NAME

rsync — a fast, versatile, remote (and local) file-copying tool

SYNOPSIS

Local: rsync [OPTION...] SRC... [DEST]

Access via remote shell:
Pull: rsync [OPTION...] [USER@]HOST:SRC... [DEST]
Push: rsync [OPTION...] SRC... [USER@]HOST:DEST

Access via rsync daemon:
Pull: rsync [OPTION...] [USER@]HOST::SRC... [DEST]
rsync [OPTION...] rsync://[USER@]HOST[:PORT]/SRC... [DEST]
Push: rsync [OPTION...] SRC... [USER@]HOST::DEST
rsync [OPTION...] SRC... rsync://[USER@]HOST[:PORT]/DEST

Usages with just one SRC arg and no DEST arg will list the source files instead of copying.

DESCRIPTION

Rsync is a fast and extraordinarily versatile file copying tool. It can copy locally, to/from another host over any remote shell, or to/from a remote rsync daemon. It offers a large number of options that control every aspect of its behavior and permit very flexible specification of the set of files to be copied. It is famous for its delta-transfer algorithm, which reduces the amount of data sent over the network by sending only the differences between the source files and the existing files in the destination. Rsync is widely used for backups and mirroring and as an improved copy command for everyday use.

Rsync finds files that need to be transferred using a "quick check" algorithm (by default) that looks for files that have changed in size or in last-modified time. Any changes in the other preserved attributes (as requested by options) are made on the destination file directly when the quick check indicates that the file's data does not need to be updated.

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 super-user 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 single source arg is specified without a destination, the 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).

Rsync refers to the local side as the “client” and the remote side as the “server”. Don’t confuse “server” with an rsync daemon — a daemon is always a server, but a server can be either a daemon or a remote-shell spawned process.

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.

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.

Finally, you can list all the (listable) modules available from a particular rsync daemon by leaving off the module name:

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

See the following section for more details.

ADVANCED USAGE

The syntax for requesting multiple files from a remote host is done by specifying additional remote-host args in the same style as the first, or with the hostname omitted. For instance, all these work:

```
rsync -av host:file1 :file2 host:file{3,4} /dest/
rsync -av host::modname/file{1,2} host::modname/file3 /dest/
rsync -av host::modname/file1 ::modname/file{3,4}
```

Older versions of rsync required using quoted spaces in the SRC, like these examples:

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

This word-splitting still works (by default) in the latest rsync, but is not as easy to use as the first method.

If you need to transfer a filename that contains whitespace, you can either specify the **--protect-args** (**-s**) option, or you'll need to escape the whitespace in a way that the remote shell will understand. For instance:

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

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 of the "path" is actually 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.
- you must not specify the **--rsh** (**-e**) option.

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.

You may also establish a daemon connection using a program as a proxy by setting the environment variable RSYNC_CONNECT_PROG to the commands you wish to run in place of making a direct socket connection. The string may contain the escape "%H" to represent the hostname specified in the rsync command (so use "%%" if you need a single "%" in your string). For example:

```
export RSYNC_CONNECT_PROG='ssh proxyhost nc %H 873'
rsync −av targethost1::module/src/ /dest/
rsync −av rsync://targethost2/module/src/ /dest/
```

The command specified above uses ssh to run nc (netcat) on a proxyhost, which forwards all data to port 873 (the rsync daemon) on the targethost (%H).

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, as in this example that uses the short version of the **−−rsh** option:

```
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
    −−no−motd               suppress daemon-mode MOTD (see caveat)
−c, −−checksum              skip based on checksum, not mod-time & size
−a, −−archive               archive mode; equals −rlptgoD (no −H,−A,−X)
    −−no−OPTION             turn off an implied OPTION (e.g. −−no−D)
−r, −−recursive             recurse into directories
−R, −−relative              use relative path names
    −−no−implied−dirs       don't send implied dirs with −−relative
−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
    −−append                append data onto shorter files
    −−append−verify         −−append w/old data in file checksum
−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
−k, −−copy−dirlinks         transform symlink to dir into referent dir
−K, −−keep−dirlinks         treat symlinked dir on receiver as dir
−H, −−hard−links            preserve hard links
−p, −−perms                 preserve permissions
−E, −−executability         preserve executability
    −−chmod=CHMOD           affect file and/or directory permissions
−A, −−acls                  preserve ACLs (implies −p)
−X, −−xattrs                preserve extended attributes
−o, −−owner                 preserve owner (super-user only)
```

−g, −−group preserve group
−−devices preserve device files (super-user only)
−−specials preserve special files
−D same as −−devices −−specials
−t, −−times preserve modification times
−O, −−omit−dir−times omit directories from −−times
−−super receiver attempts super-user activities
−−fake−super store/recover privileged attrs using xattrs
−S, −−sparse handle sparse files efficiently
−n, −−dry−run perform a trial run with no changes made
−W, −−whole−file copy files whole (w/o delta-xfer 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 skip creating new files on receiver
−−ignore−existing skip updating files that exist on receiver
−−remove−source−files sender removes synchronized files (non-dir)
−−del an alias for −−delete−during
−−delete delete extraneous files from dest dirs
−−delete−before receiver deletes before transfer (default)
−−delete−during receiver deletes during xfer, not before
−−delete−delay find deletions during, delete after
−−delete−after receiver deletes after transfer, not before
−−delete−excluded also delete excluded files from dest dirs
−−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
−−min−size=SIZE don’t transfer any file smaller 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
−m, −−prune−empty−dirs prune empty directory chains from file-list
−−numeric−ids don’t map uid/gid values by user/group name
−−timeout=SECONDS set I/O timeout in seconds
−−contimeout=SECONDS set daemon connection 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
−−compress−level=NUM explicitly set compression level
−−skip−compress=LIST skip compressing files with suffix in LIST
−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
-s, --protect-args          no space-splitting; wildcard chars only
    --address=ADDRESS       bind address for outgoing socket to daemon
    --port=PORT             specify double-colon alternate port number
    --sockopts=OPTIONS      specify custom TCP options
    --blocking-io           use blocking I/O for the remote shell
    --stats                 give some file-transfer stats
-8, --8-bit-output          leave high-bit chars unescaped in output
-h, --human-readable        output numbers in a human-readable format
    --progress              show progress during transfer
-P                          same as --partial --progress
-i, --itemize-changes       output a change-summary for all updates
    --out-format=FORMAT     output updates using the specified FORMAT
    --log-file=FILE         log what we're doing to the specified FILE
    --log-file-format=FMT   log updates using the specified FMT
    --password-file=FILE    read daemon-access 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
    --iconv=CONVERT_SPEC    request charset conversion of filenames
    --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 (see below for -h comment)
```

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
    --log-file=FILE         override the "log file" setting
    --log-file-format=FMT   override the "log format" setting
    --sockopts=OPTIONS      specify custom TCP options
-v, --verbose               increase verbosity
-4, --ipv4                  prefer IPv4
-6, --ipv6                  prefer IPv6
-h, --help                  show this help (if used after --daemon)
```

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.

