## NAME

rsync − faster, flexible replacement for rcp

## SYNOPSIS

rsync [OPTION]... SRC [SRC]... [USER@]HOST:DEST

rsync [OPTION]... [USER@]HOST:SRC [DEST]

rsync [OPTION]... SRC [SRC]... DEST

rsync [OPTION]... [USER@]HOST::SRC [DEST]

rsync [OPTION]... SRC [SRC]... [USER@]HOST::DEST

rsync [OPTION]... rsync://[USER@]HOST[:PORT]/SRC [DEST]

rsync [OPTION]... SRC [SRC]... rsync://[USER@]HOST[:PORT]/DEST

## DESCRIPTION

rsync is a program that behaves in much the same way that rcp does, but has many more options and uses the rsync remote-update protocol to greatly speed up file transfers when the destination file is being updated.

The rsync remote-update protocol allows rsync to transfer just the differences between two sets of files across the network connection, using an efficient checksum-search algorithm described in the technical report that accompanies this package.

Some of the additional features of rsync are:

- o support for copying links, devices, owners, groups, and permissions
- o exclude and exclude-from options similar to GNU tar
- o a CVS exclude mode for ignoring the same files that CVS would ignore
- o can use any transparent remote shell, including ssh or rsh
- o does not require root privileges
- o pipelining of file transfers to minimize latency costs
- o support for anonymous or authenticated rsync daemons (ideal for mirroring)

## GENERAL

Rsync copies files either to or from a remote host, or locally on the current host (it does not support copying files between two remote hosts).

There are two different ways for rsync to contact a remote system: using a remote-shell program as the transport (such as ssh or rsh) or contacting an rsync daemon directly via TCP. The remote-shell transport is used whenever the source or destination path contains a single colon (:) separator after a host specification. Contacting an rsync daemon directly happens when the source or destination path contains a double colon (::) separator after a host specification, OR when an rsync:// URL is specified (see also the "USING RSYNC-DAEMON FEATURES VIA A REMOTE-SHELL CONNECTION" section for an exception to this latter rule).

As a special case, if a remote source is specified without a destination, the remote files are listed in an output format similar to "ls -l".

As expected, if neither the source or destination path specify a remote host, the copy occurs locally (see also the **--list-only** option).

## SETUP

See the file README for installation instructions.

Once installed, you can use rsync to any machine that you can access via a remote shell (as well as some that you can access using the rsync daemon-mode protocol). For remote transfers, a modern rsync uses ssh for its communications, but it may have been configured to use a different remote shell by default, such as

rsh or remsh.

You can also specify any remote shell you like, either by using the **-e** command line option, or by setting the RSYNC_RSH environment variable.

One common substitute is to use ssh, which offers a high degree of security.

Note that rsync must be installed on both the source and destination machines.

## USAGE

You use rsync in the same way you use rcp. You must specify a source and a destination, one of which may be remote.

Perhaps the best way to explain the syntax is with some examples:

```
rsync -t *.c foo:src/
```

This would transfer all files matching the pattern *.c from the current directory to the directory src on the machine foo. If any of the files already exist on the remote system then the rsync remote-update protocol is used to update the file by sending only the differences. See the tech report for details.

```
rsync -avz foo:src/bar /data/tmp
```

This would recursively transfer all files from the directory src/bar on the machine foo into the /data/tmp/bar directory on the local machine. The files are transferred in "archive" mode, which ensures that symbolic links, devices, attributes, permissions, ownerships, etc. are preserved in the transfer. Additionally, compression will be used to reduce the size of data portions of the transfer.

```
rsync -avz foo:src/bar/ /data/tmp
```

A trailing slash on the source changes this behavior to avoid creating an additional directory level at the destination. You can think of a trailing / on a source as meaning "copy the contents of this directory" as opposed to "copy the directory by name", but in both cases the attributes of the containing directory are transferred to the containing directory on the destination. In other words, each of the following commands copies the files in the same way, including their setting of the attributes of /dest/foo:

```
rsync -av /src/foo /dest
rsync -av /src/foo/ /dest/foo
```

Note also that host and module references don't require a trailing slash to copy the contents of the default directory. For example, both of these copy the remote directory's contents into "/dest":

```
rsync -av host: /dest
rsync -av host::module /dest
```

You can also use rsync in local-only mode, where both the source and destination don't have a ':' in the name. In this case it behaves like an improved copy command.

```
rsync somehost.mydomain.com::
```

This would list all the anonymous rsync modules available on the host somehost.mydomain.com. (See the following section for more details.)

## ADVANCED USAGE

The syntax for requesting multiple files from a remote host involves using quoted spaces in the SRC. Some examples:

```
rsync host::'modname/dir1/file1 modname/dir2/file2' /dest
```

This would copy file1 and file2 into /dest from an rsync daemon. Each additional arg must include the same "modname/" prefix as the first one, and must be preceded by a single space. All other spaces are assumed to be a part of the filenames.

```
rsync -av host:'dir1/file1 dir2/file2' /dest
```

This would copy file1 and file2 into /dest using a remote shell. This word-splitting is done by the remote shell, so if it doesn't work it means that the remote shell isn't configured to split its args based on

whitespace (a very rare setting, but not unknown). If you need to transfer a filename that contains whitespace, you'll need to either escape the whitespace in a way that the remote shell will understand, or use wildcards in place of the spaces. Two examples of this are:

```
rsync -av host:'file\ name\ with\ spaces' /dest
rsync -av host:file?name?with?spaces /dest
```

This latter example assumes that your shell passes through unmatched wildcards. If it complains about "no match", put the name in quotes.

## CONNECTING TO AN RSYNC DAEMON

It is also possible to use rsync without a remote shell as the transport. In this case you will directly connect to a remote rsync daemon, typically using TCP port 873. (This obviously requires the daemon to be running on the remote system, so refer to the STARTING AN RSYNC DAEMON TO ACCEPT CONNECTIONS section below for information on that.)

Using rsync in this way is the same as using it with a remote shell except that:

- you either use a double colon :: instead of a single colon to separate the hostname from the path, or you use an rsync:// URL.
- the first word after the :: is a module name.
- the remote daemon may print a message of the day when you connect.
- if you specify no path name on the remote daemon then the list of accessible paths on the daemon will be shown.
- if you specify no local destination then a listing of the specified files on the remote daemon is provided.

An example that copies all the files in a remote module named "src":

```
rsync -av host::src /dest
```

Some modules on the remote daemon may require authentication. If so, you will receive a password prompt when you connect. You can avoid the password prompt by setting the environment variable RSYNC_PASSWORD to the password you want to use or using the **--password-file** option. This may be useful when scripting rsync.

WARNING: On some systems environment variables are visible to all users. On those systems using **--password-file** is recommended.

You may establish the connection via a web proxy by setting the environment variable RSYNC_PROXY to a hostname:port pair pointing to your web proxy. Note that your web proxy's configuration must support proxy connections to port 873.

## USING RSYNC-DAEMON FEATURES VIA A REMOTE-SHELL CONNECTION

It is sometimes useful to use various features of an rsync daemon (such as named modules) without actually allowing any new socket connections into a system (other than what is already required to allow remote-shell access). Rsync supports connecting to a host using a remote shell and then spawning a single-use "daemon" server that expects to read its config file in the home dir of the remote user. This can be useful if you want to encrypt a daemon-style transfer's data, but since the daemon is started up fresh by the remote user, you may not be able to use features such as chroot or change the uid used by the daemon. (For another way to encrypt a daemon transfer, consider using ssh to tunnel a local port to a remote machine and configure a normal rsync daemon on that remote host to only allow connections from "localhost".)

From the user's perspective, a daemon transfer via a remote-shell connection uses nearly the same command-line syntax as a normal rsync-daemon transfer, with the only exception being that you must explicitly

set the remote shell program on the command-line with the **--rsh=COMMAND** option. (Setting the RSYNC_RSH in the environment will not turn on this functionality.) For example:

```
rsync -av --rsh=ssh host::module /dest
```

If you need to specify a different remote-shell user, keep in mind that the user@ prefix in front of the host is specifying the rsync-user value (for a module that requires user-based authentication). This means that you must give the '-l user' option to ssh when specifying the remote-shell:

```
rsync -av -e "ssh -l ssh-user" rsync-user@host::module /dest
```

The "ssh-user" will be used at the ssh level; the "rsync-user" will be used to log-in to the "module".

## STARTING AN RSYNC DAEMON TO ACCEPT CONNECTIONS

In order to connect to an rsync daemon, the remote system needs to have a daemon already running (or it needs to have configured something like inetd to spawn an rsync daemon for incoming connections on a particular port). For full information on how to start a daemon that will handling incoming socket connections, see the rsyncd.conf(5) man page -- that is the config file for the daemon, and it contains the full details for how to run the daemon (including stand-alone and inetd configurations).

If you're using one of the remote-shell transports for the transfer, there is no need to manually start an rsync daemon.

## EXAMPLES

Here are some examples of how I use rsync.

To backup my wife's home directory, which consists of large MS Word files and mail folders, I use a cron job that runs

```
rsync -Cavz . arvidsjaur:backup
```

each night over a PPP connection to a duplicate directory on my machine "arvidsjaur".

To synchronize my samba source trees I use the following Makefile targets:

```
get:
    rsync -avuzb --exclude '*~' samba:samba/ .
put:
    rsync -Cavuzb . samba:samba/
sync: get put
```

this allows me to sync with a CVS directory at the other end of the connection. I then do CVS operations on the remote machine, which saves a lot of time as the remote CVS protocol isn't very efficient.

I mirror a directory between my "old" and "new" ftp sites with the command:

```
rsync -az -e ssh --delete ~ftp/pub/samba nimbus:"~ftp/pub/tridge"
```

This is launched from cron every few hours.

