Manual

FreeBSD 온라인 매뉴얼 일부를 한글로 번역했습니다 .

- rsyncd.conf (작업중)
- CHROOT
- rsync

rsyncd.conf (작업중)

rsyncd.conf 파일은 rsync 데몬으로 실행할 때 rsync를 위한 런타임 구성 파일입니다.

rsyncd.conf 파일은 인증, 액세스, 로깅 및 사용 가능한 모듈을 제어합니다.

FILE FORMAT

파일은 모듈과 매개변수로 구성됩니다. 모듈은 대괄호 안의 모듈 이름으로 시작하여 다음 모듈이 시작될 때까지 계속됩니다. 모듈에는 name = value 형식의 매개변수가 포함됩니다.

파일은 줄 기반이며, 줄 바꿈으로 끝나는 각 줄은 주석, 모듈 이름 또는 매개변수를 나타냅니다.

매개변수에서 첫 번째 등호만 의미가 있습니다. 첫 번째 등호 앞뒤의 공백은 무시됩니다. 모듈 및 매개변수 이름의 선행 공백, 후행 공백, 내부 공백은 상관없습니다. 매개변수 값의 선행 공백과 후행 공백은 무시됩니다. 매개변수 값 내의 내부 공백은 그대로 유지됩니다.

해시(#)로 시작하는 줄은 공백만 포함된 줄과 마찬가지로 무시됩니다. (선행 공백 이외의 다른 공백 뒤에 해시가 오는 경우 해당 해시는 줄 내용의 일부로 간주됩니다.)

\ 로 끝나는 줄은 일반적인 UNIX 방식에 따라 다음 줄에서 "계속"됩니다.

매개변수에서 등호 뒤에 오는 값은 모두 문자열(따옴표가 필요 없음) 또는 부울이며, yes/no, 0/1 또는 true/false으로 지정할 수 있습니다. 대소문자는 부울 값에서는 중요하지 않지만 문자열 값에서는 유지됩니다.

LAUNCHING THE RSYNC DAEMON

rsync 데몬은 --daemon 옵션을 지정하여 실행합니다.

chroot를 사용하거나, 1024 미만의 포트(기본값은 873)에 바인딩하거나, 파일 소유권을 설정하려면 데몬이 루트 권한으로 실행되어야 합니다. 그렇지 않으면 적절한 데이터 읽기 및 쓰기 권한, 로그 및 파일 잠금 권한만 있으면 됩니다.

독립 실행형 데몬으로 inetd를 통해 실행하거나 원격 셸을 통해 rsync 클라이언트에서 실행할 수 있습니다. 독립 실행형 데몬으로 실행하려면 적절한 시작 스크립트에서 "rsync --daemon" 명령을 실행하면 됩니다.

inetd를 통해 실행하는 경우 /etc/services에 다음과 같은 줄을 추가해야 합니다:

```
rsync           873/tcp
```

그리고 /etc/inetd.conf에 다음과 같이 한 줄을 추가합니다:

rsync stream tcp nowait root /usr/bin/rsync rsyncd --daemon

"/usr/bin/rsync" rsync . HUP inetd .

rsync HUP rsyncd.conf .

GLOBAL PARAMETERS

([ ]):

motd file

'(MOTD) . MOTD . --dparam=motdfile=FILE .

pid file

ID rsync . rsync . --dparam=pidfile=FILE .

port

(873). inetd --port .

address

IP . inetd --address .

socket options

setsockopt() . --sockopts .

listen backlog

5 .

MODULE PARAMETERS .

%VAR% RSYNC_USER_NAME rsync (: true/false) (: %VAR% %%).

MODULE PARAMETERS

[module] "[global]" GLOBAL PARAMETERS .

comment

path

rsyncd.conf .

"/./" use chroot false dot dir .

```
path = /var/rsync/./module1
```

(chroot) "/var/rsync" chroot chroot "/module1" .

변수 이름을 퍼센트 기호로 둘러싸서 환경 변수를 기준으로 경로 값을 설정할 수 있습니다. 사용자가 연결할 때 rsync에 의해 설정된 변수를 참조할 수도 있습니다. 예를 들어, 경로에 인증 사용자의 이름을 사용합니다:

```
path = /home/%RSYNC_USER_NAME%
```

경로에 내부 공백이 포함되어 있어도 괜찮습니다. 공백은 그대로 유지되므로 공백을 이스케이프하려고 하면 안 됩니다. 최종 디렉터리에 후행 공백이 있는 경우(수정하고 싶지 않은 경우) 경로에 후행 슬래시를 추가하여 후행 공백이 손실되지 않도록 하세요.

