deal with one client at a time.

4.6 Using Unison for All Your Files

Once you are comfortable with the basic operation of Unison, you may find yourself wanting to use it regularly to synchronize your commonly used files. There are several possible ways of going about this:

1. Synchronize your whole home directory, using the Ignore facility (see Section 6.12 [Ignore]) to avoid synchronizing temporary files and things that only belong on one host.

2. Create a subdirectory called `shared` (or `current`, or whatever) in your home directory on each host, and put all the files you want to synchronize into this directory.

3. Create a subdirectory called `shared` (or `current`, or whatever) in your home directory on each host, and put *links to* all the files you want to synchronize into this directory. Use the `follow` preference (see Section 6.13 [Symbolic Links]) to make Unison treat these links as transparent.

4. Make your home directory the root of the synchronization, but tell Unison to synchronize only some of the files and subdirectories within it on any given run. This can be accomplished by using the `-path` switch on the command line:

   ```
   unison /home/username ssh://remotehost//home/username -path shared
   ```

   The `-path` option can be used as many times as needed, to synchronize several files or subdirectories:

   ```
   unison /home/username ssh://remotehost//home/username \
      -path shared \
      -path pub \
      -path .netscape/bookmarks.html
   ```

   These `-path` arguments can also be put in your preference file. See Section 6.4 [Preferences] for an example.

Most people find that they only need to maintain a profile (or profiles) on one of the hosts that they synchronize, since Unison is always initiated from this host. (For example, if you're synchronizing a laptop with a fileserver, you'll probably always run Unison on the laptop.) This is a bit different from the usual situation with asymmetric mirroring programs like `rdist`, where the mirroring operation typically needs to be initiated from the machine with the most recent changes. Section 6.5 [Profile] covers the syntax of Unison profiles, together with some sample profiles.

Some tips on improving Unison's performance can be found on the Frequently Asked Questions page[3].

4.7 Using Unison to Synchronize More Than Two Machines

Unison is designed for synchronizing pairs of replicas. However, it is possible to use it to keep larger groups of machines in sync by performing multiple pairwise synchronizations.

If you need to do this, the most reliable way to set things up is to organize the machines into a "star topology," with one machine designated as the "hub" and the rest as "spokes," and with each spoke machine synchronizing only with the hub. The big advantage of the star topology is that it eliminates the possibility of confusing "spurious conflicts" arising from the fact that a separate archive is maintained by Unison for every pair of hosts that it synchronizes.

4.8 Going Further

On-line documentation for the various features of Unison can be obtained either by typing

```
unison -doc topics
```

at the command line, or by selecting the Help menu in the graphical user interface. The on-line information and the printed manual are essentially identical.

If you use Unison regularly, you should subscribe to one of the mailing lists, to receive announcements of new versions. See Section 2.2 [Mailing Lists].

[3] `http://www.cis.upenn.edu/~bcpierce/unison/faq.html`

5 Basic Concepts

To understand how Unison works, it is necessary to discuss a few straightforward concepts. These concepts are developed more rigorously and at more length in a number of papers, available at `http://www.cis.upenn.edu/~bcpierce/` But the informal presentation here should be enough for most users.

5.1 Roots

A replica's *root* tells Unison where to find a set of files to be synchronized, either on the local machine or on a remote host. For example,

relative/path/of/root

specifies a local root relative to the directory where Unison is started, while

/absolute/path/of/root

specifies a root relative to the top of the local filesystem, independent of where Unison is running. Remote roots can begin with `ssh://`, `rsh://` to indicate that the remote server should be started with rsh or ssh:

```
ssh://remotehost//absolute/path/of/root
rsh://user@remotehost/relative/path/of/root
```

If the remote server is already running (in the socket mode), then the syntax

```
socket://remotehost:portnum//absolute/path/of/root
socket://remotehost:portnum/relative/path/of/root
```

is used to specify the hostname and the port that the client Unison should use to contact it.

The syntax for roots is based on that of URIs (described in RFC 2396). The full grammar is:

```
replica ::= [protocol:]//[user@][host][:port][/path]
         |  path

protocol ::= file
          |  socket
          |  ssh
          |  rsh

user ::= [-_a-zA-Z0-9]+

host ::= [-_a-zA-Z0-9.]+

port ::= [0-9]+
```

When `path` is given without any protocol prefix, the protocol is assumed to be `file:`. Under Windows, it is possible to synchronize with a remote directory using the `file:` protocol over the Windows Network Neighborhood. For example,

```
unison foo //host/drive/bar
```

synchronizes the local directory `foo` with the directory `drive:\bar` on the machine `host`, provided that `host` is accessible via Network Neighborhood. When the `file:` protocol is used in this way, there is no need for a Unison server to be running on the remote host. However, running Unison this way is only a good idea if the remote host is reached by a very fast network connection, since the full contents of every file in the remote replica will have to be transferred to the local machine to detect updates.

The names of roots are *canonized* by Unison before it uses them to compute the names of the corresponding archive files, so `//saul//home/bcpierce/common` and `//saul.cis.upenn.edu/common` will be recognized as the same replica under different names.

5.2 Paths

A *path* refers to a point *within* a set of files being synchronized; it is specified relative to the root of the replica.

Formally, a path is just a sequence of names, separated by `/`. Note that the path separator character is always a forward slash, no matter what operating system Unison is running on. Forward slashes are converted to backslashes as necessary when paths are converted to filenames in the local filesystem on a particular host. (For example, suppose that we run Unison on a Windows system, synchronizing the local root `c:\pierce` with the root `ssh://saul.cis.upenn.edu/home/bcpierce` on a Unix server. Then the path `current/todo.txt` refers to the file `c:\pierce\current\todo.txt` on the client and `/home/bcpierce/current/todo.txt` on the server.)

The empty path (i.e., the empty sequence of names) denotes the whole replica. Unison displays the empty path as "`[root]`."

If `p` is a path and `q` is a path beginning with `p`, then `q` is said to be a *descendant* of `p`. (Each path is also a descendant of itself.)

5.3 What is an Update?

The *contents* of a path `p` in a particular replica could be a file, a directory, a symbolic link, or absent (if `p` does not refer to anything at all in that replica). More specifically:

- If `p` refers to an ordinary file, then the contents of `p` are the actual contents of this file (a string of bytes) plus the current permission bits of the file.
- If `p` refers to a symbolic link, then the contents of `p` are just the string specifying where the link points.
- If `p` refers to a directory, then the contents of `p` are just the token "DIRECTORY" plus the current permission bits of the directory.
- If `p` does not refer to anything in this replica, then the contents of `p` are the token "ABSENT."

Unison keeps a record of the contents of each path after each successful synchronization of that path (i.e., it remembers the contents at the last moment when they were the same in the two replicas).

We say that a path is *updated* (in some replica) if its current contents are different from its contents the last time it was successfully synchronized. Note that whether a path is updated has nothing to do with its last modification time—Unison considers only the contents when determining whether an update has occurred. This means that touching a file without changing its contents will *not* be recognized as an update. A file can even be changed several times and then changed back to its original contents; as long as Unison is only run at the end of this process, no update will be recognized.

What Unison actually calculates is a close approximation to this definition; see Section 5.7 [Caveats and Shortcomings].

5.4 What is a Conflict?

A path is said to be *conflicting* if the following conditions all hold:

1. it has been updated in one replica,
2. it or any of its descendants has been updated in the other replica, and
3. its contents in the two replicas are not identical.

5.5 Reconciliation

Unison operates in several distinct stages:

1. On each host, it compares its archive file (which records the state of each path in the replica when it was last synchronized) with the current contents of the replica, to determine which paths have been updated.

2. It checks for "false conflicts" — paths that have been updated on both replicas, but whose current values are identical. These paths are silently marked as synchronized in the archive files in both replicas.

3. It displays all the updated paths to the user. For updates that do not conflict, it suggests a default action (propagating the new contents from the updated replica to the other). Conflicting updates are just displayed. The user is given an opportunity to examine the current state of affairs, change the default actions for nonconflicting updates, and choose actions for conflicting updates.

4. It performs the selected actions, one at a time. Each action is performed by first transferring the new contents to a temporary file on the receiving host, then atomically moving them into place.

5. It updates its archive files to reflect the new state of the replicas.

5.6 Invariants

Given the importance and delicacy of the job that it performs, it is important to understand both what a synchronizer does under normal conditions and what can happen under unusual conditions such as system crashes and communication failures.

Unison is careful to protect both its internal state and the state of the replicas at every point in this process. Specifically, the following guarantees are enforced:

- At every moment, each path in each replica has either (1) its *original* contents (i.e., no change at all has been made to this path), or (2) its *correct* final contents (i.e., the value that the user expected to be propagated from the other replica).

- At every moment, the information stored on disk about Unison's private state can be either (1) unchanged, or (2) updated to reflect those paths that have been successfully synchronized.

The upshot is that it is safe to interrupt Unison at any time, either manually or accidentally. [Caveat: the above is *almost* true there are occasionally brief periods where it is not (and, because of shortcoming of the Posix filesystem API, cannot be); in particular, when it is copying a file onto a directory or vice versa, it must first move the original contents out of the way. If Unison gets interrupted during one of these periods, some manual cleanup may be required. In this case, a file called `DANGER.README` will be left in your home directory, containing information about the operation that was interrupted. The next time you try to run Unison, it will notice this file and warn you about it.]

If an interruption happens while it is propagating updates, then there may be some paths for which an update has been propagated but which have not been marked as synchronized in Unison's archives. This is no problem: the next time Unison runs, it will detect changes to these paths in both replicas, notice that the contents are now equal, and mark the paths as successfully updated when it writes back its private state at the end of this run.

If Unison is interrupted, it may sometimes leave temporary working files (with suffix `.tmp`) in the replicas. It is safe to delete these files. Also, if the `backups` flag is set, Unison will leave around old versions of files that it overwrites, with names like `file.0.unison.bak`. These can be deleted safely when they are no longer wanted.

Unison is not bothered by clock skew between the different hosts on which it is running. It only performs comparisons between timestamps obtained from the same host, and the only assumption it makes about them is that the clock on each system always runs forward.

If Unison finds that its archive files have been deleted (or that the archive format has changed and they cannot be read, or that they don't exist because this is the first run of Unison on these particular roots), it takes a conservative approach: it behaves as though the replicas had both been completely empty at the point of the last synchronization. The effect of this is that, on the first run, files that exist in only one replica will be propagated to the other, while files that exist in both replicas but are unequal will be marked as conflicting.

Touching a file without changing its contents should never affect whether or not Unison does an update. (When running with the fastcheck preference set to true—the default on Unix systems—Unison uses file modtimes for a quick first pass to tell which files have definitely not changed; then, for each file that might have changed, it computes a fingerprint of the file's contents and compares it against the last-synchronized contents. Also, the `-times` option allows you to synchronize file times, but it does not cause identical files to be changed; Unison will only modify the file times.)

It is safe to "brainwash" Unison by deleting its archive files *on both replicas*. The next time it runs, it will assume that all the files it sees in the replicas are new.

It is safe to modify files while Unison is working. If Unison discovers that it has propagated an out-of-date change, or that the file it is updating has changed on the target replica, it will signal a failure for that file. Run Unison again to propagate the latest change.

Changes to the ignore patterns from the user interface (e.g., using the 'i' key) are immediately reflected in the current profile.

5.7 Caveats and Shortcomings

Here are some things to be careful of when using Unison.

- In the interests of speed, the update detection algorithm may (depending on which OS architecture that you run Unison on) actually use an approximation to the definition given in Section 5.3 [What is an Update?].

  In particular, the Unix implementation does not compare the actual contents of files to their previous contents, but simply looks at each file's inode number and modtime; if neither of these have changed, then it concludes that the file has not been changed.

  Under normal circumstances, this approximation is safe, in the sense that it may sometimes detect "false updates" but will never miss a real one. However, it is possible to fool it, for example by using `retouch` to change a file's modtime back to a time in the past.

- If you synchronize between a single-user filesystem and a shared Unix server, you should pay attention to your permission bits: by default, Unison will synchronize permissions verbatim, which may leave group-writable files on the server that could be written over by a lot of people.

  You can control this by setting your `umask` on both computers to something like 022, masking out the "world write" and "group write" permission bits.

  Unison does not synchronize the `setuid` and `setgid` bits, for security.

- The graphical user interface is single-threaded. This means that if Unison is performing some long-running operation, the display will not be repainted until it finishes. We recommend not trying to do anything with the user interface while Unison is in the middle of detecting changes or propagating files.

- Unison does not understand hard links.

- It is important to be a little careful when renaming directories containing `ignore`d files.

  For example, suppose Unison is synchronizing directory A between the two machines called the "local" and the "remote" machine; suppose directory A contains a subdirectory D; and suppose D on the local machine contains a file or subdirectory P that matches an ignore directive in the profile used to synchronize. Thus path A/D/P exists on the local machine but not on the remote machine.

  If D is renamed to D' on the remote machine, and this change is propagated to the local machine, all such files or subdirectories P will be deleted. This is because Unison sees the rename as a delete and a separate create: it deletes the old directory (including the ignored files) and creates a new one (*not* including the ignored files, since they are completely invisible to it).

6 Reference Guide

This section covers the features of Unison in detail.

6.1 Running Unison

There are several ways to start Unison.

- Typing "`unison` *`profile`*" on the command line. Unison will look for a file *`profile`*`.prf` in the `.unison` directory. If this file does not specify a pair of roots, Unison will prompt for them and add them to the information specified by the profile.
- Typing "`unison` *`profile root1 root2`*" on the command line. In this case, Unison will use *`profile`*, which should not contain any `root` directives.
- Typing "`unison` *`root1 root2`*" on the command line. This has the same effect as typing "`unison default` *`root1 root2`*."
- Typing just "`unison`" (or invoking Unison by clicking on a desktop icon). In this case, Unison will ask for the profile to use for synchronization (or create a new one, if necessary).

6.2 The `.unison` Directory

Unison stores a variety of information in a private directory on each host. If the environment variable `UNISON` is defined, then its value will be used as the name of this directory. If `UNISON` is not defined, then the name of the directory depends on which operating system you are using. In Unix, the default is to use `$HOME/.unison`. In Windows, if the environment variable `USERPROFILE` is defined, then the directory will be `$USERPROFILE\.unison`; otherwise if `HOME` is defined, it will be `$HOME\.unison`; otherwise, it will be `c:\.unison`.

The archive file for each replica is found in the `.unison` directory on that replica's host. Profiles (described below) are always taken from the `.unison` directory on the client host.

Note that Unison maintains a completely different set of archive files for each pair of roots.

We do not recommend synchronizing the whole `.unison` directory, as this will involve frequent propagation of large archive files. It should be safe to do it, though, if you really want to. Synchronizing just the profile files in the `.unison` directory is definitely OK.

6.3 Archive Files

The name of the archive file on each replica is calculated from

- the *canonical names* of all the hosts (short names like `saul` are converted into full addresses like `saul.cis.upenn.edu`),
- the paths to the replicas on all the hosts (again, relative pathnames, symbolic links, etc. are converted into full, absolute paths), and
- an internal version number that is changed whenever a new Unison release changes the format of the information stored in the archive.

This method should work well for most users. However, it is occasionally useful to change the way archive names are generated. Unison provides two ways of doing this.

The function that finds the canonical hostname of the local host (which is used, for example, in calculating the name of the archive file used to remember which files have been synchronized) normally uses the `gethostname` operating system call. However, if the environment variable `UNISONLOCALHOSTNAME` is set, its value will be used instead. This makes it easier to use Unison in situations where a machine's name changes frequently (e.g., because it is a laptop and gets moved around a lot).

A more powerful way of changing archive names is provided by the `rootalias` preference. The preference file may contain any number of lines of the form:

```
rootalias = //hostnameA//path-to-replicaA -> //hostnameB/path-to-replicaB
```

When calculating the name of the archive files for a given pair of roots, Unison replaces any root that matches the left-hand side of any rootalias rule by the corresponding right-hand side.

So, if you need to relocate a root on one of the hosts, you can add a rule of the form:

```
rootalias = //new-hostname//new-path -> //old-hostname/old-path
```

Note that root aliases are case-sensitive, even on case-insensitive file systems.

Warning: The `rootalias` option is dangerous and should only be used if you are sure you know what you're doing. In particular, it should only be used if you are positive that either (1) both the original root and the new alias refer to the same set of files, or (2) the files have been relocated so that the original name is now invalid and will never be used again. (If the original root and the alias refer to different sets of files, Unison's update detector could get confused.) After introducing a new `rootalias`, it is a good idea to run Unison a few times interactively (with the `batch` flag off, etc.) and carefully check that things look reasonable—in particular, that update detection is working as expected.

6.4 Preferences

Many details of Unison's behavior are configurable by user-settable "preferences."

Some preferences are boolean-valued; these are often called *flags*. Others take numeric or string arguments, indicated in the preferences list by `n` or `xxx`. Most of the string preferences can be given several times; the arguments are accumulated into a list internally.

There are two ways to set the values of preferences: temporarily, by providing command-line arguments to a particular run of Unison, or permanently, by adding commands to a *profile* in the `.unison` directory on the client host. The order of preferences (either on the command line or in preference files) is not significant. On the command line, preferences and other arguments (the profile name and roots) can be intermixed in any order.

To set the value of a preference `p` from the command line, add an argument `-p` (for a boolean flag) or `-p n` or `-p xxx` (for a numeric or string preference) anywhere on the command line. To set a boolean flag to `false` on the command line, use `-p=false`.