## OPTIONS SUMMARY

Here is a short summary of the options available in rsync. Please refer to the detailed description below for a complete description.

```
-v, --verbose               increase verbosity
-q, --quiet                 suppress non-error messages
-c, --checksum              skip based on checksum, not mod-time & size
-a, --archive               archive mode; same as -rlptgoD (no -H)
-r, --recursive             recurse into directories
-R, --relative              use relative path names
    --no-relative           turn off --relative
    --no-implied-dirs       don't send implied dirs with -R
-b, --backup                make backups (see --suffix & --backup-dir)
    --backup-dir=DIR        make backups into hierarchy based in DIR
    --suffix=SUFFIX         backup suffix (default ~ w/o --backup-dir)
-u, --update                skip files that are newer on the receiver
    --inplace               update destination files in-place
-d, --dirs                  transfer directories without recursing
-l, --links                 copy symlinks as symlinks
-L, --copy-links            transform symlink into referent file/dir
    --copy-unsafe-links     only "unsafe" symlinks are transformed
    --safe-links            ignore symlinks that point outside the tree
-H, --hard-links            preserve hard links
-K, --keep-dirlinks         treat symlinked dir on receiver as dir
-p, --perms                 preserve permissions
-o, --owner                 preserve owner (root only)
-g, --group                 preserve group
-D, --devices               preserve devices (root only)
-t, --times                 preserve times
-O, --omit-dir-times        omit directories when preserving times
-S, --sparse                handle sparse files efficiently
-n, --dry-run               show what would have been transferred
-W, --whole-file            copy files whole (without rsync algorithm)
    --no-whole-file         always use incremental rsync algorithm
-x, --one-file-system       don't cross filesystem boundaries
-B, --block-size=SIZE       force a fixed checksum block-size
-e, --rsh=COMMAND           specify the remote shell to use
    --rsync-path=PROGRAM    specify the rsync to run on remote machine
    --existing              only update files that already exist
    --ignore-existing       ignore files that already exist on receiver
    --remove-sent-files     sent files/symlinks are removed from sender
    --del                   an alias for --delete-during
    --delete                delete files that don't exist on sender
    --delete-before         receiver deletes before transfer (default)
    --delete-during         receiver deletes during xfer, not before
    --delete-after          receiver deletes after transfer, not before
    --delete-excluded       also delete excluded files on receiver
    --ignore-errors         delete even if there are I/O errors
    --force                 force deletion of dirs even if not empty
    --max-delete=NUM        don't delete more than NUM files
    --max-size=SIZE         don't transfer any file larger than SIZE
    --partial               keep partially transferred files
    --partial-dir=DIR       put a partially transferred file into DIR
```

--delay-updates put all updated files into place at end
--numeric-ids don't map uid/gid values by user/group name
--timeout=TIME set I/O timeout in seconds
-I, --ignore-times don't skip files that match size and time
--size-only skip files that match in size
--modify-window=NUM compare mod-times with reduced accuracy
-T, --temp-dir=DIR create temporary files in directory DIR
-y, --fuzzy find similar file for basis if no dest file
--compare-dest=DIR also compare received files relative to DIR
--copy-dest=DIR ... and include copies of unchanged files
--link-dest=DIR hardlink to files in DIR when unchanged
-z, --compress compress file data during the transfer
-C, --cvs-exclude auto-ignore files in the same way CVS does
-f, --filter=RULE add a file-filtering RULE
-F same as --filter='dir-merge /.rsync-filter'
repeated: --filter='- .rsync-filter'
--exclude=PATTERN exclude files matching PATTERN
--exclude-from=FILE read exclude patterns from FILE
--include=PATTERN don't exclude files matching PATTERN
--include-from=FILE read include patterns from FILE
--files-from=FILE read list of source-file names from FILE
-0, --from0 all *from/filter files are delimited by 0s
--address=ADDRESS bind address for outgoing socket to daemon
--port=PORT specify double-colon alternate port number
--blocking-io use blocking I/O for the remote shell
--no-blocking-io turn off blocking I/O when it is default
--stats give some file-transfer stats
--progress show progress during transfer
-P same as --partial --progress
-i, --itemize-changes output a change-summary for all updates
--log-format=FORMAT output filenames using the specified format
--password-file=FILE read password from FILE
--list-only list the files instead of copying them
--bwlimit=KBPS limit I/O bandwidth; KBytes per second
--write-batch=FILE write a batched update to FILE
--only-write-batch=FILE like --write-batch but w/o updating dest
--read-batch=FILE read a batched update from FILE
--protocol=NUM force an older protocol version to be used
--checksum-seed=NUM set block/file checksum seed (advanced)
-4, --ipv4 prefer IPv4
-6, --ipv6 prefer IPv6
--version print version number
-h, --help show this help screen

Rsync can also be run as a daemon, in which case the following options are accepted:

--daemon run as an rsync daemon
--address=ADDRESS bind to the specified address
--bwlimit=KBPS limit I/O bandwidth; KBytes per second
--config=FILE specify alternate rsyncd.conf file
--no-detach do not detach from the parent
--port=PORT listen on alternate port number

-v, --verbose increase verbosity
-4, --ipv4 prefer IPv4
-6, --ipv6 prefer IPv6
-h, --help show this help screen

## OPTIONS

rsync uses the GNU long options package. Many of the command line options have two variants, one short and one long. These are shown below, separated by commas. Some options only have a long variant. The '=' for options that take a parameter is optional; whitespace can be used instead.

**-h, --help**

Print a short help page describing the options available in rsync.

**--version**

print the rsync version number and exit.

**-v, --verbose**

This option increases the amount of information you are given during the transfer. By default, rsync works silently. A single **-v** will give you information about what files are being transferred and a brief summary at the end. Two **-v** flags will give you information on what files are being skipped and slightly more information at the end. More than two **-v** flags should only be used if you are debugging rsync.

Note that the names of the transferred files that are output are done using a default **--log-format** of "%n%L", which tells you just the name of the file and, if the item is a link, where it points. At the single **-v** level of verbosity, this does not mention when a file gets its attributes changed. If you ask for an itemized list of changed attributes (either **--itemize-changes** or adding "%i" to the **--log-format** setting), the output (on the client) increases to mention all items that are changed in any way. See the **--log-format** option for more details.

**-q, --quiet**

This option decreases the amount of information you are given during the transfer, notably suppressing information messages from the remote server. This flag is useful when invoking rsync from cron.

**-I, --ignore-times**

Normally rsync will skip any files that are already the same size and have the same modification time-stamp. This option turns off this "quick check" behavior.

**--size-only**

Normally rsync will not transfer any files that are already the same size and have the same modification time-stamp. With the **--size-only** option, files will not be transferred if they have the same size, regardless of timestamp. This is useful when starting to use rsync after using another mirroring system which may not preserve timestamps exactly.

**--modify-window**

When comparing two timestamps, rsync treats the timestamps as being equal if they differ by no more than the modify-window value. This is normally 0 (for an exact match), but you may find it useful to set this to a larger value in some situations. In particular, when transferring to or from an MS Windows FAT filesystem (which represents times with a 2-second resolution), **--modify-window=1** is useful (allowing times to differ by up to 1 second).

**-c, --checksum**

This forces the sender to checksum all files using a 128-bit MD4 checksum before transfer. The checksum is then explicitly checked on the receiver and any files of the same name which already

exist and have the same checksum and size on the receiver are not transferred. This option can be quite slow.

**-a, --archive**

This is equivalent to **-rlptgoD**. It is a quick way of saying you want recursion and want to preserve almost everything. The only exception to this is if **--files-from** was specified, in which case **-r** is not implied.

Note that **-a does not preserve hardlinks**, because finding multiply-linked files is expensive. You must separately specify **-H**.

**-r, --recursive**

This tells rsync to copy directories recursively. See also **--dirs** (**-d**).

**-R, --relative**

Use relative paths. This means that the full path names specified on the command line are sent to the server rather than just the last parts of the filenames. This is particularly useful when you want to send several different directories at the same time. For example, if you used the command

```
rsync /foo/bar/foo.c remote:/tmp/
```

then this would create a file called foo.c in /tmp/ on the remote machine. If instead you used

```
rsync -R /foo/bar/foo.c remote:/tmp/
```

then a file called /tmp/foo/bar/foo.c would be created on the remote machine -- the full path name is preserved. To limit the amount of path information that is sent, do something like this:

```
cd /foo
rsync -R bar/foo.c remote:/tmp/
```

That would create /tmp/bar/foo.c on the remote machine.

**--no-relative**

Turn off the **--relative** option. This is only needed if you want to use **--files-from** without its implied **--relative** file processing.

**--no-implied-dirs**

When combined with the **--relative** option, the implied directories in each path are not explicitly duplicated as part of the transfer. This makes the transfer more optimal and also allows the two sides to have non-matching symlinks in the implied part of the path. For instance, if you transfer the file "/path/foo/file" with **-R**, the default is for rsync to ensure that "/path" and "/path/foo" on the destination exactly match the directories/symlinks of the source. Using the **--no-implied-dirs** option would omit both of these implied dirs, which means that if "/path" was a real directory on one machine and a symlink of the other machine, rsync would not try to change this.

**-b, --backup**

With this option, preexisting destination files are renamed as each file is transferred or deleted. You can control where the backup file goes and what (if any) suffix gets appended using the **--backup-dir** and **--suffix** options. Note that if you don't specify **--backup-dir**, the **--omit-dir-times** option will be enabled.

**--backup-dir=DIR**

In combination with the **--backup** option, this tells rsync to store all backups in the specified directory. This is very useful for incremental backups. You can additionally specify a backup suffix using the **--suffix** option (otherwise the files backed up in the specified directory will keep their original filenames).

**--suffix=SUFFIX**

This option allows you to override the default backup suffix used with the **--backup** (**-b**) option. The default suffix is a ~ if no **--backup-dir** was specified, otherwise it is an empty string.

**-u, --update**

This forces rsync to skip any files which exist on the destination and have a modified time that is newer than the source file. (If an existing destination file has a modify time equal to the source file's, it will be updated if the sizes are different.)

