Unison File Synchronizer

`http://www.cis.upenn.edu/∼bcpierce/unison`

Version 2.9.20

User Manual and Reference

Contents

Unison is a file-synchronization tool for Unix and Windows. (It also works on OSX to some extent, but it does not yet deal with 'resource forks' correctly; more information on OSX usage can be found on the `unison-users` mailing list archives.) 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, and 2k) and Unix (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 hack (or own!) 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 `rsh` or 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.

1 Preface

1.1 People

Benjamin Pierce is the Unison project leader. The current version of Unison was designed and implemented by Trevor Jim, Benjamin Pierce, and Jérôme Vouillon, with Zhe Yang, Sylvain Gommier, and Matthieu Goulay.

Our implementation of the rsync protocol was built by Norman Ramsey and Sylvain Gommier. It is is based on Andrew Tridgell's thesis work and inspired by his rsync utility. The mirroring and merging functionality was implemented by Sylvain Roy. Jacques Garrigue contributed the original Gtk version of the user interface. 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.

1.2 Mailing Lists and Bug Reporting

Mailing Lists: It is strongly recommended that all Unison users subscribe to one of the first two:

- **unison-announce** is a moderated list where new Unison releases are announced. It is very low volume, averaging about one message per month.

  To subscribe, you can either visit `http://groups.yahoo.com/group/unison-announce` (you will be asked to create a Yahoo groups account if you do not already have one), or else send an email to **unison-announce-subscribe@groups.yahoo.com** (which will simply add you to the list, whether you have a Yahoo account or not).

- **unison-users** is a somewhat-higher-volume list for users of unison. It is used for discussions of many sorts — proposals and designs for new features, installation and configuration questions, usage tips, etc. It is also moderated, but just to filter spam.

  To subscribe, you can either visit `http://groups.yahoo.com/group/unison-users` or else send an email to **unison-users-subscribe@groups.yahoo.com**.

  Release announcements are made on both of these lists, so there is no need to subscribe to both.

- **unison-hackers** is for informal discussion among Unison developers.. Anyone who considers themselves a Unison expert and wishes to lend a hand with maintaining and improving Unison is welcome to join. This list is moderated to filter spam.

  To subscribe, you can either visit `http://groups.yahoo.com/group/unison-hackers` or else send an email to **unison-hackers-subscribe@groups.yahoo.com**.

Archives of all the lists are available (and publically visible) via the above links.

Reporting bugs: If Unison is not working the way you expect, here are some steps to follow.

- First, take a look at the Unison documentation, especially the FAQ section. Lots of questions are answered there.
- Next, try running Unison with the `-debug all` command line option. This will cause Unison to generate a detailed trace of what it's doing, which may help pinpoint where the problem is occurring.
- If this doesn't clarify matters, try sending an email describing your problem to the users list at `unison-users@groups.yahoo.com`. (Make sure you subscribe first, so that you'll see people's responses in case they reply only to the list!) Please include the version of Unison you are using (`unison -version`), the kind of machine(s) you are running it on, a record of what gets printed when the `-debug all` option is included, and as much information as you can about what went wrong.

Feature Requests: Please post your feature requests, suggestions, etc. to the **unison-users** list.

1.3 Development Status

Unison is no longer under active development as a research project. (Our research efforts in this area 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 **unison-users** mailing list.

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

Patches are even more welcome. They should be sent directly to the project leader, Benjamin Pierce, at `bcpierce@cis.upenn.edu`. (Caveat: 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`.

1.4 Copying

Unison is free software. You are free to change and redistribute it under the terms of the GNU General Public License. Please see the file COPYING in the Unison distribution for more information.

1.5 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).

2 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 in Section 6 [FAQ] helpful.)

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 Gtk.

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.

2.1 Downloading 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 2.5 [Building Unison]. (There are also a small number of "contributed ports" to other architectures that are not maintained by us. See Section 2.4 [Contributed Ports] 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 several different modes: with different directories on a single machine, with a remote machine over a direct socket connection, with a remote machine using `rsh` (on Unix systems), or with a remote Unix system (from either a Unix or a Windows client) 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].

2.2 Running Unison

Once you've got Unison installed on at least one system, read Section 3 [Tutorial] of the user manual (or type `unison -doc tutorial`) for instructions on how to get started.

2.3 Upgrading

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

Before upgrading, it is a good idea to use the *old* version 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 and need to be resolved manually.

2.4 Contributed Ports

A few people have offered to maintain pre-built executables, easy installation scripts, etc., for particular architectures. They are not maintained by us and are not guaranteed to work, be kept up to date with our latest releases, etc., but you may find them useful. Here's what's available at the moment:

- Dan Pelleg has ported unison to FreeBSD. This means that any FreeBSD user with an up-to-date "ports" collection can install unison by doing: `cd /usr/ports/net/unison; make && make install`. (Make sure your "ports" collection is fully up to date before doing this, to ensure that you get the most recent Unison version that has been compiled for FreeBSD.)

  FreeBSD binaries can also be obtained directly from

  `http://www.freebsd.org/cgi/ports.cgi?query=unison&stype=all`.

- Andrew Pitts has built binaries for some versions of Unison for the Linux-PPC platform. They can be found in `ftp://ftp.cl.cam.ac.uk/papers/amp12/unison/`.

- Robert McQueen maintains a Debian package for Unison. The homepage is located at

  `http://packages.debian.org/testing/non-us/unison.html`.