Here are all the preferences supported by Unison. This list can be obtained by typing `unison -help`.

```
Usage: unison [options]
    or unison root1 root2 [options]
    or unison profilename [options]

Basic options:
 -auto               automatically accept default (nonconflicting) actions
 -batch              batch mode: ask no questions at all
 -doc xxx            show documentation ('-doc topics' lists topics)
 -fat                use appropriate options for FAT filesystems
 -group              synchronize group attributes
 -ignore xxx         add a pattern to the ignore list
 -ignorenot xxx      add a pattern to the ignorenot list
 -nocreation xxx     prevent file creations on one replica
 -nodeletion xxx     prevent file deletions on one replica
 -noupdate xxx       prevent file updates and deletions on one replica
 -owner              synchronize owner
 -path xxx           path to synchronize
 -perms n            part of the permissions which is synchronized
 -root xxx           root of a replica (should be used exactly twice)
 -silent             print nothing except error messages
 -terse              suppress status messages
 -testserver         exit immediately after the connection to the server
```

```
 -times             synchronize modification times
 -version           print version and exit

Advanced options:
 -addprefsto xxx    file to add new prefs to
 -addversionno      add version number to name of unison on server
 -backup xxx        add a pattern to the backup list
 -backupcurr xxx    add a pattern to the backupcurr list
 -backupcurrnot xxx add a pattern to the backupcurrnot list
 -backupdir xxx     directory for storing centralized backups
 -backuploc xxx     where backups are stored ('local' or 'central')
 -backupnot xxx     add a pattern to the backupnot list
 -backupprefix xxx  prefix for the names of backup files
 -backups           keep backup copies of all files (see also 'backup')
 -backupsuffix xxx  a suffix to be added to names of backup files
 -confirmbigdel     ask about whole-replica (or path) deletes (default true)
 -confirmmerge      ask for confirmation before commiting results of a merge
 -contactquietly    suppress the 'contacting server' message during startup
 -copymax n         maximum number of simultaneous copyprog transfers
 -copyprog xxx      external program for copying large files
 -copyprogrest xxx  variant of copyprog for resuming partial transfers
 -copyquoterem xxx  add quotes to remote file name for copyprog (true/false/default)
 -copythreshold n   use copyprog on files bigger than this (if >=0, in Kb)
 -debug xxx         debug module xxx ('all' -> everything, 'verbose' -> more)
 -diff xxx          set command for showing differences between files
 -dontchmod         when set, never use the chmod system call
 -dumbtty           do not change terminal settings in text UI (default true)
 -fastcheck xxx     do fast update detection (true/false/default)
 -follow xxx        add a pattern to the follow list
 -force xxx         force changes from this replica to the other
 -forcepartial xxx  add a pattern to the forcepartial list
 -halfduplex        force half-duplex communication with the server
 -height n          height (in lines) of main window in graphical interface
 -host xxx          bind the socket to this host name in server socket mode
 -ignorearchives    ignore existing archive files
 -ignorecase xxx    identify upper/lowercase filenames (true/false/default)
 -ignoreinodenumbers ignore inode number changes when detecting updates
 -ignorelocks       ignore locks left over from previous run (dangerous!)
 -immutable xxx     add a pattern to the immutable list
 -immutablenot xxx  add a pattern to the immutablenot list
 -key xxx           define a keyboard shortcut for this profile (in some UIs)
 -killserver        kill server when done (even when using sockets)
 -label xxx         provide a descriptive string label for this profile
 -links xxx         allow the synchronization of symbolic links (true/false/default)
 -log               record actions in logfile (default true)
 -logfile xxx       logfile name
 -maxbackups n      number of backed up versions of a file
 -maxerrors n       maximum number of errors before a directory transfer is aborted
 -maxthreads n      maximum number of simultaneous file transfers
 -merge xxx         add a pattern to the merge list
 -mountpoint xxx    abort if this path does not exist
 -nocreationpartial xxx add a pattern to the nocreationpartial list
 -nodeletionpartial xxx add a pattern to the nodeletionpartial list
```

```
-noupdatepartial xxx add a pattern to the noupdatepartial list
-numericids        don't map uid/gid values by user/group names
-prefer xxx        choose this replica's version for conflicting changes
-preferpartial xxx add a pattern to the preferpartial list
-repeat xxx        synchronize repeatedly (text interface only)
-retry n           re-try failed synchronizations N times (text ui only)
-rootalias xxx     register alias for canonical root names
-rsrc xxx          synchronize resource forks (true/false/default)
-rsync             activate the rsync transfer mode (default true)
-selftest          run internal tests and exit
-servercmd xxx     name of unison executable on remote server
-showarchive       show 'true names' (for rootalias) of roots and archive
-socket xxx        act as a server on a socket
-sortbysize        list changed files by size, not name
-sortfirst xxx     add a pattern to the sortfirst list
-sortlast xxx      add a pattern to the sortlast list
-sortnewfirst      list new before changed files
-sshargs xxx       other arguments (if any) for remote shell command
-sshcmd xxx        path to the ssh executable
-stream            use a streaming protocol for transferring file contents (default true)
-ui xxx            select UI ('text' or 'graphic'); command-line only
-unicode xxx       assume Unicode encoding in case insensitive mode
-xferbycopying     optimize transfers using local copies (default true)
```

Here, in more detail, is what they do. Many are discussed in greater detail in other sections of the manual.

addprefsto `xxx` By default, new preferences added by Unison (e.g., new `ignore` clauses) will be appended to whatever preference file Unison was told to load at the beginning of the run. Setting the preference `addprefsto` *`filename`* makes Unison add new preferences to the file named *`filename`* instead.

addversionno When this flag is set to `true`, Unison will use `unison-`*`currentversionnumber`* instead of just `unison` as the remote server command. This allows multiple binaries for different versions of unison to coexist conveniently on the same server: whichever version is run on the client, the same version will be selected on the server.

auto When set to `true`, this flag causes the user interface to skip asking for confirmations on non-conflicting changes. (More precisely, when the user interface is done setting the propagation direction for one entry and is about to move to the next, it will skip over all non-conflicting entries and go directly to the next conflict.)

backup `xxx` Including the preference `-backup` *`pathspec`* causes Unison to keep backup files for each path that matches *`pathspec`*. These backup files are kept in the directory specified by the `backuplocation` preference. The backups are named according to the `backupprefix` and `backupsuffix` preferences. The number of versions that are kept is determined by the `maxbackups` preference.

The syntax of *`pathspec`* is described in Section 6.11 [Path Specification].

backupcurr `xxx` Including the preference `-backupcurr` *`pathspec`* causes Unison to keep a backup of the *current* version of every file matching *`pathspec`*. This file will be saved as a backup with version number 000. Such backups can be used as inputs to external merging programs, for instance. See the documentatation for the `merge` preference. For more details, see Section 6.8 [Merging Conflicting Versions].

The syntax of *`pathspec`* is described in Section 6.11 [Path Specification].

backupcurrnot `xxx` Exceptions to `backupcurr`, like the `ignorenot` preference.

backupdir `xxx` If this preference is set, Unison will use it as the name of the directory used to store backup files specified by the `backup` preference, when `backuplocation` is set to `central`. It is checked *after* the `UNISONBACKUPDIR` environment variable.

backuploc `xxx` This preference determines whether backups should be kept locally, near the original files, or in a central directory specified by the `backupdir` preference. If set to `local`, backups will be kept in the same directory as the original files, and if set to `central`, `backupdir` will be used instead.

backupnot `xxx` The values of this preference specify paths or individual files or regular expressions that should *not* be backed up, even if the `backup` preference selects them—i.e., it selectively overrides `backup`. The same caveats apply here as with `ignore` and `ignorenot`.

backupprefix `xxx` When a backup for a file `NAME` is created, it is stored in a directory specified by `backuplocation`, in a file called `backupprefixNAMEbackupsuffix`. `backupprefix` can include a directory name (causing Unison to keep all backup files for a given directory in a subdirectory with this name), and both `backupprefix` and `backupsuffix` can contain the string *`$VERSION`*, which will be replaced by the *age* of the backup (1 for the most recent, 2 for the second most recent, and so on...). This keyword is ignored if it appears in a directory name in the prefix; if it does not appear anywhere in the prefix or the suffix, it will be automatically placed at the beginning of the suffix.

One thing to be careful of: If the `backuploc` preference is set to `local`, Unison will automatically ignore *all* files whose prefix and suffix match `backupprefix` and `backupsuffix`. So be careful to choose values for these preferences that are sufficiently different from the names of your real files.

backups Setting this flag to true is equivalent to setting `backuplocation` to `local` and `backup` to `Name *`.

backupsuffix `xxx` See `backupprefix` for full documentation.

batch When this is set to `true`, the user interface will ask no questions at all. Non-conflicting changes will be propagated; conflicts will be skipped.

confirmbigdel When this is set to `true`, Unison will request an extra confirmation if it appears that the entire replica has been deleted, before propagating the change. If the `batch` flag is also set, synchronization will be aborted. When the `path` preference is used, the same confirmation will be requested for top-level paths. (At the moment, this flag only affects the text user interface.) See also the `mountpoint` preference.

confirmmerge Setting this preference causes both the text and graphical interfaces to ask the user if the results of a merge command may be commited to the replica or not. Since the merge command works on temporary files, the user can then cancel all the effects of applying the merge if it turns out that the result is not satisfactory. In batch-mode, this preference has no effect. Default is false.

contactquietly If this flag is set, Unison will skip displaying the 'Contacting server' message (which some users find annoying) during startup.

copymax `n` A number indicating how many instances of the external copying utility Unison is allowed to run simultaneously (default to 1).

copyprog `xxx` A string giving the name of an external program that can be used to copy large files efficiently (plus command-line switches telling it to copy files in-place). The default setting invokes `rsync` with appropriate options—most users should not need to change it.

copyprogrest `xxx` A variant of `copyprog` that names an external program that should be used to continue the transfer of a large file that has already been partially transferred. Typically, `copyprogrest` will just be `copyprog` with one extra option (e.g., `--partial`, for rsync). The default setting invokes `rsync` with appropriate options—most users should not need to change it.

copyquoterem `xxx` When set to `true`, this flag causes Unison to add an extra layer of quotes to the remote path passed to the external copy program. This is needed by rsync, for example, which internally uses an ssh connection requiring an extra level of quoting for paths containing spaces. When this flag is set to `default`, extra quotes are added if the value of `copyprog` contains the string `rsync`.

copythreshold `n` A number indicating above what filesize (in kilobytes) Unison should use the external copying utility specified by `copyprog`. Specifying 0 will cause *all* copies to use the external program; a negative number will prevent any files from using it. The default is -1. See Section 6.17 [Making Unison Faster on Large Files] for more information.

debug `xxx` This preference is used to make Unison print various sorts of information about what it is doing internally on the standard error stream. It can be used many times, each time with the name of a module for which debugging information should be printed. Possible arguments for `debug` can be found by looking for calls to `Util.debug` in the sources (using, e.g., `grep`). Setting `-debug all` causes information from *all* modules to be printed (this mode of usage is the first one to try, if you are trying to understand something that Unison seems to be doing wrong); `-debug verbose` turns on some additional debugging output from some modules (e.g., it will show exactly what bytes are being sent across the network).

diff `xxx` This preference can be used to control the name and command-line arguments of the system utility used to generate displays of file differences. The default is '`diff -u CURRENT2 CURRENT1`'. If the value of this preference contains the substrings CURRENT1 and CURRENT2, these will be replaced by the names of the files to be diffed. If not, the two filenames will be appended to the command. In both cases, the filenames are suitably quoted.

doc `xxx` The command-line argument `-doc` *`secname`* causes unison to display section *`secname`* of the manual on the standard output and then exit. Use `-doc all` to display the whole manual, which includes exactly the same information as the printed and HTML manuals, modulo formatting. Use `-doc topics` to obtain a list of the names of the various sections that can be printed.

dontchmod By default, Unison uses the 'chmod' system call to set the permission bits of files after it has copied them. But in some circumstances (and under some operating systems), the chmod call always fails. Setting this preference completely prevents Unison from ever calling chmod.

dumbtty When set to `true`, this flag makes the text mode user interface avoid trying to change any of the terminal settings. (Normally, Unison puts the terminal in 'raw mode', so that it can do things like overwriting the current line.) This is useful, for example, when Unison runs in a shell inside of Emacs.

When `dumbtty` is set, commands to the user interface need to be followed by a carriage return before Unison will execute them. (When it is off, Unison recognizes keystrokes as soon as they are typed.)

This preference has no effect on the graphical user interface.

dumparchives When this preference is set, Unison will create a file unison.dump on each host, containing a text summary of the archive, immediately after loading it.

fastcheck `xxx` When this preference is set to `true`, Unison will use the modification time and length of a file as a 'pseudo inode number' when scanning replicas for updates, instead of reading the full contents of every file. Under Windows, this may cause Unison to miss propagating an update if the modification time and length of the file are both unchanged by the update. However, Unison will never *overwrite* such an update with a change from the other replica, since it always does a safe check for updates just before propagating a change. Thus, it is reasonable to use this switch under Windows most of the time and occasionally run Unison once with `fastcheck` set to `false`, if you are worried that Unison may have overlooked an update. For backward compatibility, `yes`, `no`, and `default` can be used in place of `true`, `false`, and `auto`. See Section 6.18 [Fast Checking] for more information.

fat When this is set to `true`, Unison will use appropriate options to synchronize efficiently and without error a replica located on a FAT filesystem on a non-Windows machine: do not synchronize permissions (`perms = 0`); never use chmod (t dontchmod = true); treat filenames as case insensitive (`ignorecase = true`); do not attempt to synchronize symbolic links (`links = false`); ignore inode number changes when detecting updates (`ignoreinodenumbers = true`). Any of these change can be overridden by explicitely setting the corresponding preference in the profile.

follow `xxx` Including the preference `-follow` ***`pathspec`*** causes Unison to treat symbolic links matching ***`pathspec`*** as 'invisible' and behave as if the object pointed to by the link had appeared literally at this position in the replica. See Section 6.13 [Symbolic Links] for more details. The syntax of ***`pathspec`*** is described in Section 6.11 [Path Specification].

force `xxx` Including the preference `-force` *`root`* causes Unison to resolve all differences (even non-conflicting changes) in favor of *`root`*. This effectively changes Unison from a synchronizer into a mirroring utility.

You can also specify `-force newer` (or `-force older`) to force Unison to choose the file with the later (earlier) modtime. In this case, the `-times` preference must also be enabled.

This preference is overridden by the `forcepartial` preference.

This preference should be used only if you are *sure* you know what you are doing!

forcepartial `xxx` Including the preference `forcepartial =` *`PATHSPEC`* `->` *`root`* causes Unison to resolve all differences (even non-conflicting changes) in favor of *`root`* for the files in *`PATHSPEC`* (see Section 6.11 [Path Specification] for more information). This effectively changes Unison from a synchronizer into a mirroring utility.

You can also specify `forcepartial PATHSPEC -> newer` (or `forcepartial PATHSPEC older`) to force Unison to choose the file with the later (earlier) modtime. In this case, the `-times` preference must also be enabled.

This preference should be used only if you are *sure* you know what you are doing!

group When this flag is set to `true`, the group attributes of the files are synchronized. Whether the group names or the group identifiers are synchronized depends on the preference `numerids`.

halfduplex When this flag is set to `true`, Unison network communication is forced to be half duplex (the client and the server never simultaneously emit data). If you experience unstabilities with your network link, this may help. The communication is always half-duplex when synchronizing with a Windows machine due to a limitation of Unison current implementation that could result in a deadlock.

height `n` Used to set the height (in lines) of the main window in the graphical user interface.

ignore `xxx` Including the preference `-ignore` ***`pathspec`*** causes Unison to completely ignore paths that match ***`pathspec`*** (as well as their children). This is useful for avoiding synchronizing temporary files, object files, etc. The syntax of ***`pathspec`*** is described in Section 6.11 [Path Specification], and further details on ignoring paths is found in Section 6.12 [Ignoring Paths].

ignorearchives When this preference is set, Unison will ignore any existing archive files and behave as though it were being run for the first time on these replicas. It is not a good idea to set this option in a profile: it is intended for command-line use.

ignorecase `xxx` When set to `true`, this flag causes Unison to treat filenames as case insensitive—i.e., files in the two replicas whose names differ in (upper- and lower-case) 'spelling' are treated as the same file. When the flag is set to `false`, Unison will treat all filenames as case sensitive. Ordinarily, when the flag is set to `default`, filenames are automatically taken to be case-insensitive if either host is running Windows or OSX. In rare circumstances it may be useful to set the flag manually.

ignoreinodenumbers When set to true, this preference makes Unison not take advantage of inode numbers during fast update detection. This switch should be used with care, as it is less safe than the standard update detection method, but it can be useful with filesystems which do not support inode numbers.

ignorelocks When this preference is set, Unison will ignore any lock files that may have been left over from a previous run of Unison that was interrupted while reading or writing archive files; by default, when Unison sees these lock files it will stop and request manual intervention. This option should be set only if you are *positive* that no other instance of Unison might be concurrently accessing the same archive files (e.g., because there was only one instance of unison running and it has just crashed or you have just killed it). It is probably not a good idea to set this option in a profile: it is intended for command-line use.

ignorenot `xxx` This preference overrides the preference `ignore`. It gives a list of patterns (in the same format as `ignore`) for paths that should definitely *not* be ignored, whether or not they happen to match one of the `ignore` patterns.