−−help Print a short help page describing the options available in rsync and exit. For backward-compatibility with older versions of rsync, the help will also be output if you use the **−h** option without any other args.

−−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** options will give you information on what files are being skipped and slightly more information at the end. More than two **−v** options 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 **−−out−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 **−−out−format** setting), the output (on the client) increases to mention all items that are changed in any way. See the **−−out−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 option is useful when invoking rsync from cron.

−−no−motd
This option affects the information that is output by the client at the start of a daemon transfer. This suppresses the message-of-the-day (MOTD) text, but it also affects the list of modules that the daemon sends in response to the "rsync host::" request (due to a limitation in the rsync protocol), so omit this option if you want to request the list of modules from the daemon.

−I, −−ignore−times
Normally rsync will skip any files that are already the same size and have the same modification timestamp. This option turns off this "quick check" behavior, causing all files to be updated.

−−size−only
This modifies rsync's "quick check" algorithm for finding files that need to be transferred, changing it from the default of transferring files with either a changed size or a changed last-modified time to just looking for files that have changed in size. 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 changes the way rsync checks if the files have been changed and are in need of a transfer. Without this option, rsync uses a "quick check" that (by default) checks if each file's size and time of last modification match between the sender and receiver. This option changes this to compare a 128-bit checksum for each file that has a matching size. Generating the checksums means that both sides will expend a lot of disk I/O reading all the data in the files in the transfer (and this is prior to any reading that will be done to transfer changed files), so this can slow things down significantly.

The sending side generates its checksums while it is doing the file-system scan that builds the list of the available files. The receiver generates its checksums when it is scanning for changed files, and will checksum any file that has the same size as the corresponding sender's file: files with either a changed size or a changed checksum are selected for transfer.

Note that rsync always verifies that each *transferred* file was correctly reconstructed on the receiving side by checking a whole-file checksum that is generated as the file is transferred, but that automatic after-the-transfer verification has nothing to do with this option's before-the-transfer "Does this file need to be updated?" check.

For protocol 30 and beyond (first supported in 3.0.0), the checksum used is MD5. For older protocols, the checksum used is MD4.

−a, −−archive

This is equivalent to **−rlptgoD**. It is a quick way of saying you want recursion and want to preserve almost everything (with −H being a notable omission). The only exception to the above equivalence is when **−−files−from** is 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**.

−−no−OPTION

You may turn off one or more implied options by prefixing the option name with "no−". Not all options may be prefixed with a "no−": only options that are implied by other options (e.g. **−−no−D**, **−−no−perms**) or have different defaults in various circumstances (e.g. **−−no−whole−file**, **−−no−blocking−io**, **−−no−dirs**). You may specify either the short or the long option name after the "no−" prefix (e.g. **−−no−R** is the same as **−−no−relative**).

For example: if you want to use **−a** (**−−archive**) but don't want **−o** (**−−owner**), instead of converting **−a** into **−rlptgD**, you could specify **−a −−no−o** (or **−a −−no−owner**).

The order of the options is important: if you specify **−−no−r −a**, the **−r** option would end up being turned on, the opposite of **−a −−no−r**. Note also that the side-effects of the **−−files−from** option are NOT positional, as it affects the default state of several options and slightly changes the meaning of **−a** (see the **−−files−from** option for more details).

−r, −−recursive

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

Beginning with rsync 3.0.0, the recursive algorithm used is now an incremental scan that uses much less memory than before and begins the transfer after the scanning of the first few directories have been completed. This incremental scan only affects our recursion algorithm, and does not change a non-recursive transfer. It is also only possible when both ends of the transfer are at least version 3.0.0.

Some options require rsync to know the full file list, so these options disable the incremental recursion mode. These include: **−−delete−before**, **−−delete−after**, **−−prune−empty−dirs**, and **−−delay−updates**. Because of this, the default delete mode when you specify **−−delete** is now **−−delete−during** when both ends of the connection are at least 3.0.0 (use **−−del** or **−−delete−during** to request this improved deletion mode explicitly). See also the **−−delete−delay** option that is a better choice than using **−−delete−after**.

Incremental recursion can be disabled using the **−−no−inc−recursive** option or its shorter **−−no−i−r** alias.

−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 this command:

```
rsync -av /foo/bar/baz.c remote:/tmp/
```

... this would create a file named baz.c in /tmp/ on the remote machine. If instead you used

```
rsync -avR /foo/bar/baz.c remote:/tmp/
```

then a file named /tmp/foo/bar/baz.c would be created on the remote machine, preserving its full path. These extra path elements are called "implied directories" (i.e. the "foo" and the "foo/bar" directories in the above example).

Beginning with rsync 3.0.0, rsync always sends these implied directories as real directories in the file list, even if a path element is really a symlink on the sending side. This prevents some really unexpected behaviors when copying the full path of a file that you didn't realize had a symlink in its path. If you want to duplicate a server-side symlink, include both the symlink via its path, and referent directory via its real path. If you're dealing with an older rsync on the sending side, you may need to use the **--no-implied-dirs** option.

It is also possible to limit the amount of path information that is sent as implied directories for each path you specify. With a modern rsync on the sending side (beginning with 2.6.7), you can insert a dot and a slash into the source path, like this:

```
rsync -avR /foo/./bar/baz.c remote:/tmp/
```

That would create /tmp/bar/baz.c on the remote machine. (Note that the dot must be followed by a slash, so "/foo/." would not be abbreviated.) (2) For older rsync versions, you would need to use a chdir to limit the source path. For example, when pushing files:

```
(cd /foo; rsync -avR bar/baz.c remote:/tmp/)
```

(Note that the parens put the two commands into a sub-shell, so that the "cd" command doesn't remain in effect for future commands.) If you're pulling files from an older rsync, use this idiom (but only for a non-daemon transfer):

```
rsync -avR --rsync-path="cd /foo; rsync" \
    remote:bar/baz.c /tmp/
```

--no-implied-dirs

This option affects the default behavior of the **--relative** option. When it is specified, the attributes of the implied directories from the source names are not included in the transfer. This means that the corresponding path elements on the destination system are left unchanged if they exist, and any missing implied directories are created with default attributes. This even allows these implied path elements to have big differences, such as being a symlink to a directory on the receiving side.

For instance, if a command-line arg or a files-from entry told rsync to transfer the file "path/foo/file", the directories "path" and "path/foo" are implied when **--relative** is used. If "path/foo" is a symlink to "bar" on the destination system, the receiving rsync would ordinarily delete "path/foo", recreate it as a directory, and receive the file into the new directory. With **--no-implied-dirs**, the receiving rsync updates "path/foo/file" using the existing path elements, which means that the file ends up being created in "path/bar". Another way to accomplish this link preservation is to use the **--keep-dirlinks** option (which will also affect symlinks to directories in the rest of the transfer).

When pulling files from an rsync older than 3.0.0, you may need to use this option if the sending side has a symlink in the path you request and you wish the implied directories to be transferred as normal directories.