- Chris Cocosco provides binaries for Unison under SGI IRIX (6.5). They can be found in

  `www.bic.mni.mcgill.ca/users/crisco/unison.irix/`.

- Christian Marker has translated the Unison manual into German:

  `http://www.linux-is-cool.de/projects/doc/unison`

2.5 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. In particular, it has been tested on many flavors of Windows (98, NT, 2000) and Unix (Solaris, Linux, FreeBSD), and on both 32- and 64-bit architectures. Unison *partly* works on Mac OSX (see Section 6 [FAQ] for caveats); it does not work on earlier MacOS systems.

2.5.1 Unix

You'll need the Objective Caml compiler (version 3.04 or later[1]), which is available from its official site `http://caml.inria.fr`. Building and installing OCaml on Unix systems is very straightforward; follow the instructions in the distribution. You'll probably want to build the native-code compiler in addition to the bytecode compiler, 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, then 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.)

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
```

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 a graphical user interface, choose one of the following:

[1]If you are compiling Unison 2.7.7 or an earlier version, you need to

- insert a line "`CAMLFLAGS+=-nolabels` to the file named "`Makefile.OCaml`" in the source directory, and
- install LablGtk 1.1.1 instead of the latest version.

- Gtk interface:
  You will need Gtk (version 1.2 or later, available from `http://www.gtk.org` and standard on many Unix installations).

  You also need the get LablGtk (version 1.1.3 is known to work). 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, typing

  ```
      make
  ```

  again should build a `unison` executable with a Gtk graphical interface. (In previous releases of Unison, it was necessary to add `UISTYLE=gtk` to the 'make' command above. This requirement has been removed: the makefile should detect automatically when lablgtk is present and set this flag automatically.)

If this step does not work, don't worry: Unison works fine with the textual interface.

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.

2.5.2 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 3.04 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: To build a more efficient, native version of Unison on Windows, you can choose between two options. Both options require the OCaml distribution version 3.04 (later versions will probably work too, but there have been some changes in the lablgtk distribution beginning with 3.05 that should make some of the following steps unnecessary) as well as the Cygwin layer, which provides certain GNU tools. The two options differ in the C compiler employed: MS Visual C++ (MSVC) vs. Cygwin GNU C.

The tradeoff?

- Only the MSVC option can produce statically linked Unison executable.
- The Cygwin GNU C option requires only free software.

The files "INSTALL.win32-msvc" and "INSTALL.win32-cygwin-gnuc" describe the building procedures for the respective options.

2.5.3 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.

3 Tutorial

3.1 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.

The command-line arguments to both versions are identical. The graphical version can be run directly by clicking on its icon, but this requires a little set-up (see Section 5.18 [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 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.*

3.2 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 2 [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
```

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**ollw 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 overriden 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 a.tmp to propagate to b.tmp) or the right-arrow or "`>`" key (which makes the b.tmp version override a.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.

3.3 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 5.4 [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` or `rsh`. 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 3.4 [Remote Shell Method] or Section 3.5 [Socket Method], as appropriate.

3.4 Remote Shell Method

The standard remote shell facility on Unix systems is `rsh`. A drop-in replacement for `rsh` is `ssh`, which provides the same functionality but much better security. (Ssh is available from `ftp://ftp.cs.hut.fi/pub/ssh/`; up-to-date binaries for some architectures can also be found at `ftp://ftp.faqs.org/ssh/contrib`. See section A.2 for installation instructions for the Windows version.) Both `rsh` and `ssh` require 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 `rsh` or `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 5.5 [Profile]). Similarly, you can specify a explicit path for the `rsh` or `ssh` program using the option `-rshcmd` or `-sshcmd`.

3.5 Socket 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.

> **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.)

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.

3.6 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 5.12 [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 5.13 [Symbolic Links]) to make sure that all these links are treated transparently by Unison.

4. Make your home directory the root of the synchronization, but tell Unison to synchronize only some of the files and subdirectories within it. 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 5.4 [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 5.5 [Profile] covers the syntax of Unison profiles, together with some sample profiles.

Some tips on improving Unison's performance can be found in Section 6.3 [Tips and Tricks].

3.7 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 1.2 [Mailing Lists].

4 Basic Concepts

Unison deals in a few straightforward concepts. (A more mathematical development of these concepts can be found in "*What is a File Synchronizer?*" (http://www.cis.upenn.edu/~bcpierce/papers/snc-mobicom.ps.gz) by Sundar Balasubramaniam and Benjamin Pierce [MobiCom 1998]. A more up-to-date version can be found in a recent set of slides (http://www.cis.upenn.edu/~bcpierce/papers/snc-tacs-2001Oct.ps).)

4.1 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.

4.2 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.)

4.3 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.

(What Unison actually calculates is a slight approximation to this definition; see Section 4.7 [Caveats and Shortcomings].)

4.4 What is a Conflict?

A path is said to be *conflicting* if

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.

4.5 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.

4.6 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 (deprecated) `backups` flag is set, Unison will leave around old versions of files, 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 Unison's behavior. (On Unix, it 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.)

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.

4.7 Caveats and Shortcomings

Here are some things to be careful of when using Unison. A complete list of bugs can be found in the file BUGS.txt in the source distribution.

- 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 4.3 [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" 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.

- The graphical user interface is currently 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 currently understand hard links.

5 Reference

This section covers the features of Unison in detail.

5.1 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).

5.2 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 the profile files in the `.unison` directory is definitely OK.)