Note that the semantics of `ignore` and `ignorenot` is a little counter-intuitive. When detecting updates, Unison examines paths in depth-first order, starting from the roots of the replicas and working downwards. Before examining each path, it checks whether it matches `ignore` and does not match `ignorenot`; in this case it skips this path *and all its descendants.* This means that, if some parent of a given path matches an `ignore` pattern, then it will be skipped even if the path itself matches an `ignorenot` pattern. In particular, putting `ignore = Path *` in your profile and then using `ignorenot` to select particular paths to be synchronized will not work. Instead, you should use the `path` preference to choose particular paths to synchronize.

immutable `xxx` This preference specifies paths for directories whose immediate children are all immutable files — i.e., once a file has been created, its contents never changes. When scanning for updates, Unison does not check whether these files have been modified; this can speed update detection significantly (in particular, for mail directories).

immutablenot `xxx` This preference overrides `immutable`.

key `xxx` Used in a profile to define a numeric key (0-9) that can be used in the graphical user interface to switch immediately to this profile.

killserver When set to `true`, this flag causes Unison to kill the remote server process when the synchronization is finished. This behavior is the default for `ssh` connections, so this preference is not normally needed when running over `ssh`; it is provided so that socket-mode servers can be killed off after a single run of Unison, rather than waiting to accept future connections. (Some users prefer to start a remote socket server for each run of Unison, rather than leaving one running all the time.)

label `xxx` Used in a profile to provide a descriptive string documenting its settings. (This is useful for users that switch between several profiles, especially using the 'fast switch' feature of the graphical user interface.)

links `xxx` When set to `true`, this flag causes Unison to synchronize symbolic links. When the flag is set to `false`, symbolic links will result in an error during update detection. Ordinarily, when the flag is set to `default`, symbolic links are synchronized except when one of the hosts is running Windows. In rare circumstances it may be useful to set the flag manually.

log When this flag is set, Unison will log all changes to the filesystems on a file.

logfile `xxx` By default, logging messages will be appended to the file `unison.log` in your HOME directory. Set this preference if you prefer another file.

maxbackups `n` This preference specifies the number of backup versions that will be kept by unison, for each path that matches the predicate `backup`. The default is 2.

maxerrors `n` This preference controls after how many errors Unison aborts a directory transfer. Setting it to a large number allows Unison to transfer most of a directory even when some files fail to be copied. The default is 1. If the preference is set too high, Unison may take a long time to abort in case of repeated failures (for instance, when the disk is full).

maxthreads `n` This preference controls how much concurrency is allowed during the transport phase. Normally, it should be set reasonably high to maximize performance, but when Unison is used over a low-bandwidth link it may be helpful to set it lower (e.g. to 1) so that Unison doesn't soak up all the available bandwidth. The default is the special value 0, which mean 20 threads when file content streaming is desactivated and 1000 threads when it is activated.

merge `xxx` This preference can be used to run a merge program which will create a new version for each of the files and the backup, with the last backup and the both replicas. Setting the `merge` preference for a path will also cause this path to be backed up, just like t backup. The syntax of *`pathspec>cmd`* is described in Section 6.11 [Path Specification], and further details on Merging functions are present in Section 6.8 [Merging files].

mountpoint `xxx` Including the preference `-mountpoint PATH` causes Unison to double-check, at the end of update detection, that `PATH` exists and abort if it does not. This is useful when Unison is used to synchronize removable media. This preference can be given more than once. See Section 6.19 [Mount Points].

nocreation `xxx` Including the preference `-nocreation` *`root`* prevents Unison from performing any file creation on root *`root`*.

This preference can be included twice, once for each root, if you want to prevent any creation.

nocreationpartial `xxx` Including the preference `nocreationpartial =` *`PATHSPEC`* `->` *`root`* prevents Unison from performing any file creation in *`PATHSPEC`* on root *`root`* (see Section 6.11 [Path Specification] for more information). It is recommended to use `BelowPath` patterns when selecting a directory and all its contents.

nodeletion `xxx` Including the preference `-nodeletion` *`root`* prevents Unison from performing any file deletion on root *`root`*.

This preference can be included twice, once for each root, if you want to prevent any deletion.

nodeletionpartial `xxx` Including the preference `nodeletionpartial =` *`PATHSPEC`* `->` *`root`* prevents Unison from performing any file deletion in *`PATHSPEC`* on root *`root`* (see Section 6.11 [Path Specification] for more information). It is recommended to use `BelowPath` patterns when selecting a directory and all its contents.

noupdate `xxx` Including the preference `-noupdate` *`root`* prevents Unison from performing any file update or deletion on root *`root`*.

This preference can be included twice, once for each root, if you want to prevent any update.

noupdatepartial `xxx` Including the preference `noupdatepartial =` *`PATHSPEC`* `->` *`root`* prevents Unison from performing any file update or deletion in *`PATHSPEC`* on root *`root`* (see Section 6.11 [Path Specification] for more information). It is recommended to use `BelowPath` patterns when selecting a directory and all its contents.

numericids When this flag is set to `true`, groups and users are synchronized numerically, rather than by name.

The special uid 0 and the special group 0 are never mapped via user/group names even if this preference is not set.

owner When this flag is set to `true`, the owner attributes of the files are synchronized. Whether the owner names or the owner identifiers are synchronizeddepends on the preference `numerids`.

path `xxx` When no `path` preference is given, Unison will simply synchronize the two entire replicas, beginning from the given pair of roots. If one or more `path` preferences are given, then Unison will synchronize only these paths and their children. (This is useful for doing a fast sync of just one directory, for example.) Note that `path` preferences are intepreted literally—they are not regular expressions.

perms `n` The integer value of this preference is a mask indicating which permission bits should be synchronized. It is set by default to $0o1777$: all bits but the set-uid and set-gid bits are synchronised (synchronizing theses latter bits can be a security hazard). If you want to synchronize all bits, you can set the value of this preference to -1. If one of the replica is on a FAT [Windows] filesystem, you should consider using the t fat preference instead of this preference. If you need Unison not to set permissions at all, set the value of this preference to 0 and set the preference t dontchmod to t true.

prefer `xxx` Including the preference `-prefer` *`root`* causes Unison always to resolve conflicts in favor of *`root`*, rather than asking for guidance from the user. (The syntax of *`root`* is the same as for the `root` preference, plus the special values `newer` and `older`.)

This preference is overridden by the `preferpartial` preference.

This preference should be used only if you are *sure* you know what you are doing!

preferpartial `xxx` Including the preference `preferpartial =` *`PATHSPEC`* `->` *`root`* causes Unison always to resolve conflicts in favor of *`root`*, rather than asking for guidance from the user, for the files in *`PATHSPEC`* (see Section 6.11 [Path Specification] for more information). (The syntax of *`root`* is the same as for the `root` preference, plus the special values `newer` and `older`.)

This preference should be used only if you are *sure* you know what you are doing!

repeat `xxx` Setting this preference causes the text-mode interface to synchronize repeatedly, rather than doing it just once and stopping. If the argument is a number, Unison will pause for that many seconds before beginning again.

retry `n` Setting this preference causes the text-mode interface to try again to synchronize updated paths where synchronization fails. Each such path will be tried N times.

root `xxx` Each use of this preference names the root of one of the replicas for Unison to synchronize. Exactly two roots are needed, so normal modes of usage are either to give two values for `root` in the profile, or to give no values in the profile and provide two on the command line. Details of the syntax of roots can be found in Section 5.1 [Roots].

The two roots can be given in either order; Unison will sort them into a canonical order before doing anything else. It also tries to 'canonize' the machine names and paths that appear in the roots, so that, if Unison is invoked later with a slightly different name for the same root, it will be able to locate the correct archives.

rootalias `xxx` When calculating the name of the archive files for a given pair of roots, Unison replaces any roots matching the left-hand side of any rootalias rule by the corresponding right-hand side.

rshargs `xxx` The string value of this preference will be passed as additional arguments (besides the host name and the name of the Unison executable on the remote system) to the `rsh` command used to invoke the remote server.

rshcmd `xxx` This preference can be used to explicitly set the name of the rsh executable (e.g., giving a full path name), if necessary.

rsrc `xxx` When set to `true`, this flag causes Unison to synchronize resource forks and HFS meta-data. On filesystems that do not natively support resource forks, this data is stored in Carbon-compatible ._ AppleDouble files. When the flag is set to `false`, Unison will not synchronize these data. Ordinarily, the flag is set to `default`, and these data are automatically synchronized if either host is running OSX. In rare circumstances it is useful to set the flag manually.

rsync Unison uses the 'rsync algorithm' for 'diffs-only' transfer of updates to large files. Setting this flag to false makes Unison use whole-file transfers instead. Under normal circumstances, there is no reason to do this, but if you are having trouble with repeated 'rsync failure' errors, setting it to false should permit you to synchronize the offending files.

selftest Run internal tests and exit. This option is mostly for developers and must be used carefully: in particular, it will delete the contents of both roots, so that it can install its own files for testing. This flag only makes sense on the command line. When it is provided, no preference file is read: all preferences must be specified on thecommand line. Also, since the self-test procedure involves overwriting the roots and backup directory, the names of the roots and of the backupdir preference must include the string "test" or else the tests will be aborted. (If these are not given on the command line, dummy subdirectories in the current directory will be created automatically.)

servercmd `xxx` This preference can be used to explicitly set the name of the Unison executable on the remote server (e.g., giving a full path name), if necessary.

showarchive When this preference is set, Unison will print out the 'true names'of the roots, in the same form as is expected by the `rootalias`preference.

silent When this preference is set to `true`, the textual user interface will print nothing at all, except in the case of errors. Setting `silent` to true automatically sets the `batch` preference to `true`.

sortbysize When this flag is set, the user interface will list changed files by size (smallest first) rather than by name. This is useful, for example, for synchronizing over slow links, since it puts very large files at the end of the list where they will not prevent smaller files from being transferred quickly.

This preference (as well as the other sorting flags, but not the sorting preferences that require patterns as arguments) can be set interactively and temporarily using the 'Sort' menu in the graphical user interface.

sortfirst `xxx` Each argument to `sortfirst` is a pattern *`pathspec`*, which describes a set of paths. Files matching any of these patterns will be listed first in the user interface. The syntax of *`pathspec`* is described in Section 6.11 [Path Specification].

sortlast `xxx` Similar to `sortfirst`, except that files matching one of these patterns will be listed at the very end.

sortnewfirst When this flag is set, the user interface will list newly created files before all others. This is useful, for example, for checking that newly created files are not 'junk', i.e., ones that should be ignored or deleted rather than synchronized.

sshargs `xxx` The string value of this preference will be passed as additional arguments (besides the host name and the name of the Unison executable on the remote system) to the `ssh` command used to invoke the remote server.

sshcmd `xxx` This preference can be used to explicitly set the name of the ssh executable (e.g., giving a full path name), if necessary.

sshversion `xxx` This preference can be used to control which version of ssh should be used to connect to the server. Legal values are 1 and 2, which will cause unison to try to use `ssh1` or`ssh2` instead of just `ssh` to invoke ssh. The default value is empty, which will make unison use whatever version of ssh is installed as the default 'ssh' command.

stream When this preference is set, Unison will use an experimental streaming protocol for transferring file contents more efficiently. The default value is `true`.

terse When this preference is set to `true`, the user interface will not print status messages.

testserver Setting this flag on the command line causes Unison to attempt to connect to the remote server and, if successful, print a message and immediately exit. Useful for debugging installation problems. Should not be set in preference files.

times When this flag is set to `true`, file modification times (but not directory modtimes) are propagated.

ui `xxx` This preference selects either the graphical or the textual user interface. Legal values are `graphic` or `text`.

Because this option is processed specially during Unison's start-up sequence, it can *only* be used on the command line. In preference files it has no effect.

If the Unison executable was compiled with only a textual interface, this option has no effect. (The pre-compiled binaries are all compiled with both interfaces available.)

unicode `xxx` When set to `true`, this flag causes Unison to perform case insensitive file comparisons assuming Unicode encoding. This is the default. When the flag is set to `false`, a Latin 1 encoding is assumed. When Unison runs in case sensitive mode, this flag only makes a difference if one host is running Windows or Mac OS X. Under Windows, the flag selects between using the Unicode or 8bit Windows API for accessing the filesystem. Under Mac OS X, it selects whether comparing the filenames up to decomposition, or byte-for-byte.

version Print the current version number and exit. (This option only makes sense on the command line.)

xferbycopying When this preference is set, Unison will try to avoid transferring file contents across the network by recognizing when a file with the required contents already exists in the target replica. This usually allows file moves to be propagated very quickly. The default value is`true`.

6.5 Profiles

A *profile* is a text file that specifies permanent settings for roots, paths, ignore patterns, and other preferences, so that they do not need to be typed at the command line every time Unison is run. Profiles should reside in the `.unison` directory on the client machine. If Unison is started with just one argument *`name`* on the command line, it looks for a profile called *`name`*`.prf` in the `.unison` directory. If it is started with no arguments, it scans the `.unison` directory for files whose names end in `.prf` and offers a menu (provided that the Unison executable is compiled with the graphical user interface). If a file named `default.prf` is found, its settings will be offered as the default choices.

To set the value of a preference `p` permanently, add to the appropriate profile a line of the form

```
p = true
```

for a boolean flag or

```
p = <value>
```

for a preference of any other type.

Whitespaces around `p` and `xxx` are ignored. A profile may also include blank lines and lines beginning with `#`; both are ignored.

When Unison starts, it first reads the profile and then the command line, so command-line options will override settings from the profile.

Profiles may also include lines of the form `include` *`name`*, which will cause the file *`name`* (or *`name`*`.prf`, if *`name`* does not exist in the `.unison` directory) to be read at the point, and included as if its contents, instead of the `include` line, was part of the profile. Include lines allows settings common to several profiles to be stored in one place.

A profile may include a preference '`label =` *`desc`*' to provide a description of the options selected in this profile. The string *`desc`* is listed along with the profile name in the profile selection dialog, and displayed in the top-right corner of the main Unison window in the graphical user interface.

The graphical user-interface also supports one-key shortcuts for commonly used profiles. If a profile contains a preference of the form '`key =` *`n`*', where *`n`* is a single digit, then pressing this digit key will cause Unison to immediately switch to this profile and begin synchronization again from scratch. In this case, all actions that have been selected for a set of changes currently being displayed will be discarded.

6.6 Sample Profiles

6.6.1 A Minimal Profile

Here is a very minimal profile file, such as might be found in `.unison/default.prf`:

```
# Roots of the synchronization
root = /home/bcpierce
root = ssh://saul//home/bcpierce

# Paths to synchronize
path = current
path = common
path = .netscape/bookmarks.html
```

6.6.2 A Basic Profile

Here is a more sophisticated profile, illustrating some other useful features.

```
# Roots of the synchronization
root = /home/bcpierce
root = ssh://saul//home/bcpierce

# Paths to synchronize
path = current
path = common
path = .netscape/bookmarks.html

# Some regexps specifying names and paths to ignore
ignore = Name temp.*
ignore = Name *~
ignore = Name .*~
ignore = Path */pilot/backup/Archive_*
ignore = Name *.o
ignore = Name *.tmp

# Window height
height = 37

# Keep a backup copy of every file in a central location
backuplocation = central
backupdir = /home/bcpierce/backups
backup = Name *
backupprefix = $VERSION.
backupsuffix =

# Use this command for displaying diffs
diff = diff -y -W 79 --suppress-common-lines

# Log actions to the terminal
log = true
```

6.6.3 A Power-User Profile

When Unison is used with large replicas, it is often convenient to be able to synchronize just a part of the replicas on a given run (this saves the time of detecting updates in the other parts). This can be accomplished by splitting up the profile into several parts — a common part containing most of the preference settings, plus one “top-level” file for each set of paths that need to be synchronized. (The `include` mechanism can also be used to allow the same set of preference settings to be used with different roots.)

The collection of profiles implementing this scheme might look as follows. The file `default.prf` is empty except for an `include` directive:

```
# Include the contents of the file common
include common
```

Note that the name of the common file is `common`, not `common.prf`; this prevents Unison from offering `common` as one of the list of profiles in the opening dialog (in the graphical UI).

The file `common` contains the real preferences:

```
# Roots of the synchronization
root = /home/bcpierce
root = ssh://saul//home/bcpierce

# (... other preferences ...)

# If any new preferences are added by Unison (e.g. 'ignore'
# preferences added via the graphical UI), then store them in the
# file 'common' rathen than in the top-level preference file
addprefsto = common

# Names and paths to ignore:
ignore = Name temp.*
ignore = Name *~
ignore = Name .*~
ignore = Path */pilot/backup/Archive_*
ignore = Name *.o
ignore = Name *.tmp
```

Note that there are no `path` preferences in `common`. This means that, when we invoke Unison with the default profile (e.g., by typing '`unison default`' or just '`unison`' on the command line), the whole replicas will be synchronized. (If we *never* want to synchronize the whole replicas, then `default.prf` would instead include settings for all the paths that are usually synchronized.)

To synchronize just part of the replicas, Unison is invoked with an alternate preference file—e.g., doing '`unison workingset`', where the preference file `workingset.prf` contains

```
path = current/papers
path = Mail/inbox
path = Mail/drafts
include common
```

causes Unison to synchronize just the listed subdirectories.

The `key` preference can be used in combination with the graphical UI to quickly switch between different sets of paths. For example, if the file `mail.prf` contains

```
path = Mail
batch = true
key = 2
include common
```

then pressing 2 will cause Unison to look for updates in the `Mail` subdirectory and (because the `batch` flag is set) immediately propagate any that it finds.

6.7 Keeping Backups

When Unison overwrites a file or directory by propagating a new version from the other replica, it can keep the old version around as a backup. There are several preferences that control precisely where these backups are stored and how they are named.

To enable backups, you must give one or more `backup` preferences. Each of these has the form

```
backup = <pathspec>
```

where `<pathspec>` has the same form as for the `ignore` preference. For example,

```
backup = Name *
```

causes Unison to keep backups of *all* files and directories. The `backupnot` preference can be used to give a few exceptions: it specifies which files and directories should *not* be backed up, even if they match the `backup` pathspec.

It is important to note that the `pathspec` is matched against the path that is being updated by Unison, not its descendants. For example, if you set `backup = Name *.txt` and then delete a whole directory named `foo` containing some text files, these files will not be backed up because Unison will just check that `foo` does not match `*.txt`. Similarly, if the directory itself happened to be called `foo.txt`, then the whole directory and all the files in it will be backed up, regardless of their names.

Backup files can be stored either *centrally* or *locally*. This behavior is controlled by the preference `backuplocation`, whose value must be either `central` or `local`. (The default is `central`.)

When backups are stored locally, they are kept in the same directory as the original.