use chroot

"use chroot"가 참이면, rsync 데몬은 클라이언트와 파일 전송을 시작하기 전에 "path"로 루팅합니다. 이 방법은 구현상의 보안 허점을 방지할 수 있다는 장점이 있지만, 수퍼유저 권한이 필요하고, 절대적이거나 새 루트 경로를 벗어난 심볼릭 링크를 따라갈 수 없으며, 이름별 사용자 및 그룹 보존이 복잡해진다는 단점이 있습니다(아래 참조).

use chroot가 설정되지 않은 경우, 기본값은 기본적으로 chroot 활성화를 시도하지만 실패하면(경고를 로깅한 후) 데몬이 계속할 수 있도록 허용합니다. 한 가지 예외는 모듈의 경로에 "/./" chroot 구분자가 있는 경우로, 이 경우 설정되지 않은 값이 해당 모듈에 대해 참으로 취급됩니다.

rsync 3.2.7 이전에는 기본값이 "true"였습니다. 새로운 기본값인 "unset"을 사용하면 루트 사용자가 아닌 사용자로 rsync 데몬을 설정하거나 chroot 설정이 실패한 시스템에서 데몬을 실행하기가 더 쉬워집니다. rsyncd.conf에서 이 값을 "true"로 명시적으로 설정하면 언제나 chroot가 성공해야 합니다.

모듈의 "path"에 dot-dir을 지정하여 경로의 앞부분에 chdir을 지정한 다음 경로의 뒷부분에 있는 파일을 서비스할 수 있습니다(sanitizing 및 기본 symlink munging 사용). 이 방법은 chroot 내부에 일부 라이브러리 디렉터리가 필요하지만(일반적으로 uid 및 gid 조회를 위해) lib 디렉터리를 서비스되는 경로의 맨 위에 넣고 싶지 않은 경우 유용할 수 있습니다(제외 지시어로 숨길 수 있지만). 그러나 최신 rsync 설정에서 더 나은 선택은 이름 변환기를 사용하는 것 "이며 내부 lib 디렉터리를 완전히 피하는 것입니다. 또한 경로와 관련된 chroot을 수행하기 전에 rsync가 자체 chroot 영역으로 chroot하도록 하는 데몬 chroot 매개변수를 참조하세요.

데몬이 "/" 디렉터리를 서비스하는 경우(직접적으로 또는 모듈의 경로에 루팅되어 있기 때문에), rsync는 경로 sanitizing 또는 (기본값인) munging을 수행하지 않습니다.

When it has to limit access to a particular subdir (either due to chroot being disabled or having an inside-chroot path set), rsync will munge symlinks (by default) and sanitize paths. Those that dislike munged symlinks (and really, really trust their users to not break out of the subdir) can disable the symlink munging via the "munge symlinks" parameter.

When rsync is sanitizing paths, it trims ".." path elements from args that it believes would escape the module hierarchy. It also substitutes leading slashes in absolute paths with the module's path (so that options such as --backup-dir & --compare-dest interpret an absolute path as rooted in the module's "path" dir).

When a chroot is in effect and the "name converter" parameter is not set, the "numeric ids" parameter will default to being enabled (disabling name lookups). This means that if you manually

setup name-lookup libraries in your chroot (instead of using a name converter) that you need to explicitly set numeric ids = false for rsync to do name lookups.

If you copy library resources into the module's chroot area, you should protect them through your OS's normal user/group or ACL settings (to prevent the rsync module's user from being able to change them), and then hide them from the user's view via "exclude" (see how in the discussion of that parameter). However, it's easier and safer to setup a name converter.

daemon chroot

This parameter specifies a path to which the daemon will chroot before beginning communication with clients. Module paths (and any "use chroot" settings) will then be related to this one. This lets you choose if you want the whole daemon to be chrooted (with this setting), just the transfers to be chrooted (with "use chroot"), or both. Keep in mind that the "daemon chroot" area may need various OS/lib/etc files installed to allow the daemon to function. By default the daemon runs without any chrooting.

proxy protocol

When this parameter is enabled, all incoming connections must start with a V1 or V2 proxy protocol header. If the header is not found, the connection is closed.

Setting this to true requires a proxy server to forward source IP information to rsync, allowing you to log proper IP/host info and make use of client-oriented IP restrictions. The default of false means that the IP information comes directly from the socket's metadata. If rsync is not behind a proxy, this should be disabled.

CAUTION: using this option can be dangerous if you do not ensure that only the proxy is allowed to connect to the rsync port. If any non-proxied connections are allowed through, the client will be able to use a modified rsync to spoof any remote IP address that they desire. You can lock this down using something like iptables -uid-owner root rules (for strict localhost access), various firewall rules, or you can require password authorization so that any spoofing by users will not grant extra access.