5.3 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
```

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.

5.4 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.

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]

Options:
  -addprefsto xxx     file to add new prefs to
  -addversionno       add version number to name of unison executable on server
  -auto               automatically accept default actions
  -backup xxx         add a regexp to the backup list
  -backupdir xxx      Location for backups created by -backup
  -backupnot xxx      add a regexp to the backupnot list
  -backups            keep backup copies of files (deprecated: use 'backup')
  -batch              batch mode: ask no questions at all
  -contactquietly      Suppress the 'contacting server' message during startup
  -debug xxx          debug module xxx ('all' -> everything, 'verbose' -> more)
  -doc xxx            show documentation ('-doc topics' lists topics)
  -dumbtty            do not try to change terminal settings in text UI
  -editor xxx         command for displaying the output of the merge program
  -fastcheck xxx      do fast update detection (‘true’, ‘false’, or ‘default’)
  -follow xxx         add a regexp to the follow list
  -force xxx          force changes from this replica to the other
  -group              synchronize group
  -height n           height (in lines) of main window in graphical interface
  -ignore xxx         add a regexp to the ignore list
  -ignorecase         ignore upper/lowercase spelling of filenames
```

```
-ignorenot xxx      add a regexp to the ignorenot list
-key xxx            define a keyboard shortcut for this profile
-killserver         kill server when done (even when using sockets)
-label xxx          provide a descriptive string label for this profile
-log                record actions in file specified by logfile preference
-logfile xxx        Log file name
-maxbackups n       number of backed up versions of a file
-maxthreads n       maximum number of simultaneous file transfers
-merge xxx          command for merging conflicting files
-merge2 xxx         command for merging files (when no common version exists)
-numericids         don't map uid/gid values by user/group names
-owner              synchronize owner
-path xxx           path to synchronize
-perms n            part of the permissions which is synchronized
-prefer xxx         choose this replica's version for conflicting changes
-pretendwin         Use creation times for detecting updates
-root xxx           root of a replica
-rootalias xxx      Register alias for canonical root names
-rshargs xxx        other arguments (if any) for remote shell command
-rshcmd xxx         path to the rsh executable
-rsync              activate the rsync transfer mode
-servercmd xxx      name of unison executable on remote server
-silent             print nothing (except error messages)
-socket xxx         act as a server on a socket
-sortbysize         list changed files by size, not name
-sortfirst xxx      add a regexp to the sortfirst list
-sortlast xxx       add a regexp to the sortlast list
-sortnewfirst       list new before changed files
-sshcmd xxx         path to the ssh executable
-statusdepth n      status display depth for local files
-terse              suppress status messages
-testserver         exit immediately after the connection to the server
-times              synchronize modification times
-ui xxx             select user interface ('text' or 'graphic')
-version            print version and exit
-xferbycopying      optimize transfers using local copies, if possible
```

Here, in more detail, are what they do. Many are discussed in even 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 make back up for each path that matches *`pathspec`*. More precisely, for each path that matches this *`pathspec`*, Unison will keep several old versions of a file as a backup whenever a change is propagated. These backup files are left in the directory specified by the environment variable `UNISONBACKUPDIR`, if it is set; otherwise in the directory named by the `backupdir` preference, if it is non-null; otherwise in `.unison/backup/` by default. The newest backed up copy willhave the same name as the original; older versions will be named with extensions `.n.unibck`. The number of versions that are kept is determined by the `maxbackups` preference.

The syntax of *`pathspec`* is described in Section 5.11 [Path Specification].

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. It is checked *after* the `UNISONBACKUPDIR` environment variable.

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 t ignorenot.

backups When this flag is `true`, Unison will keep the old version of a file as a backup whenever a change is propagated. These backup files are left in the same directory, with extension `.bak`. This flag is probably less useful for most users than the t backup flag.

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.

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

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`'. The diff program should expect two file names as arguments

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.

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.

editor `xxx` This preference is used when unison wants to display the output of the merge program when its return value is not 0. User changes the file as he wants and then save it, unison will take this version for the synchronisation. By default the value is 'emacs'.

fastcheck `xxx` When this preference is set to `true`, Unison will use file creation times as 'pseudo inode numbers' 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 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 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. The default value of the preference is `auto`, which causes Unison to use fast checking on Unix replicas (where it is safe) and slow checking on Windows replicas. For backward compatibility, `yes`, `no`, and `default` can be used in place of `true`, `false`, and `auto`. See Section 5.17 [Fast Checking] for more information.

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 5.13 [Symbolic Links] for more details. The syntax of ***`pathspec>`*** is described in Section 5.11 [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 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 synchronizeddepends on the preference `numerids`.

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 5.11 [Path Specification], and further details on ignoring paths is found in Section 5.12 [Ignoring Paths].

ignorecase 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. This flag is set automatically when either host is running Windows or OSX. In rare circumstances it is also useful to set it manually (e.g. when running Unison on a Unix system with a FAT [Windows] volume mounted).

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 t 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 t ignore and does not match t 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 t ignorenot to select particular paths to be synchronized will not work. Instead, you should use the `path` preference to choose particular paths to synchronize.

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.)

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.

maxthreads `n` This preference controls how much concurrency is allowed during the transport phase. Normally, it should be set reasonably high (default is 20) 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.

merge `xxx` This preference can be used to run a merge program which will create a new version of the file with the last backup and the both replicas. This new version will be used for the synchronization. See Section 5.8 [Merging Conflicting Versions] for further detail.

merge2 `xxx` This preference can be used to run a merge program which will create a new version of the file with the last backup and the both replicas. This new version will be used for the synchronization. See Section 5.8 [Merging Conflicting Versions] for further detail.