When backups are stored centrally, the directory used to hold them is controlled by the preference `backupdir` and the environment variable `UNISONBACKUPDIR`. (The environment variable is checked first.) If neither of these are set, then the directory `.unison/backup` in the user's home directory is used.

The preference `maxbackups` controls how many previous versions of each file are kept (including the current version).

By default, backup files are named `.bak.VERSION.FILENAME`, where `FILENAME` is the original filename and `VERSION` is the backup number (1 for the most recent, 2 for the next most recent, etc.). This can be changed by setting the preferences `backupprefix` and/or `backupsuffix`. If desired, `backupprefix` may include a directory prefix; this can be used with `backuplocation = local` to put all backup files for each directory into a single subdirectory. For example, setting

```
backuplocation = local
backupprefix = .unison/$VERSION.
backupsuffix =
```

will put all backups in a local subdirectory named `.unison`. Also, note that the string `$VERSION` in either `backupprefix` or `backupsuffix` (it must appear in one or the other) is replaced by the version number. This can be used, for example, to ensure that backup files retain the same extension as the originals.

For backward compatibility, the `backups` preference is also supported. It simply means `backup = Name *` and `backuplocation = local`.

6.8 Merging Conflicting Versions

Unison can invoke external programs to merge conflicting versions of a file. The preference `merge` controls this process.

The `merge` preference may be given once or several times in a preference file (it can also be given on the command line, of course, but this tends to be awkward because of the spaces and special characters involved). Each instance of the preference looks like this:

```
merge = <PATHSPEC> -> <MERGECMD>
```

The <PATHSPEC> here has exactly the same format as for the ignore preference (see Section 6.11 [Path specification]). For example, using "Name *.txt" as the <PATHSPEC> tells Unison that this command should be used whenever a file with extension .txt needs to be merged.

Many external merging programs require as inputs not just the two files that need to be merged, but also a file containing the *last synchronized version*. You can ask Unison to keep a copy of the last synchronized version for some files using the backupcurrent preference. This preference is used in exactly the same way as backup and its meaning is similar, except that it causes backups to be kept of the *current* contents of each file after it has been synchronized by Unison, rather than the *previous* contents that Unison overwrote. These backups are kept on *both* replicas in the same place as ordinary backup files—i.e. according to the backuplocation and backupdir preferences. They are named like the original files if backupslocation is set to 'central' and otherwise, Unison uses the backupprefix and backupsuffix preferences and assumes a version number 000 for these backups.

The <MERGECMD> part of the preference specifies what external command should be invoked to merge files at paths matching the <PATHSPEC>. Within this string, several special substrings are recognized; these will be substituted with appropriate values before invoking a sub-shell to execute the command.

- CURRENT1 is replaced by the name of (a temporary copy of) the local variant of the file.
- CURRENT2 is replaced by the name of a temporary file, into which the contents of the remote variant of the file have been transferred by Unison prior to performing the merge.
- CURRENTARCH is replaced by the name of the backed up copy of the original version of the file (i.e., the file saved by Unison if the current filename matches the path specifications for the backupcurrent preference, as explained above), if one exists. If no archive exists and CURRENTARCH appears in the merge command, then an error is signalled.
- CURRENTARCHOPT is replaced by the name of the backed up copy of the original version of the file (i.e., its state at the end of the last successful run of Unison), if one exists, or the empty string if no archive exists.
- NEW is replaced by the name of a temporary file that Unison expects to be written by the merge program when it finishes, giving the desired new contents of the file.
- PATH is replaced by the path (relative to the roots of the replicas) of the file being merged.
- NEW1 and NEW2 are replaced by the names of temporary files that Unison expects to be written by the merge program when it is only able to partially merge the originals; in this case, NEW1 will be written back to the local replica and NEW2 to the remote replica; NEWARCH, if present, will be used as the "last common state" of the replicas. (These three options are provided for later compatibility with the Harmony data synchronizer.)

To accommodate the wide variety of programs that users might want to use for merging, Unison checks for several possible situations when the merge program exits:

- If the merge program exits with a non-zero status, then merge is considered to have failed and the replicas are not changed.
- If the file NEW has been created, it is written back to both replicas (and stored in the backup directory). Similarly, if just the file NEW1 has been created, it is written back to both replicas.
- If neither NEW nor NEW1 have been created, then Unison examines the temporary files CURRENT1 and CURRENT2 that were given as inputs to the merge program. If either has been changed (or both have been changed in identical ways), then its new contents are written back to both replicas. If either CURRENT1 or CURRENT2 has been *deleted*, then the contents of the other are written back to both replicas.

- If the files NEW1, NEW2, and NEWARCH have all been created, they are written back to the local replica, remote replica, and backup directory, respectively. If the files NEW1, NEW2 have been created, but NEWARCH has not, then these files are written back to the local replica and remote replica, respectively. Also, if NEW1 and NEW2 have identical contents, then the same contents are stored as a backup (if the backupcurrent preference is set for this path) to reflect the fact that the path is currently in sync.

- If NEW1 and NEW2 (resp. CURRENT1 and CURRENT2) are created (resp. overwritten) with different contents but the merge command did not fail (i.e., it exited with status code 0), then we copy NEW1 (resp. CURRENT1) to the other replica and to the archive.

  This behavior is a design choice made to handle the case where a merge command only synchronizes some specific contents between two files, skipping some irrelevant information (order between entries, for instance). We assume that, if the merge command exits normally, then the two resulting files are "as good as equal." (The reason we copy one on top of the other is to avoid Unison detecting that the files are unequal the next time it is run and trying again to merge them when, in fact, the merge program has already made them as similar as it is able to.)

If the confirmmerge preference is set and Unison is not run in batch mode, then Unison will always ask for confirmation before actually committing the results of the merge to the replicas.

A large number of external merging programs are available. For example, on Unix systems setting the merge preference to

```
merge = Name *.txt -> diff3 -m CURRENT1 CURRENTARCH CURRENT2
                        > NEW || echo "differences detected"
```

will tell Unison to use the external diff3 program for merging. Alternatively, users of emacs may find the following settings convenient:

```
merge = Name *.txt -> emacs -q --eval '(ediff-merge-files-with-ancestor
                         "CURRENT1" "CURRENT2" "CURRENTARCH" nil "NEW")'
```

(These commands are displayed here on two lines to avoid running off the edge of the page. In your preference file, each command should be written on a single line.)

Users running emacs under windows may find something like this useful:

```
merge = Name * -> C:\Progra~1\Emacs\emacs\bin\emacs.exe -q --eval
                        "(ediff-files """CURRENT1""" """CURRENT2""")"
```

Users running Mac OS X (you may need the Developer Tools installed to get the opendiff utility) may prefer

```
merge = Name *.txt -> opendiff CURRENT1 CURRENT2 -ancestor CURRENTARCH -merge NEW
```

Here is a slightly more involved hack. The opendiff program can operate either with or without an archive file. A merge command of this form

```
merge = Name *.txt ->
          if [ CURRENTARCHOPTx = x ];
          then opendiff CURRENT1 CURRENT2 -merge NEW;
          else opendiff CURRENT1 CURRENT2 -ancestor CURRENTARCHOPT -merge NEW;
          fi
```

(still all on one line in the preference file!) will test whether an archive file exists and use the appropriate variant of the arguments to opendiff.

Ordinarily, external merge programs are only invoked when Unison is *not* running in batch mode. To specify an external merge program that should be used no matter the setting of the batch flag, use the mergebatch preference instead of merge.

> *Please post suggestions for other useful values of the* merge *preference to the* unison-users *mailing list—we'd like to give several examples here.*

6.9 The User Interface

Both the textual and the graphical user interfaces are intended to be mostly self-explanatory. Here are just a few tricks:

- By default, when running on Unix the textual user interface will try to put the terminal into the "raw mode" so that it reads the input a character at a time rather than a line at a time. (This means you can type just the single keystroke "`>`" to tell Unison to propagate a file from left to right, rather than "> Enter.")

  There are some situations, though, where this will not work — for example, when Unison is running in a shell window inside Emacs. Setting the `dumbtty` preference will force Unison to leave the terminal alone and process input a line at a time.

6.10 Exit code

When running in the textual mode, Unison returns an exit status, which describes whether, and at which level, the synchronization was successful. The exit status could be useful when Unison is invoked from a script. Currently, there are four possible values for the exit status:

0 : successful synchronization; everything is up-to-date now.

1 : some files were skipped, but all file transfers were successful.

2 : non-fatal failures occurred during file transfer.

3 : a fatal error occurred, or the execution was interrupted.

The graphical interface does not return any useful information through the exit status.

6.11 Path specification

Several Unison preferences (e.g., `ignore/ignorenot`, `follow`, `sortfirst/sortlast`, `backup`, `merge`, etc.) specify individual paths or sets of paths. These preferences share a common syntax based on regular-expressions. Each preference is associated with a list of path patterns; the paths specified are those that match any one of the path pattern.

- Pattern preferences can be given on the command line, or, more often, stored in profiles, using the same syntax as other preferences. For example, a profile line of the form

  `ignore =` *`pattern`*

  adds *`pattern`* to the list of patterns to be ignored.

- Each *`pattern`* can have one of three forms. The most general form is a Posix extended regular expression introduced by the keyword `Regex`. (The collating sequences and character classes of full Posix regexps are not currently supported).

  `Regex` *`regexp`*

  For convenience, three other styles of pattern are also recognized:

  `Name` *`name`*

  matches any path in which the last component matches *`name`*,

  `Path` *`path`*

matches exactly the path ***path***, and

```
BelowPath path
```

matches the path ***path*** and any path below. The ***name*** and ***path*** arguments of the latter forms of patterns are *not* regular expressions. Instead, standard "globbing" conventions can be used in ***name*** and ***path***:

- a `*` matches any sequence of characters not including `/` (and not beginning with `.`, when used at the beginning of a ***name***)
- a `?` matches any single character except `/` (and leading `.`)
- `[xyz]` matches any character from the set {`x`, `y`, `z`}
- `{a,bb,ccc}` matches any one of `a`, `bb`, or `ccc`.

- The path separator in path patterns is always the forward-slash character "/" — even when the client or server is running under Windows, where the normal separator character is a backslash. This makes it possible to use the same set of path patterns for both Unix and Windows file systems.

Some examples of path patterns appear in Section 6.12 [Ignoring Paths].

6.12 Ignoring Paths

Most users of Unison will find that their replicas contain lots of files that they don't ever want to synchronize — temporary files, very large files, old stuff, architecture-specific binaries, etc. They can instruct Unison to ignore these paths using patterns introduced in Section 6.11 [Path Patterns].

For example, the following pattern will make Unison ignore any path containing the name `CVS` or a name ending in `.cmo`:

```
ignore = Name {CVS,*.cmo}
```

The next pattern makes Unison ignore the path `a/b`:

```
ignore = Path a/b
```

Path patterns do *not* skip filesnames beginning with `.` (as Name patterns do). For example,

```
ignore = Path */tmp
```

will include `.foo/tmp` in the set of ignore directories, as it is a path, not a name, that is ignored.

The following pattern makes Unison ignore any path beginning with `a/b` and ending with a name ending by `.ml`.

```
ignore = Regex a/b/.*\.ml
```

Note that regular expression patterns are "anchored": they must match the whole path, not just a substring of the path.

Here are a few extra points regarding the `ignore` preference.

- If a directory is ignored, all its descendents will be too.
- The user interface provides some convenient commands for adding new patterns to be ignored. To ignore a particular file, select it and press "`i`". To ignore all files with the same extension, select it and press "`E`" (with the shift key). To ignore all files with the same name, no matter what directory they appear in, select it and press "`N`". These new patterns become permanent: they are immediately added to the current profile on disk.

- If you use the `include` directive to include a common collection of preferences in several top-level preference files, you will probably also want to set the `addprefsto` preference to the name of this file. This will cause any new ignore patterns that you add from inside Unison to be appended to this file, instead of whichever top-level preference file you started Unison with.
- Ignore patterns can also be specified on the command line, if you like (this is probably not very useful), using an option like `-ignore 'Name temp.txt'`.
- Be careful about renaming directories containing ignored files. Because Unison understands the rename as a delete plus a create, any ignored files in the directory will be lost (since they are invisible to Unison and therefore they do not get recreated in the new version of the directory).
- There is also an `ignorenot` preference, which specifies a set of patterns for paths that should *not* be ignored, even if they match an `ignore` pattern. However, the interaction of these two sets of patterns can be a little tricky. Here is exactly how it works:
  - Unison starts detecting updates from the root of the replicas—i.e., from the empty path. If the empty path matches an `ignore` pattern and does not match an `ignorenot` pattern, then the whole replica will be ignored. (For this reason, it is not a good idea to include `Name *` as an `ignore` pattern. If you want to ignore everything except a certain set of files, use `Name ?*`.)
  - If the root is a directory, Unison continues looking for updates in all the immediate children of the root. Again, if the name of some child matches an `ignore` pattern and does not match an `ignorenot` pattern, then this whole path *including everything below it* will be ignored.
  - If any of the non-ignored children are directories, then the process continues recursively.

6.13 Symbolic Links

Ordinarily, Unison treats symbolic links in Unix replicas as "opaque": it considers the contents of the link to be just the string specifying where the link points, and it will propagate changes in this string to the other replica.

It is sometimes useful to treat a symbolic link "transparently," acting as though whatever it points to were physically *in* the replica at the point where the symbolic link appears. To tell Unison to treat a link in this manner, add a line of the form

```
follow = pathspec
```

to the profile, where ***`pathspec`*** is a path pattern as described in Section 6.11 [Path Patterns].

Windows file systems do not support symbolic links; Unison will refuse to propagate an opaque symbolic link from Unix to Windows and flag the path as erroneous. When a Unix replica is to be synchronized with a Windows system, all symbolic links should match either an `ignore` pattern or a `follow` pattern.

6.14 Permissions

Synchronizing the permission bits of files is slightly tricky when two different filesytems are involved (e.g., when synchronizing a Windows client and a Unix server). In detail, here's how it works:

- When the permission bits of an existing file or directory are changed, the values of those bits that make sense on *both* operating systems will be propagated to the other replica. The other bits will not be changed.
- When a newly created file is propagated to a remote replica, the permission bits that make sense in both operating systems are also propagated. The values of the other bits are set to default values (they are taken from the current umask, if the receiving host is a Unix system).
- For security reasons, the Unix `setuid` and `setgid` bits are not propagated.
- The Unix owner and group ids are not propagated. (What would this mean, in general?) All files are created with the owner and group of the server process.

6.15 Cross-Platform Synchronization

If you use Unison to synchronize files between Windows and Unix systems, there are a few special issues to be aware of.

Case conflicts. In Unix, filenames are case sensitive: `foo` and `FOO` can refer to different files. In Windows, on the other hand, filenames are not case sensitive: `foo` and `FOO` can only refer to the same file. This means that a Unix `foo` and `FOO` cannot be synchronized onto a Windows system — Windows won't allow two different files to have the "same" name. Unison detects this situation for you, and reports that it cannot synchronize the files.

You can deal with a case conflict in a couple of ways. If you need to have both files on the Windows system, your only choice is to rename one of the Unix files to avoid the case conflict, and re-synchronize. If you don't need the files on the Windows system, you can simply disregard Unison's warning message, and go ahead with the synchronization; Unison won't touch those files. If you don't want to see the warning on each synchronization, you can tell Unison to ignore the files (see Section 6.12 [Ignore]).

Illegal filenames. Unix allows some filenames that are illegal in Windows. For example, colons (':') are not allowed in Windows filenames, but they are legal in Unix filenames. This means that a Unix file `foo:bar` can't be synchronized to a Windows system. As with case conflicts, Unison detects this situation for you, and you have the same options: you can either rename the Unix file and re-synchronize, or you can ignore it.

6.16 Slow Links

Unison is built to run well even over relatively slow links such as modems and DSL connections.

Unison uses the "rsync protocol" designed by Andrew Tridgell and Paul Mackerras to greatly speed up transfers of large files in which only small changes have been made. More information about the rsync protocol can be found at the rsync web site (`http://samba.anu.edu.au/rsync/`).

If you are using Unison with `ssh`, you may get some speed improvement by enabling `ssh`'s compression feature. Do this by adding the option "`-sshargs -C`" to the command line or "`sshargs = -C`" to your profile.

6.17 Making Unison Faster on Large Files

Unison's built-in implementation of the rsync algorithm makes transferring updates to existing files pretty fast. However, for whole-file copies of newly created files, the built-in transfer method is not highly optimized. Also, if Unison is interrupted in the middle of transferring a large file, it will attempt to retransfer the whole thing on the next run.

These shortcomings can be addressed with a little extra work by telling Unison to use an external file copying utility for whole-file transfers. The recommended one is the standalone `rsync` tool, which is available by default on most Unix systems and can easily be installed on Windows systems using Cygwin.

If you have `rsync` installed on both hosts, you can make Unison use it simply by setting the `copythreshold` flag to something non-negative. If you set it to 0, Unison will use the external copy utility for *all* whole-file transfers. (This is probably slower than letting Unison copy small files by itself, but can be useful for testing.) If you set it to a larger value, Unison will use the external utility for all files larger than this size (which is given in kilobytes, so setting it to 1000 will cause the external tool to be used for all transfers larger than a megabyte).

If you want to use a different external copy utility, set both the `copyprog` and `copyprogpartial` preferences—the former is used for the first transfer of a file, while the latter is used when Unison sees a partially transferred temp file on the receiving host. Be careful here: Your external tool needs to be instructed to copy files in place (otherwise if the transfer is interrupted Unison will not notice that some of the data has already been transferred, the next time it tries). The default values are:

```
copyprog      =   rsync --inplace --compress
copyprogrest  =   rsync --partial --inplace --compress
```

You may also need to set the `copyquoterem` preference. When it is set to `true`, this causes Unison to add an extra layer of quotes to the remote path passed to the external copy program. This is is needed by rsync, for example, which internally uses an ssh connection, requiring an extra level of quoting for paths containing spaces. When this flag is set to `default`, extra quotes are added if the value of `copyprog` contains the string `rsync`. The default value is `default`, naturally.