This setting is global. If you need some modules to require this and not others, then you will need to setup multiple rsync daemon processes on different ports.

name converter

This parameter lets you specify a program that will be run by the rsync daemon to do user & group conversions between names & ids. This script is started prior to any chroot being setup, and runs as the daemon user (not the transfer user). You can specify a fully qualified pathname or a program name that is on the $PATH.

The program can be used to do normal user & group lookups without having to put any extra files into the chroot area of the module or you can do customized conversions.

The nameconvert program has access to all of the environment variables that are described in the section on pre-xfer exec. This is useful if you want to customize the conversion using information about the module and/or the copy request.

There is a sample python script in the support dir named "nameconvert" that implements the normal user & group lookups. Feel free to customize it or just use it as documentation to implement your own.

numeric ids

Enabling this parameter disables the mapping of users and groups by name for the current daemon module. This prevents the daemon from trying to load any user/group-related files or libraries. This enabling makes the transfer behave as if the client had passed the --numeric-ids command-line option. By default, this parameter is enabled for chroot modules and disabled for non-chroot modules. Also keep in mind that uid/gid preservation requires the module to be running as root (see "uid") or for "fake super" to be configured.

A chroot-enabled module should not have this parameter set to false unless you're using a "name converter" program or you've taken steps to ensure that the module has the necessary resources it needs to translate names and that it is not possible for a user to change those resources.

munge symlinks

This parameter tells rsync to modify all symlinks in the same way as the (non-daemon-affecting) --munge-links command-line option (using a method described below). This should help protect your files from user trickery when your daemon module is writable. The default is disabled when "use chroot" is on with an inside-chroot path of "/", OR if "daemon chroot" is on, otherwise it is enabled.

If you disable this parameter on a daemon that is not read-only, there are tricks that a user can play with uploaded symlinks to access daemon-excluded items (if your module has any), and, if "use chroot" is off, rsync can even be tricked into showing or changing data that is outside the module's path (as access-permissions allow).

The way rsync disables the use of symlinks is to prefix each one with the string "/rsyncd-munged/". This prevents the links from being used as long as that directory does not exist. When this parameter is enabled, rsync will refuse to run if that path is a directory or a symlink to a directory. When using the "munge symlinks" parameter in a chroot area that has an inside-chroot path of "/", you should add "/rsyncd-munged/" to the exclude setting for the module so that a user can't try to create it.

Note: rsync makes no attempt to verify that any pre-existing symlinks in the module's hierarchy are as safe as you want them to be (unless, of course, it just copied in the whole hierarchy). If you setup an rsync daemon on a new area or locally add symlinks, you can manually protect your symlinks from being abused by prefixing "/rsyncd-munged/" to the start of every symlink's value. There is a perl script in the support directory of the source code named "munge-symlinks" that can be used to add or remove this prefix from your symlinks.

When this parameter is disabled on a writable module and "use chroot" is off (or the inside-chroot path is not "/"), incoming symlinks will be modified to drop a leading slash and to remove ".." path elements that rsync believes will allow a symlink to escape the module's hierarchy. There

are tricky ways to work around this, though, so you had better trust your users if you choose this combination of parameters.

charset

This specifies the name of the character set in which the module's filenames are stored. If the client uses an --iconv option, the daemon will use the value of the "charset" parameter regardless of the character set the client actually passed. This allows the daemon to support charset conversion in a chroot module without extra files in the chroot area, and also ensures that name-translation is done in a consistent manner. If the "charset" parameter is not set, the --iconv option is refused, just as if "iconv" had been specified via "refuse options".

If you wish to force users to always use --iconv for a particular module, add "no-iconv" to the "refuse options" parameter. Keep in mind that this will restrict access to your module to very new rsync clients.

max connections

This parameter allows you to specify the maximum number of simultaneous connections you will allow. Any clients connecting when the maximum has been reached will receive a message telling them to try later. The default is 0, which means no limit. A negative value disables the module. See also the "lock file" parameter.

log file

When the "log file" parameter is set to a non-empty string, the rsync daemon will log messages to the indicated file rather than using syslog. This is particularly useful on systems (such as AIX) where syslog() doesn't work for chrooted programs. The file is opened before chroot() is called, allowing it to be placed outside the transfer. If this value is set on a per-module basis instead of globally, the global log will still contain any authorization failures or config-file error messages.

If the daemon fails to open the specified file, it will fall back to using syslog and output an error about the failure. (Note that the failure to open the specified log file used to be a fatal error.)