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 extttnumerids.

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 synch 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 0*o*1777: 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.

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

pretendwin When set to true, this preference makes Unison use Windows-style fast update detection (using file creation times as "pseudo-inode-numbers"), even when running on a Unix system. This switch should be used with care, as it is less safe than the standard update detection method, but it can be useful for synchronizing VFAT filesystems (which do not support inode numbers) mounted on Unix systems. The `fastcheck` option should also be set to true.

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 4.1 [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 `ssh` or `rsh` command used to invoke the remote server. (This option is used for passing arguments to both `rsh` or `ssh`—that's why its name is `rshargs` rather than `sshargs`.)

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.

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.

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.

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 5.11 [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.

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.

statusdepth `n` This preference suppresses the display of status messages during update detection on the local machine for paths deeper than the specified cutoff. (Displaying too many local status messages can slow down update detection somewhat.)

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`.

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.)

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`.

5.5 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 kinds of lines 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.

5.6 Sample Profiles

5.6.1 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
```

5.6.2 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 the entire replica
backup = Name *

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

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

5.6.3 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:

```
# (... 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

# 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
```

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 papers`', where the preference file `papers.prf` contains

```
path = current/papers
path = older/papers
include common
```

causes Unison to synchronize just the subdirectories `current/papers` and `older/papers`.

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.

5.7 Keeping Backups

Unison can maintain 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`. 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.

It is safe 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.

The preference `backup` controls which files are actually backed up: for example, giving the preference '`backup = Name *`' causes backing up of all files. The preference `backupversions` controls how many previous versions of each file are kept. The default is value 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.

5.8 Merging Conflicting Versions

Both user interfaces offer a 'merge' command that can be used to interactively merge conflicting versions of a file. It is invoked by selecting a conflicting file and pressing '`m`'.

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 substrings 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. Substitution of `OLD` 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 settings 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 command 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.)*

5.9 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.

5.10 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.

5.11 Path specification

Several Unison preferences (e.g., `ignore/ignorenot`, `follow`, `sortfirst/sortlast`, `backup`) 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, two other styles of pattern are also recognized:

  ```
  Name name
  ```

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

  ```
  Path path
  ```

matches exactly the path `path`. 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 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`.

- 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 5.12 [Ignoring Paths].

5.12 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 5.11 [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
```

This 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'`.

5.13 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 5.11 [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.

5.14 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.

5.15 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 5.12 [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.

5.16 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 "`-rshargs -C`" to the command line or "`rshargs = -C`" to your profile.

5.17 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 `yes`, 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 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 `no`, if you are worried that Unison may have overlooked an update.

5.18 Click-starting Unison

On Windows NT/2k 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 5.5 [Profile]). Use your favorite editor for this.

6 Frequently Asked Questions

6.1 General Questions

- *Does Unison work on Mac OSX?*

  Unison has not been extensively tested on OSX, though several people say they are using it successfully. There are several known caveats:

  - It does not understand resource forks and will not synchronize them properly. Resource forks are not used by standard unix applications, but *are* used by many native mac applications.
  - OSX native filesystems are case-insensitive (i.e., 'a' and 'A' are the same file), but Unison doesn't recognize this. A workaround is to add the line

    ```
    ignorecase = true
    ```

    to your profile.
  - Unison will be confused by some files that are frequently updated by OSX, and will report lots of errors of the form "XXX has been modified during synchronization." These files — in particular, files with names like `.FBCLockFolder` and `.FBCIndex` — should be ignored by adding

    ```
    ignore = Name .FBCIndex
    ignore = Name .FBCLockFolder
    ```

    to your profile.
  - We do not distribute pre-built binaries for OSX. The textual user interface should compile "straight out of the box" according to the standard Unix installation instructions (once OCaml is installed, either from Fink or by compiling it from sources, which is straightforward).

    Compiling the graphical user interface is also fairly straightforward, assuming you have already installed Fink (a very large collection of Unix tools with a nice package system). Proceed as follows:

    * Use Fink to install an X server (e.g., XDarwin)
    * Use Fink to install the `gtk` package (you may already have installed it without realizing it, since it is required by a number of other common packages).
    * Grab the `lablgtk` distribution from

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

      and install it according to the enclosed instructions. (Make sure to build the native-code versions of the libraries by doing 'make opt' before 'make install.')
    * Go to the main unison directory and do 'make'. (The `UISTYLE=gtk` option should be selected automatically.)

  Unison does not run on Mac OS 9 or earlier.

- *What are the differences between Unison and rsync?*

  Rsync is a mirroring tool; Unison is a synchronizer. That is, rsync needs to be told "this replica contains the true versions of all the files; please make the other replica look exactly the same." Unison is capable of recognizing updates in both replicas and deciding which way they should be propagated.

  Both Unison and rsync use the so-called "rsync algorithm," by Andrew Tridgell and Paul Mackerras, for performing updates. This algorithm streamlines updates in small parts of large files by transferring only the parts that have changed.

- *What are the differences between Unison and CVS?*

  Both CVS and Unison can be used to keep a remote replica of a directory structure up to date with a central repository. Both are capable of propagating updates in both directions and recognizing conflicting updates. Both use the rsync protocol for file transfer.