If a *directory* transfer is interrupted, the next run of Unison will automatically skip any files that were completely transferred before the interruption. (This behavior is always on: it does not depend on the setting of the `copythreshold` preference.) Note, though, that the new directory will not appear in the destination filesystem until everything has been transferred—partially transferred directories are kept in a temporary location (with names like `.unison.DIRNAME....`) until the transfer is complete.

6.18 Fast Update Detection

If your replicas are large and at least one of them is on a Windows system, you may find that Unison's default method for detecting changes (which involves scanning the full contents of every file on every sync—the only completely safe way to do it under Windows) is too slow. Unison provides a preference `fastcheck` that, when set to `true`, causes it to use file creation times as 'pseudo inode numbers' when scanning replicas for updates, instead of reading the full contents of every file.

When `fastcheck` is set to `no`, Unison will perform slow checking—re-scanning the contents of each file on each synchronization—on all replicas. When `fastcheck` is set to `default` (which, naturally, is the default), Unison will use fast checks on Unix replicas and slow checks on Windows replicas.

This strategy may cause Unison to miss propagating an update if the modification time and length of the file are both unchanged by the update. However, Unison will never *overwrite* such an update with a change from the other replica, since it always does a safe check for updates just before propagating a change. Thus, it is reasonable to use this switch most of the time and occasionally run Unison once with `fastcheck` set to `no`, if you are worried that Unison may have overlooked an update.

Fastcheck is (always) automatically disabled for files with extension `.xls` or `.mpp`, to prevent Unison from being confused by the habits of certain programs (Excel, in particular) of updating files without changing their modification times.

6.19 Mount Points and Removable Media

Using Unison removable media such as USB drives can be dangerous unless you are careful. If you synchronize a directory that is stored on removable media when the media is not present, it will look to Unison as though the whole directory has been deleted, and it will proceed to delete the directory from the other replica—probably not what you want!

To prevent accidents, Unison provides a preference called `mountpoint`. Including a line like

```
mountpoint = foo
```

in your preference file will cause Unison to check, after it finishes detecting updates, that something actually exists at the path `foo` on both replicas; if it does not, the Unison run will abort.

6.20 Click-starting Unison

On Windows NT/2k/XP systems, the graphical version of Unison can be invoked directly by clicking on its icon. On Windows 95/98 systems, click-starting also works, *as long as you are not using ssh.* Due to an incompatibility with ocaml and Windows 95/98 that is not under our control, you must start Unison from a DOS window in Windows 95/98 if you want to use ssh.

When you click on the Unison icon, two windows will be created: Unison's regular window, plus a console window, which is used only for giving your password to ssh (if you do not use ssh to connect, you can ignore this window). When your password is requested, you'll need to activate the console window (e.g., by clicking in it) before typing. If you start Unison from a DOS window, Unison's regular window will appear and you will type your password in the DOS window you were using.

To use Unison in this mode, you must first create a profile (see Section 6.5 [Profile]). Use your favorite editor for this.

A Installing Ssh

Warning: These instructions may be out of date. More current information can be found the Unison Wiki[4].

Your local host will need just an ssh client; the remote host needs an ssh server (or daemon), which is available on Unix systems. Unison is known to work with ssh version 1.2.27 (Unix) and version 1.2.14 (Windows); other versions may or may not work.

A.1 Unix

Most modern Unix installations come with `ssh` pre-installed.

A.2 Windows

Many Windows implementations of ssh only provide graphical interfaces, but Unison requires an ssh client that it can invoke with a command-line interface. A suitable version of ssh can be installed as follows.

1. Download an `ssh` executable.

   Warning: there are many implementations and ports of ssh for Windows, and not all of them will work with Unison. We have gotten Unison to work with Cygwin's port of openssh, and we suggest you try that one first. Here's how to install it:

   (a) First, create a new folder on your desktop to hold temporary installation files. It can have any name you like, but in these instructions we'll assume that you call it `Foo`.

   (b) Direct your web browser to www.cygwin.com, and click on the "Install now!" link. This will download a file, `setup.exe`; save it in the directory `Foo`. The file `setup.exe` is a small program that will download the actual install files from the Internet when you run it.

   (c) Start `setup.exe` (by double-clicking). This brings up a series of dialogs that you will have to go through. Select "Install from Internet." For "Local Package Directory" select the directory `Foo`. For "Select install root directory" we recommend that you use the default, `C:\cygwin`. The next dialog asks you to select the way that you want to connect to the network to download the installation files; we have used "Use IE5 Settings" successfully, but you may need to make a different selection depending on your networking setup. The next dialog gives a list of mirrors; select one close to you.

   Next you are asked to select which packages to install. The default settings in this dialog download a lot of packages that are not strictly necessary to run Unison with ssh. If you don't want to install a package, click on it until "skip" is shown. For a minimum installation, select only the packages "cygwin" and "openssh," which come to about 1900KB; the full installation is much larger.

   > *Note that you are plan to build unison using the free CygWin GNU C compiler, you need to install essential development packages such as "gcc", "make", "fileutil", etc; we refer to the file "INSTALL.win32-cygwin-gnuc" in the source distribution for further details.*

   After the packages are downloaded and installed, the next dialog allows you to choose whether to "Create Desktop Icon" and "Add to Start Menu." You make the call.

   (d) You can now delete the directory `Foo` and its contents.

   Some people have reported problems using Cygwin's ssh with Unison. If you have trouble, you might try other ones instead:

   `http://linuxmafia.com/ssh/win32.html`

[4] `http://alliance.seas.upenn.edu/~bcpierce/wiki/index.php?n=Main.UnisonFAQOSSpecific`

2. You must set the environment variables HOME and PATH. Ssh will create a directory `.ssh` in the directory given by HOME, so that it has a place to keep data like your public and private keys. PATH must be set to include the Cygwin `bin` directory, so that Unison can find the ssh executable.

   - On Windows 95/98, add the lines

     ```
     set PATH=%PATH%;<SSHDIR>
     set HOME=<HOMEDIR>
     ```

     to the file `C:\AUTOEXEC.BAT`, where `<HOMEDIR>` is the directory where you want ssh to create its `.ssh` directory, and `<SSHDIR>` is the directory where the executable `ssh.exe` is stored; if you've installed Cygwin in the default location, this is `C:\cygwin\bin`. You will have to reboot your computer to take the changes into account.
   - On Windows NT/2k/XP, open the environment variables dialog box:
     - Windows NT: My Computer/Properties/Environment
     - Windows 2k: My Computer/Properties/Advanced/Environment variables

     then select Path and edit its value by appending `;<SSHDIR>` to it, where `<SSHDIR>` is the full name of the directory that includes the ssh executable; if you've installed Cygwin in the default location, this is `C:\cygwin\bin`.

3. Test ssh from a DOS shell by typing

   ```
   ssh <remote host> -l <login name>
   ```

   You should get a prompt for your password on `<remote host>`, followed by a working connection.

4. Note that `ssh-keygen` may not work (fails with "gethostname: no such file or directory") on some systems. This is OK: you can use ssh with your regular password for the remote system.

5. You should now be able to use Unison with an ssh connection. If you are logged in with a different user name on the local and remote hosts, provide your remote user name when providing the remote root (i.e., `//username@host/path...`).

B Changes in Version 2.40.128

Changes since 2.40.1:

- Added "BelowPath" patterns, that match a path as well as all paths below (convenient to use with nodeletion,update,creationpartial preferences)
- Added a "fat" preference that makes Unison use the right options when one of the replica is on a FAT filesystem.
- Allow "prefer/force=newer" even when not synchronizing modification times. (The reconciler will not be aware of the modification time of unchanged files, so the synchronization choices of Unison can be different from when "times=true", but the behavior remains sane: changed files with the most recent modification time will be propagated.)
- Minor fixes and improvements:
  - Compare filenames up to decomposition in case sensitive mode when one host is running MacOSX and the unicode preference is set to true.
  - Rsync: somewhat faster compressor
  - Make Unicode the default on all architectures (it was only the default when a Mac OS X or Windows machine was involved).

Changes since 2.32:

- Major enhancement: Unicode support.
  - Unison should now handle unicode filenames correctly on all platforms.
  - This functionality is controlled by a new preference `unicode`.
  - Unicode mode is now the default when one of the hosts is under Windows or MacOS. This may make upgrades a bit more painful (the archives cannot be reused), but this is a much saner default.
- Partial transfer of directories. If an error occurs while transferring a directory, the part transferred so far is copied into place (and the archives are updated accordingly). The "maxerrors" preference controls how many transfer error Unison will accept before stopping the transfer of a directory (by default, only one). This makes it possible to transfer most of a directory even if there are some errors. Currently, only the first error is reported by the GUIs.

  Also, allow partial transfer of a directory when there was an error deep inside this directory during update detection. At the moment, this is only activated with the text and GTK UIs, which have been modified so that they show that the transfer is going to be partial and so that they can display all errors.
- Improvement to the code for resuming directory transfers:
  - if a file was not correctly transferred (or the source has been modified since, with unchanged size), Unison performs a new transfer rather than failing
  - spurious files are deleted (this can happen if a file is deleted on the source replica before resuming the transfer; not deleting the file would result in it reappearing on the target replica)
- Experimental streaming protocol for transferring file contents (can be disabled by setting the directive "stream" to false): file contents is transfered asynchronously (without waiting for a response from the destination after each chunk sent) rather than using the synchronous RPC mechanism. As a consequence:
  - Unison now transfers the contents of a single file at a time (Unison used to transfer several contents simultaneously in order to hide the connection latency.)

- the transfer of large files uses the full available bandwidth and is not slowed done due to the connection latency anymore
- we get performance improvement for small files as well by scheduling many files simultaneously (as scheduling a file for transfer consume little ressource: it does not mean allocating a large buffer anymore)

- Changes to the internal implementation of the rsync algorithm:
  - use longer blocks for large files (the size of a block is the square root of the size of the file for large files);
  - transmit less checksum information per block (we still have less than one chance in a hundred million of transferring a file incorrectly, and Unison will catch any transfer error when fingerprinting the whole file)
  - avoid transfer overhead (which was 4 bytes per block)

  For a 1G file, the first optimization saves a factor 50 on the amount of data transferred from the target to the source (blocks are 32768 bytes rather than just 700 bytes). The two other optimizations save another factor of 2 (from 24 bytes per block down to 10).

- Implemented an on-disk file fingerprint cache to speed-up update detection after a crash: this way, Unison does not have do recompute all the file fingerprints from scratch.
  - When Unison detects that the archive case-sensitivity mode does not match the current settings, it populates the fingerprint cache using the archive contents. This way, changing the case-sensitivity mode should be reasonably fast.

- New preferences ”noupdate=root”, ”nodeletion=root”, ”nocreation=root” that prevent Unison from performing files updates, deletions or creations on the given root. Also ’partial’ versions of ’noupdate’, ’nodeletion’ and ’nocreation’
- Limit the number of simultaneous external copy program (”copymax” preference)
- New ”links” preference. When set to false, Unison will report an error on symlinks during update detection. (This is the default when one host is running Windows but not Cygwin.) This is better than failing during propagation.
- Added a preference ”halfduplex” to force half-duplex communication with the server. This may be useful on unreliable links (as a more efficient alternative to ”maxthreads = 1”).
- Renamed preference ”pretendwin” to ”ignoreinodenumbers” (an alias is kept for backwards compatibility).
- Ignore one-second differences when synchronizing modification time. (Technically, this is an incompatible archive format change, but it is backward compatible. To trigger a problem, a user would have to synchronize modification times on a filesystem with a two-second granularity and then downgrade to a previous version of Unison, which does not work well in such a case. Thus, it does not seem worthwhile to increment the archive format number, which would impact all users.)
- Do not keep many files simultaneously opened anymore when the rsync algorithm is in use.
- Add “ignorearchives” preference to ignore existing archives (to avoid forcing users to delete them manually, in situations where one archive has gotten deleted or corrupted).
- Mac OS
  - fixed rsync bug which could result in an ”index out of bounds” error when transferring resource forks.

  - Fixed bug which made Unison ignore finder information and resource fork when compiled to 64bit on Mac OSX.
  - should now be 64 bit clean (the Growl framework is not up to date, though)
  - Made the bridge between Objective C and Ocaml code GC friendly (it was allocating ML values and putting them in an array which was not registered with the GC)
  - use darker grey arrows (patch contributed by Eric Y. Kow)

- GTK user interface
  - assistant for creating profiles
  - profile editor
  - pop up a summary window when the replicas are not fully synchronized after transport
  - display estimated remaining time and transfer rate on the progress bar
  - allow simultaneous selection of several items
  - Do not reload the preference file before a new update detection if it is unchanged
  - disabled scrolling to the first unfinished item during transport. It goes way too fast when lot of small files are synchronized, and it makes it impossible to browse the file list during transport.
  - take into account the "height" preference again
  - the internal list of selected reconciler item was not always in sync with what was displayed (GTK bug?); workaround implemented
  - Do not display "Looking for change" messages during propagation (when checking the targe is unchanged) but only during update detection
  - Apply patch to fix some crashes in the OSX GUI, thanks to Onne Gorter.

- Text UI
  - During update detection, display status by updating a single line rather than generating a new line of output every so often. Should be less confusing.

- Windows
  - Fastcheck is now the default under Windows. People mostly use NTFS nowadays and the Unicode API provides an equivalent to inode numbers for this filesystem.
  - Only use long UNC path for accessing replicas (as '..' is not handled with this format of paths, but can be useful)
  - Windows text UI: now put the console into UTF-8 output mode. This is the right thing to do when in Unicode mode, and is no worse than what we had previously otherwise (the console use some esoteric encoding by default). This only works when using a Unicode font instead of the default raster font.
  - Don't get the home directory from environment variable HOME under Windows (except for Cygwin binaries): we don't want the behavior of Unison to depends on whether it is run from a Cygwin shell (where HOME is set) or in any other way (where HOME is usually not set).

- Miscellaneous fixes and improvements
  - Made a server waiting on a socket more resilient to unexpected lost connections from the client.
  - Small patch to property setting code suggested by Ulrich Gernkow.
  - Several fixes to the change transfer functions (both the internal ones and external transfers using rsync). In particular, limit the number of simultaneous transfer using an rsync (as the rsync algorithm can use a large amount of memory when processing huge files)

  - Keep track of which file contents are being transferred, and delay the transfer of a file when another file with the same contents is currently being transferred. This way, the second transfer can be skipped and replaced by a local copy.
  - Experimental update detection optimization: do not read the contents of unchanged directories
  - When a file transfer fails, turn off fastcheck for this file on the next sync.
  - Fixed bug with case insensitive mode on a case sensitive filesystem:
    * if file "a/a" is created on one replica and directory "A" is created on the other, the file failed to be synchronized the first time Unison is run afterwards, as Unison uses the wrong path "a/a" (if Unison is run again, the directories are in the archive, so the right path is used);
    * if file "a" appears on one replica and file "A" appears on the other with different contents, Unison was unable to synchronize them.
  - Improved error reporting when the destination is updated during synchronization: Unison now tells which file has been updated, and how.
  - Limit the length of temporary file names
  - Case sensitivity information put in the archive (in a backward compatible way) and checked when the archive is loaded
  - Got rid of the 16mb marshalling limit by marshalling to a bigarray.
  - Resume copy of partially transferred files.

Changes since 2.31:

- Small user interface changes
  - Small change to text UI "scanning..." messages, to print just directories (hopefully making it clearer that individual files are not necessarily being fingerprinted).
- Minor fixes and improvements:
  - Ignore one hour differences when deciding whether a file may have been updated. This avoids slow update detection after daylight saving time changes under Windows. This makes Unison slightly more likely to miss an update, but it should be safe enough.
  - Fix a small bug that was affecting mainly windows users. We need to commit the archives at the end of the sync even if there are no updates to propagate because some files (in fact, if we've just switched to DST on windows, a LOT of files) might have new modtimes in the archive. (Changed the text UI only. It's less clear where to change the GUI.)
  - Don't delete the temp file when a transfer fails due to a fingerprint mismatch (so that we can have a look and see why!) We've also added more debugging code togive more informative error messages when we encounter the dreaded and longstanding "assert failed during file transfer" bug
  - Incorrect paths ("path" directive) now result in an error update item rather than a fatal error.
  - Create parent directories (with correct permissions) during transport for paths which point to non-existent locations in the destination replica.

Changes since 2.27:

- If Unison is interrupted during a directory transfer, it will now leave the partially transferred directory intact in a temporary location. (This maintains the invariant that new files/directories are transferred either completely or not at all.) The next time Unison is run, it will continue filling in this temporary directory, skipping transferring files that it finds are already there.
- We've added experimental support for invoking an external file transfer tool for whole-file copies instead of Unison's built-in transfer protocol. Three new preferences have been added:

  - `copyprog` is a string giving the name (and command-line switches, if needed) of an external program that can be used to copy large files efficiently. By default, rsync is invoked, but other tools such as scp can be used instead by changing the value of this preference. (Although this is not its primary purpose, rsync is actually a pretty fast way of copying files that don't already exist on the receiving host.) For files that do already exist on (but that have been changed in one replica), Unison will always use its built-in implementation of the rsync algorithm.
  - Added a "copyprogrest" preference, so that we can give different command lines for invoking the external copy utility depending on whether a partially transferred file already exists or not. (Rsync doesn't seem to care about this, but other utilities may.)
  - `copythreshold` is an integer (-1 by default), indicating above what filesize (in megabytes) Unison should use the external copying utility specified by copyprog. Specifying 0 will cause ALL copies to use the external program; a negative number will prevent any files from using it. (Default is -1.)

  Thanks to Alan Schmitt for a huge amount of hacking and to an anonymous sponsor for suggesting and underwriting this extension.