This setting can be overridden by using the --log-file=FILE or --dparam=logfile=FILE command-line options. The former overrides all the log-file parameters of the daemon and all module settings. The latter sets the daemon's log file and the default for all the modules, which still allows modules to override the default setting.

syslog facility

This parameter allows you to specify the syslog facility name to use when logging messages from the rsync daemon. You may use any standard syslog facility name which is defined on your system. Common names are auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, security, syslog, user, uucp, local0, local1, local2, local3, local4, local5, local6 and local7. The default is daemon. This setting has no effect if the "log file" setting is a non-empty string (either set in the per-modules settings, or inherited from the global settings).

syslog tag

This parameter allows you to specify the syslog tag to use when logging messages from the rsync daemon. The default is "rsyncd". This setting has no effect if the "log file" setting is a non-

empty string (either set in the per-modules settings, or inherited from the global settings).

For example, if you wanted each authenticated user's name to be included in the syslog tag, you could do something like this:

```
syslog tag = rsyncd.%RSYNC_USER_NAME%
```

max verbosity

This parameter allows you to control the maximum amount of verbose information that you'll allow the daemon to generate (since the information goes into the log file). The default is 1, which allows the client to request one level of verbosity.

This also affects the user's ability to request higher levels of --info and --debug logging. If the max value is 2, then no info and/or debug value that is higher than what would be set by -vv will be honored by the daemon in its logging. To see how high of a verbosity level you need to accept for a particular info/debug level, refer to rsync --info=help and rsync --debug=help. For instance, it takes max-verbosity 4 to be able to output debug TIME2 and FLIST3.

lock file

This parameter specifies the file to use to support the "max connections" parameter. The rsync daemon uses record locking on this file to ensure that the max connections limit is not exceeded for the modules sharing the lock file. The default is /var/run/rsyncd.lock.

read only

This parameter determines whether clients will be able to upload files or not. If "read only" is true then any attempted uploads will fail. If "read only" is false then uploads will be possible if file permissions on the daemon side allow them. The default is for all modules to be read only.

Note that "auth users" can override this setting on a per-user basis.

write only

This parameter determines whether clients will be able to download files or not. If "write only" is true then any attempted downloads will fail. If "write only" is false then downloads will be possible if file permissions on the daemon side allow them. The default is for this parameter to be disabled.

Helpful hint: you probably want to specify "refuse options = delete" for a write-only module.

open noatime

When set to True, this parameter tells the rsync daemon to open files with the O_NOATIME flag (on systems that support it) to avoid changing the access time of the files that are being transferred. If your OS does not support the O_NOATIME flag then rsync will silently ignore this option. Note also that some filesystems are mounted to avoid updating the atime on read access even without the O_NOATIME flag being set.

When set to False, this parameters ensures that files on the server are not opened with O_NOATIME.

When set to Unset (the default) the user controls the setting via --open-noatime.

list

This parameter determines whether this module is listed when the client asks for a listing of available modules. In addition, if this is false, the daemon will pretend the module does not exist when a client denied by "hosts allow" or "hosts deny" attempts to access it. Realize that if "reverse lookup" is disabled globally but enabled for the module, the resulting reverse lookup to a potentially client-controlled DNS server may still reveal to the client that it hit an existing module. The default is for modules to be listable.

uid

This parameter specifies the user name or user ID that file transfers to and from that module should take place as when the daemon was run as root. In combination with the "gid" parameter this determines what file permissions are available. The default when run by a super-user is to switch to the system's "nobody" user. The default for a non-super-user is to not try to change the user. See also the "gid" parameter.

The RSYNC_USER_NAME environment variable may be used to request that rsync run as the authorizing user. For example, if you want a rsync to run as the same user that was received for the rsync authentication, this setup is useful:

```
uid = %RSYNC_USER_NAME%
gid = *
```

gid

This parameter specifies one or more group names/IDs that will be used when accessing the module. The first one will be the default group, and any extra ones be set as supplemental groups. You may also specify a "*" as the first gid in the list, which will be replaced by all the normal groups for the transfer's user (see "uid"). The default when run by a super-user is to switch to your OS's "nobody" (or perhaps "nogroup") group with no other supplementary groups. The default for a non-super-user is to not change any group attributes (and indeed, your OS may not allow a non-super-user to try to change their group settings).

The specified list is normally split into tokens based on spaces and commas. However, if the list starts with a comma, then the list is only split on commas, which allows a group name to contain a space. In either case any leading and/or trailing whitespace is removed from the tokens and empty tokens are ignored.

daemon uid

This parameter specifies a uid under which the daemon will run. The daemon usually runs as user root, and when this is left unset the user is left unchanged. See also the "uid" parameter.

daemon gid

This parameter specifies a gid under which the daemon will run. The daemon usually runs as group root, and when this is left unset, the group is left unchanged. See also the "gid" parameter.

fake super

Setting "fake super = yes" for a module causes the daemon side to behave as if the --fake-super command-line option had been specified. This allows the full attributes of a file to be stored without having to have the daemon actually running as root.

filter

The daemon has its own filter chain that determines what files it will let the client access. This chain is not sent to the client and is independent of any filters the client may have specified. Files excluded by the daemon filter chain (daemon-excluded files) are treated as non-existent if the client tries to pull them, are skipped with an error message if the client tries to push them (triggering exit code 23), and are never deleted from the module. You can use daemon filters to prevent clients from downloading or tampering with private administrative files, such as files you may add to support uid/gid name translations.