Unison's main advantage is being somewhat more automatic and easier to use, especially on large groups of files. CVS requires manual notification whenever files are added or deleted. Moving files is a bit tricky. And if you decide to move a directory... well, heaven help you.

CVS, on the other hand, is a full-blown version control system, and it has *lots* of other features (version history, multiple branches, etc.) that Unison (which is just a file synchronizer) doesn't have.

- *Has anybody tried to use Unison in conjunction with CVS in order to replicate a CVS repository for active development in more than one geographical location at the same time? Do you forsee any issues with trying to do such a thing, or do you have any tips as to how to get a setup like that working?*

  Unison and CVS can be used together. The easiest way is to replicate your files with Unison but keep your CVS repository on just one machine (and do a commit on that machine after each time you synchronize with Unison, if files in that directory have changed).

  More complex schemes are also possible (e.g., using a remote CVS server and checking in from any host with one of the replicas), but should be used with care. In particular, if you use a remote CVS server, it is important that you do *not* tell Unison to `ignore` the files in the `CVS` subdirectory.

- *Is it OK to mount my remote filesystem using NFS and run unison locally, or should I run a remote server process?*

  NFS-mounting the replicas is fine, as long as the local network is fast enough. Unison needs to read a lot of files (in particular, it needs to check the last-modified time of every file in the repository every time it runs), so if the link bandwidth is low then running a remote server is much better.

- *When I run Unison on Windows, it creates two different windows, the main user interface and a blank console window. Is there any way to get rid of the second one?*

  The extra console window is there for ssh to use to get your password. Unfortunately, in the present version of unison the window will appear whether you're using ssh or not.

  Karl Moerder contributed some scripts that he uses to make the command window a bit more attractive. He starts unison from a shortcut to a `.cmd` file. This lets him control the attributes of the command window, making it small and gray and centering the passphrase request. His scripts can be found at `http://www.cis.upenn.edu/~bcpierce/unison/download/resources/karls-winhax.zip`.

  It is also possible to get rid of the window entirely (for users that only want socket mode connections) by playing games with icons. If you make a symbolic link to the executable, you can edit the properties box to make this window come up iconic. That way when you click on the link, you seem to just get a unison window (except on the task bar, where the text window shows).

- *Will unison behave correctly if used transitively? That is, if I synchronize both between* `host1:dir` *and* `host2:dir` *and between* `host2:dir` *and* `host3:dir` *at different times? Are there any problems if the "connectivity graph" has loops?*

  This mode of usage will work fine. As far as each "host pair" is concerned, filesystem updates made by Unison when synchronizing any other pairs of hosts are exactly the same as ordinary user changes to the filesystem. So if a file started out having been modified on just one machine, then every time Unison is run on a pair of hosts where one has heard about the change and the other hasn't will result in the change being propagated to the other host. Running unison between machines where both have already heard about the change will leave that file alone. So, no matter what the connectivity graph looks like (as long as it is not partitioned), eventually everyone will agree on the new value of the file.

  The only thing to be careful of is changing the file *again* on the first machine (or, in fact, any other machine) before all the machines have heard about the first change – this can result in Unison reporting conflicting changes to the file, which you'll then have to resolve by hand.

- *What will happen if I try to synchronize a special file (e.g., something in* `/dev`*,* `/proc`*, etc.)?*

  Unison will refuse to synchronize such files. It only understands ordinary files, directories, and symlinks.

- *Is it OK to run several copies of Unison concurrently?*

  Unison is built to handle this case, but this functionality has not been extensively tested. Keep your eyes open.

- *What will happen if I do a local (or NFS, etc.) sync and some file happens to be part of both replicas?*

  It will look to Unison as though somebody else has been modifying the files it is trying to synchronize, and it will fail (safely) on these files.

- *What happens if Unison gets killed while it is working? Do I have to kill it nicely, or can I use* `kill -9`*? What if the network goes down during a synchronization? What if one machine crashes but the other keeps running?*

  Don't worry; be happy. See Section 4.6 [Invariants].

- *What about race conditions when both Unison and some other program or user are both trying to write to a file at exactly the same moment?*

  Unison works hard to make these "windows of danger" as short as possible, but they cannot be eliminated completely.

- *I've heard that the Unix file locking mechanism doesn't work very well under NFS. Is this a problem for Unison?*

  No.

- *On Windows systems, it looks like the* `root` *preferences are specified using backslashes, but* `path` *and* `ignore` *preferences are specified with forward slashes. What's up with that?*

  Unison uses two sorts of paths: native filesystem paths, which use the syntax of the host filesystem, and "portable" paths relative to the roots of the replicas, which always use / to separate the path components. Roots are native filesystem paths; the others are root-relative.

- *What will happen if I run Unison after my archive files get deleted or damaged?*

  A missing or damaged archive is treated the same as a completely empty one. This means that Unison will consider *all* the files in both replicas to be new. Any files that exist only in one replica will be transferred to the other replica (because it will look as though they have just been created); files that exist on both replicas but have different contents will be flagged as conflicts; files that have the same contents on both replicas will simply be noted in the rebuilt archive.

  If just one of the archive files is missing or damaged, Unison will ignore the other one and start from an empty archive.

6.2 Common Problems

A general recommendation is that, if you've gotten into a state you don't understand, deleting the archive files on both replicas (files with names like `arNNNNNNNNNNNNNNN` in the `.unison` directory) will return you to a blank slate. If the replicas are identical, then deleting the archives is always safe. If they are not identical, then deleting the archives will cause all files that exist on one side but not the other to be copied, and will report conflicts for all non-identical files that do exist on both sides.