- Small improvements:
  - Added a new preference, `dontchmod`. By default, Unison uses the `chmod` system call to set the permission bits of files after it has copied them. But in some circumstances (and under some operating systems), the chmod call always fails. Setting this preference completely prevents Unison from ever calling `chmod`.
  - Don't ignore files that look like backup files if the `backuplocation` preference is set to `central`
  - Shortened the names of several preferences. The old names are also still supported, for backwards compatibility, but they do not appear in the documentation.
  - Lots of little documentation tidying. (In particular, preferences are separated into Basic and Advanced! This should hopefully make Unison a little more approachable for new users.
  - Unison can sometimes fail to transfer a file, giving the unhelpful message "Destination updated during synchronization" even though the file has not been changed. This can be caused by programs that change either the file's contents *or* the file's extended attributes without changing its modification time. It's not clear what is the best fix for this – it is not Unison's fault, but it makes Unison's behavior puzzling – but at least Unison can be more helpful about suggesting a workaround (running once with `fastcheck` set to false). The failure message has been changed to give this advice.
  - Further improvements to the OS X GUI (thanks to Alan Schmitt and Craig Federighi).
- Very preliminary support for triggering Unison from an external filesystem-watching utility. The current implementation is very simple, not efficient, and almost completely untested—not ready for real users. But if someone wants to help improve it (e.g., by writing a filesystem watcher for your favorite OS), please make yourself known!

  On the Unison side, the new behavior is very simple:

  - use the text UI
  - start Unison with the command-line flag "-repeat FOO", where FOO is name of a file where Unison should look for notifications of changes
  - when it starts up, Unison will read the whole contents of this file (on both hosts), which should be a newline-separated list of paths (relative to the root of the synchronization) and synchronize just these paths, as if it had been started with the "-path=xxx" option for each one of them
  - when it finishes, it will sleep for a few seconds and then examine the watchfile again; if anything has been added, it will read the new paths, synchronize them, and go back to sleep
  - that's it!

To use this to drive Unison "incrementally," just start it in this mode and start up a tool (on each host) to watch for new changes to the filesystem and append the appropriate paths to the watchfile. Hopefully such tools should not be too hard to write.

- Bug fixes:
  - Fixed a bug that was causing new files to be created with permissions 0x600 instead of using a reasonable default (like 0x644), if the 'perms' flag was set to 0. (Bug reported by Ben Crowell.)
  - Follow maxthreads preference when transferring directories.

Changes since 2.17:

- Major rewrite and cleanup of the whole Mac OS X graphical user interface by Craig Federighi. Thanks, Craig!!!
- Small fix to ctime (non-)handling in update detection under windows with fastcheck.
- Several small fixes to the GTK2 UI to make it work better under Windows [thanks to Karl M for these].
- The backup functionality has been completely rewritten. The external interface has not changed, but numerous bugs, irregular behaviors, and cross-platform inconsistencies have been corrected.
- The Unison project now accepts donations via PayPal. If you'd like to donate, you can find a link to the donation page on the Unison home page.
- Some important safety improvements:
  - Added a new `mountpoint` preference, which can be used to specify a path that must exist in both replicas at the end of update detection (otherwise Unison aborts). This can be used to avoid potentially dangerous situations when Unison is used with removable media such as external hard drives and compact flash cards.
  - The confirmation of "big deletes" is now controlled by a boolean preference `confirmbigdeletes`. Default is true, which gives the same behavior as previously. (This functionality is at least partly superceded by the `mountpoint` preference, but it has been left in place in case it is useful to some people.)
  - If Unison is asked to "follow" a symbolic link but there is nothing at the other end of the link, it will now flag this path as an error, rather than treating the symlink itself as missing or deleted. This avoids a potentially dangerous situation where a followed symlink points to an external filesystem that might be offline when Unison is run (whereupon Unison would cheerfully delete the corresponding files in the other replica!).
- Smaller changes:
  - Added `forcepartial` and `preferpartial` preferences, which behave like `force` and `prefer` but can be specified on a per-path basis. [Thanks to Alan Schmitt for this.]
  - A bare-bones self test feature was added, which runs unison through some of its paces and checks that the results are as expected. The coverage of the tests is still very limited, but the facility has already been very useful in debugging the new backup functionality (especially in exposing some subtle cross-platform issues).
  - Refined debugging code so that the verbosity of individual modules can be controlled separately. Instead of just putting '-debug verbose' on the command line, you can put '-debug update+', which causes all the extra messages in the Update module, but not other modules, to be printed. Putting '-debug verbose' causes all modules to print with maximum verbosity.
  - Removed `mergebatch` preference. (It never seemed very useful, and its semantics were confusing.)

- Rewrote some of the merging functionality, for better cooperation with external Harmony instances.
- Changed the temp file prefix from `.#` to `.unison`.
- Compressed the output from the text user interface (particularly when run with the `-terse` flag) to make it easier to interpret the results when Unison is run several times in succession from a script.
- Diff and merge functions now work under Windows.
- Changed the order of arguments to the default diff command (so that the + and - annotations in diff's output are reversed).
- Added `.mpp` files to the "never fastcheck" list (like `.xls` files).

- Many small bugfixes, including:
  - Fixed a longstanding bug regarding fastcheck and daylight saving time under Windows when Unison is set up to synchronize modification times. (Modification times cannot be updated in the archive in this case, so we have to ignore one hour differences.)
  - Fixed a bug that would occasionally cause the archives to be left in non-identical states on the two hosts after synchronization.
  - Fixed a bug that prevented Unison from communicating correctly between 32- and 64-bit architectures.
  - On windows, file creation times are no longer used as a proxy for inode numbers. (This is unfortunate, as it makes fastcheck a little less safe. But it turns out that file creation times are not reliable under Windows: if a file is removed and a new file is created in its place, the new one will sometimes be given the same creation date as the old one!)
  - Set read-only file to R/W on OSX before attempting to change other attributes.
  - Fixed bug resulting in spurious "Aborted" errors during transport (thanks to Jerome Vouillon)
  - Enable diff if file contents have changed in one replica, but only properties in the other.
  - Removed misleading documentation for 'repeat' preference.
  - Fixed a bug in merging code where Unison could sometimes deadlock with the external merge program, if the latter produced large amounts of output.
  - Workaround for a bug compiling gtk2 user interface against current versions of gtk2+ libraries.
  - Added a better error message for "ambiguous paths".
  - Squashed a longstanding bug that would cause file transfer to fail with the message "Failed: Error in readWrite: Is a directory."
  - Replaced symlinks with copies of their targets in the Growl framework in src/uimac. This should make the sources easier to check out from the svn repository on WinXP systems.
  - Added a workaround (suggested by Karl M.) for the problem discussed on the unison users mailing list where, on the Windows platform, the server would hang when transferring files. I conjecture that the problem has to do with the RPC mechanism, which was used to make a call *back* from the server to the client (inside the Trace.log function) so that the log message would be appended to the log file on the client. The workaround is to dump these messages (about when xferbycopying shortcuts are applied and whether they succeed) just to the standard output of the Unison process, not to the log file.

Changes since 2.13.0:

- The features for performing backups and for invoking external merge programs have been completely rewritten by Stephane Lescuyer (thanks, Stephane!). The user-visible functionality should not change, but the internals have been rationalized and there are a number of new features. See the manual (in particular, the description of the `backupXXX` preferences) for details.

- Incorporated patches for ipv6 support, contributed by Samuel Thibault. (Note that, due to a bug in the released OCaml 3.08.3 compiler, this code will not actually work with ipv6 unless compiled with the CVS version of the OCaml compiler, where the bug has been fixed; however, ipv4 should continue to work normally.)
- OSX interface:
  - Incorporated Ben Willmore's cool new icon for the Mac UI.
- Small fixes:
  - Fixed off by one error in month numbers (in printed dates) reported by Bob Burger

Changes since 2.12.0:

- New convention for release numbering: Releases will continue to be given numbers of the form `X.Y.Z`, but, from now on, just the major version number (`X.Y`) will be considered significant when checking compatibility between client and server versions. The third component of the version number will be used only to identify "patch levels" of releases.

  This change goes hand in hand with a change to the procedure for making new releases. Candidate releases will initially be given "beta release" status when they are announced for public consumption. Any bugs that are discovered will be fixed in a separate branch of the source repository (without changing the major version number) and new tarballs re-released as needed. When this process converges, the patched beta version will be dubbed stable.
- Warning (failure in batch mode) when one path is completely emptied. This prevents Unison from deleting everything on one replica when the other disappear.
- Fix diff bug (where no difference is shown the first time the diff command is given).
- User interface changes:
  - Improved workaround for button focus problem (GTK2 UI)
  - Put leading zeroes in date fields
  - More robust handling of character encodings in GTK2 UI
  - Changed format of modification time displays, from `modified at hh:mm:ss on dd MMM, yyyy` to `modified on yyyy-mm-dd hh:mm:ss`
  - Changed time display to include seconds (so that people on FAT filesystems will not be confused when Unison tries to update a file time to an odd number of seconds and the filesystem truncates it to an even number!)
  - Use the diff "-u" option by default when showing differences between files (the output is more readable)
  - In text mode, pipe the diff output to a pager if the environment variable PAGER is set
  - Bug fixes and cleanups in ssh password prompting. Now works with the GTK2 UI under Linux. (Hopefully the Mac OS X one is not broken!)
  - Include profile name in the GTK2 window name
  - Added bindings ',' (same as '¡') and '.' (same as '¿') in the GTK2 UI
- Mac GUI:
  - actions like ¡ and ¿ scroll to the next item as necessary.
  - Restart has a menu item and keyboard shortcut (command-R).
  - Added a command-line tool for Mac OS X. It can be installed from the Unison menu.
  - New icon.

- Handle the "help" command-line argument properly.
- Handle profiles given on the command line properly.
- When a profile has been selected, the profile dialog is replaced by a "connecting" message while the connection is being made. This gives better feedback.
- Size of left and right columns is now large enough so that "PropsChanged" is not cut off.

- Minor changes:
  - Disable multi-threading when both roots are local
  - Improved error handling code. In particular, make sure all files are closed in case of a transient failure
  - Under Windows, use `$UNISON` for home directory as a last resort (it was wrongly moved before `$HOME` and `$USERPROFILE` in Unison 2.12.0)
  - Reopen the logfile if its name changes (profile change)
  - Double-check that permissions and modification times have been properly set: there are some combination of OS and filesystem on which setting them can fail in a silent way.
  - Check for bad Windows filenames for pure Windows synchronization also (not just cross architecture synchronization). This way, filenames containing backslashes, which are not correctly handled by unison, are rejected right away.
  - Attempt to resolve issues with synchronizing modification times of read-only files under Windows
  - Ignore chmod failures when deleting files
  - Ignore trailing dots in filenames in case insensitive mode
  - Proper quoting of paths, files and extensions ignored using the UI
  - The strings CURRENT1 and CURRENT2 are now correctly substitued when they occur in the diff preference
  - Improvements to syncing resource forks between Macs via a non-Mac system.

Changes since 2.10.2:

- **Incompatible change:** Archive format has changed.
- Source code availability: The Unison sources are now managed using Subversion. One nice side-effect is that anonymous checkout is now possible, like this:

  ```
  svn co https://cvs.cis.upenn.edu:3690/svnroot/unison/
  ```

  We will also continue to export a "developer tarball" of the current (modulo one day) sources in the web export directory. To receive commit logs for changes to the sources, subscribe to the `unison-hackers` list (`http://www.cis.upenn.edu/~bcpierce/unison/lists.html`).

- Text user interface:
  - Substantial reworking of the internal logic of the text UI to make it a bit easier to modify.
  - The `dumbtty` flag in the text UI is automatically set to true if the client is running on a Unix system and the `EMACS` environment variable is set to anything other than the empty string.

- Native OS X gui:
  - Added a synchronize menu item with keyboard shortcut
  - Added a merge menu item, still needs to be debugged
  - Fixes to compile for Panther

  - Miscellaneous improvements and bugfixes

- Small changes:
  - Changed the filename checking code to apply to Windows only, instead of OS X as well.
  - Finder flags now synchronized
  - Fallback in copy.ml for filesystem that do not support `O_EXCL`
  - Changed buffer size for local file copy (was highly inefficient with synchronous writes)
  - Ignore chmod failure when deleting a directory
  - Fixed assertion failure when resolving a conflict content change / permission changes in favor of the content change.
  - Workaround for transferring large files using rsync.
  - Use buffered I/O for files (this is the only way to open files in binary mode under Cygwin).
  - On non-Cygwin Windows systems, the UNISON environment variable is now checked first to determine where to look for Unison's archive and preference files, followed by `HOME` and `USERPROFILE` in that order. On Unix and Cygwin systems, `HOME` is used.
  - Generalized `diff` preference so that it can be given either as just the command name to be used for calculating diffs or else a whole command line, containing the strings `CURRENT1` and `CURRENT2`, which will be replaced by the names of the files to be diff'ed before the command is called.
  - Recognize password prompts in some newer versions of ssh.

Changes since 2.9.20:

- **Incompatible change:** Archive format has changed.
- Major functionality changes:
  - Major tidying and enhancement of 'merge' functionality. The main user-visible change is that the external merge program may either write the merged output to a single new file, as before, or it may modify one or both of its input files, or it may write *two* new files. In the latter cases, its modifications will be copied back into place on both the local and the remote host, and (if the two files are now equal) the archive will be updated appropriately. More information can be found in the user manual. Thanks to Malo Denielou and Alan Schmitt for these improvements.

    Warning: the new merging functionality is not completely compatible with old versions! Check the manual for details.
  - Files larger than 2Gb are now supported.
  - Added preliminary (and still somewhat experimental) support for the Apple OS X operating system.
    - Resource forks should be transferred correctly. (See the manual for details of how this works when synchronizing HFS with non-HFS volumes.) Synchronization of file type and creator information is also supported.
    - On OSX systems, the name of the directory for storing Unison's archives, preference files, etc., is now determined as follows:
      - if `~/.unison` exists, use it
      - otherwise, use `~/Library/Application Support/Unison`, creating it if necessary.
    - A preliminary native-Cocoa user interface is under construction. This still needs some work, and some users experience unpredictable crashes, so it is only for hackers for now. Run make with `UISTYLE=mac` to build this interface.
- Minor functionality changes:

- Added an `ignorelocks` preference, which forces Unison to override left-over archive locks. (Setting this preference is dangerous! Use it only if you are positive you know what you are doing.)
- Added a new preference `assumeContentsAreImmutable`. If a directory matches one of the patterns set in this preference, then update detection is skipped for files in this directory. (The purpose is to speed update detection for cases like Mail folders, which contain lots and lots of immutable files.) Also a preference `assumeContentsAreImmutableNot`, which overrides the first, similarly to `ignorenot`. (Later amendment: these preferences are now called `immutable` and `immutablenot`.)
- The `ignorecase` flag has been changed from a boolean to a three-valued preference. The default setting, called `default`, checks the operating systems running on the client and server and ignores filename case if either of them is OSX or Windows. Setting ignorecase to `true` or `false` overrides this behavior. If you have been setting `ignorecase` on the command line using `-ignorecase=true` or `-ignorecase=false`, you will need to change to `-ignorecase true` or `-ignorecase false`.
- a new preference, 'repeat', for the text user interface (only). If 'repeat' is set to a number, then, after it finishes synchronizing, Unison will wait for that many seconds and then start over, continuing this way until it is killed from outside. Setting repeat to true will automatically set the batch preference to true.
- Excel files are now handled specially, so that the `fastcheck` optimization is skipped even if the `fastcheck` flag is set. (Excel does some naughty things with modtimes, making this optimization unreliable and leading to failures during change propagation.)
- The ignorecase flag has been changed from a boolean to a three-valued preference. The default setting, called 'default', checks the operating systems running on the client and server and ignores filename case if either of them is OSX or Windows. Setting ignorecase to 'true' or 'false' overrides this behavior.
- Added a new preference, 'repeat', for the text user interface (only, at the moment). If 'repeat' is set to a number, then, after it finishes synchronizing, Unison will wait for that many seconds and then start over, continuing this way until it is killed from outside. Setting repeat to true will automatically set the batch preference to true.
- The 'rshargs' preference has been split into 'rshargs' and 'sshargs' (mainly to make the documentation clearer). In fact, 'rshargs' is no longer mentioned in the documentation at all, since pretty much everybody uses ssh now anyway.

• Documentation

  - The web pages have been completely redesigned and reorganized. (Thanks to Alan Schmitt for help with this.)

• User interface improvements

  - Added a GTK2 user interface, capable (among other things) of displaying filenames in any locale encoding. Kudos to Stephen Tse for contributing this code!
  - The text UI now prints a list of failed and skipped transfers at the end of synchronization.
  - Restarting update detection from the graphical UI will reload the current profile (which in particular will reset the -path preference, in case it has been narrowed by using the "Recheck unsynchronized items" command).
  - Several small improvements to the text user interface, including a progress display.

• Bug fixes (too numerous to count, actually, but here are some):

  - The `maxthreads` preference works now.
  - Fixed bug where warning message about uname returning an unrecognized result was preventing connection to server. (The warning is no longer printed, and all systems where 'uname' returns anything other than 'Darwin' are assumed not to be running OS X.)

- Fixed a problem on OS X that caused some valid file names (e.g., those including colons) to be considered invalid.
- Patched Path.followLink to follow links under cygwin in addition to Unix (suggested by Matt Swift).
- Small change to the storeRootsName function, suggested by bliviero at ichips.intel.com, to fix a problem in unison with the 'rootalias' option, which allows you to tell unison that two roots contain the same files. Rootalias was being applied after the hosts were sorted, so it wouldn't work properly in all cases.
- Incorporated a fix by Dmitry Bely for setting utimes of read-only files on Win32 systems.