−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**, (1) the **−−omit−dir−times** option will be implied, and (2) if **−−delete** is also in effect (without **−−delete−excluded**), rsync will add a "protect" filter-rule for the backup suffix to the end of all your existing excludes (e.g. **−f "P *~"**). This will prevent previously backed-up files from being deleted. Note that if you are supplying your own filter rules, you may need to manually insert your own exclude/protect rule 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 inclusion/exclusion of '*', the auto-added rule would never be reached).

−−backup−dir=DIR

In combination with the **−−backup** option, this tells rsync to store all backups in the specified directory on the receiving side. This can be used 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 modification time equal to the source file's, it will be updated if the sizes are different.)

Note that this does not affect the copying of symlinks or other special files. Also, 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 where the destination has a file, the transfer would occur regardless of the timestamps.

This option is a transfer rule, not an exclude, so it doesn't affect the data that goes into the file-lists, and thus it doesn't affect deletions. It just limits the files that the receiver requests to be transferred.

−−inplace

This option changes how rsync transfers a file when the file's data needs to be updated: instead of the default method of creating a new copy of the file and moving it into place when it is complete, rsync instead writes the updated data directly to the destination file.

This has several effects: (1) in-use binaries cannot be updated (either the OS will prevent this from happening, or binaries that attempt to swap-in their data will misbehave or crash), (2) the file's data will be in an inconsistent state during the transfer, (3) a file's data may be left in an inconsistent state after the transfer if the transfer is interrupted or if an update fails, (4) a file that does not have write permissions can not be updated, and (5) the efficiency of rsync's delta-transfer algorithm may be reduced if some data in the destination file is overwritten before it can be copied to a position later in the file (one exception to this is if you combine this option with **−−backup**, since rsync is smart enough to use the backup file as the basis file for the transfer).

WARNING: you should not use this option to update files that are being accessed by others, so be careful when choosing to use this for a copy.

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**.

−−append
This causes rsync to update a file by appending data onto the end of the file, which presumes that the data that already exists on the receiving side is identical with the start of the file on the sending side. If a file needs to be transferred and its size on the receiver is the same or longer than the size on the sender, the file is skipped. This does not interfere with the updating of a file's non-content attributes (e.g. permissions, ownership, etc.) when the file does not need to be transferred, nor does it affect the updating of any non-regular files. Implies **−−inplace**, but does not conflict with **−−sparse** (since it is always extending a file's length).

−−append−verify
This works just like the **−−append** option, but the existing data on the receiving side is included in the full-file checksum verification step, which will cause a file to be resent if the final verification step fails (rsync uses a normal, non-appending **−−inplace** transfer for the resend).

Note: prior to rsync 3.0.0, the **−−append** option worked like **−−append−verify**, so if you are interacting with an older rsync (or the transfer is using a protocol prior to 30), specifying either append option will initiate an **−−append−verify** transfer.

−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 name specified is "." or ends with a trailing slash (e.g. ".", "dir/.", "dir/", etc.). Without this option or the **−−recursive** option, rsync will skip all directories it encounters (and output a message to that effect for each one). If you specify both **−−dirs** and **−−recursive**, **−−recursive** takes precedence.

The **−−dirs** option is implied by the **−−files−from** option or the **−−list−only** option (including an implied **−−list−only** usage) if **−−recursive** wasn't specified (so that directories are seen in the listing). Specify **−−no−dirs** (or **−−no−d**) if you want to turn this off.

There is also a backward-compatibility helper option, **−−old−dirs** (or **−−old−d**) that tells rsync to use a hack of "−r −−exclude='/*/*'" to get an older rsync to list a single directory without recursing.

−l, −−links
When symlinks are encountered, recreate the symlink on the destination.

−L, −−copy−links
When symlinks are encountered, the item 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. This option has no additional effect if **−−copy−links** was also specified.

−−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.

−k, −−copy−dirlinks
This option causes the sending side to treat a symlink to a directory as though it were a real directory. This is useful if you don't want symlinks to non-directories to be affected, as they would be using **−−copy−links**.

Without this option, if the sending side has replaced a directory with a symlink to a directory, the receiving side will delete anything that is in the way of the new symlink, including a directory hierarchy (as long as **−−force** or **−−delete** is in effect).

See also **−−keep−dirlinks** for an analogous option for the receiving side.

−K, −−keep−dirlinks
This option causes the receiving side to treat a symlink to a directory as though it were a real directory, but only if it matches a real directory from the sender. Without this option, the receiver's symlink would be deleted and replaced with a real directory.

For example, suppose you transfer a directory "foo" that contains a file "file", but "foo" is a symlink to directory "bar" on the receiver. Without **−−keep−dirlinks**, the receiver deletes symlink "foo", recreates it as a directory, and receives the file into the new directory. With **−−keep−dirlinks**, the receiver keeps the symlink and "file" ends up in "bar".

One note of caution: if you use **−−keep−dirlinks**, you must trust all the symlinks in the copy! If it is possible for an untrusted user to create their own symlink to any directory, the user could then (on a subsequent copy) replace the symlink with a real directory and affect the content of whatever directory the symlink references. For backup copies, you are better off using something like a bind mount instead of a symlink to modify your receiving hierarchy.

See also **−−copy−dirlinks** for an analogous option for the sending side.

−H, −−hard−links
This tells rsync to look for hard-linked files in the transfer and link together the corresponding files on the receiving side. Without this option, hard-linked files in the transfer are treated as though they were separate files.

When you are updating a non-empty destination, this option only ensures that files that are hard-linked together on the source are hard-linked together on the destination. It does NOT currently endeavor to break already existing hard links on the destination that do not exist between the source files. Note, however, that if one or more extra-linked files have content changes, they will become unlinked when updated (assuming you are not using the **−−inplace** option).

Note that rsync can only detect hard links between files that are inside the transfer set. If rsync updates a file that has extra hard-link connections to files outside the transfer, that linkage will be broken. If you are tempted to use the **−−inplace** option to avoid this breakage, be very careful that you know how your files are being updated so that you are certain that no unintended changes happen due to lingering hard links (and see the **−−inplace** option for more caveats).

If incremental recursion is active (see **−−recursive**), rsync may transfer a missing hard-linked file before it finds that another link for that contents exists elsewhere in the hierarchy. This does not affect the accuracy of the transfer, just its efficiency. One way to avoid this is to disable incremental recursion using the **−−no−inc−recursive** option.

−p, −−perms
This option causes the receiving rsync to set the destination permissions to be the same as the source permissions. (See also the **−−chmod** option for a way to modify what rsync considers to be the source permissions.)

When this option is *off*, permissions are set as follows:

- Existing files (including updated files) retain their existing permissions, though the **−−executability** option might change just the execute permission for the file.

- New files get their "normal" permission bits set to the source file's permissions masked with the receiving directory's default permissions (either the receiving process's umask, or the permissions specified via the destination directory's default ACL), and their special permission bits disabled except in the case where a new directory inherits a setgid bit from its parent directory.