(If you think the behavior you're observing is an actual bug, then you might consider *moving* the archives to somewhere else instead of deleting them, so that you can try to replicate the bad behavior and report more clearly what happened.)

- *The text mode user interface fails with "*`Uncaught exception Sys_blocked_io`*" when running over ssh2.*

  The problem here is that ssh2 puts its standard file descriptors into non-blocking mode. But unison and ssh share the same stderr (so that error messages from the server are displayed), and the nonblocking setting interferes with Unison's interaction with the user. This can be corrected by redirecting the stderr when invoking Unison:

```
unison -ui text <other args> 2>/dev/tty
```

(The redirection syntax is a bit shell-specific. On some shells, e.g., csh and tcsh, you may need to write

```
unison -ui text <other args>  > & /dev/tty
```

instead.)

- *What does the following mean?*

  ```
  Propagating updates [accounting/fedscwh3qt2000.wb3]
  failed: error in renaming locally:
  /DANGER.README: permission denied
  ```

  It means that unison is having trouble creating the temporary file DANGER.README, which it uses as a "commit log" for operations (such as renaming its temporary file `accounting/fedscwh3qt2000.wb3.unison.tmp` to the real location `accounting/fedscwh3qt2000.wb3`) that may leave the filesystem in a bad state if they are interrupted in the middle. This is pretty unlikely, since the rename operation happens fast, but it is possible; if it happens, the commit log will be left around and Unison will notice (and tell you) the next time it runs that the consistency of that file needs to be checked.

  The specific problem here is that Unison is trying to create DANGER.README in the directory specified by your HOME environment variable, which seems to be set to /, where you do not have write permission.

- *The command line*

  ```
  unison work ssh://remote.dcs.ed.ac.uk/work
  ```

  fails, with "`fatal error:  could not connect to server.`*" But when I connect directly with* `ssh remote.dcs.ed.ac.uk/work`*, I see that my* `PATH` *variable is correctly set, and the unison executable is found.*

  In the first case, Unison is using `ssh` to execute a command, and in the second, it is giving you an interactive remote shell. Under some ssh configurations, these two use different startup sequences. You can test whether this is the problem here by trying, e.g.,

  ```
  ssh remote.dcs.ed.ac.uk 'echo $PATH'
  ```

  and seeing whether your `PATH` is the same as when you do

  ```
  ssh remote.dcs.ed.ac.uk
  [give password and wait for connection]
  echo $PATH
  ```

  One method that should always work is this [thanks to Richard Atterer for this]:

  1. log into the machine, set up PATH so the program is found
  2. execute

     ```
     echo "PATH=$PATH" >>~/.ssh/environment
     ```

  All this seems to be controlled by the configuration of ssh, but we have not understood the details—if someone does, please let us know.

- *I'm having trouble getting unison working with openssh under Windows. Any suggestions?*

  Antony Courtney contributed the following comment.

I ran in to some difficulties trying to use this ssh client with Unison, and tracked down at least one of the problems. I thought I'd share my experiences, and provide a 'known good' solution for other users who might want to use this Windows / Unison / ssh / Cygwin combination.

If you launch Unison from bash, it fails (at least for me). Running `unison_win32-gtkui.exe`, I get a dialog box that reads:

```
Fatal error: Error in checkServer: Broken pipe [read()]
```

and a message is printed to stderr in the bash window that reads:

```
ssh: unison_win32-gtkui.exe: no address associated with hostname.
```

My guess is that this is caused by some incompatibility between the Ocaml Win32 library routines and Cygwin with regard to setting up argv[] for child processes.

The solution is to launch Unison from a DOS command prompt instead; or see section 5.18.

- *When I use ssh to log into the server, everything looks fine (and I can see the Unison binary in my path). But when I do '*`ssh <server> unison`*' it fails. Why?*

  [Thanks to Nick Phillips for the following explanation.]

  It's simple. If you start ssh, enter your password etc. and then end up in a shell, you have a login shell.

  If you do "ssh myhost.com unison" then unison is not run in a login shell.

  This means that different shell init scripts are used, and most people seem to have their shell init scripts set up all wrong.

  With bash, for example, your `.bash_profile` *only* gets used if you start a login shell. This usually means that you've logged in on the system console, on a terminal, or remotely. If you start an xterm from the command line you won't get a login shell in it. If you start a command remotely from the ssh or rsh command line you also won't get a login shell to run it in (this is of course a Good Thing – you may want to run interactive commands from it, for example to ask what type of terminal they're using today).

  If people insist on setting their `PATH` in their `.bash_profile`, then they should probably do at least one of the following:

  1. stop it;
  2. read the bash manual, section "INVOCATION";
  3. set their path in their `.bashrc`;
  4. get their sysadmin to set a sensible system-wide default path;
  5. source their `.bash_profile` from their `.bashrc` ...

  It's pretty similar for most shells.

- *Unison crashes with an "out of memory" error when used to synchronize really huge directories (e.g., with hundreds of thousands of files).*

  You may need to increase your maximum stack size. On Linux and Solaris systems, for example, you can do this using the `ulimit` command (see the `bash` documentation for details).

- *Unison seems to be unable to copy a single really huge file. I get something like this:*

  ```
  Error in querying file information:
  Value too large for defined data type [lstat(...)]
  ```

This is a limitation in the OCaml interface to the Unix system calls. (The problem is that the OCaml library uses 32-bit integers to represent file positions. The maximal positive 'int' in OCaml is about 2.1E9. We hope that the OCaml team will someday provide an alternative interface that uses 64-bit integers.