- Installation / portability:
  - Unison now compiles with OCaml version 3.07 and later out of the box.
  - Makefile.OCaml fixed to compile out of the box under OpenBSD.
  - a few additional ports (e.g. OpenBSD, Zaurus/IPAQ) are now mentioned in the documentation
  - Unison can now be installed easily on OSX systems using the Fink package manager

Changes since 2.9.1:

- Added a preference `maxthreads` that can be used to limit the number of simultaneous file transfers.
- Added a `backupdir` preference, which controls where backup files are stored.
- Basic support added for OSX. In particular, Unison now recognizes when one of the hosts being synchronized is running OSX and switches to a case-insensitive treatment of filenames (i.e., 'foo' and 'FOO' are considered to be the same file). (OSX is not yet fully working, however: in particular, files with resource forks will not be synchronized correctly.)
- The same hash used to form the archive name is now also added to the names of the temp files created during file transfer. The reason for this is that, during update detection, we are going to silently delete any old temp files that we find along the way, and we want to prevent ourselves from deleting temp files belonging to other instances of Unison that may be running in parallel, e.g. synchronizing with a different host. Thanks to Ruslan Ermilov for this suggestion.
- Several small user interface improvements
- Documentation
  - FAQ and bug reporting instructions have been split out as separate HTML pages, accessible directly from the unison web page.
  - Additions to FAQ, in particular suggestions about performance tuning.
- Makefile
  - Makefile.OCaml now sets UISTYLE=text or UISTYLE=gtk automatically, depending on whether it finds lablgtk installed
  - Unison should now compile "out of the box" under OSX

Changes since 2.8.1:

- Changing profile works again under Windows
- File movement optimization: Unison now tries to use local copy instead of transfer for moved or copied files. It is controled by a boolean option "xferbycopying".
- Network statistics window (transfer rate, amount of data transferred). [NB: not available in Windows-Cygwin version.]

- symlinks work under the cygwin version (which is dynamically linked).
- Fixed potential deadlock when synchronizing between Windows and Unix
- Small improvements:
  - If neither the `USERPROFILE` nor the `HOME` environment variables are set, then Unison will put its temporary commit log (called `DANGER.README`) into the directory named by the `UNISON` environment variable, if any; otherwise it will use `C:`.
  - alternative set of values for fastcheck: yes = true; no = false; default = auto.
  - -silent implies -contactquietly
- Source code:
  - Code reorganization and tidying. (Started breaking up some of the basic utility modules so that the non-unison-specific stuff can be made available for other projects.)
  - several Makefile and docs changes (for release);
  - further comments in "update.ml";
  - connection information is not stored in global variables anymore.

Changes since 2.7.78:

- Small bugfix to textual user interface under Unix (to avoid leaving the terminal in a bad state where it would not echo inputs after Unison exited).

Changes since 2.7.39:

- Improvements to the main web page (stable and beta version docs are now both accessible).
- User manual revised.
- Added some new preferences:
  - "sshcmd" and "rshcmd" for specifying paths to ssh and rsh programs.
  - "contactquietly" for suppressing the "contacting server" message during Unison startup (under the graphical UI).
- Bug fixes:
  - Fixed small bug in UI that neglected to change the displayed column headers if loading a new profile caused the roots to change.
  - Fixed a bug that would put the text UI into an infinite loop if it encountered a conflict when run in batch mode.
  - Added some code to try to fix the display of non-Ascii characters in filenames on Windows systems in the GTK UI. (This code is currently untested—if you're one of the people that had reported problems with display of non-ascii filenames, we'd appreciate knowing if this actually fixes things.)
  - '`-prefer/-force newer`' works properly now. (The bug was reported by Sebastian Urbaniak and Sean Fulton.)
- User interface and Unison behavior:
  - Renamed 'Proceed' to 'Go' in the graphical UI.
  - Added exit status for the textual user interface.
  - Paths that are not synchronized because of conflicts or errors during update detection are now noted in the log file.

- [END] messages in log now use a briefer format
- Changed the text UI startup sequence so that ./unison -ui text will use the default profile instead of failing.
- Made some improvements to the error messages.
- Added some debugging messages to remote.ml.

Changes since 2.7.7:

- Incorporated, once again, a multi-threaded transport sub-system. It transfers several files at the same time, thereby making much more effective use of available network bandwidth. Unlike the earlier attempt, this time we do not rely on the native thread library of OCaml. Instead, we implement a light-weight, non-preemptive multi-thread library in OCaml directly. This version appears stable.

  Some adjustments to unison are made to accommodate the multi-threaded version. These include, in particular, changes to the user interface and logging, for example:

  - Two log entries for each transferring task, one for the beginning, one for the end.
  - Suppressed warning messages against removing temp files left by a previous unison run, because warning does not work nicely under multi-threading. The temp file names are made less likely to coincide with the name of a file created by the user. They take the form .#<filename>.<serial>.unison.tmp. [N.b. This was later changed to .unison.<filename>.<serial>.unison.

- Added a new command to the GTK user interface: pressing 'f' causes Unison to start a new update detection phase, using as paths *just* those paths that have been detected as changed and not yet marked as successfully completed. Use this command to quickly restart Unison on just the set of paths still needing attention after a previous run.

- Made the ignorecase preference user-visible, and changed the initialization code so that it can be manually set to true, even if neither host is running Windows. (This may be useful, e.g., when using Unison running on a Unix system with a FAT volume mounted.)

- Small improvements and bug fixes:

  - Errors in preference files now generate fatal errors rather than warnings at startup time. (I.e., you can't go on from them.) Also, we fixed a bug that was preventing these warnings from appearing in the text UI, so some users who have been running (unsuspectingly) with garbage in their prefs files may now get error reports.
  - Error reporting for preference files now provides file name and line number.
  - More intelligible message in the case of identical change to the same files: "Nothing to do: replicas have been changed only in identical ways since last sync."
  - Files with prefix '.#' excluded when scanning for preference files.
  - Rsync instructions are send directly instead of first marshaled.
  - Won't try forever to get the fingerprint of a continuously changing file: unison will give up after certain number of retries.
  - Other bug fixes, including the one reported by Peter Selinger (force=older preference not working).

- Compilation:

  - Upgraded to the new OCaml 3.04 compiler, with the LablGtk 1.2.3 library (patched version used for compiling under Windows).
  - Added the option to compile unison on the Windows platform with Cygwin GNU C compiler. This option only supports building dynamically linked unison executables.

Changes since 2.7.4:

- Fixed a silly (but debilitating) bug in the client startup sequence.

Changes since 2.7.1:

- Added `addprefsto` preference, which (when set) controls which preference file new preferences (e.g. new ignore patterns) are added to.
- Bug fix: read the initial connection header one byte at a time, so that we don't block if the header is shorter than expected. (This bug did not affect normal operation — it just made it hard to tell when you were trying to use Unison incorrectly with an old version of the server, since it would hang instead of giving an error message.)

Changes since 2.6.59:

- Changed `fastcheck` from a boolean to a string preference. Its legal values are `yes` (for a fast check), `no` (for a safe check), or `default` (for a fast check—which also happens to be safe—when running on Unix and a safe check when on Windows). The default is `default`.
- Several preferences have been renamed for consistency. All preference names are now spelled out in lowercase. For backward compatibility, the old names still work, but they are not mentioned in the manual any more.
- The temp files created by the 'diff' and 'merge' commands are now named by *pre*pending a new prefix to the file name, rather than appending a suffix. This should avoid confusing diff/merge programs that depend on the suffix to guess the type of the file contents.
- We now set the keepalive option on the server socket, to make sure that the server times out if the communication link is unexpectedly broken.
- Bug fixes:
  - When updating small files, Unison now closes the destination file.
  - File permissions are properly updated when the file is behind a followed link.
  - Several other small fixes.

Changes since 2.6.38:

- Major Windows performance improvement!

  We've added a preference `fastcheck` that makes Unison look only at a file's creation time and last-modified time to check whether it has changed. This should result in a huge speedup when checking for updates in large replicas.

  When this switch is set, Unison will use file creation times as 'pseudo inode numbers' when scanning Windows replicas for updates, instead of reading the full contents of every file. This may cause Unison to miss propagating an update if the create time, modification time, and length of the file are all unchanged by the update (this is not easy to achieve, but it can be done). However, Unison will never *overwrite* such an update with a change from the other replica, since it always does a safe check for updates just before propagating a change. Thus, it is reasonable to use this switch most of the time and occasionally run Unison once with `fastcheck` set to false, if you are worried that Unison may have overlooked an update.

  Warning: This change is has not yet been thoroughly field-tested. If you set the `fastcheck` preference, pay careful attention to what Unison is doing.

- New functionality: centralized backups and merging

- This version incorporates two pieces of major new functionality, implemented by Sylvain Roy during a summer internship at Penn: a *centralized backup* facility that keeps a full backup of (selected files in) each replica, and a *merging* feature that allows Unison to invoke an external file-merging tool to resolve conflicting changes to individual files.
- Centralized backups:
  * Unison now maintains full backups of the last-synchronized versions of (some of) the files in each replica; these function both as backups in the usual sense and as the "common version" when invoking external merge programs.
  * The backed up files are stored in a directory /.unison/backup on each host. (The name of this directory can be changed by setting the environment variable `UNISONBACKUPDIR`.)
  * The predicate `backup` controls which files are actually backed up: giving the preference '`backup = Path *`' causes backing up of all files.
  * Files are added to the backup directory whenever unison updates its archive. This means that
    · When unison reconstructs its archive from scratch (e.g., because of an upgrade, or because the archive files have been manually deleted), all files will be backed up.
    · Otherwise, each file will be backed up the first time unison propagates an update for it.
  * The preference `backupversions` controls how many previous versions of each file are kept. The default is 2 (i.e., the last synchronized version plus one backup).
  * For backward compatibility, the `backups` preference is also still supported, but `backup` is now preferred.
  * It is OK to manually delete files from the backup directory (or to throw away the directory itself). Before unison uses any of these files for anything important, it checks that its fingerprint matches the one that it expects.
- Merging:
  * Both user interfaces offer a new 'merge' command, invoked by pressing 'm' (with a changed file selected).
  * The actual merging is performed by an external program. The preferences `merge` and `merge2` control how this program is invoked. If a backup exists for this file (see the `backup` preference), then the `merge` preference is used for this purpose; otherwise `merge2` is used. In both cases, the value of the preference should be a string representing the command that should be passed to a shell to invoke the merge program. Within this string, the special substrings `CURRENT1`, `CURRENT2`, `NEW`, and `OLD` may appear at any point. Unison will substitute these as follows before invoking the command:
    · `CURRENT1` is replaced by the name of the local copy of the file;
    · `CURRENT2` is replaced by the name of a temporary file, into which the contents of the remote copy of the file have been transferred by Unison prior to performing the merge;
    · `NEW` is replaced by the name of a temporary file that Unison expects to be written by the merge program when it finishes, giving the desired new contents of the file; and
    · `OLD` is replaced by the name of the backed up copy of the original version of the file (i.e., its state at the end of the last successful run of Unison), if one exists (applies only to `merge`, not `merge2`).

    For example, on Unix systems setting the `merge` preference to

    ```
    merge = diff3 -m CURRENT1 OLD CURRENT2 > NEW
    ```

    will tell Unison to use the external `diff3` program for merging.

    A large number of external merging programs are available. For example, `emacs` users may find the following convenient:

    ```
    merge2 = emacs -q --eval '(ediff-merge-files "CURRENT1" "CURRENT2"
               nil "NEW")'
    merge = emacs -q --eval '(ediff-merge-files-with-ancestor
               "CURRENT1" "CURRENT2" "OLD" nil "NEW")'
    ```

(These commands are displayed here on two lines to avoid running off the edge of the page. In your preference file, each should be written on a single line.)

  * If the external program exits without leaving any file at the path `NEW`, Unison considers the merge to have failed. If the merge program writes a file called `NEW` but exits with a non-zero status code, then Unison considers the merge to have succeeded but to have generated conflicts. In this case, it attempts to invoke an external editor so that the user can resolve the conflicts. The value of the `editor` preference controls what editor is invoked by Unison. The default is `emacs`.
  * Please send us suggestions for other useful values of the `merge2` and `merge` preferences – we'd like to give several examples in the manual.

- Smaller changes:
  - When one preference file includes another, unison no longer adds the suffix '`.prf`' to the included file by default. If a file with precisely the given name exists in the .unison directory, it will be used; otherwise Unison will add `.prf`, as it did before. (This change means that included preference files can be named `blah.include` instead of `blah.prf`, so that unison will not offer them in its 'choose a preference file' dialog.)
  - For Linux systems, we now offer both a statically linked and a dynamically linked executable. The static one is larger, but will probably run on more systems, since it doesn't depend on the same versions of dynamically linked library modules being available.
  - Fixed the `force` and `prefer` preferences, which were getting the propagation direction exactly backwards.
  - Fixed a bug in the startup code that would cause unison to crash when the default profile (`~/.unison/default.prf`) does not exist.
  - Fixed a bug where, on the run when a profile is first created, Unison would confusingly display the roots in reverse order in the user interface.
- For developers:
  - We've added a module dependency diagram to the source distribution, in `src/DEPENDENCIES.ps`, to help new prospective developers with navigating the code.

Changes since 2.6.11:

- **Incompatible change:** Archive format has changed.
- **Incompatible change:** The startup sequence has been completely rewritten and greatly simplified. The main user-visible change is that the `defaultpath` preference has been removed. Its effect can be approximated by using multiple profiles, with `include` directives to incorporate common settings. All uses of `defaultpath` in existing profiles should be changed to `path`.

  Another change in startup behavior that will affect some users is that it is no longer possible to specify roots *both* in the profile *and* on the command line.

  You can achieve a similar effect, though, by breaking your profile into two:

```
default.prf =
    root = blah
    root = foo
    include common

common.prf =
    <everything else>
```

Now do

```
unison common root1 root2
```

when you want to specify roots explicitly.

- The `-prefer` and `-force` options have been extended to allow users to specify that files with more recent modtimes should be propagated, writing either `-prefer newer` or `-force newer`. (For symmetry, Unison will also accept `-prefer older` or `-force older`.) The `-force older/newer` options can only be used when `-times` is also set.

  The graphical user interface provides access to these facilities on a one-off basis via the `Actions` menu.

- Names of roots can now be “aliased” to allow replicas to be relocated without changing the name of the archive file where Unison stores information between runs. (This feature is for experts only. See the “Archive Files” section of the manual for more information.)

- Graphical user-interface:
  - A new command is provided in the Synchronization menu for switching to a new profile without restarting Unison from scratch.
  - The GUI also supports one-key shortcuts for commonly used profiles. If a profile contains a preference of the form ’`key = n`’, where `n` is a single digit, then pressing this key will cause Unison to immediately switch to this profile and begin synchronization again from scratch. (Any actions that may have been selected for a set of changes currently being displayed will be discarded.)
  - Each profile may include a preference ’`label = <string>`’ giving a descriptive string that described the options selected in this profile. The string is listed along with the profile name in the profile selection dialog, and displayed in the top-right corner of the main Unison window.

- Minor:
  - Fixed a bug that would sometimes cause the ’diff’ display to order the files backwards relative to the main user interface. (Thanks to Pascal Brisset for this fix.)
  - On Unix systems, the graphical version of Unison will check the `DISPLAY` variable and, if it is not set, automatically fall back to the textual user interface.
  - Synchronization paths (`path` preferences) are now matched against the ignore preferences. So if a path is both specified in a `path` preference and ignored, it will be skipped.
  - Numerous other bugfixes and small improvements.

Changes since 2.6.1:

- The synchronization of modification times has been disabled for directories.
- Preference files may now include lines of the form `include <name>`, which will cause `name.prf` to be read at that point.
- The synchronization of permission between Windows and Unix now works properly.
- A binding `CYGWIN=binmode` in now added to the environment so that the Cygwin port of OpenSSH works properly in a non-Cygwin context.
- The `servercmd` and `addversionno` preferences can now be used together: `-addversionno` appends an appropriate `-NNN` to the server command, which is found by using the value of the `-servercmd` preference if there is one, or else just `unison`.
- Both `’-pref=val’` and `’-pref val’` are now allowed for boolean values. (The former can be used to set a preference to false.)

- Lot of small bugs fixed.

Changes since 2.5.31:

- The `log` preference is now set to `true` by default, since the log file seems useful for most users.
- Several miscellaneous bugfixes (most involving symlinks).

Changes since 2.5.25:

- **Incompatible change:** Archive format has changed (again).
- Several significant bugs introduced in 2.5.25 have been fixed.

Changes since 2.5.1:

- **Incompatible change:** Archive format has changed. Make sure you synchronize your replicas before upgrading, to avoid spurious conflicts. The first sync after upgrading will be slow.
- New functionality:
  - Unison now synchronizes file modtimes, user-ids, and group-ids.

    These new features are controlled by a set of new preferences, all of which are currently `false` by default.
    * When the `times` preference is set to `true`, file modification times are propaged. (Because the representations of time may not have the same granularity on both replicas, Unison may not always be able to make the modtimes precisely equal, but it will get them as close as the operating systems involved allow.)
    * When the `owner` preference is set to `true`, file ownership information is synchronized.
    * When the `group` preference is set to `true`, group information is synchronized.
    * When the `numericIds` preference is set to `true`, owner and group information is synchronized numerically. By default, owner and group numbers are converted to names on each replica and these names are synchronized. (The special user id 0 and the special group 0 are never mapped via user/group names even if this preference is not set.)
  - Added an integer-valued preference `perms` that can be used to control the propagation of permission bits. The value of this preference is a mask indicating which permission bits should be synchronized. It is set by default to $0o1777$: all bits but the set-uid and set-gid bits are synchronised (synchronizing theses latter bits can be a security hazard). If you want to synchronize all bits, you can set the value of this preference to -1.
  - Added a `log` preference (default `false`), which makes Unison keep a complete record of the changes it makes to the replicas. By default, this record is written to a file called `unison.log` in the user's home directory (the value of the `HOME` environment variable). If you want it someplace else, set the `logfile` preference to the full pathname you want Unison to use.
  - Added an `ignorenot` preference that maintains a set of patterns for paths that should definitely *not* be ignored, whether or not they match an `ignore` pattern. (That is, a path will now be ignored iff it matches an ignore pattern and does not match any ignorenot patterns.)
- User-interface improvements:
  - Roots are now displayed in the user interface in the same order as they were given on the command line or in the preferences file.
  - When the `batch` preference is set, the graphical user interface no longer waits for user confirmation when it displays a warning message: it simply pops up an advisory window with a Dismiss button at the bottom and keeps on going.