Thus, when **--perms** and **--executability** are both disabled, rsync's behavior is the same as that of other file-copy utilities, such as **cp**(1) and **tar**(1).

In summary: to give destination files (both old and new) the source permissions, use **--perms**. To give new files the destination-default permissions (while leaving existing files unchanged), make sure that the **--perms** option is off and use **--chmod=ugo=rwX** (which ensures that all non-masked bits get enabled). If you'd care to make this latter behavior easier to type, you could define a popt alias for it, such as putting this line in the file ~/.popt (the following defines the **-Z** option, and includes --no-g to use the default group of the destination dir):

```
rsync alias -Z --no-p --no-g --chmod=ugo=rwX
```

You could then use this new option in a command such as this one:

```
rsync -avZ src/ dest/
```

(Caveat: make sure that **-a** does not follow **-Z**, or it will re-enable the two "--no-*" options mentioned above.)

The preservation of the destination's setgid bit on newly-created directories when **--perms** is off was added in rsync 2.6.7. Older rsync versions erroneously preserved the three special permission bits for newly-created files when **--perms** was off, while overriding the destination's setgid bit setting on a newly-created directory. Default ACL observance was added to the ACL patch for rsync 2.6.7, so older (or non-ACL-enabled) rsyncs use the umask even if default ACLs are present. (Keep in mind that it is the version of the receiving rsync that affects these behaviors.)

-E, --executability

This option causes rsync to preserve the executability (or non-executability) of regular files when **--perms** is not enabled. A regular file is considered to be executable if at least one 'x' is turned on in its permissions. When an existing destination file's executability differs from that of the corresponding source file, rsync modifies the destination file's permissions as follows:

- To make a file non-executable, rsync turns off all its 'x' permissions.
- To make a file executable, rsync turns on each 'x' permission that has a corresponding 'r' permission enabled.

If **--perms** is enabled, this option is ignored.

-A, --acls

This option causes rsync to update the destination ACLs to be the same as the source ACLs. The option also implies **--perms**.

The source and destination systems must have compatible ACL entries for this option to work properly. See the **--fake-super** option for a way to backup and restore ACLs that are not compatible.

-X, --xattrs

This option causes rsync to update the remote extended attributes to be the same as the local ones.

For systems that support extended-attribute namespaces, a copy being done by a super-user copies all namespaces except system.*. A normal user only copies the user.* namespace. To be able to

backup and restore non-user namespaces as a normal user, see the **--fake-super** option.

--chmod

This option tells rsync to apply one or more comma-separated "chmod" strings to the permission of the files in the transfer. The resulting value is treated as though it was the permissions that the sending side supplied for the file, which means that this option can seem to have no effect on existing files if **--perms** is not enabled.

In addition to the normal parsing rules specified in the **chmod**(1) manpage, you can specify an item that should only apply to a directory by prefixing it with a 'D', or specify an item that should only apply to a file by prefixing it with a 'F'. For example:

--chmod=Dg+s,ug+w,Fo-w,+X

It is also legal to specify multiple **--chmod** options, as each additional option is just appended to the list of changes to make.

See the **--perms** and **--executability** options for how the resulting permission value can be applied to the files in the transfer.

-o, --owner

This option causes rsync to set the owner of the destination file to be the same as the source file, but only if the receiving rsync is being run as the super-user (see also the **--super** and **--fake-super** options). Without this option, the owner of new and/or transferred files are set to the invoking user on the receiving side.

The preservation of ownership will associate matching names by default, but may fall back to using the ID number in some circumstances (see also 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 (or if **--no-super** was specified), only groups that the invoking user on the receiving side is a member of will be preserved. Without this option, the group is set to the default group of the invoking user on the receiving side.

The preservation of group information will associate matching names by default, but may fall back to using the ID number in some circumstances (see also the **--numeric-ids** option for a full discussion).

--devices

This option causes rsync to transfer character and block device files to the remote system to recreate these devices. This option has no effect if the receiving rsync is not run as the super-user (see also the **--super** and **--fake-super** options).

--specials

This option causes rsync to transfer special files such as named sockets and fifos.

-D The **-D** option is equivalent to **--devices --specials**.

-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 rsync's delta-transfer 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**.

−−super

This tells the receiving side to attempt super-user activities even if the receiving rsync wasn't run by the super-user. These activities include: preserving users via the **−−owner** option, preserving all groups (not just the current user's groups) via the **−−groups** option, and copying devices via the **−−devices** option. This is useful for systems that allow such activities without being the super-user, and also for ensuring that you will get errors if the receiving side isn't being run as the super-user. To turn off super-user activities, the super-user can use **−−no−super**.

−−fake−super

When this option is enabled, rsync simulates super-user activities by saving/restoring the privileged attributes via special extended attributes that are attached to each file (as needed). This includes the file's owner and group (if it is not the default), the file's device info (device & special files are created as empty text files), and any permission bits that we won't allow to be set on the real file (e.g. the real file gets u-s,g-s,o-t for safety) or that would limit the owner's access (since the real super-user can always access/change a file, the files we create can always be accessed/changed by the creating user). This option also handles ACLs (if **−−acls** was specified) and non-user extended attributes (if **−−xattrs** was specified).

This is a good way to backup data without using a super-user, and to store ACLs from incompatible systems.

The **−−fake−super** option only affects the side where the option is used. To affect the remote side of a remote-shell connection, specify an rsync path:

```
rsync −av −−rsync−path="rsync −−fake−super" /src/ host:/dest/
```

Since there is only one "side" in a local copy, this option affects both the sending and receiving of files. You'll need to specify a copy using "localhost" if you need to avoid this, possibly using the "lsh" shell script (from the support directory) as a substitute for an actual remote shell (see **−−rsh**).

This option is overridden by both **−−super** and **−−no−super**.

See also the "fake super" setting in the daemon's rsyncd.conf file.

−S, −−sparse

Try to handle sparse files efficiently so they take up less space on the destination. Conflicts with **−−inplace** because it's not possible to overwrite data in a sparse fashion.

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.

−n, −−dry−run

This makes rsync perform a trial run that doesn't make any changes (and produces mostly the same output as a real run). It is most commonly used in combination with the **−v, −−verbose** and/or **−i, −−itemize−changes** options to see what an rsync command is going to do before one actually runs it.

The output of **−−itemize−changes** is supposed to be exactly the same on a dry run and a subsequent real run (barring intentional trickery and system call failures); if it isn't, that's a bug. Other output is the same to the extent practical, but may differ in some areas. Notably, a dry run does not send the actual data for file transfers, so **−−progress** has no effect, the "bytes sent", "bytes received", "literal data", and "matched data" statistics are too small, and the "speedup" value is equivalent to a run where no file transfers are needed.

−W, −−whole−file

With this option rsync's delta-transfer 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.