- *Why does unison run so slowly the first time I start it?*

  On the first synchronization, unison doesn't have any "memory" of what your replicas used to look like, so it has to go through, fingerprint every file, transfer the fingerprints across the network, and compare them to what's on the other side. Having done this once, it stashes away the information so that in future runs almost all of the work can be done locally on each side.

- *I can't seem to override the paths selected in the profile by using a* `-path` *argument on the command line.*

  Right: the `path` preference is additive (each use adds an entry to the list of paths within the replicas that Unison will try to synchronize), and there is no way to remove entries once they have gotten into this list. The solution is to split your preference file into different "top-level" files containing different sets of `path` preferences and make them all include a common preference file to avoid repeating the non-path preferences. See Section 5.6 [Profile Examples] for a complete example.

- *I can't seem to override the roots selected in the profile by listing the roots on the command line. I get "Fatal error: Wrong number of roots (2 expected; 4 provided)."*

  Roots should be provided *either* in the preference file *or* on the command line, not both. See Section 5.6 [Profile Examples] for further advice.

- *I am trying to compile unison 2.7.7 using OCaml 3.04. I get "Values do not match" error.*

  Unison 2.7.7 compiles with Ocaml 3.02. Later versions of OCaml, include version 3.04, require by default all parameter labels for function calls if they are declared in the interface. Adding the compilation option "`-nolabels`" (by inserting a line "`CAMLFLAGS+=-nolabels`" to the file named "`Makefile.OCaml`") should solve the problem. To compile the graphical user interface for Unison 2.7.7, use LablGtk 1.1.2 instead of LablGtk 1.1.3.

- *I get a persistent 'rsync failure' error when transferring a particular file. What can I do?*

  We're not sure what causes this failure, but a workaround is to set the `rsync` flag to false.

6.3 Tricks and Tips

- *I want to use Unison to synchronize really big replicas. How can I improve performance?*

  When you synchronize a large directory structure for the first time, Unison will need to spend a lot of time walking over all the files and building an internal data structure called an archive. There is no way around this: Unison uses these archives in a critical way to do its work. While you're getting things set up, you'll probably save time if you start off focusing Unison's attention on just a subset of your files, by including the option `-path` *some/small/subdirectory* on the command line. When this is working to your satisfaction, take away the `-path` option and go get lunch while Unison works. This rebuilding operation will sometimes need to be repeated when you upgrade Unison (major upgrades often involve changes to the format of the archive files; minor upgrades generally do not.)

  Next, you make sure that you are not "remote mounting" either of your replicas over a network connection. Unison needs to run close to the files that it is managing, otherwise performance will be very poor. Set up a client-server configuration as described in the installation section of the manual.

  If your replicas are large and at least one of them is on a Windows system, you will probably 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. In this case, you may be interested in the `fastcheck` preference, documented in Section 5.17 [Fast Update Checking].

In normal operation, the longest part of a Unison run is usually the time that it takes to scan the replicas for updates. This requires examining the filesystem entry for every file (i.e., doing an `fstat` on each inode) in the replica. This means that the total number of inodes in the replica, rather than the total size of the data, is the main factor limiting Unison's performance.

Update detection times can be improved (sometimes dramatically) by telling Unison to ignore certain files or directories. See the description of the `ignore` and `ignorenot` preferences in Section 5.4 [Preferences].

(One could also imagine improving Unison's update detection by giving it access to the filesystem logs kept by some modern "journaling filesystems" such as ext3 or ReiserFS, but this has not been implemented. We have some ideas for how to make it work, but it will require a bit of systems hacking that no one has volunteered for yet.)

Another way of making Unison detect updates faster is by "aiming" it at just a portion of the replicas by giving it one or more `path` preferences. For example, if you want to synchronize several large subdirectories of your home directory between two hosts, you can set things up like this:

– Create a common profile (called, e.g., `common`) containing most of your preferences, including the two roots:

  ```
  root = /home/bcpierce
  root = ssh://saul.cis.upenn.edu//home/bcpierce
  ignore = Name *.o
  ignore = Name *.tmp
  etc.
  ```

– Create a default profile `default.prf` with `path` preferences for all of the top-level subdirectories that you want to keep in sync, plus an instruction to read the `common` profile:

  ```
  path = current
  path = archive
  path = src
  path = Mail
  include common
  ```

  Running `unison default` will synchronize everything.

  (If you want to synchronize *everything* in your home directory, you can omit the `path` preferences from `default.prf`.)

– Create several more preference files similar to `default.prf` but containing smaller sets of `path` preferences. For example, `mail.prf` might contain:

  ```
  path = Mail
  include common
  ```

  Now running `unison mail` will scan and synchronize just your `Mail` subdirectory.

Once update detection is finished, Unison needs to transfer the changed files. This is done using a variant of the `rsync` protocol, so if you have made only small changes in a large file, the amount of data transferred across the network will be relatively small.

Unison carries out many file transfers at the same time, so the per-file set up time is not a significant performance factor.

• *Is it possible to run Unison from* `inetd` *(the Unix internet services daemon)?*

  Toby Johnson has contributed a detailed chroot min-HOWTO describing how to do this. (Yan Seiner wrote an earlier howto, on which Toby's is based.)