  - Added a new preference for controlling how many status messages are printed during update detection: `statusdepth` controls the maximum depth for paths on the local machine (longer paths are not displayed, nor are non-directory paths). The value should be an integer; default is 1.
  - Removed the `trace` and `silent` preferences. They did not seem very useful, and there were too many preferences for controlling output in various ways.
  - The text UI now displays just the default command (the one that will be used if the user just types `<return>`) instead of all available commands. Typing `?` will print the full list of possibilities.
  - The function that finds the canonical hostname of the local host (which is used, for example, in calculating the name of the archive file used to remember which files have been synchronized) normally uses the `gethostname` operating system call. However, if the environment variable `UNISONLOCALHOSTNAME` is set, its value will now be used instead. This makes it easier to use Unison in situations where a machine's name changes frequently (e.g., because it is a laptop and gets moved around a lot).
  - File owner and group are now displayed in the "detail window" at the bottom of the screen, when unison is configured to synchronize them.

- For hackers:

  - Updated to Jacques Garrigue's new version of `lablgtk`, which means we can throw away our local patched version.

    If you're compiling the GTK version of unison from sources, you'll need to update your copy of lablgtk to the developers release. (Warning: installing lablgtk under Windows is currently a bit challenging.)
  - The TODO.txt file (in the source distribution) has been cleaned up and reorganized. The list of pending tasks should be much easier to make sense of, for people that may want to contribute their programming energies. There is also a separate file BUGS.txt for open bugs.
  - The Tk user interface has been removed (it was not being maintained and no longer compiles).
  - The `debug` preference now prints quite a bit of additional information that should be useful for identifying sources of problems.
  - The version number of the remote server is now checked right away during the connection setup handshake, rather than later. (Somebody sent a bug report of a server crash that turned out to come from using inconsistent versions: better to check this earlier and in a way that can't crash either client or server.)
  - Unison now runs correctly on 64-bit architectures (e.g. Alpha linux). We will not be distributing binaries for these architectures ourselves (at least for a while) but if someone would like to make them available, we'll be glad to provide a link to them.

- Bug fixes:

  - Pattern matching (e.g. for `ignore`) is now case-insensitive when Unison is in case-insensitive mode (i.e., when one of the replicas is on a windows machine).
  - Some people had trouble with mysterious failures during propagation of updates, where files would be falsely reported as having changed during synchronization. This should be fixed.
  - Numerous smaller fixes.

Changes since 2.4.1:

- Added a number of 'sorting modes' for the user interface. By default, conflicting changes are displayed at the top, and the rest of the entries are sorted in alphabetical order. This behavior can be changed in the following ways:

  - Setting the sortnewfirst preference to true causes newly created files to be displayed before changed files.
  - Setting sortbysize causes files to be displayed in increasing order of size.
  - Giving the preference sortfirst=<pattern> (where <pattern> is a path descriptor in the same format as 'ignore' and 'follow' patterns, causes paths matching this pattern to be displayed first.
  - Similarly, giving the preference sortlast=<pattern> causes paths matching this pattern to be displayed last.

  The sorting preferences are described in more detail in the user manual. The sortnewfirst and sortbysize flags can also be accessed from the 'Sort' menu in the grpahical user interface.

- Added two new preferences that can be used to change unison's fundamental behavior to make it more like a mirroring tool instead of a synchronizer.
  - Giving the preference prefer with argument <root> (by adding -prefer <root> to the command line or prefer=<root>) to your profile) means that, if there is a conflict, the contents of <root> should be propagated to the other replica (with no questions asked). Non-conflicting changes are treated as usual.
  - Giving the preference force with argument <root> will make unison resolve *all* differences in favor of the given root, even if it was the other replica that was changed.

  These options should be used with care! (More information is available in the manual.)

- Small changes:
  - Changed default answer to 'Yes' in all two-button dialogs in the graphical interface (this seems more intuitive).
  - The rsync preference has been removed (it was used to activate rsync compression for file transfers, but rsync compression is now enabled by default).
  - In the text user interface, the arrows indicating which direction changes are being propagated are printed differently when the user has overridded Unison's default recommendation (====> instead of ---->). This matches the behavior of the graphical interface, which displays such arrows in a different color.
  - Carriage returns (Control-M's) are ignored at the ends of lines in profiles, for Windows compatibility.
  - All preferences are now fully documented in the user manual.

Changes since 2.3.12:

- **Incompatible change:** Archive format has changed. Make sure you synchronize your replicas before upgrading, to avoid spurious conflicts. The first sync after upgrading will be slow.
- New/improved functionality:
  - A new preference -sortbysize controls the order in which changes are displayed to the user: when it is set to true, the smallest changed files are displayed first. (The default setting is false.)
  - A new preference -sortnewfirst causes newly created files to be listed before other updates in the user interface.
  - We now allow the ssh protocol to specify a port.
  - Incompatible change: The unison: protocol is deprecated, and we added file: and socket:. You may have to modify your profiles in the .unison directory. If a replica is specified without an explicit protocol, we now assume it refers to a file. (Previously "//saul/foo" meant to use SSH to connect to saul, then access the foo directory. Now it means to access saul via a remote file mechanism such as samba; the old effect is now achieved by writing ssh://saul/foo.)

- Changed the startup sequence for the case where roots are given but no profile is given on the command line. The new behavior is to use the default profile (creating it if it does not exist), and temporarily override its roots. The manual claimed that this case would work by reading no profile at all, but AFAIK this was never true.
  - In all user interfaces, files with conflicts are always listed first
  - A new preference 'sshversion' can be used to control which version of ssh should be used to connect to the server. Legal values are 1 and 2. (Default is empty, which will make unison use whatever version of ssh is installed as the default 'ssh' command.)
  - The situation when the permissions of a file was updated the same on both side is now handled correctly (we used to report a spurious conflict)

- Improvements for the Windows version:
  - The fact that filenames are treated case-insensitively under Windows should now be handled correctly. The exact behavior is described in the cross-platform section of the manual.
  - It should be possible to synchronize with Windows shares, e.g., //host/drive/path.
  - Workarounds to the bug in syncing root directories in Windows. The most difficult thing to fix is an ocaml bug: Unix.opendir fails on c: in some versions of Windows.

- Improvements to the GTK user interface (the Tk interface is no longer being maintained):
  - The UI now displays actions differently (in blue) when they have been explicitly changed by the user from Unison's default recommendation.
  - More colorful appearance.
  - The initial profile selection window works better.
  - If any transfers failed, a message to this effect is displayed along with 'Synchronization complete' at the end of the transfer phase (in case they may have scrolled off the top).
  - Added a global progress meter, displaying the percentage of *total* bytes that have been transferred so far.

- Improvements to the text user interface:
  - The file details will be displayed automatically when a conflict is been detected.
  - when a warning is generated (e.g. for a temporary file left over from a previous run of unison) Unison will no longer wait for a response if it is running in -batch mode.
  - The UI now displays a short list of possible inputs each time it waits for user interaction.
  - The UI now quits immediately (rather than looping back and starting the interaction again) if the user presses 'q' when asked whether to propagate changes.
  - Pressing 'g' in the text user interface will proceed immediately with propagating updates, without asking any more questions.

- Documentation and installation changes:
  - The manual now includes a FAQ, plus sections on common problems and on tricks contributed by users.
  - Both the download page and the download directory explicitly say what are the current stable and beta-test version numbers.
  - The OCaml sources for the up-to-the-minute developers' version (not guaranteed to be stable, or even to compile, at any given time!) are now available from the download page.
  - Added a subsection to the manual describing cross-platform issues (case conflicts, illegal filenames)

- Many small bug fixes and random improvements.

Changes since 2.3.1:

- Several bug fixes. The most important is a bug in the rsync module that would occasionally cause change propagation to fail with a 'rename' error.

Changes since 2.2:

- The multi-threaded transport system is now disabled by default. (It is not stable enough yet.)
- Various bug fixes.
- A new experimental feature:

  The final component of a -path argument may now be the wildcard specifier `*`. When Unison sees such a path, it expands this path on the client into into the corresponding list of paths by listing the contents of that directory.

  Note that if you use wildcard paths from the command line, you will probably need to use quotes or a backslash to prevent the * from being interpreted by your shell.

  If both roots are local, the contents of the first one will be used for expanding wildcard paths. (Nb: this is the first one *after* the canonization step – i.e., the one that is listed first in the user interface – not the one listed first on the command line or in the preferences file.)

Changes since 2.1:

- The transport subsystem now includes an implementation by Sylvain Gommier and Norman Ramsey of Tridgell and Mackerras's `rsync` protocol. This protocol achieves much faster transfers when only a small part of a large file has been changed by sending just diffs. This feature is mainly helpful for transfers over slow links—on fast local area networks it can actually degrade performance—so we have left it off by default. Start unison with the `-rsync` option (or put `rsync=true` in your preferences file) to turn it on.
- "Progress bars" are now diplayed during remote file transfers, showing what percentage of each file has been transferred so far.
- The version numbering scheme has changed. New releases will now be have numbers like 2.2.30, where the second component is incremented on every significant public release and the third component is the "patch level."
- Miscellaneous improvements to the GTK-based user interface.
- The manual is now available in PDF format.
- We are experimenting with using a multi-threaded transport subsystem to transfer several files at the same time, making much more effective use of available network bandwidth. This feature is not completely stable yet, so by default it is disabled in the release version of Unison.

  If you want to play with the multi-threaded version, you'll need to recompile Unison from sources (as described in the documentation), setting the THREADS flag in Makefile.OCaml to true. Make sure that your OCaml compiler has been installed with the `-with-pthreads` configuration option. (You can verify this by checking whether the file `threads/threads.cma` in the OCaml standard library directory contains the string `-lpthread` near the end.)

Changes since 1.292:

- Reduced memory footprint (this is especially important during the first run of unison, where it has to gather information about all the files in both repositories).
- Fixed a bug that would cause the socket server under NT to fail after the client exits.

- Added a SHIFT modifier to the Ignore menu shortcut keys in GTK interface (to avoid hitting them accidentally).

Changes since 1.231:

- Tunneling over ssh is now supported in the Windows version. See the installation section of the manual for detailed instructions.
- The transport subsystem now includes an implementation of the `rsync` protocol, built by Sylvain Gommier and Norman Ramsey. This protocol achieves much faster transfers when only a small part of a large file has been changed by sending just diffs. The rsync feature is off by default in the current version. Use the `-rsync` switch to turn it on. (Nb. We still have a lot of tuning to do: you may not notice much speedup yet.)
- We're experimenting with a multi-threaded transport subsystem, written by Jerome Vouillon. The downloadable binaries are still single-threaded: if you want to try the multi-threaded version, you'll need to recompile from sources. (Say `make THREADS=true`.) Native thread support from the compiler is required. Use the option `-threads N` to select the maximal number of concurrent threads (default is 5). Multi-threaded and single-threaded clients/servers can interoperate.
- A new GTK-based user interface is now available, thanks to Jacques Garrigue. The Tk user interface still works, but we'll be shifting development effort to the GTK interface from now on.
- OCaml 3.00 is now required for compiling Unison from sources. The modules `uitk` and `myfileselect` have been changed to use labltk instead of camltk. To compile the Tk interface in Windows, you must have ocaml-3.00 and tk8.3. When installing tk8.3, put it in `c:\Tcl` rather than the suggested `c:\Program Files\Tcl`, and be sure to install the headers and libraries (which are not installed by default).
- Added a new `-addversionno` switch, which causes unison to use `unison-<currentversionnumber>` instead of just `unison` as the remote server command. This allows multiple versions of unison to coexist conveniently on the same server: whichever version is run on the client, the same version will be selected on the server.

Changes since 1.219:

- **Incompatible change:** Archive format has changed. Make sure you synchronize your replicas before upgrading, to avoid spurious conflicts. The first sync after upgrading will be slow.
- This version fixes several annoying bugs, including:
  - Some cases where propagation of file permissions was not working.
  - umask is now ignored when creating directories
  - directories are create writable, so that a read-only directory and its contents can be propagated.
  - Handling of warnings generated by the server.
  - Synchronizing a path whose parent is not a directory on both sides is now flagged as erroneous.
  - Fixed some bugs related to symnbolic links and nonexistant roots.
    * When a change (deletion or new contents) is propagated onto a 'follow'ed symlink, the file pointed to by the link is now changed. (We used to change the link itself, which doesn't fit our assertion that 'follow' means the link is completely invisible)
    * When one root did not exist, propagating the other root on top of it used to fail, because unison could not calculate the working directory into which to write changes. This should be fixed.
- A human-readable timestamp has been added to Unison's archive files.

- The semantics of Path and Name regular expressions now correspond better.
- Some minor improvements to the text UI (e.g. a command for going back to previous items)
- The organization of the export directory has changed — should be easier to find / download things now.

Changes since 1.200:

- **Incompatible change:** Archive format has changed. Make sure you synchronize your replicas before upgrading, to avoid spurious conflicts. The first sync after upgrading will be slow.
- This version has not been tested extensively on Windows.
- Major internal changes designed to make unison safer to run at the same time as the replicas are being changed by the user.
- Internal performance improvements.

Changes since 1.190:

- **Incompatible change:** Archive format has changed. Make sure you synchronize your replicas before upgrading, to avoid spurious conflicts. The first sync after upgrading will be slow.
- A number of internal functions have been changed to reduce the amount of memory allocation, especially during the first synchronization. This should help power users with very big replicas.
- Reimplementation of low-level remote procedure call stuff, in preparation for adding rsync-like smart file transfer in a later release.
- Miscellaneous bug fixes.

Changes since 1.180:

- **Incompatible change:** Archive format has changed. Make sure you synchronize your replicas before upgrading, to avoid spurious conflicts. The first sync after upgrading will be slow.
- Fixed some small bugs in the interpretation of ignore patterns.
- Fixed some problems that were preventing the Windows version from working correctly when click-started.
- Fixes to treatment of file permissions under Windows, which were causing spurious reports of different permissions when synchronizing between windows and unix systems.
- Fixed one more non-tail-recursive list processing function, which was causing stack overflows when synchronizing very large replicas.

Changes since 1.169:

- The text user interface now provides commands for ignoring files.
- We found and fixed some *more* non-tail-recursive list processing functions. Some power users have reported success with very large replicas.
- **Incompatible change:** Files ending in `.tmp` are no longer ignored automatically. If you want to ignore such files, put an appropriate ignore pattern in your profile.
- **Incompatible change:** The syntax of `ignore` and `follow` patterns has changed. Instead of putting a line of the form

```
ignore = <regexp>
```

in your profile (`.unison/default.prf`), you should put:

```
ignore = Regex <regexp>
```

Moreover, two other styles of pattern are also recognized:

```
ignore = Name <name>
```

matches any path in which one component matches `<name>`, while

```
ignore = Path <path>
```

matches exactly the path `<path>`.

Standard "globbing" conventions can be used in `<name>` and `<path>`:

- a `?` matches any single character except `/`
- a `*` matches any sequence of characters not including `/`
- `[xyz]` matches any character from the set {`x`, `y`, `z`}
- `{a,bb,ccc}` matches any one of `a`, `bb`, or `ccc`.

See the user manual for some examples.

Changes since 1.146:

- Some users were reporting stack overflows when synchronizing huge directories. We found and fixed some non-tail-recursive list processing functions, which we hope will solve the problem. Please give it a try and let us know.
- Major additions to the documentation.

Changes since 1.142:

- Major internal tidying and many small bugfixes.
- Major additions to the user manual.
- Unison can now be started with no arguments – it will prompt automatically for the name of a profile file containing the roots to be synchronized. This makes it possible to start the graphical UI from a desktop icon.
- Fixed a small bug where the text UI on NT was raising a 'no such signal' exception.

Changes since 1.139:

- The precompiled windows binary in the last release was compiled with an old OCaml compiler, causing propagation of permissions not to work (and perhaps leading to some other strange behaviors we've heard reports about). This has been corrected. If you're using precompiled binaries on Windows, please upgrade.
- Added a `-debug` command line flag, which controls debugging of various modules. Say `-debug XXX` to enable debug tracing for module `XXX`, or `-debug all` to turn on absolutely everything.
- Fixed a small bug where the text UI on NT was raising a 'no such signal' exception.

Changes since 1.111:

- **Incompatible change:** The names and formats of the preference files in the .unison directory have changed. In particular:

  - the file “prefs” should be renamed to default.prf
  - the contents of the file “ignore” should be merged into default.prf. Each line of the form `REGEXP` in ignore should become a line of the form `ignore = REGEXP` in default.prf.

- Unison now handles permission bits and symbolic links. See the manual for details.
- You can now have different preference files in your .unison directory. If you start unison like this

  ```
  unison profilename
  ```

  (i.e. with just one “anonymous” command-line argument), then the file `~/.unison/profilename.prf` will be loaded instead of `default.prf`.
- Some improvements to terminal handling in the text user interface
- Added a switch -killServer that terminates the remote server process when the unison client is shutting down, even when using sockets for communication. (By default, a remote server created using ssh/rsh is terminated automatically, while a socket server is left running.)
- When started in 'socket server' mode, unison prints 'server started' on stderr when it is ready to accept connections. (This may be useful for scripts that want to tell when a socket-mode server has finished initalization.)
- We now make a nightly mirror of our current internal development tree, in case anyone wants an up-to-the-minute version to hack around with.
- Added a file CONTRIB with some suggestions for how to help us make Unison better.