In the current implementation of **--update**, a difference of file format between the sender and receiver is always considered to be important enough for an update, no matter what date is on the objects. In other words, if the source has a directory or a symlink where the destination has a file, the transfer would occur regardless of the timestamps. This might change in the future (feel free to comment on this on the mailing list if you have an opinion).

**--inplace**

This causes rsync not to create a new copy of the file and then move it into place. Instead rsync will overwrite the existing file, meaning that the rsync algorithm can't accomplish the full amount of network reduction it might be able to otherwise (since it does not yet try to sort data matches). One exception to this is if you combine the option with **--backup**, since rsync is smart enough to use the backup file as the basis file for the transfer.

This option is useful for transfer of large files with block-based changes or appended data, and also on systems that are disk bound, not network bound.

The option implies **--partial** (since an interrupted transfer does not delete the file), but conflicts with **--partial-dir** and **--delay-updates**. Prior to rsync 2.6.4 **--inplace** was also incompatible with **--compare-dest** and **--link-dest**.

WARNING: The file's data will be in an inconsistent state during the transfer (and possibly afterward if the transfer gets interrupted), so you should not use this option to update files that are in use. Also note that rsync will be unable to update a file in-place that is not writable by the receiving user.

**-d, --dirs**

Tell the sending side to include any directories that are encountered. Unlike **--recursive**, a directory's contents are not copied unless the directory was specified on the command-line as either "." or a name with a trailing slash (e.g. "foo/"). Without this option or the **--recursive** option, rsync will skip all directories it encounters (and output a message to that effect for each one).

**-l, --links**

When symlinks are encountered, recreate the symlink on the destination.

**-L, --copy-links**

When symlinks are encountered, the file that they point to (the referent) is copied, rather than the symlink. In older versions of rsync, this option also had the side-effect of telling the receiving side to follow symlinks, such as symlinks to directories. In a modern rsync such as this one, you'll need to specify **--keep-dirlinks** (**-K**) to get this extra behavior. The only exception is when sending files to an rsync that is too old to understand **-K** -- in that case, the **-L** option will still have the side-effect of **-K** on that older receiving rsync.

**--copy-unsafe-links**

This tells rsync to copy the referent of symbolic links that point outside the copied tree. Absolute symlinks are also treated like ordinary files, and so are any symlinks in the source path itself when **--relative** is used.

**--safe-links**

This tells rsync to ignore any symbolic links which point outside the copied tree. All absolute symlinks are also ignored. Using this option in conjunction with **--relative** may give unexpected results.

**-H, --hard-links**

This tells rsync to recreate hard links on the remote system to be the same as the local system. Without this option hard links are treated like regular files.

Note that rsync can only detect hard links if both parts of the link are in the list of files being sent.

This option can be quite slow, so only use it if you need it.

**-K, --keep-dirlinks**

On the receiving side, if a symlink is pointing to a directory, it will be treated as matching a directory from the sender.

**-W, --whole-file**

With this option the incremental rsync algorithm is not used and the whole file is sent as-is instead. The transfer may be faster if this option is used when the bandwidth between the source and destination machines is higher than the bandwidth to disk (especially when the "disk" is actually a networked filesystem). This is the default when both the source and destination are specified as local paths.

**--no-whole-file**

Turn off **--whole-file**, for use when it is the default.

**-p, --perms**

This option causes rsync to set the destination permissions to be the same as the source permissions.

Without this option, all existing files (including updated files) retain their existing permissions, while each new file gets its permissions set based on the source file's permissions, but masked by the receiving end's umask setting (which is the same behavior as other file-copy utilities, such as cp).

**-o, --owner**

This option causes rsync to set the owner of the destination file to be the same as the source file. On most systems, only the super-user can set file ownership. By default, the preservation is done by name, but may fall back to using the ID number in some circumstances. See the **--numeric-ids** option for a full discussion.

**-g, --group**

This option causes rsync to set the group of the destination file to be the same as the source file. If the receiving program is not running as the super-user, only groups that the receiver is a member of will be preserved. By default, the preservation is done by name, but may fall back to using the ID number in some circumstances. See the **--numeric-ids** option for a full discussion.

**-D, --devices**

This option causes rsync to transfer character and block device information to the remote system to recreate these devices. This option is only available to the super-user.

**-t, --times**

This tells rsync to transfer modification times along with the files and update them on the remote system. Note that if this option is not used, the optimization that excludes files that have not been modified cannot be effective; in other words, a missing **-t** or **-a** will cause the next transfer to behave as if it used **-I**, causing all files to be updated (though the rsync algorithm will make the update fairly efficient if the files haven't actually changed, you're much better off using **-t**).

**-O, --omit-dir-times**

This tells rsync to omit directories when it is preserving modification times (see **--times**). If NFS is sharing the directories on the receiving side, it is a good idea to use **-O**. This option is inferred if you use **--backup** without **--backup-dir**.

**-n, --dry-run**

This tells rsync to not do any file transfers, instead it will just report the actions it would have taken.

**-S, --sparse**

Try to handle sparse files efficiently so they take up less space on the destination.

NOTE: Don't use this option when the destination is a Solaris "tmpfs" filesystem. It doesn't seem to handle seeks over null regions correctly and ends up corrupting the files.

**-x, --one-file-system**

This tells rsync not to cross filesystem boundaries when recursing. This is useful for transferring the contents of only one filesystem.

**--existing**

This tells rsync not to create any new files -- only update files that already exist on the destination.

**--ignore-existing**

This tells rsync not to update files that already exist on the destination.

**--remove-sent-files**

This tells rsync to remove from the sending side the files and/or symlinks that are newly created or whose content is updated on the receiving side. Directories and devices are not removed, nor are files/symlinks whose attributes are merely changed.

**--delete**

This tells rsync to delete extraneous files from the receiving side (ones that aren't on the sending side), but only for the directories that are being synchronized. You must have asked rsync to send the whole directory (e.g. "dir" or "dir/") without using a wildcard for the directory's contents (e.g. "dir/*") since the wildcard is expanded by the shell and rsync thus gets a request to transfer individual files, not the files' parent directory. Files that are excluded from transfer are also excluded from being deleted unless you use the **--delete-excluded** option or mark the rules as only matching on the sending side (see the include/exclude modifiers in the FILTER RULES section).

This option has no effect unless directory recursion is enabled.

This option can be dangerous if used incorrectly! It is a very good idea to run first using the **--dry-run** option (**-n**) to see what files would be deleted to make sure important files aren't listed.

If the sending side detects any I/O errors, then the deletion of any files at the destination will be automatically disabled. This is to prevent temporary filesystem failures (such as NFS errors) on the sending side causing a massive deletion of files on the destination. You can override this with the **--ignore-errors** option.

The **--delete** option may be combined with one of the --delete-WHEN options without conflict, as well as **--delete-excluded**. However, if none of the --delete-WHEN options are specified, rsync will currently choose the **--delete-before** algorithm. A future version may change this to choose the **--delete-during** algorithm. See also **--delete-after**.

**--delete-before**

Request that the file-deletions on the receiving side be done before the transfer starts. This is the default if **--delete** or **--delete-excluded** is specified without one of the --delete-WHEN options. See **--delete** (which is implied) for more details on file-deletion.

Deleting before the transfer is helpful if the filesystem is tight for space and removing extraneous files would help to make the transfer possible. However, it does introduce a delay before the start of the transfer, and this delay might cause the transfer to timeout (if **--timeout** was specified).

**--delete-during, --del**

Request that the file-deletions on the receiving side be done incrementally as the transfer happens. This is a faster method than choosing the before- or after-transfer algorithm, but it is only supported beginning with rsync version 2.6.4. See **--delete** (which is implied) for more details on file-deletion.

**--delete-after**

Request that the file-deletions on the receiving side be done after the transfer has completed. This is useful if you are sending new per-directory merge files as a part of the transfer and you want their exclusions to take effect for the delete phase of the current transfer. See **--delete** (which is implied) for more details on file-deletion.

**--delete-excluded**

In addition to deleting the files on the receiving side that are not on the sending side, this tells rsync to also delete any files on the receiving side that are excluded (see **--exclude**). See the FILTER RULES section for a way to make individual exclusions behave this way on the receiver, and for a way to protect files from **--delete-excluded**. See **--delete** (which is implied) for more details on file-deletion.

**--ignore-errors**

Tells **--delete** to go ahead and delete files even when there are I/O errors.

**--force** This options tells rsync to delete directories even if they are not empty when they are to be replaced by non-directories. This is only relevant without **--delete** because deletions are now done depth-first. Requires the **--recursive** option (which is implied by **-a**) to have any effect.

**--max-delete=NUM**

This tells rsync not to delete more than NUM files or directories (NUM must be non-zero). This is useful when mirroring very large trees to prevent disasters.

**--max-size=SIZE**

This tells rsync to avoid transferring any file that is larger than the specified SIZE. The SIZE value can be suffixed with a letter to indicate a size multiplier (K, M, or G) and may be a fractional value (e.g. "**--max-size=1.5m**").

**-B, --block-size=BLOCKSIZE**

This forces the block size used in the rsync algorithm to a fixed value. It is normally selected based on the size of each file being updated. See the technical report for details.

**-e, --rsh=COMMAND**

This option allows you to choose an alternative remote shell program to use for communication between the local and remote copies of rsync. Typically, rsync is configured to use ssh by default, but you may prefer to use rsh on a local network.

If this option is used with **[user@]host::module/path**, then the remote shell *COMMAND* will be used to run an rsync daemon on the remote host, and all data will be transmitted through that remote shell connection, rather than through a direct socket connection to a running rsync daemon on the remote host. See the section "USING RSYNC-DAEMON FEATURES VIA A REMOTE-SHELL CONNECTION" above.

Command-line arguments are permitted in COMMAND provided that COMMAND is presented to rsync as a single argument. For example:

```
-e "ssh -p 2234"
```

(Note that ssh users can alternately customize site-specific connect options in their .ssh/config file.)

You can also choose the remote shell program using the RSYNC_RSH environment variable, which accepts the same range of values as **-e**.