−x, −−one−file−system

This tells rsync to avoid crossing a filesystem boundary when recursing. This does not limit the user's ability to specify items to copy from multiple filesystems, just rsync's recursion through the hierarchy of each directory that the user specified, and also the analogous recursion on the receiving side during deletion. Also keep in mind that rsync treats a "bind" mount to the same device as being on the same filesystem.

If this option is repeated, rsync omits all mount-point directories from the copy. Otherwise, it includes an empty directory at each mount-point it encounters (using the attributes of the mounted directory because those of the underlying mount-point directory are inaccessible).

If rsync has been told to collapse symlinks (via **−−copy−links** or **−−copy−unsafe−links**), a symlink to a directory on another device is treated like a mount-point. Symlinks to non-directories are unaffected by this option.

−−existing, −−ignore−non−existing

This tells rsync to skip creating files (including directories) that do not exist yet on the destination. If this option is combined with the **−−ignore−existing** option, no files will be updated (which can be useful if all you want to do is delete extraneous files).

This option is a transfer rule, not an exclude, so it doesn't affect the data that goes into the filelists, and thus it doesn't affect deletions. It just limits the files that the receiver requests to be transferred.

−−ignore−existing

This tells rsync to skip updating files that already exist on the destination (this does *not* ignore existing directories, or nothing would get done). See also **−−existing**.

This option is a transfer rule, not an exclude, so it doesn't affect the data that goes into the filelists, and thus it doesn't affect deletions. It just limits the files that the receiver requests to be transferred.

This option can be useful for those doing backups using the **−−link−dest** option when they need to continue a backup run that got interrupted. Since a **−−link−dest** run is copied into a new directory hierarchy (when it is used properly), using **−−ignore existing** will ensure that the already-handled files don't get tweaked (which avoids a change in permissions on the hard-linked files). This does mean that this option is only looking at the existing files in the destination hierarchy itself.

−−remove−source−files

This tells rsync to remove from the sending side the files (meaning non-directories) that are a part of the transfer and have been successfully duplicated on the receiving side.

−−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 the 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).

Prior to rsync 2.6.7, this option would have no effect unless **--recursive** was enabled. Beginning with 2.6.7, deletions will also occur when **--dirs** (**-d**) is enabled, but only for directories whose contents are being copied.

This option can be dangerous if used incorrectly! It is a very good idea to first try a run using the **--dry-run** option (**-n**) to see what files are going to be deleted.

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 choose the **--delete-during** algorithm when talking to rsync 3.0.0 or newer, and the **--delete-before** algorithm when talking to an older rsync. See also **--delete-delay** and **--delete-after**.

--delete-before

Request that the file-deletions on the receiving side be done before the transfer starts. 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). It also forces rsync to use the old, non-incremental recursion algorithm that requires rsync to scan all the files in the transfer into memory at once (see **--recursive**).

--delete-during, --del

Request that the file-deletions on the receiving side be done incrementally as the transfer happens. The per-directory delete scan is done right before each directory is checked for updates, so it behaves like a more efficient **--delete-before**, including doing the deletions prior to any per-directory filter files being updated. This option was first added in rsync version 2.6.4. See **--delete** (which is implied) for more details on file-deletion.

--delete-delay

Request that the file-deletions on the receiving side be computed during the transfer (like **--delete-during**), and then removed after the transfer completes. This is useful when combined with **--delay-updates** and/or **--fuzzy**, and is more efficient than using **--delete-after** (but can behave differently, since **--delete-after** computes the deletions in a separate pass after all updates are done). If the number of removed files overflows an internal buffer, a temporary file will be created on the receiving side to hold the names (it is removed while open, so you shouldn't see it during the transfer). If the creation of the temporary file fails, rsync will try to fall back to using **--delete-after** (which it cannot do if **--recursive** is doing an incremental scan). 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. It also forces rsync to use the old, non-incremental recursion algorithm that requires rsync to scan all the files in the transfer into memory at once (see **--recursive**). 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 option tells rsync to delete a non-empty directory when it is to be replaced by a non-directory. This is only relevant if deletions are not active (see **−−delete** for details).

Note for older rsync versions: **−−force** used to still be required when using **−−delete−after**, and it used to be non-functional unless the **−−recursive** option was also enabled.

−−max−delete=NUM
This tells rsync not to delete more than NUM files or directories. If that limit is exceeded, a warning is output and rsync exits with an error code of 25 (new for 3.0.0).

Also new for version 3.0.0, you may specify **−−max−delete=0** to be warned about any extraneous files in the destination without removing any of them. Older clients interpreted this as "unlimited", so if you don't know what version the client is, you can use the less obvious **−−max−delete=−1** as a backward-compatible way to specify that no deletions be allowed (though older versions didn't warn when the limit was exceeded).

−−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 string to indicate a size multiplier, and may be a fractional value (e.g. "**−−max−size=1.5m**").

This option is a transfer rule, not an exclude, so it doesn't affect the data that goes into the file-lists, and thus it doesn't affect deletions. It just limits the files that the receiver requests to be transferred.

The suffixes are as follows: "K" (or "KiB") is a kibibyte (1024), "M" (or "MiB") is a mebibyte (1024*1024), and "G" (or "GiB") is a gibibyte (1024*1024*1024). If you want the multiplier to be 1000 instead of 1024, use "KB", "MB", or "GB". (Note: lower-case is also accepted for all values.) Finally, if the suffix ends in either "+1" or "−1", the value will be offset by one byte in the indicated direction.

Examples: −−max−size=1.5mb−1 is 1499999 bytes, and −−max−size=2g+1 is 2147483649 bytes.

−−min−size=SIZE
This tells rsync to avoid transferring any file that is smaller than the specified SIZE, which can help in not transferring small, junk files. See the **−−max−size** option for a description of SIZE and other information.

−B, −−block−size=BLOCKSIZE
This forces the block size used in rsync's delta-transfer 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. You must use spaces (not tabs or other whitespace) to separate the command and args from each other, and you can use single- and/or double-quotes to preserve spaces in an argument (but not backslashes). Note that doubling a single-quote inside a single-quoted string gives you a single-quote; likewise for double-quotes (though you need to pay attention to which quotes your shell is parsing and which quotes rsync is parsing). Some examples:

```
-e 'ssh -p 2234'
-e 'ssh -o "ProxyCommand nohup ssh firewall nc -w1 %h %p"'
```

(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" host: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 a similar algorithm to CVS to determine if a file should be ignored.

The exclude list is initialized to exclude the following items (these initial items are marked as perishable — see the FILTER RULES section):

```
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/ .git/ .bzr/
```

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. If the filter contains whitespace, be sure to quote it so that the shell gives the rule to rsync as a single argument. The text below also mentions that you can use an underscore to replace the space that separates a rule from its arg.

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='dir-merge /.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='exclude .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 related to the **−−exclude** option, but it specifies a FILE that contains exclude patterns (one per line). Blank lines in the 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 option is related to the **−−include** option, but it specifies a FILE that contains include patterns (one per line). Blank lines in the file and lines starting with ‘;’ or ‘#’ are ignored. 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** or **−−no−R** 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 (use **−−no−dirs** or **−−no−d** if you want to turn that off).