The daemon filter chain is built from the "filter", "include from", "include", "exclude from", and "exclude" parameters, in that order of priority. Anchored patterns are anchored at the root of the module. To prevent access to an entire subtree, for example, "/secret", you must exclude everything in the subtree; the easiest way to do this is with a triple-star pattern like "/secret/***".

The "filter" parameter takes a space-separated list of daemon filter rules, though it is smart enough to know not to split a token at an internal space in a rule (e.g. "- /foo - /bar" is parsed as two rules). You may specify one or more merge-file rules using the normal syntax. Only one "filter" parameter can apply to a given module in the config file, so put all the rules you want in a single parameter. Note that per-directory merge-file rules do not provide as much protection as global rules, but they can be used to make --delete work better during a client download operation if the per-dir merge files are included in the transfer and the client requests that they be used.

exclude

This parameter takes a space-separated list of daemon exclude patterns. As with the client --exclude option, patterns can be qualified with "- " or "+ " to explicitly indicate exclude/include. Only one "exclude" parameter can apply to a given module. See the "filter" parameter for a description of how excluded files affect the daemon.

include

Use an "include" to override the effects of the "exclude" parameter. Only one "include" parameter can apply to a given module. See the "filter" parameter for a description of how excluded files affect the daemon.

exclude from

This parameter specifies the name of a file on the daemon that contains daemon exclude patterns, one per line. Only one "exclude from" parameter can apply to a given module; if you have multiple exclude-from files, you can specify them as a merge file in the "filter" parameter. See the "filter" parameter for a description of how excluded files affect the daemon.

include from

Analogue of "exclude from" for a file of daemon include patterns. Only one "include from" parameter can apply to a given module. See the "filter" parameter for a description of how

excluded files affect the daemon.
incoming chmod

This parameter allows you to specify a set of comma-separated chmod strings that will affect the permissions of all incoming files (files that are being received by the daemon). These changes happen after all other permission calculations, and this will even override destination-default and/or existing permissions when the client does not specify --perms. See the description of the --chmod rsync option and the chmod(1) manpage for information on the format of this string.
outgoing chmod

This parameter allows you to specify a set of comma-separated chmod strings that will affect the permissions of all outgoing files (files that are being sent out from the daemon). These changes happen first, making the sent permissions appear to be different than those stored in the filesystem itself. For instance, you could disable group write permissions on the server while having it appear to be on to the clients. See the description of the --chmod rsync option and the chmod(1) manpage for information on the format of this string.
auth users

This parameter specifies a comma and/or space-separated list of authorization rules. In its simplest form, you list the usernames that will be allowed to connect to this module. The usernames do not need to exist on the local system. The rules may contain shell wildcard characters that will be matched against the username provided by the client for authentication. If "auth users" is set then the client will be challenged to supply a username and password to connect to the module. A challenge response authentication protocol is used for this exchange. The plain text usernames and passwords are stored in the file specified by the "secrets file" parameter. The default is for all users to be able to connect without a password (this is called "anonymous rsync").

In addition to username matching, you can specify groupname matching via a '@' prefix. When using groupname matching, the authenticating username must be a real user on the system, or it will be assumed to be a member of no groups. For example, specifying "@rsync" will match the authenticating user if the named user is a member of the rsync group.

Finally, options may be specified after a colon (:). The options allow you to "deny" a user or a group, set the access to "ro" (read-only), or set the access to "rw" (read/write). Setting an auth-rule-specific ro/rw setting overrides the module's "read only" setting.

Be sure to put the rules in the order you want them to be matched, because the checking stops at the first matching user or group, and that is the only auth that is checked. For example:

```
auth users = joe:deny @guest:deny admin:rw @rsync:ro susan joe sam
```

In the above rule, user joe will be denied access no matter what. Any user that is in the group "guest" is also denied access. The user "admin" gets access in read/write mode, but only if the admin user is not in group "guest" (because the admin user-matching rule would never be reached if the user is in group "guest"). Any other user who is in group "rsync" will get read-only access. Finally, users susan, joe, and sam get the ro/rw setting of the module, but only if the user didn't

match an earlier group-matching rule.

If you need to specify a user or group name with a space in it, start your list with a comma to indicate that the list should only be split on commas (though leading and trailing whitespace will also be removed, and empty entries are just ignored). For example:

```
auth users = , joe:deny, @Some Group:deny, admin:rw, @RO Group:ro
```

See the description of the secrets file for how you can have per-user passwords as well as per-group passwords. It also explains how a user can authenticate using their user password or (when applicable) a group password, depending on what rule is being authenticated.