See also the **--blocking-io** option which is affected by this option.

**--rsync-path=PROGRAM**

Use this to specify what program is to be run on the remote machine to start-up rsync. Often used when rsync is not in the default remote-shell's path (e.g. --rsync-path=/usr/local/bin/rsync). Note that PROGRAM is run with the help of a shell, so it can be any program, script, or command sequence you'd care to run, so long as it does not corrupt the standard-in & standard-out that rsync is using to communicate.

One tricky example is to set a different default directory on the remote machine for use with the **--relative** option. For instance:

```
rsync -avR --rsync-path="cd /a/b && rsync" hst:c/d /e/
```

**-C, --cvs-exclude**

This is a useful shorthand for excluding a broad range of files that you often don't want to transfer between systems. It uses the same algorithm that CVS uses to determine if a file should be ignored.

The exclude list is initialized to:

```
RCS SCCS CVS CVS.adm RCSLOG cvslog.* tags TAGS .make.state
.nse_depinfo *~ #* .#* ,* _$* *$ *.old *.bak *.BAK *.orig
*.rej .del-* *.a *.olb *.o *.obj *.so *.exe *.Z *.elc *.ln
core .svn/
```

then files listed in a $HOME/.cvsignore are added to the list and any files listed in the CVSIGNORE environment variable (all cvsignore names are delimited by whitespace).

Finally, any file is ignored if it is in the same directory as a .cvsignore file and matches one of the patterns listed therein. Unlike rsync's filter/exclude files, these patterns are split on whitespace. See the **cvs(1)** manual for more information.

If you're combining **-C** with your own **--filter** rules, you should note that these CVS excludes are appended at the end of your own rules, regardless of where the **-C** was placed on the command-line. This makes them a lower priority than any rules you specified explicitly. If you want to control where these CVS excludes get inserted into your filter rules, you should omit the **-C** as a command-line option and use a combination of **--filter=:C** and **--filter=-C** (either on your command-line or by putting the ":C" and "-C" rules into a filter file with your other rules). The first option turns on the per-directory scanning for the .cvsignore file. The second option does a one-time import of the CVS excludes mentioned above.

**-f, --filter=RULE**

This option allows you to add rules to selectively exclude certain files from the list of files to be transferred. This is most useful in combination with a recursive transfer.

You may use as many **--filter** options on the command line as you like to build up the list of files to exclude.

See the FILTER RULES section for detailed information on this option.

**-F** The **-F** option is a shorthand for adding two **--filter** rules to your command. The first time it is used is a shorthand for this rule:

```
--filter=': /.rsync-filter'
```

This tells rsync to look for per-directory .rsync-filter files that have been sprinkled through the hierarchy and use their rules to filter the files in the transfer. If **-F** is repeated, it is a shorthand for this rule:

```
--filter='- .rsync-filter'
```

This filters out the .rsync-filter files themselves from the transfer.

See the FILTER RULES section for detailed information on how these options work.

**--exclude=PATTERN**

This option is a simplified form of the **--filter** option that defaults to an exclude rule and does not allow the full rule-parsing syntax of normal filter rules.

See the FILTER RULES section for detailed information on this option.

**--exclude-from=FILE**

This option is similar to the **--exclude** option, but instead it adds all exclude patterns listed in the file FILE to the exclude list. Blank lines in FILE and lines starting with ';' or '#' are ignored. If *FILE* is **-** the list will be read from standard input.

**--include=PATTERN**

This option is a simplified form of the **--filter** option that defaults to an include rule and does not allow the full rule-parsing syntax of normal filter rules.

See the FILTER RULES section for detailed information on this option.

**--include-from=FILE**

This specifies a list of include patterns from a file. If *FILE* is "-" the list will be read from standard input.

**--files-from=FILE**

Using this option allows you to specify the exact list of files to transfer (as read from the specified FILE or "-" for standard input). It also tweaks the default behavior of rsync to make transferring just the specified files and directories easier:

- The **--relative** (**-R**) option is implied, which preserves the path information that is specified for each item in the file (use **--no-relative** if you want to turn that off).
- The **--dirs** (**-d**) option is implied, which will create directories specified in the list on the destination rather than noisily skipping them.
- The **--archive** (**-a**) option's behavior does not imply **--recursive** (**-r**), so specify it explicitly, if you want it.

The file names that are read from the FILE are all relative to the source dir -- any leading slashes are removed and no ".." references are allowed to go higher than the source dir. For example, take this command:

```
rsync -a --files-from=/tmp/foo /usr remote:/backup
```

If /tmp/foo contains the string "bin" (or even "/bin"), the /usr/bin directory will be created as /backup/bin on the remote host. If it contains "bin/" (note the trailing slash), the immediate contents of the directory would also be sent (without needing to be explicitly mentioned in the file -- this began in version 2.6.4). In both cases, if the **-r** option was enabled, that dir's entire hierarchy would also be transferred (keep in mind that **-r** needs to be specified explicitly with **--files-from**, since it is not implied by **-a**). Also note that the effect of the (enabled by default) **--relative** option is to duplicate only the path info that is read from the file -- it does not force the duplication of the source-spec path (/usr in this case).

In addition, the **--files-from** file can be read from the remote host instead of the local host if you specify a "host:" in front of the file (the host must match one end of the transfer). As a short-cut, you can specify just a prefix of ":" to mean "use the remote end of the transfer". For example:

```
rsync -a --files-from=:/path/file-list src:/ /tmp/copy
```

This would copy all the files specified in the /path/file-list file that was located on the remote "src" host.

**-0, --from0**

This tells rsync that the rules/filenames it reads from a file are terminated by a null ('\0') character, not a NL, CR, or CR+LF. This affects **--exclude-from**, **--include-from**, **--files-from**, and any merged files specified in a **--filter** rule. It does not affect **--cvs-exclude** (since all names read from a .cvsignore file are split on whitespace).

**-T, --temp-dir=DIR**

This option instructs rsync to use DIR as a scratch directory when creating temporary copies of the files transferred on the receiving side. The default behavior is to create the temporary files in the receiving directory.

**-y, --fuzzy**

This option tells rsync that it should look for a basis file for any destination file that is missing. The current algorithm looks in the same directory as the destination file for either a file that has an identical size and modified-time, or a similarly-named file. If found, rsync uses the fuzzy basis file to try to speed up the transfer.

Note that the use of the **--delete** option might get rid of any potential fuzzy-match files, so either use **--delete-after** or specify some filename exclusions if you need to prevent this.

**--compare-dest=DIR**

This option instructs rsync to use *DIR* on the destination machine as an additional hierarchy to compare destination files against doing transfers (if the files are missing in the destination directory). If a file is found in *DIR* that is identical to the sender's file, the file will NOT be transferred to the destination directory. This is useful for creating a sparse backup of just files that have changed from an earlier backup.

Beginning in version 2.6.4, multiple **--compare-dest** directories may be provided, which will cause rsync to search the list in the order specified for an exact match. If a match is found that differs only in attributes, a local copy is made and the attributes updated. If a match is not found, a basis file from one of the *DIR*s will be selected to try to speed up the transfer.

If *DIR* is a relative path, it is relative to the destination directory. See also **--copy-dest** and **--link-dest**.

**--copy-dest=DIR**

This option behaves like **--compare-dest**, but rsync will also copy unchanged files found in *DIR* to the destination directory using a local copy. This is useful for doing transfers to a new destination while leaving existing files intact, and then doing a flash-cutover when all files have been successfully transferred.

Multiple **--copy-dest** directories may be provided, which will cause rsync to search the list in the order specified for an unchanged file. If a match is not found, a basis file from one of the *DIR*s will be selected to try to speed up the transfer.

If *DIR* is a relative path, it is relative to the destination directory. See also **--compare-dest** and **--link-dest**.

**--link-dest=DIR**

This option behaves like **--copy-dest**, but unchanged files are hard linked from *DIR* to the destination directory. The files must be identical in all preserved attributes (e.g. permissions, possibly

ownership) in order for the files to be linked together. An example:

```
rsync -av --link-dest=$PWD/prior_dir host:src_dir/ new_dir/
```

Beginning in version 2.6.4, multiple **--link-dest** directories may be provided, which will cause rsync to search the list in the order specified for an exact match. If a match is found that differs only in attributes, a local copy is made and the attributes updated. If a match is not found, a basis file from one of the *DIR*s will be selected to try to speed up the transfer.

If *DIR* is a relative path, it is relative to the destination directory. See also **--compare-dest** and **--copy-dest**.

Note that rsync versions prior to 2.6.1 had a bug that could prevent **--link-dest** from working properly for a non-root user when **-o** was specified (or implied by **-a**). You can work-around this bug by avoiding the **-o** option when sending to an old rsync.

**-z, --compress**

With this option, rsync compresses the file data as it is sent to the destination machine, which reduces the amount of data being transmitted -- something that is useful over a slow connection.

Note this this option typically achieves better compression ratios that can be achieved by using a compressing remote shell or a compressing transport because it takes advantage of the implicit information in the matching data blocks that are not explicitly sent over the connection.

**--numeric-ids**

With this option rsync will transfer numeric group and user IDs rather than using user and group names and mapping them at both ends.

By default rsync will use the username and groupname to determine what ownership to give files. The special uid 0 and the special group 0 are never mapped via user/group names even if the **--numeric-ids** option is not specified.

If a user or group has no name on the source system or it has no match on the destination system, then the numeric ID from the source system is used instead. See also the comments on the "use chroot" setting in the rsyncd.conf manpage for information on how the chroot setting affects rsync's ability to look up the names of the users and groups and what you can do about it.

**--timeout=TIMEOUT**

This option allows you to set a maximum I/O timeout in seconds. If no data is transferred for the specified time then rsync will exit. The default is 0, which means no timeout.

**--address**

By default rsync will bind to the wildcard address when connecting to an rsync daemon. The **--address** option allows you to specify a specific IP address (or hostname) to bind to. See also this option in the **--daemon** mode section.