- The **--archive** (**-a**) option's behavior does not imply **--recursive** (**-r**), so specify it explicitly, if you want it.
- These side-effects change the default state of rsync, so the position of the **--files-from** option on the command-line has no bearing on how other options are parsed (e.g. **-a** works the same before or after **--files-from**, as does **--no-R** and all other options).

The filenames 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).

If the **--iconv** and **--protect-args** options are specified and the **--files-from** filenames are being sent from one host to another, the filenames will be translated from the sending host's charset to the receiving host's charset.

-s, --protect-args

This option sends all filenames and some options to the remote rsync without allowing the remote shell to interpret them. This means that spaces are not split in names, and any non-wildcard special characters are not translated (such as ~, $, ;, &, etc.). Wildcards are expanded on the remote host by rsync (instead of the shell doing it).

If you use this option with **--iconv**, the args will also be translated from the local to the remote character-set. The translation happens before wild-cards are expanded. See also the **--files-from** option.

-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 each temporary file in the same directory as the associated destination file.

This option is most often used when the receiving disk partition does not have enough free space to hold a copy of the largest file in the transfer. In this case (i.e. when the scratch directory is on a

different disk partition), rsync will not be able to rename each received temporary file over the top of the associated destination file, but instead must copy it into place. Rsync does this by copying the file over the top of the destination file, which means that the destination file will contain truncated data during this copy. If this were not done this way (even if the destination file were first removed, the data locally copied to a temporary file in the destination directory, and then renamed into place) it would be possible for the old file to continue taking up disk space (if someone had it open), and thus there might not be enough room to fit the new version on the disk at the same time.

If you are using this option for reasons other than a shortage of disk space, you may wish to combine it with the **--delay-updates** option, which will ensure that all copied files get put into subdirectories in the destination hierarchy, awaiting the end of the transfer. If you don't have enough room to duplicate all the arriving files on the destination partition, another way to tell rsync that you aren't overly concerned about disk space is to use the **--partial-dir** option with a relative path; because this tells rsync that it is OK to stash off a copy of a single file in a subdir in the destination hierarchy, rsync will use the partial-dir as a staging area to bring over the copied file, and then rename it into place from there. (Specifying a **--partial-dir** with an absolute path does not have this side-effect.)

-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/
```

If file's aren't linking, double-check their attributes. Also check if some attributes are getting forced outside of rsync's control, such a mount option that squishes root to a single user, or mounts a removable drive with generic ownership (such as OS X's "Ignore ownership on this volume" option).

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.

This option works best when copying into an empty destination hierarchy, as rsync treats existing files as definitive (so it never looks in the link-dest dirs when a destination file already exists), and as malleable (so it might change the attributes of a destination file, which affects all the hard-linked versions).

Note that if you combine this option with **--ignore-times**, rsync will not link any files together because it only links identical files together as a substitute for transferring the file, never as an additional check after the file is updated.

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-super-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 that this option typically achieves better compression ratios than 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.

See the **--skip-compress** option for the default list of file suffixes that will not be compressed.

--compress-level=NUM

Explicitly set the compression level to use (see **--compress**) instead of letting it default. If NUM is non-zero, the **--compress** option is implied.

--skip-compress=LIST

Override the list of file suffixes that will not be compressed. The **LIST** should be one or more file suffixes (without the dot) separated by slashes (/).

You may specify an empty string to indicate that no file should be skipped.

Simple character-class matching is supported: each must consist of a list of letters inside the square brackets (e.g. no special classes, such as "[:alpha:]", are supported).

The characters asterisk (*) and question-mark (?) have no special meaning.

Here's an example that specifies 6 suffixes to skip (since 1 of the 5 rules matches 2 suffixes):

--skip-compress=gz/jpg/mp[34]/7z/bz2

The default list of suffixes that will not be compressed is this (several of these are newly added for 3.0.0):

gz/zip/z/rpm/deb/iso/bz2/t[gb]z/7z/mp[34]/mov/avi/ogg/jpg/jpeg

This list will be replaced by your **--skip-compress** list in all but one situation: a copy from a daemon rsync will add your skipped suffixes to its list of non-compressing files (and its list may be configured to a different default).

--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.

--contimeout

This option allows you to set the amount of time that rsync will wait for its connection to an rsync daemon to succeed. If the timeout is reached, rsync exits with an error.

--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.

--sockopts

This option can provide endless fun for people who like to tune their systems to the utmost degree. You can set all sorts of socket options which may make transfers faster (or slower!). Read the man page for the `setsockopt()` system call for details on some of the options you may be able to set. By default no special socket options are set. This only affects direct socket connections to a remote rsync daemon. This option also exists 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.)

-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 **--out-format='%i %n%L'**. If you repeat the option, unchanged files will also be output, but only if the receiving rsync is at least version 2.6.7 (you can use **-vv** with older versions of rsync, but that also turns on the output of other verbose messages).

The "%i" escape has a cryptic output that is 11 letters long. The general format is like the string **YXcstpoguax**, where **Y** is replaced by the type 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 **Y** 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).
- A ***** means that the rest of the itemized-output area contains a message (e.g. “deleting”).

The file-types that replace the **X** are: **f** for a file, a **d** for a directory, an **L** for a symlink, a **D** for a device, and a **S** for a special file (e.g. named sockets and fifos).

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 either that a regular file has a different checksum (requires **−−checksum**) or that a symlink, device, or special file has a changed value. Note that if you are sending files to an rsync prior to 3.0.1, this change flag will be present only for checksum-differing regular files.
- A **s** means the size of a regular 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 modification time will be set to the transfer time, which happens when a file/symlink/device is updated without **−−times** and when a symlink is changed and the receiver can’t set its time. (Note: when using an rsync 3.0.0 client, you might see the **s** flag combined with **t** instead of the proper **T** flag for this time-setting failure.)
- 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 super-user 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 **u** slot is reserved for future use.
- The **a** means that the ACL information changed.
- The **x** means that the extended attribute information changed.

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).

−−out−format=FORMAT

This allows you to specify exactly what the rsync client outputs to the user on a per-update basis. The format is a text string containing embedded single-character escape sequences prefixed with a

percent (%) character. A default format of “%n%L” is assumed if **−v** is specified (which reports the name of the file and, if the item is a link, where it points). For a full list of the possible escape characters, see the “log format” setting in the rsyncd.conf manpage.

Specifying the **−−out−format** 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). In addition, if the itemize-changes escape (%i) is included in the string (e.g. if the **−−itemize−changes** option was used), 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 **−−itemize−changes** option for a description of the output of “%i”.

Rsync will output the out-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 out-format output).

−−log−file=FILE

This option causes rsync to log what it is doing to a file. This is similar to the logging that a daemon does, but can be requested for the client side and/or the server side of a non-daemon transfer. If specified as a client option, transfer logging will be enabled with a default format of “%i %n%L”. See the **−−log−file−format** option if you wish to override this.

Here’s a example command that requests the remote side to log what is happening:

```
rsync −av −−rsync−path="rsync −−log−file=/tmp/rlog" src/ dest/
```

This is very useful if you need to debug why a connection is closing unexpectedly.

−−log−file−format=FORMAT

This allows you to specify exactly what per-update logging is put into the file specified by the **−−log−file** option (which must also be specified for this option to have any effect). If you specify an empty string, updated files will not be mentioned in the log file. For a list of the possible escape characters, see the “log format” setting in the rsyncd.conf manpage.

The default FORMAT used if **−−log−file** is specified and this option is not is ’%i %n%L’.

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

The current statistics are as follows:

- **Number of files** is the count of all “files” (in the generic sense), which includes directories, symlinks, etc.
- **Number of files transferred** is the count of normal files that were updated via rsync’s delta-transfer algorithm, which does not include created dirs, symlinks, etc.
- **Total file size** is the total sum of all file sizes in the transfer. This does not count any size for directories or special files, but does include the size of symlinks.
- **Total transferred file size** is the total sum of all files sizes for just the transferred files.
- **Literal data** is how much unmatched file-update data we had to send to the receiver for it to recreate the updated files.
- **Matched data** is how much data the receiver got locally when recreating the updated files.
- **File list size** is how big the file-list data was when the sender sent it to the receiver. This is smaller than the in-memory size for the file list due to some compressing of duplicated data when rsync sends the list.

- **File list generation time** is the number of seconds that the sender spent creating the file list. This requires a modern rsync on the sending side for this to be present.
- **File list transfer time** is the number of seconds that the sender spent sending the file list to the receiver.
- **Total bytes sent** is the count of all the bytes that rsync sent from the client side to the server side.
- **Total bytes received** is the count of all non-message bytes that rsync received by the client side from the server side. "Non-message" bytes means that we don't count the bytes for a verbose message that the server sent to us, which makes the stats more consistent.

−8, −−8−bit−output

This tells rsync to leave all high-bit characters unescaped in the output instead of trying to test them to see if they're valid in the current locale and escaping the invalid ones. All control characters (but never tabs) are always escaped, regardless of this option's setting.

The escape idiom that started in 2.6.7 is to output a literal backslash (\) and a hash (#), followed by exactly 3 octal digits. For example, a newline would output as "\#012". A literal backslash that is in a filename is not escaped unless it is followed by a hash and 3 digits (0−9).

−h, −−human−readable

Output numbers in a more human-readable format. This makes big numbers output using larger units, with a K, M, or G suffix. If this option was specified once, these units are K (1000), M (1000*1000), and G (1000*1000*1000); if the option is repeated, the units are powers of 1024 instead of 1000.

−−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 delete 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 rsync's delta-transfer 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 add an exclude rule at the end of all your existing excludes. This will prevent the sending of any partial-dir files that may exist on the sending side, and will also prevent the untimely deletion of partial-dir items on the receiving side. An example: the above **−−partial−dir** option would add the equivalent of "**−f '−p .rsync−partial/'**" at the end of any other filter rules.

If you are supplying your own exclude rules, you may need to add your own exclude/hide/protect rule for the partial-dir because (1) the auto-added rule may be ineffective at the end of your other rules, or (2) you may wish to override rsync's exclude choice. For instance, if you want to make

rsync clean-up any left-over partial-dirs that may be lying around, you should specify **−−delete−after** and add a "risk" filter rule, e.g. **−f 'R .rsync−partial/'**. (Avoid using **−−delete−before** or **−−delete−during** unless you don't need rsync to use any of the left-over partial-dir data during the current run.)

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 affects 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 times that the **−−partial** option does not look for this environment value are (1) when **−−inplace** was specified (since **−−inplace** conflicts with **−−partial−dir**), and (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 if you've specified the **−−partial−dir** option, that directory will be used instead. See the comments in the **−−partial−dir** section for a discussion of how this ".˜tmp˜" dir will be excluded from the transfer, and what you can do if you want rsync to cleanup old ".˜tmp˜" dirs that might be lying around. Conflicts with **−−inplace** and **−−append**.

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 (1) 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) and (2) there are no mount points in the hierarchy (since the delayed updates will fail if they can't be renamed into place).

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).

−m, −−prune−empty−dirs

This option tells the receiving rsync to get rid of empty directories from the file-list, including nested directories that have no non-directory children. This is useful for avoiding the creation of a bunch of useless directories when the sending rsync is recursively scanning a hierarchy of files using include/exclude/filter rules.

Note that the use of transfer rules, such as the **−−min−size** option, does not affect what goes into the file list, and thus does not leave directories empty, even if none of the files in a directory match the transfer rule.

Because the file-list is actually being pruned, this option also affects what directories get deleted when a delete is active. However, keep in mind that excluded files and directories can prevent existing items from being deleted due to an exclude both hiding source files and protecting destination files. See the perishable filter-rule option for how to avoid this.

You can prevent the pruning of certain empty directories from the file-list by using a global "protect" filter. For instance, this option would ensure that the directory "emptydir" was kept in the file-list:

−−filter ’protect emptydir/’

Here’s an example that copies all .pdf files in a hierarchy, only creating the necessary destination directories to hold the .pdf files, and ensures that any superfluous files and directories in the destination are removed (note the hide filter of non-directories being used instead of an exclude):

rsync −avm −−del −−include=’*.pdf’ −f ’hide,! */’ src/ dest

If you didn’t want to remove superfluous destination files, the more time-honored options of “**−−include=’*/’ −−exclude=’*’**” would work fine in place of the hide-filter (if that is more natural to you).

−−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.

While rsync is transferring a regular file, it updates a progress line that looks like this:

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

In this example, the receiver has reconstructed 782448 bytes or 63% of the sender’s file, which is being reconstructed at a rate of 110.64 kilobytes per second, and the transfer will finish in 4 seconds if the current rate is maintained until the end.

These statistics can be misleading if rsync’s delta-transfer algorithm is in use. For example, if the sender’s file consists of the basis file followed by additional data, the reported rate will probably drop dramatically when the receiver gets to the literal data, and the transfer will probably take much longer to finish than the receiver estimated as it was finishing the matched part of the file.

When the file transfer finishes, rsync replaces the progress line with a summary line that looks like this:

1238099 100% 146.38kB/s 0:00:08 (xfer#5, to-check=169/396)

In this example, the file was 1238099 bytes long in total, the average rate of transfer for the whole file was 146.38 kilobytes per second over the 8 seconds that it took to complete, it was the 5th transfer of a regular file during the current rsync session, and there are 169 more files for the receiver to check (to see if they are up-to-date or not) remaining out of the 396 total files in the file-list.

−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 an rsync daemon. The file must not be world readable. It should contain just the password as a single line.

This option does not supply a password to a remote shell transport such as ssh; to learn how to do that, consult the remote shell’s documentation. When accessing an rsync daemon using a remote shell as the transport, this option only comes into effect after the remote shell finishes its authentication (i.e. if you have also specified a password in the daemon’s config file).

−−list−only