See also the section entitled "USING RSYNC-DAEMON FEATURES VIA A REMOTE SHELL CONNECTION" in rsync(1) for information on how handle an rsyncd.conf-level username that differs from the remote-shell-level username when using a remote shell to connect to an rsync daemon.

secrets file

This parameter specifies the name of a file that contains the username:password and/or @groupname:password pairs used for authenticating this module. This file is only consulted if the "auth users" parameter is specified. The file is line-based and contains one name:password pair per line. Any line has a hash (#) as the very first character on the line is considered a comment and is skipped. The passwords can contain any characters but be warned that many operating systems limit the length of passwords that can be typed at the client end, so you may find that passwords longer than 8 characters don't work.

The use of group-specific lines are only relevant when the module is being authorized using a matching "@groupname" rule. When that happens, the user can be authorized via either their "username:password" line or the "@groupname:password" line for the group that triggered the authentication.

It is up to you what kind of password entries you want to include, either users, groups, or both. The use of group rules in "auth users" does not require that you specify a group password if you do not want to use shared passwords.

There is no default for the "secrets file" parameter, you must choose a name (such as /etc/rsyncd.secrets). The file must normally not be readable by "other"; see "strict modes". If the file is not found or is rejected, no logins for an "auth users" module will be possible.

strict modes

This parameter determines whether or not the permissions on the secrets file will be checked. If "strict modes" is true, then the secrets file must not be readable by any user ID other than the one that the rsync daemon is running under. If "strict modes" is false, the check is not performed. The default is true. This parameter was added to accommodate rsync running on the Windows operating system.

hosts allow

This parameter allows you to specify a list of comma- and/or whitespace-separated patterns that are matched against a connecting client's hostname and IP address. If none of the patterns match, then the connection is rejected.

Each pattern can be in one of six forms:

a dotted decimal IPv4 address of the form a.b.c.d, or an IPv6 address of the form a:b:c::d:e:f. In this case the incoming machine's IP address must match exactly.

an address/mask in the form ipaddr/n where ipaddr is the IP address and n is the number of one bits in the netmask. All IP addresses which match the masked IP address will be allowed in.

an address/mask in the form ipaddr/maskaddr where ipaddr is the IP address and maskaddr is the netmask in dotted decimal notation for IPv4, or similar for IPv6, e.g. ffff:ffff:ffff:ffff:: instead of /64. All IP addresses which match the masked IP address will be allowed in.

a hostname pattern using wildcards. If the hostname of the connecting IP (as determined by a reverse lookup) matches the wildcarded name (using the same rules as normal Unix filename matching), the client is allowed in. This only works if "reverse lookup" is enabled (the default).

a hostname. A plain hostname is matched against the reverse DNS of the connecting IP (if "reverse lookup" is enabled), and/or the IP of the given hostname is matched against the connecting IP (if "forward lookup" is enabled, as it is by default). Any match will be allowed in.

an '@' followed by a netgroup name, which will match if the reverse DNS of the connecting IP is in the specified netgroup.

Note IPv6 link-local addresses can have a scope in the address specification:

fe80::1%link1
fe80::%link1/64
fe80::%link1/ffff:ffff:ffff:ffff::

You can also combine "hosts allow" with "hosts deny" as a way to add exceptions to your deny list. When both parameters are specified, the "hosts allow" parameter is checked first and a match results in the client being able to connect. A non-allowed host is then matched against the "hosts deny" list to see if it should be rejected. A host that does not match either list is allowed to connect.

The default is no "hosts allow" parameter, which means all hosts can connect.

hosts deny

This parameter allows you to specify a list of comma- and/or whitespace-separated patterns that are matched against a connecting clients hostname and IP address. If the pattern matches then the connection is rejected. See the "hosts allow" parameter for more information.

The default is no "hosts deny" parameter, which means all hosts can connect.

reverse lookup

Controls whether the daemon performs a reverse lookup on the client's IP address to determine its hostname, which is used for "hosts allow" & "hosts deny" checks and the "%h" log escape. This is enabled by default, but you may wish to disable it to save time if you know the lookup will not

return a useful result, in which case the daemon will use the name "UNDETERMINED" instead.