**--port=PORT**

This specifies an alternate TCP port number to use rather than the default of 873. This is only needed if you are using the double-colon (::) syntax to connect with an rsync daemon (since the URL syntax has a way to specify the port as a part of the URL). See also this option in the **--daemon** mode section.

**--blocking-io**

This tells rsync to use blocking I/O when launching a remote shell transport. If the remote shell is either rsh or remsh, rsync defaults to using blocking I/O, otherwise it defaults to using non-blocking I/O. (Note that ssh prefers non-blocking I/O.)

**--no-blocking-io**

Turn off **--blocking-io**, for use when it is the default.

**-i, --itemize-changes**

Requests a simple itemized list of the changes that are being made to each file, including attribute changes. This is exactly the same as specifying **--log-format='%i %n%L'**.

The "%i" escape has a cryptic output that is 9 letters long. The general format is like the string **UXcstpoga**), where **U** is replaced by the kind of update being done, **X** is replaced by the file-type, and the other letters represent attributes that may be output if they are being modified.

The update types that replace the **U** are as follows:

- A < means that a file is being transferred to the remote host (sent).
- A > means that a file is being transferred to the local host (received).
- A **c** means that a local change/creation is occurring for the item (such as the creation of a directory or the changing of a symlink, etc.).
- A **h** means that the item is a hard-link to another item (requires **--hard-links**).
- A **.** means that the item is not being updated (though it might have attributes that are being modified).

The file-types that replace the **X** are: **f** for a file, a **d** for a directory, an **L** for a symlink, and a **D** for a device.

The other letters in the string above are the actual letters that will be output if the associated attribute for the item is being updated or a "." for no change. Three exceptions to this are: (1) a newly created item replaces each letter with a "+", (2) an identical item replaces the dots with spaces, and (3) an unknown attribute replaces each letter with a "?" (this can happen when talking to an older rsync).

The attribute that is associated with each letter is as follows:

- A **c** means the checksum of the file is different and will be updated by the file transfer (requires **--checksum**).
- A **s** means the size of the file is different and will be updated by the file transfer.
- A **t** means the modification time is different and is being updated to the sender's value (requires **--times**). An alternate value of **T** means that the time will be set to the transfer time, which happens anytime a symlink is transferred, or when a file or device is transferred without **--times**.
- A **p** means the permissions are different and are being updated to the sender's value (requires **--perms**).
- An **o** means the owner is different and is being updated to the sender's value (requires **--owner** and root privileges).
- A **g** means the group is different and is being updated to the sender's value (requires **--group** and the authority to set the group).
- The **a** is reserved for a future enhanced version that supports extended file attributes, such as ACLs.

One other output is possible: when deleting files, the "%i" will output the string "*deleting" for each item that is being removed (assuming that you are talking to a recent enough rsync that it logs deletions instead of outputting them as a verbose message).

**--log-format=FORMAT**

This allows you to specify exactly what the rsync client outputs to the user on a per-file basis. The format is a text string containing embedded single-character escape sequences prefixed with a percent (%) character. For a list of the possible escape characters, see the "log format" setting in the rsyncd.conf manpage. (Note that this option does not affect what a daemon logs to its logfile.)

Specifying this option will mention each file, dir, etc. that gets updated in a significant way (a transferred file, a recreated symlink/device, or a touched directory) unless the itemized-changes escape (%i) is included in the string, in which case the logging of names increases to mention any item that is changed in any way (as long as the receiving side is at least 2.6.4). See the **--itemized-changes** option for a description of the output of "%i".

The **--verbose** option implies a format of "%n%L", but you can use **--log-format** without bv(--verbose) if you like, or you can override the format of its per-file output using this option.

Rsync will output the log-format string prior to a file's transfer unless one of the transfer-statistic escapes is requested, in which case the logging is done at the end of the file's transfer. When this late logging is in effect and **--progress** is also specified, rsync will also output the name of the file being transferred prior to its progress information (followed, of course, by the log-format output).

**--stats** This tells rsync to print a verbose set of statistics on the file transfer, allowing you to tell how effective the rsync algorithm is for your data.

**--partial**

By default, rsync will delete any partially transferred file if the transfer is interrupted. In some circumstances it is more desirable to keep partially transferred files. Using the **--partial** option tells rsync to keep the partial file which should make a subsequent transfer of the rest of the file much faster.

**--partial-dir=DIR**

A better way to keep partial files than the **--partial** option is to specify a *DIR* that will be used to hold the partial data (instead of writing it out to the destination file). On the next transfer, rsync will use a file found in this dir as data to speed up the resumption of the transfer and then deletes it after it has served its purpose. Note that if **--whole-file** is specified (or implied), any partial-dir file that is found for a file that is being updated will simply be removed (since rsync is sending files without using the incremental rsync algorithm).

Rsync will create the *DIR* if it is missing (just the last dir -- not the whole path). This makes it easy to use a relative path (such as "**--partial-dir=.rsync-partial**") to have rsync create the partial-directory in the destination file's directory when needed, and then remove it again when the partial file is deleted.

If the partial-dir value is not an absolute path, rsync will also add a directory **--exclude** of this value at the end of all your existing excludes. This will prevent partial-dir files from being transferred and also prevent the untimely deletion of partial-dir items on the receiving side. An example: the above **--partial-dir** option would add an "**--exclude=.rsync-partial/**" rule at the end of any other filter rules. Note that if you are supplying your own filter rules, you may need to manually insert a rule for this directory exclusion somewhere higher up in the list so that it has a high enough priority to be effective (e.g., if your rules specify a trailing **--exclude='*'** rule, the auto-added rule would never be reached).

IMPORTANT: the **--partial-dir** should not be writable by other users or it is a security risk. E.g. AVOID "/tmp".

You can also set the partial-dir value the RSYNC_PARTIAL_DIR environment variable. Setting this in the environment does not force **--partial** to be enabled, but rather it effects where partial files go when **--partial** is specified. For instance, instead of using **--partial-dir=.rsync-tmp** along with **--progress**, you could set RSYNC_PARTIAL_DIR=.rsync-tmp in your environment and then just use the **-P** option to turn on the use of the .rsync-tmp dir for partial transfers. The only time that the **--partial** option does not look for this environment value is (1) when **--inplace** was specified (since **--inplace** conflicts with **--partial-dir**), or (2) when **--delay-updates** was specified (see below).

For the purposes of the daemon-config's "refuse options" setting, **--partial-dir** does *not* imply **--partial**. This is so that a refusal of the **--partial** option can be used to disallow the overwriting

of destination files with a partial transfer, while still allowing the safer idiom provided by **--partial-dir**.

**--delay-updates**

This option puts the temporary file from each updated file into a holding directory until the end of the transfer, at which time all the files are renamed into place in rapid succession. This attempts to make the updating of the files a little more atomic. By default the files are placed into a directory named ".~tmp~" in each file's destination directory, but you can override this by specifying the **--partial-dir** option. (Note that RSYNC_PARTIAL_DIR has no effect on this value, nor is **--partial-dir** considered to be implied for the purposes of the daemon-config's "refuse options" setting.) Conflicts with **--inplace**.

This option uses more memory on the receiving side (one bit per file transferred) and also requires enough free disk space on the receiving side to hold an additional copy of all the updated files. Note also that you should not use an absolute path to **--partial-dir** unless there is no chance of any of the files in the transfer having the same name (since all the updated files will be put into a single directory if the path is absolute).

See also the "atomic-rsync" perl script in the "support" subdir for an update algorithm that is even more atomic (it uses **--link-dest** and a parallel hierarchy of files).

**--progress**

This option tells rsync to print information showing the progress of the transfer. This gives a bored user something to watch. Implies **--verbose** if it wasn't already specified.

When the file is transferring, the data looks like this:

```
782448  63%  110.64kB/s    0:00:04
```

This tells you the current file size, the percentage of the transfer that is complete, the current calculated file-completion rate (including both data over the wire and data being matched locally), and the estimated time remaining in this transfer.

After a file is complete, the data looks like this:

```
1238099 100%  146.38kB/s    0:00:08  (5, 57.1% of 396)
```

This tells you the final file size, that it's 100% complete, the final transfer rate for the file, the amount of elapsed time it took to transfer the file, and the addition of a total-transfer summary in parentheses. These additional numbers tell you how many files have been updated, and what percent of the total number of files has been scanned.

**-P** The **-P** option is equivalent to **--partial --progress**. Its purpose is to make it much easier to specify these two options for a long transfer that may be interrupted.

**--password-file**

This option allows you to provide a password in a file for accessing a remote rsync daemon. Note that this option is only useful when accessing an rsync daemon using the built in transport, not when using a remote shell as the transport. The file must not be world readable. It should contain just the password as a single line.

**--list-only**

This option will cause the source files to be listed instead of transferred. This option is inferred if there is no destination specified, so you don't usually need to use it explicitly. However, it can come in handy for a user that wants to avoid the "**-r --exclude='/*/*'**" options that rsync might use as a compatibility kluge when generating a non-recursive listing, or to list the files that are involved in a local copy (since the destination path is not optional for a local copy, you must specify this option explicitly and still include a destination).

**--bwlimit=KBPS**

This option allows you to specify a maximum transfer rate in kilobytes per second. This option is most effective when using rsync with large files (several megabytes and up). Due to the nature of rsync transfers, blocks of data are sent, then if rsync determines the transfer was too fast, it will wait before sending the next data block. The result is an average transfer rate equaling the specified limit. A value of zero specifies no limit.

**--write-batch=FILE**

Record a file that can later be applied to another identical destination with **--read-batch**. See the "BATCH MODE" section for details, and also the **--only-write-batch** option.

**--only-write-batch=FILE**

Works like **--write-batch**, except that no updates are made on the destination system when creating the batch. This lets you transport the changes to the destination system via some other means and then apply the changes via **--read-batch**.

Note that you can feel free to write the batch directly to some portable media: if this media fills to capacity before the end of the transfer, you can just apply that partial transfer to the destination and repeat the whole process to get the rest of the changes (as long as you don't mind a partially updated destination system while the multi-update cycle is happening).