This option will cause the source files to be listed instead of transferred. This option is inferred if there is a single source arg and no destination specified, so its main uses are: (1) to turn a copy command that includes a destination arg into a file-listing command, or (2) to be able to specify more than one source arg (note: be sure to include the destination). Caution: keep in mind that a source arg with a wild-card is expanded by the shell into multiple args, so it is never safe to try to

list such an arg without using this option. For example:

rsync −av −−list−only foo* dest/

Compatibility note: when requesting a remote listing of files from an rsync that is version 2.6.3 or older, you may encounter an error if you ask for a non-recursive listing. This is because a file listing implies the **−−dirs** option w/o **−−recursive**, and older rsyncs don't have that option. To avoid this problem, either specify the **−−no−dirs** option (if you don't need to expand a directory's content), or turn on recursion and exclude the content of subdirectories: **−r −−exclude='/*/*'**.

−−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).

−−iconv=CONVERT_SPEC

Rsync can convert filenames between character sets using this option. Using a CONVERT_SPEC of "." tells rsync to look up the default character-set via the locale setting. Alternately, you can fully specify what conversion to do by giving a local and a remote charset separated by a comma in the order **−−iconv=LOCAL,REMOTE**, e.g. **−−iconv=utf8,iso88591**. This order ensures that the option will stay the same whether you're pushing or pulling files. Finally, you can specify either **−−no−iconv** or a CONVERT_SPEC of "−" to turn off any conversion. The default setting of this option is site-specific, and can also be affected via the RSYNC_ICONV environment variable.

For a list of what charset names your local iconv library supports, you can run "iconv −−list".

If you specify the **--protect-args** option (**-s**), rsync will translate the filenames you specify on the command-line that are being sent to the remote host. See also the **--files-from** option.

Note that rsync does not do any conversion of names in filter files (including include/exclude files). It is up to you to ensure that you're specifying matching rules that can match on both sides of the transfer. For instance, you can specify extra include/exclude rules if there are filename differences on the two sides that need to be accounted for.

When you pass an **--iconv** option to an rsync daemon that allows it, the daemon uses the charset specified in its "charset" configuration parameter regardless of the remote charset you actually pass. Thus, you may feel free to specify just the local charset for a daemon transfer (e.g. **--iconv=utf8**).

-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.

If rsync was complied without support for IPv6, the **--ipv6** option will have no effect. The **--version** output will tell you if this is the case.

--checksum-seed=NUM

Set the checksum seed to the integer NUM. This 4 byte checksum seed is included in each block and file 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. 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 the super-user; 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.

−−log−file=FILE
This option tells the rsync daemon to use the given log-file name instead of using the "log file" setting in the config file.

−−log−file−format=FORMAT
This option tells the rsync daemon to use the given FORMAT string instead of using the "log format" setting in the config file. It also enables "transfer logging" unless the string is empty, in which case transfer logging is turned off.

−−sockopts
This overrides the **socket options** setting in the rsyncd.conf file and has the same syntax.

−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).

If rsync was complied without support for IPv6, the **−−ipv6** option will have no effect. The **−−version** output will tell you if this is the case.

−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:

- 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 name of “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 a name of “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 filename. 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 regular file, symlink, or device.
- rsync chooses between doing a simple string match and wildcard matching by checking if the pattern contains one of these three wildcard characters: ‘*’, ‘?’, and ‘[’ .
- a ‘*’ matches any path component, but it stops at slashes.
- use ’**’ to match anything, including slashes.
- a ‘?’ matches any character except a slash (/).
- a ‘[’ introduces a character class, such as [a−z] or [[:alpha:]].
- in a wildcard pattern, a backslash can be used to escape a wildcard character, but it is matched literally when no wildcards are present.
- 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.)

- a trailing “dir_name/***” will match both the directory (as if “dir_name/” had been specified) and everything in the directory (as if “dir_name/**” had been specified). This behavior was added in version 2.6.7.

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), and perhaps use the **--prune-empty-dirs** option. 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 names matching *.o
- “- /foo” would exclude a file (or directory) named foo in the transfer-root directory
- “- foo/” would exclude any directory named foo
- “- /foo/*/bar” would exclude any file named bar which is at two levels below a directory named foo in the transfer-root directory
- “- /foo/**/bar” would exclude any file named bar two or more levels below a directory named foo in the transfer-root directory
- The combination of “+ */”, “+ *.c”, and “- *” would include all directories and C source files but nothing else (see also the **--prune-empty-dirs** option)
- 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 “*”)

The following modifiers are accepted after a “+” or “-”:

- A / specifies that the include/exclude rule should be matched against the absolute pathname of the current item. For example, “-/ /etc/passwd” would exclude the passwd file any time the transfer was sending files from the “/etc” directory, and “-/ subdir/foo” would always exclude “foo” when it is in a dir named “subdir”, even if “foo” is at the root of the current transfer.

- 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.
- A **p** indicates that a rule is perishable, meaning that it is ignored in directories that are being deleted. For instance, the **−C** option's default rules that exclude things like "CVS" and "*.o" are marked as perishable, and will not prevent a directory that was removed on the source from being deleted on the destination.

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 (above) 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.

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.

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.

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 your convenience, a script file is also created when the write-batch option is used: it will be named the same as the batch file with ".sh" appended. This script file contains a command-line suitable for updating a

destination tree using the associated batch file. It can be executed using a Bourne (or Bourne-like) shell, optionally passing in an alternate destination tree pathname which is then used instead of the original destination path. This is useful when the destination tree path on the current host differs from the one used to create the batch file.

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

35 Timeout waiting for daemon connection

ENVIRONMENT VARIABLES

CVSIGNORE

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

RSYNC_ICONV

Specify a default **−−iconv** setting using this environment variable.

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 remote shell transport such as ssh; to learn how to do that, consult the remote shell's documentation.

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 *nix 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 web site at http://rsync.samba.org/

VERSION
This man page is current for version 3.0.6 of rsync.

INTERNAL OPTIONS
The options **--server** and **--sender** are used internally by rsync, and should never be typed by a user under normal circumstances. Some awareness of these options may be needed in certain scenarios, such as when setting up a login that can only run an rsync command. For instance, the support directory of the rsync distribution has an example script named rrsync (for restricted rsync) that can be used with a restricted ssh login.

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. Please contact the mailing-list at rsync@lists.samba.org.

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

THANKS
Especial thanks go out to: John Van Essen, Matt McCutchen, Wesley W. Terpstra, David Dykstra, Jos Backus, Sebastian Krahmer, Martin Pool, and our gone-but-not-forgotten compadre, J.W. Schultz.

Thanks also to Richard Brent, Brendan Mackay, Bill Waite, Stephen Rothwell and David Bell. I’ve probably missed some people, my apologies if I have.

AUTHOR
rsync was originally written by Andrew Tridgell and Paul Mackerras. Many people have later contributed to it. It is currently maintained by Wayne Davison.

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