- *Is there a way to get Unison not to prompt me for a password every time I run it (e.g., so that I can run it every half hour from a shell script)?*

  It's actually `ssh` that's asking for the password. If you're running the Unison client on a Unix system, you should check out the 'ssh-agent' facility in ssh. If you do

  ```
  ssh-agent bash
  ```

  (or `ssh-agent startx`, when you first log in) it will start you a shell (or an X Windows session) in which all processes and sub-processes are part of the same ssh-authorization group. If, inside any shell belonging to this authorization group, you run the `ssh-add` program, it will prompt you *once* for a password and then remember it for the duration of the bash session. You can then use Unison over `ssh`—or even run it repeatedly from a shell script—without giving your password again.

  It may also be possible to configure `ssh` so that it does not require any password: just enter an empty password when you create a pair of keys. If you think it is safe enough to keep your private key unencrypted on your client machine, this solution should work even under Windows.

- *Is there a way, under Windows, to click-start Unison and make it synchronize according to a particular profile?*

  Greg Sullivan sent us the following useful trick:

  > In order to make syncing a particular profile "clickable" from the Win98 desktop, when the profile uses `ssh`, you need to create a `.bat` file that contains nothing but "`unison profile-name`" (assuming `unison.exe` is in the `PATH`). I first tried the "obvious" strategy of creating a shortcut on the desktop with the actual command line "`unison profile`, but that hangs. The `.bat` file trick works, though, because it runs `command.com` and then invokes the `.bat` file.

- *Can Unison be used with SSH's port forwarding features?*

  Mark Thomas says the following procedure works for him:

  > After having problems with unison spawning a command line ssh in Windows I noticed that unison also supports a socket mode of communication (great software!) so I tried the port forwarding feature of ssh using a graphical SSH terminal TTSSH:
  >
  > `http://www.zip.com.au/∼roca/ttssh.html`
  >
  > To use unison I start TTSHH with port forwarding enabled and login to the Linux box where the unison server (`unison -socket xxxx`) is started automatically. In windows I just run unison and connect to localhost (`unison socket://localhost:xxxx/ ...`)

- *How can I use Unison from a laptop whose hostname changes depending on where it is plugged into the network?*

  This is partially addressed by the `rootalias` preference. See the discussion in Section 5.3 [Archive Files].

- *It's annoying that (on Unix systems) I have to type an ssh passphrase into a console window, rather than being asked for it in a dialog box. Is there a better way?*

  We have some ideas about how this might be done (by allocating a PTY and using it to talk to ssh), but we haven't implemented them yet. If you'd like to have a crack at it, we'd be glad to discuss ideas and incorporate patches.

  In the meantime, tmb has contributed a script that uses `expectk` to do what's needed. It's available at `http://www.cis.upenn.edu/∼bcpierce/unison/download/resources/expectk-startup`.

A Installing Ssh

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

1. Install `ssh`.
   (a) Become root. (If you do not have administrator permissions, ask your system manager to install an ssh client and an ssh server for you and skip this section.)
   (b) Download `ssh-1.2.27.tar.gz` from `ftp://ftp.ssh.com/pub/ssh/`.
   (c) Install it:
      - Unpack the archive (`gunzip ssh-1.2.27.tar.gz` and then `tar xvf ssh-1.2.27.tar.gz`).
      - following instructions in `INSTALL`, enter `./configure`, `make`, and `make install`.
      - to run the ssh daemon:
         – find the server daemon `sshd` (e.g., `/usr/local/sbin/sshd` on RedHat-Linux systems).
         – put its full pathname in the system initialization script to have it run at startup (this script is called `/etc/rc.d/rc.sysinit` on RedHat-Linux, for example).
   (d) Once a server is running on the remote host and a client is available on the local host, you should be able to connect with ssh in the same way as with rsh (e.g., `ssh foobar`, then enter your password).
2. If you like, you can now set up ssh so that you only need to type your password once per X session, rather than every time you run Unison (this is not necessary for using ssh with Unison, but it saves typing).
   (a) Build your keys :
      - enter `ssh-keygen` and type a passphrase as required.
      - your private key is now in `~/.ssh/identity` (this file must remain private) and your public key in `~/.ssh/identity.pub`.
   (b) Allow user-mode secure connection.
      - append contents of the local file `~/.ssh/identity.pub` to the file `~/.ssh/authorized_keys` on the remote system.
      - Test that you can connect by starting `ssh` and giving the passphrase you just chose instead of your remote password.
   (c) Create an agent to manage authentication for you :
      - start `ssh-agent` with the parent program whose children will be granted automatic connections (e.g., `ssh-agent bash` or `ssh-agent startx`).
      - enter `ssh-add` to enter your passphrase and enable automatic login for connections to come.
      - you should now be able to run Unison using SSH without giving any passphrase or password.
      - to kill the agent, enter `ssh-agent -k`, or simply exit the program you launched using `ssh-agent`.

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 this one instead:

   ```
   http://opensores.thebunker.net/pub/mirrors/ssh/contrib/ssh-1.2.14-win32bin.zip
   ```

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, 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 2.9.20

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
    tt USERPROFILE nor the
    tt HOME environment variables are set, then Unison will put its temporary commit log (called
    tt DANGER.README) into the directory named by the
    tt UNISON environment variable, if any; otherwise it will use
    tt 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
    tt ./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`.

- 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 0*o*1777: 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, available from `http://wwwfun.kurims.kyoto-u.ac.jp/soft/olabl/lablgt`

    (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, becuase 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 = Regexp <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.