Also note that you only save bandwidth when pushing changes to a remote system because this allows the batched data to be diverted from the sender into the batch file without having to flow over the wire to the receiver (when pulling, the sender is remote, and thus can't write the batch).

**--read-batch=FILE**

Apply all of the changes stored in FILE, a file previously generated by **--write-batch**. If *FILE* is "-" the batch data will be read from standard input. See the "BATCH MODE" section for details.

**--protocol=NUM**

Force an older protocol version to be used. This is useful for creating a batch file that is compatible with an older version of rsync. For instance, if rsync 2.6.4 is being used with the **--write-batch** option, but rsync 2.6.3 is what will be used to run the **--read-batch** option, you should use "--protocol=28" when creating the batch file to force the older protocol version to be used in the batch file (assuming you can't upgrade the rsync on the reading system).

**-4, --ipv4** or **-6, --ipv6**

Tells rsync to prefer IPv4/IPv6 when creating sockets. This only affects sockets that rsync has direct control over, such as the outgoing socket when directly contacting an rsync daemon. See also these options in the **--daemon** mode section.

**--checksum-seed=NUM**

Set the MD4 checksum seed to the integer NUM. This 4 byte checksum seed is included in each block and file MD4 checksum calculation. By default the checksum seed is generated by the server and defaults to the current time(). This option is used to set a specific checksum seed, which is useful for applications that want repeatable block and file checksums, or in the case where the user wants a more random checksum seed. Note that setting NUM to 0 causes rsync to use the default of time() for checksum seed.

## DAEMON OPTIONS

The options allowed when starting an rsync daemon are as follows:

**--daemon**

This tells rsync that it is to run as a daemon. The daemon you start running may be accessed using an rsync client using the **host::module** or **rsync://host/module/** syntax.

If standard input is a socket then rsync will assume that it is being run via inetd, otherwise it will detach from the current terminal and become a background daemon. The daemon will read the config file (rsyncd.conf) on each connect made by a client and respond to requests accordingly. See the rsyncd.conf(5) man page for more details.

**--address**

By default rsync will bind to the wildcard address when run as a daemon with the **--daemon** option. The **--address** option allows you to specify a specific IP address (or hostname) to bind to. This makes virtual hosting possible in conjunction with the **--config** option. See also the "address" global option in the rsyncd.conf manpage.

**--bwlimit=KBPS**

This option allows you to specify a maximum transfer rate in kilobytes per second for the data the daemon sends. The client can still specify a smaller **--bwlimit** value, but their requested value will be rounded down if they try to exceed it. See the client version of this option (above) for some extra details.

**--config=FILE**

This specifies an alternate config file than the default. This is only relevant when **--daemon** is specified. The default is /etc/rsyncd.conf unless the daemon is running over a remote shell program and the remote user is not root; in that case the default is rsyncd.conf in the current directory (typically $HOME).

**--no-detach**

When running as a daemon, this option instructs rsync to not detach itself and become a background process. This option is required when running as a service on Cygwin, and may also be useful when rsync is supervised by a program such as **daemontools** or AIX's **System Resource Controller**. **--no-detach** is also recommended when rsync is run under a debugger. This option has no effect if rsync is run from inetd or sshd.

**--port=PORT**

This specifies an alternate TCP port number for the daemon to listen on rather than the default of 873. See also the "port" global option in the rsyncd.conf manpage.

**-v, --verbose**

This option increases the amount of information the daemon logs during its startup phase. After the client connects, the daemon's verbosity level will be controlled by the options that the client used and the "max verbosity" setting in the module's config section.

**-4, --ipv4** or **-6, --ipv6**

Tells rsync to prefer IPv4/IPv6 when creating the incoming sockets that the rsync daemon will use to listen for connections. One of these options may be required in older versions of Linux to work around an IPv6 bug in the kernel (if you see an "address already in use" error when nothing else is using the port, try specifying **--ipv6** or **--ipv4** when starting the daemon).

**-h, --help**

When specified after **--daemon**, print a short help page describing the options available for starting an rsync daemon.

## FILTER RULES

The filter rules allow for flexible selection of which files to transfer (include) and which files to skip (exclude). The rules either directly specify include/exclude patterns or they specify a way to acquire more include/exclude patterns (e.g. to read them from a file).

As the list of files/directories to transfer is built, rsync checks each name to be transferred against the list of include/exclude patterns in turn, and the first matching pattern is acted on: if it is an exclude pattern, then that file is skipped; if it is an include pattern then that filename is not skipped; if no matching pattern is found, then the filename is not skipped.

Rsync builds an ordered list of filter rules as specified on the command-line. Filter rules have the following syntax:

```
RULE [PATTERN_OR_FILENAME]
RULE,MODIFIERS [PATTERN_OR_FILENAME]
```

You have your choice of using either short or long RULE names, as described below. If you use a short-named rule, the ',' separating the RULE from the MODIFIERS is optional. The PATTERN or FILENAME that follows (when present) must come after either a single space or an underscore (_). Here are the available rule prefixes:

**exclude, -** specifies an exclude pattern.
**include, +** specifies an include pattern.
**merge, .** specifies a merge-file to read for more rules.
**dir-merge, :** specifies a per-directory merge-file.
**hide, H** specifies a pattern for hiding files from the transfer.
**show, S** files that match the pattern are not hidden.
**protect, P** specifies a pattern for protecting files from deletion.
**risk, R** files that match the pattern are not protected.
**clear, !** clears the current include/exclude list (takes no arg)

When rules are being read from a file, empty lines are ignored, as are comment lines that start with a "#".

Note that the **--include**/**--exclude** command-line options do not allow the full range of rule parsing as described above -- they only allow the specification of include/exclude patterns plus a "!" token to clear the list (and the normal comment parsing when rules are read from a file). If a pattern does not begin with "- " (dash, space) or "+ " (plus, space), then the rule will be interpreted as if "+ " (for an include option) or "- " (for an exclude option) were prefixed to the string. A **--filter** option, on the other hand, must always contain either a short or long rule name at the start of the rule.

Note also that the **--filter**, **--include**, and **--exclude** options take one rule/pattern each. To add multiple ones, you can repeat the options on the command-line, use the merge-file syntax of the **--filter** option, or the **--include-from**/**--exclude-from** options.

## INCLUDE/EXCLUDE PATTERN RULES

You can include and exclude files by specifying patterns using the "+", "-", etc. filter rules (as introduced in the FILTER RULES section above). The include/exclude rules each specify a pattern that is matched against the names of the files that are going to be transferred. These patterns can take several forms:

- o if the pattern starts with a / then it is anchored to a particular spot in the hierarchy of files, otherwise it is matched against the end of the pathname. This is similar to a leading ^ in regular expressions. Thus "/foo" would match a file called "foo" at either the "root of the transfer" (for a global rule) or in the merge-file's directory (for a per-directory rule). An unqualified "foo" would match any file or directory named "foo" anywhere in the tree because the algorithm is applied recursively from the top down; it behaves as if each path component gets a turn at being the end of the file name. Even the unanchored "sub/foo" would match at any point in the hierarchy where a "foo" was found within a directory named "sub". See the section on ANCHORING INCLUDE/EXCLUDE PATTERNS for a full discussion of how to specify a pattern that matches at the root of the transfer.

- if the pattern ends with a / then it will only match a directory, not a file, link, or device.
- if the pattern contains a wildcard character from the set *?[ then expression matching is applied using the shell filename matching rules. Otherwise a simple string match is used.
- the double asterisk pattern "**" will match slashes while a single asterisk pattern "*" will stop at slashes.
- if the pattern contains a / (not counting a trailing /) or a "**" then it is matched against the full pathname, including any leading directories. If the pattern doesn't contain a / or a "**", then it is matched only against the final component of the filename. (Remember that the algorithm is applied recursively so "full filename" can actually be any portion of a path from the starting directory on down.)

Note that, when using the **--recursive** (**-r**) option (which is implied by **-a**), every subcomponent of every path is visited from the top down, so include/exclude patterns get applied recursively to each subcomponent's full name (e.g. to include "/foo/bar/baz" the subcomponents "/foo" and "/foo/bar" must not be excluded). The exclude patterns actually short-circuit the directory traversal stage when rsync finds the files to send. If a pattern excludes a particular parent directory, it can render a deeper include pattern ineffectual because rsync did not descend through that excluded section of the hierarchy. This is particularly important when using a trailing '*' rule. For instance, this won't work:

```
+ /some/path/this-file-will-not-be-found
+ /file-is-included
- *
```

This fails because the parent directory "some" is excluded by the '*' rule, so rsync never visits any of the files in the "some" or "some/path" directories. One solution is to ask for all directories in the hierarchy to be included by using a single rule: "+ */" (put it somewhere before the "- *" rule). Another solution is to add specific include rules for all the parent dirs that need to be visited. For instance, this set of rules works fine:

```
+ /some/
+ /some/path/
+ /some/path/this-file-is-found
+ /file-also-included
- *
```

Here are some examples of exclude/include matching:

- "- *.o" would exclude all filenames matching *.o
- "- /foo" would exclude a file called foo in the transfer-root directory
- "- foo/" would exclude any directory called foo
- "- /foo/*/bar" would exclude any file called bar two levels below a directory called foo in the transfer-root directory
- "- /foo/**/bar" would exclude any file called bar two or more levels below a directory called foo in the transfer-root directory
- The combination of "+ */", "+ *.c", and "- *" would include all directories and C source files but nothing else.
- The combination of "+ foo/", "+ foo/bar.c", and "- *" would include only the foo directory and foo/bar.c (the foo directory must be explicitly included or it would be excluded by the "*")

## MERGE-FILE FILTER RULES

You can merge whole files into your filter rules by specifying either a merge (.) or a dir-merge (:) filter rule (as introduced in the FILTER RULES section above).