If this parameter is enabled globally (even by default), rsync performs the lookup as soon as a client connects, so disabling it for a module will not avoid the lookup. Thus, you probably want to disable it globally and then enable it for modules that need the information.

forward lookup

Controls whether the daemon performs a forward lookup on any hostname specified in an hosts allow/deny setting. By default this is enabled, allowing the use of an explicit hostname that would not be returned by reverse DNS of the connecting IP.

ignore errors

This parameter tells rsyncd to ignore I/O errors on the daemon when deciding whether to run the delete phase of the transfer. Normally rsync skips the --delete step if any I/O errors have occurred in order to prevent disastrous deletion due to a temporary resource shortage or other I/O error. In some cases this test is counter productive so you can use this parameter to turn off this behavior.

ignore nonreadable

This tells the rsync daemon to completely ignore files that are not readable by the user. This is useful for public archives that may have some non-readable files among the directories, and the sysadmin doesn't want those files to be seen at all.

transfer logging

This parameter enables per-file logging of downloads and uploads in a format somewhat similar to that used by ftp daemons. The daemon always logs the transfer at the end, so if a transfer is aborted, no mention will be made in the log file.

If you want to customize the log lines, see the "log format" parameter.

log format

This parameter allows you to specify the format used for logging file transfers when transfer logging is enabled. The format is a text string containing embedded single-character escape sequences prefixed with a percent (%) character. An optional numeric field width may also be specified between the percent and the escape letter (e.g. "%-50n %8l %07p"). In addition, one or more apostrophes may be specified prior to a numerical escape to indicate that the numerical value should be made more human-readable. The 3 supported levels are the same as for the --human-readable command-line option, though the default is for human-readability to be off. Each added apostrophe increases the level (e.g. "%''l %'b %f").

The default log format is "%o %h [%a] %m (%u) %f %l", and a "%t [%p] " is always prefixed when using the "log file" parameter. (A perl script that will summarize this default log format is included in the rsync source code distribution in the "support" subdirectory: rsyncstats.)

The single-character escapes that are understood are as follows:

%a the remote IP address (only available for a daemon)
%b the number of bytes actually transferred

%B the permission bits of the file (e.g. rwxrwxrwt)

%c the total size of the block checksums received for the basis file (only when sending)

%C the full-file checksum if it is known for the file. For older rsync protocols/versions, the checksum was salted, and is thus not a useful value (and is not displayed when that is the case). For the checksum to output for a file, either the --checksum option must be in-effect or the file must have been transferred without a salted checksum being used. See the --checksum-choice option for a way to choose the algorithm.

%f the filename (long form on sender; no trailing "/")

%G the gid of the file (decimal) or "DEFAULT"

%h the remote host name (only available for a daemon)

%i an itemized list of what is being updated

%l the length of the file in bytes

%L the string " -> SYMLINK", " => HARDLINK", or "" (where SYMLINK or HARDLINK is a filename)

%m the module name

%M the last-modified time of the file

%n the filename (short form; trailing "/" on dir)

%o the operation, which is "send", "recv", or "del." (the latter includes the trailing period)

%p the process ID of this rsync session

%P the module path

%t the current date time

%u the authenticated username or an empty string

%U the uid of the file (decimal)

For a list of what the characters mean that are output by "%i", see the --itemize-changes option in the rsync manpage.

Note that some of the logged output changes when talking with older rsync versions. For instance, deleted files were only output as verbose messages prior to rsync 2.6.4.

timeout

This parameter allows you to override the clients choice for I/O timeout for this module. Using this parameter you can ensure that rsync won't wait on a dead client forever. The timeout is specified in seconds. A value of zero means no timeout and is the default. A good choice for anonymous rsync daemons may be 600 (giving a 10 minute timeout).

refuse options

This parameter allows you to specify a space-separated list of rsync command-line options that will be refused by your rsync daemon. You may specify the full option name, its one-letter abbreviation, or a wild-card string that matches multiple options. Beginning in 3.2.0, you can also negate a match term by starting it with a "!".

When an option is refused, the daemon prints an error message and exits.

For example, this would refuse --checksum (-c) and all the various delete options:

```
refuse options = c delete
```

The reason the above refuses all delete options is that the options imply --delete, and implied options are refused just like explicit options.

The use of a negated match allows you to fine-tune your refusals after a wild-card, such as this:

```
refuse options = delete-* !delete-during
```

Negated matching can also turn your list of refused options into a list of accepted options. To do this, begin the list with a "*" (to refuse all options) and then specify one or more negated matches to accept. For example:

```
refuse options = * !a !v !compress*
```

Don't worry that the "*" will refuse certain vital options such as --dry-run, --server, --no-iconv, --seclude-args, etc. These important options are not matched by wild-card, so they must be overridden by their exact name. For instance, if you're forcing iconv transfers you could use something like this:

```
refuse options = * no-iconv !a !v
```

As an additional aid (beginning in 3.2.0), refusing (or "!refusing") the "a" or "archive" option also affects all the options that the --archive option implies (-rdlptgoD), but only if the option is matched explicitly (not using a wildcard). If you want to do something tricky, you can use "archive*" to avoid this side-effect, but keep in mind that no normal rsync client ever sends the actual archive option to the server.