There are two kinds of merged files -- single-instance ('.') and per-directory (':'). A single-instance merge file is read one time, and its rules are incorporated into the filter list in the place of the "." rule. For per-

directory merge files, rsync will scan every directory that it traverses for the named file, merging its contents when the file exists into the current list of inherited rules. These per-directory rule files must be created on the sending side because it is the sending side that is being scanned for the available files to transfer. These rule files may also need to be transferred to the receiving side if you want them to affect what files don't get deleted (see PER-DIRECTORY RULES AND DELETE below).

Some examples:

```
merge /etc/rsync/default.rules
. /etc/rsync/default.rules
dir-merge .per-dir-filter
dir-merge,n- .non-inherited-per-dir-excludes
:n- .non-inherited-per-dir-excludes
```

The following modifiers are accepted after a merge or dir-merge rule:

- A **-** specifies that the file should consist of only exclude patterns, with no other rule-parsing except for in-file comments.
- A **+** specifies that the file should consist of only include patterns, with no other rule-parsing except for in-file comments.
- A **C** is a way to specify that the file should be read in a CVS-compatible manner. This turns on 'n', 'w', and '-', but also allows the list-clearing token (!) to be specified. If no filename is provided, ".cvsignore" is assumed.
- A **e** will exclude the merge-file name from the transfer; e.g. "dir-merge,e .rules" is like "dir-merge .rules" and "- .rules".
- An **n** specifies that the rules are not inherited by subdirectories.
- A **w** specifies that the rules are word-split on whitespace instead of the normal line-splitting. This also turns off comments. Note: the space that separates the prefix from the rule is treated specially, so "- foo + bar" is parsed as two rules (assuming that prefix-parsing wasn't also disabled).
- You may also specify any of the modifiers for the "+" or "-" rules (below) in order to have the rules that are read-in from the file default to having that modifier set. For instance, "merge,-/ .excl" would treat the contents of .excl as absolute-path excludes, while "dir-merge,s .filt" and ":sC" would each make all their per-directory rules apply only on the sending side.

The following modifiers are accepted after a "+" or "-":

- A "/" specifies that the include/exclude should be treated as an absolute path, relative to the root of the filesystem. For example, "-/ /etc/passwd" would exclude the passwd file any time the transfer was sending files from the "/etc" directory.
- A "!" specifies that the include/exclude should take effect if the pattern fails to match. For instance, "-! */" would exclude all non-directories.
- A **C** is used to indicate that all the global CVS-exclude rules should be inserted as excludes in place of the "-C". No arg should follow.
- An **s** is used to indicate that the rule applies to the sending side. When a rule affects the sending side, it prevents files from being transferred. The default is for a rule to affect both sides unless **--delete-excluded** was specified, in which case default rules become sender-side only. See also the hide (H) and show (S) rules, which are an alternate way to specify sending-side includes/excludes.
- An **r** is used to indicate that the rule applies to the receiving side. When a rule affects the receiving side, it prevents files from being deleted. See the **s** modifier for more info. See also the protect (P) and risk (R) rules, which are an alternate way to specify receiver-side includes/excludes.

Per-directory rules are inherited in all subdirectories of the directory where the merge-file was found unless the 'n' modifier was used. Each subdirectory's rules are prefixed to the inherited per-directory rules from

its parents, which gives the newest rules a higher priority than the inherited rules. The entire set of dir-merge rules are grouped together in the spot where the merge-file was specified, so it is possible to override dir-merge rules via a rule that got specified earlier in the list of global rules. When the list-clearing rule ("!") is read from a per-directory file, it only clears the inherited rules for the current merge file.

Another way to prevent a single rule from a dir-merge file from being inherited is to anchor it with a leading slash. Anchored rules in a per-directory merge-file are relative to the merge-file's directory, so a pattern "/foo" would only match the file "foo" in the directory where the dir-merge filter file was found.

Here's an example filter file which you'd specify via **--filter=". file":**

```
merge /home/user/.global-filter
- *.gz
dir-merge .rules
+ *.[ch]
- *.o
```

This will merge the contents of the /home/user/.global-filter file at the start of the list and also turns the ".rules" filename into a per-directory filter file. All rules read-in prior to the start of the directory scan follow the global anchoring rules (i.e. a leading slash matches at the root of the transfer).

If a per-directory merge-file is specified with a path that is a parent directory of the first transfer directory, rsync will scan all the parent dirs from that starting point to the transfer directory for the indicated per-directory file. For instance, here is a common filter (see **-F**):

```
--filter=': /.rsync-filter'
```

That rule tells rsync to scan for the file .rsync-filter in all directories from the root down through the parent directory of the transfer prior to the start of the normal directory scan of the file in the directories that are sent as a part of the transfer. (Note: for an rsync daemon, the root is always the same as the module's "path".)

Some examples of this pre-scanning for per-directory files:

```
rsync -avF /src/path/ /dest/dir
rsync -av --filter=': ../../.rsync-filter' /src/path/ /dest/dir
rsync -av --filter=': .rsync-filter' /src/path/ /dest/dir
```

The first two commands above will look for ".rsync-filter" in "/" and "/src" before the normal scan begins looking for the file in "/src/path" and its subdirectories. The last command avoids the parent-dir scan and only looks for the ".rsync-filter" files in each directory that is a part of the transfer.

If you want to include the contents of a ".cvsignore" in your patterns, you should use the rule ":C", which creates a dir-merge of the .cvsignore file, but parsed in a CVS-compatible manner. You can use this to affect where the **--cvs-exclude** (**-C**) option's inclusion of the per-directory .cvsignore file gets placed into your rules by putting the ":C" wherever you like in your filter rules. Without this, rsync would add the dir-merge rule for the .cvsignore file at the end of all your other rules (giving it a lower priority than your command-line rules). For example:

```
cat <<EOT | rsync -avC --filter=. - a/ b
+ foo.o
:C
- *.old
EOT
rsync -avC --include=foo.o -f :C --exclude='*.old' a/ b
```

Both of the above rsync commands are identical. Each one will merge all the per-directory .cvsignore rules in the middle of the list rather than at the end. This allows their dir-specific rules to supersede the rules that follow the :C instead of being subservient to all your rules. To affect the other CVS exclude rules (i.e. the default list of exclusions, the contents of $HOME/.cvsignore, and the value of $CVSIGNORE) you should omit the **-C** command-line option and instead insert a "-C" rule into your filter rules; e.g. "--filter=-C".

## LIST-CLEARING FILTER RULE

You can clear the current include/exclude list by using the "!" filter rule (as introduced in the FILTER RULES section above). The "current" list is either the global list of rules (if the rule is encountered while parsing the filter options) or a set of per-directory rules (which are inherited in their own sub-list, so a subdirectory can use this to clear out the parent's rules).

## ANCHORING INCLUDE/EXCLUDE PATTERNS

As mentioned earlier, global include/exclude patterns are anchored at the "root of the transfer" (as opposed to per-directory patterns, which are anchored at the merge-file's directory). If you think of the transfer as a subtree of names that are being sent from sender to receiver, the transfer-root is where the tree starts to be duplicated in the destination directory. This root governs where patterns that start with a / match.

Because the matching is relative to the transfer-root, changing the trailing slash on a source path or changing your use of the **--relative** option affects the path you need to use in your matching (in addition to changing how much of the file tree is duplicated on the destination host). The following examples demonstrate this.

Let's say that we want to match two source files, one with an absolute path of "/home/me/foo/bar", and one with a path of "/home/you/bar/baz". Here is how the various command choices differ for a 2-source transfer:

Example cmd: rsync -a /home/me /home/you /dest
+/- pattern: /me/foo/bar
+/- pattern: /you/bar/baz
Target file: /dest/me/foo/bar
Target file: /dest/you/bar/baz

Example cmd: rsync -a /home/me/ /home/you/ /dest
+/- pattern: /foo/bar (note missing "me")
+/- pattern: /bar/baz (note missing "you")
Target file: /dest/foo/bar
Target file: /dest/bar/baz

Example cmd: rsync -a --relative /home/me/ /home/you /dest
+/- pattern: /home/me/foo/bar (note full path)
+/- pattern: /home/you/bar/baz (ditto)
Target file: /dest/home/me/foo/bar
Target file: /dest/home/you/bar/baz

Example cmd: cd /home; rsync -a --relative me/foo you/ /dest
+/- pattern: /me/foo/bar (starts at specified path)
+/- pattern: /you/bar/baz (ditto)
Target file: /dest/me/foo/bar
Target file: /dest/you/bar/baz

The easiest way to see what name you should filter is to just look at the output when using **--verbose** and put a / in front of the name (use the **--dry-run** option if you're not yet ready to copy any files).

## PER-DIRECTORY RULES AND DELETE

Without a delete option, per-directory rules are only relevant on the sending side, so you can feel free to exclude the merge files themselves without affecting the transfer. To make this easy, the 'e' modifier adds this exclude for you, as seen in these two equivalent commands:

```
rsync -av --filter=': .excl' --exclude=.excl host:src/dir /dest
rsync -av --filter=':e .excl' host:src/dir /dest
```

However, if you want to do a delete on the receiving side AND you want some files to be excluded from being deleted, you'll need to be sure that the receiving side knows what files to exclude. The easiest way is to include the per-directory merge files in the transfer and use **--delete-after**, because this ensures that the receiving side gets all the same exclude rules as the sending side before it tries to delete anything:

```
rsync -avF --delete-after host:src/dir /dest
```

However, if the merge files are not a part of the transfer, you'll need to either specify some global exclude rules (i.e. specified on the command line), or you'll need to maintain your own per-directory merge files on the receiving side. An example of the first is this (assume that the remote .rules files exclude themselves):

```
rsync -av --filter=': .rules' --filter='. /my/extra.rules'
  --delete host:src/dir /dest
```

In the above example the extra.rules file can affect both sides of the transfer, but (on the sending side) the rules are subservient to the rules merged from the .rules files because they were specified after the per-directory merge rule.

In one final example, the remote side is excluding the .rsync-filter files from the transfer, but we want to use our own .rsync-filter files to control what gets deleted on the receiving side. To do this we must specifically exclude the per-directory merge files (so that they don't get deleted) and then put rules into the local files to control what else should not get deleted. Like one of these commands:

```
rsync -av --filter=':e /.rsync-filter' --delete \
   host:src/dir /dest
rsync -avFF --delete host:src/dir /dest
```

## BATCH MODE

Batch mode can be used to apply the same set of updates to many identical systems. Suppose one has a tree which is replicated on a number of hosts. Now suppose some changes have been made to this source tree and those changes need to be propagated to the other hosts. In order to do this using batch mode, rsync is run with the write-batch option to apply the changes made to the source tree to one of the destination trees. The write-batch option causes the rsync client to store in a "batch file" all the information needed to repeat this operation against other, identical destination trees.

To apply the recorded changes to another destination tree, run rsync with the read-batch option, specifying the name of the same batch file, and the destination tree. Rsync updates the destination tree using the information stored in the batch file.

For convenience, one additional file is creating when the write-batch option is used. This file's name is created by appending ".sh" to the batch filename. The .sh file contains a command-line suitable for updating a destination tree using that batch file. It can be executed using a Bourne(-like) shell, optionally passing in an alternate destination tree pathname which is then used instead of the original path. This is useful when the destination tree path differs from the original destination tree path.

Generating the batch file once saves having to perform the file status, checksum, and data block generation more than once when updating multiple destination trees. Multicast transport protocols can be used to transfer the batch update files in parallel to many hosts at once, instead of sending the same data to every host individually.

Examples:

```
$ rsync --write-batch=foo -a host:/source/dir/ /adest/dir/
$ scp foo* remote:
$ ssh remote ./foo.sh /bdest/dir/
```

```
$ rsync --write-batch=foo -a /source/dir/ /adest/dir/
$ ssh remote rsync --read-batch=- -a /bdest/dir/ <foo
```

In these examples, rsync is used to update /adest/dir/ from /source/dir/ and the information to repeat this operation is stored in "foo" and "foo.sh". The host "remote" is then updated with the batched data going into the directory /bdest/dir. The differences between the two examples reveals some of the flexibility you have in how you deal with batches:

- The first example shows that the initial copy doesn't have to be local -- you can push or pull data to/from a remote host using either the remote-shell syntax or rsync daemon syntax, as desired.
- The first example uses the created "foo.sh" file to get the right rsync options when running the read-batch command on the remote host.
- The second example reads the batch data via standard input so that the batch file doesn't need to be copied to the remote machine first. This example avoids the foo.sh script because it needed to use a modified **--read-batch** option, but you could edit the script file if you wished to make use of it (just be sure that no other option is trying to use standard input, such as the "**--exclude-from=-**" option).

Caveats:

The read-batch option expects the destination tree that it is updating to be identical to the destination tree that was used to create the batch update fileset. When a difference between the destination trees is encountered the update might be discarded with a warning (if the file appears to be up-to-date already) or the file-update may be attempted and then, if the file fails to verify, the update discarded with an error. This means that it should be safe to re-run a read-batch operation if the command got interrupted. If you wish to force the batched-update to always be attempted regardless of the file's size and date, use the **-I** option (when reading the batch). If an error occurs, the destination tree will probably be in a partially updated state. In that case, rsync can be used in its regular (non-batch) mode of operation to fix up the destination tree.

The rsync version used on all destinations must be at least as new as the one used to generate the batch file. Rsync will die with an error if the protocol version in the batch file is too new for the batch-reading rsync to handle. See also the **--protocol** option for a way to have the creating rsync generate a batch file that an older rsync can understand. (Note that batch files changed format in version 2.6.3, so mixing versions older than that with newer versions will not work.)

When reading a batch file, rsync will force the value of certain options to match the data in the batch file if you didn't set them to the same as the batch-writing command. Other options can (and should) be changed. For instance **--write-batch** changes to **--read-batch**, **--files-from** is dropped, and the **--filter**/**--include**/**--exclude** options are not needed unless one of the **--delete** options is specified.

The code that creates the BATCH.sh file transforms any filter/include/exclude options into a single list that is appended as a "here" document to the shell script file. An advanced user can use this to modify the exclude list if a change in what gets deleted by **--delete** is desired. A normal user can ignore this detail and just use the shell script as an easy way to run the appropriate **--read-batch** command for the batched data.

The original batch mode in rsync was based on "rsync+", but the latest version uses a new implementation.

## SYMBOLIC LINKS

Three basic behaviors are possible when rsync encounters a symbolic link in the source directory.

By default, symbolic links are not transferred at all. A message "skipping non-regular" file is emitted for any symlinks that exist.

If **--links** is specified, then symlinks are recreated with the same target on the destination. Note that **--archive** implies **--links**.

If **--copy-links** is specified, then symlinks are "collapsed" by copying their referent, rather than the symlink.

rsync also distinguishes "safe" and "unsafe" symbolic links. An example where this might be used is a web site mirror that wishes ensure the rsync module they copy does not include symbolic links to **/etc/passwd** in the public section of the site. Using **--copy-unsafe-links** will cause any links to be copied as the file they point to on the destination. Using **--safe-links** will cause unsafe links to be omitted altogether. (Note that

you must specify **--links** for **--safe-links** to have any effect.)

Symbolic links are considered unsafe if they are absolute symlinks (start with /), empty, or if they contain enough "**..**" components to ascend from the directory being copied.

Here's a summary of how the symlink options are interpreted. The list is in order of precedence, so if your combination of options isn't mentioned, use the first line that is a complete subset of your options:

**--copy-links**
Turn all symlinks into normal files (leaving no symlinks for any other options to affect).

**--links --copy-unsafe-links**
Turn all unsafe symlinks into files and duplicate all safe symlinks.

**--copy-unsafe-links**
Turn all unsafe symlinks into files, noisily skip all safe symlinks.

**--links --safe-links**
Duplicate safe symlinks and skip unsafe ones.

**--links** Duplicate all symlinks.

## DIAGNOSTICS

rsync occasionally produces error messages that may seem a little cryptic. The one that seems to cause the most confusion is "protocol version mismatch -- is your shell clean?".

This message is usually caused by your startup scripts or remote shell facility producing unwanted garbage on the stream that rsync is using for its transport. The way to diagnose this problem is to run your remote shell like this:

```
ssh remotehost /bin/true > out.dat
```

then look at out.dat. If everything is working correctly then out.dat should be a zero length file. If you are getting the above error from rsync then you will probably find that out.dat contains some text or data. Look at the contents and try to work out what is producing it. The most common cause is incorrectly configured shell startup scripts (such as .cshrc or .profile) that contain output statements for non-interactive logins.

If you are having trouble debugging filter patterns, then try specifying the **-vv** option. At this level of verbosity rsync will show why each individual file is included or excluded.

## EXIT VALUES

**0** Success

**1** Syntax or usage error

**2** Protocol incompatibility

**3** Errors selecting input/output files, dirs

**4** Requested action not supported: an attempt was made to manipulate 64-bit files on a platform that cannot support them; or an option was specified that is supported by the client and not by the server.

**5** Error starting client-server protocol

**6** Daemon unable to append to log-file

**10** Error in socket I/O

**11** Error in file I/O

**12** Error in rsync protocol data stream

**13** Errors with program diagnostics

**14** Error in IPC code

**20** Received SIGUSR1 or SIGINT

**21** Some error returned by waitpid()

**22** Error allocating core memory buffers

**23** Partial transfer due to error

**24** Partial transfer due to vanished source files

**25** The --max-delete limit stopped deletions

**30** Timeout in data send/receive

## ENVIRONMENT VARIABLES

### CVSIGNORE

The CVSIGNORE environment variable supplements any ignore patterns in .cvsignore files. See the **--cvs-exclude** option for more details.

### RSYNC_RSH

The RSYNC_RSH environment variable allows you to override the default shell used as the transport for rsync. Command line options are permitted after the command name, just as in the **-e** option.

### RSYNC_PROXY

The RSYNC_PROXY environment variable allows you to redirect your rsync client to use a web proxy when connecting to a rsync daemon. You should set RSYNC_PROXY to a hostname:port pair.

### RSYNC_PASSWORD

Setting RSYNC_PASSWORD to the required password allows you to run authenticated rsync connections to an rsync daemon without user intervention. Note that this does not supply a password to a shell transport such as ssh.

### USER or LOGNAME

The USER or LOGNAME environment variables are used to determine the default username sent to an rsync daemon. If neither is set, the username defaults to "nobody".

### HOME

The HOME environment variable is used to find the user's default .cvsignore file.

## FILES

/etc/rsyncd.conf or rsyncd.conf

## SEE ALSO

rsyncd.conf(5)

## BUGS

times are transferred as unix time_t values

When transferring to FAT filesystems rsync may re-sync unmodified files. See the comments on the **--modify-window** option.

file permissions, devices, etc. are transferred as native numerical values

see also the comments on the **--delete** option

Please report bugs! See the website at http://rsync.samba.org/

## VERSION

This man page is current for version 2.6.6 of rsync.

## CREDITS

rsync is distributed under the GNU public license. See the file COPYING for details.

A WEB site is available at http://rsync.samba.org/. The site includes an FAQ-O-Matic which may cover questions unanswered by this manual page.

The primary ftp site for rsync is ftp://rsync.samba.org/pub/rsync.

We would be delighted to hear from you if you like this program.

This program uses the excellent zlib compression library written by Jean-loup Gailly and Mark Adler.

## THANKS

Thanks to Richard Brent, Brendan Mackay, Bill Waite, Stephen Rothwell and David Bell for helpful suggestions, patches and testing of rsync. I've probably missed some people, my apologies if I have.

Especial thanks also to: David Dykstra, Jos Backus, Sebastian Krahmer, Martin Pool, Wayne Davison, J.W. Schultz.

## AUTHOR

rsync was originally written by Andrew Tridgell and Paul Mackerras. Many people have later contributed to it.

Mailing lists for support and development are available at http://lists.samba.org