As an additional safety feature, the refusal of "delete" also refuses remove-source-files when the daemon is the sender; if you want the latter without the former, instead refuse "delete-*" as that refuses all the delete modes without affecting --remove-source-files. (Keep in mind that the client's --delete option typically results in --delete-during.)

When un-refusing delete options, you should either specify "!delete*" (to accept all delete options) or specify a limited set that includes "delete", such as:

```
refuse options = * !a !delete !delete-during
```

... whereas this accepts any delete option except --delete-after:

```
refuse options = * !a !delete* delete-after
```

A note on refusing "compress": it may be better to set the "dont compress" daemon parameter to "*" and ensure that RSYNC_COMPRESS_LIST=zlib is set in the environment of the daemon in order to disable compression silently instead of returning an error that forces the client to remove the -z option.

If you are un-refusing the compress option, you may want to match "!compress*" if you also want to allow the --compress-level option.

Note that the "copy-devices" & "write-devices" options are refused by default, but they can be explicitly accepted with "!copy-devices" and/or "!write-devices". The options "log-file" and "log-file-format" are forcibly refused and cannot be accepted.

Here are all the options that are not matched by wild-cards:

--server: Required for rsync to even work.
--rsh, -e: Required to convey compatibility flags to the server.
--out-format: This is required to convey output behavior to a remote receiver. While rsync passes the older alias --log-format for compatibility reasons, this options should not be confused with --log-file-format.
--sender: Use "write only" parameter instead of refusing this.
--dry-run, -n: Who would want to disable this?
--seclude-args, -s: Is the oldest arg-protection method.
--from0, -0: Makes it easier to accept/refuse --files-from without affecting this helpful modifier.
--iconv: This is auto-disabled based on "charset" parameter.
--no-iconv: Most transfers use this option.
--checksum-seed: Is a fairly rare, safe option.
--write-devices: Is non-wild but also auto-disabled.

dont compress

NOTE: This parameter currently has no effect except in one instance: if it is set to "*" then it minimizes or disables compression for all files (for those that don't want to refuse the --compress option completely).

This parameter allows you to select filenames based on wildcard patterns that should not be compressed when pulling files from the daemon (no analogous parameter exists to govern the pushing of files to a daemon). Compression can be expensive in terms of CPU usage, so it is usually good to not try to compress files that won't compress well, such as already compressed files.

The "dont compress" parameter takes a space-separated list of case-insensitive wildcard patterns. Any source filename matching one of the patterns will be compressed as little as possible during the transfer. If the compression algorithm has an "off" level, then no compression occurs for those files. If an algorithms has the ability to change the level in mid-stream, it will be minimized to reduce the CPU usage as much as possible.

See the --skip-compress parameter in the rsync(1) manpage for the list of file suffixes that are skipped by default if this parameter is not set.
early exec, pre-xfer exec, post-xfer exec

You may specify a command to be run in the early stages of the connection, or right before and/or after the transfer. If the early exec or pre-xfer exec command returns an error code, the transfer is aborted before it begins. Any output from the pre-xfer exec command on stdout (up to

several KB) will be displayed to the user when aborting, but is not displayed if the script returns success. The other programs cannot send any text to the user. All output except for the pre-xfer exec stdout goes to the corresponding daemon's stdout/stderr, which is typically discarded. See the --no-detatch option for a way to see the daemon's output, which can assist with debugging.

Note that the early exec command runs before any part of the transfer request is known except for the module name. This helper script can be used to setup a disk mount or decrypt some data into a module dir, but you may need to use lock file and max connections to avoid concurrency issues. If the client rsync specified the --early-input=FILE option, it can send up to about 5K of data to the stdin of the early script. The stdin will otherwise be empty.

Note that the post-xfer exec command is still run even if one of the other scripts returns an error code. The pre-xfer exec command will not be run, however, if the early exec command fails.

The following environment variables will be set, though some are specific to the pre-xfer or the post-xfer environment:

RSYNC_MODULE_NAME: The name of the module being accessed.
RSYNC_MODULE_PATH: The path configured for the module.
RSYNC_HOST_ADDR: The accessing host's IP address.
RSYNC_HOST_NAME: The accessing host's name.
RSYNC_USER_NAME: The accessing user's name (empty if no user).
RSYNC_PID: A unique number for this transfer.
RSYNC_REQUEST: (pre-xfer only) The module/path info specified by the user. Note that the user can specify multiple source files, so the request can be something like "mod/path1 mod/path2", etc.
RSYNC_ARG#: (pre-xfer only) The pre-request arguments are set in these numbered values. RSYNC_ARG0 is always "rsyncd", followed by the options that were used in RSYNC_ARG1, and so on. There will be a value of "." indicating that the options are done and the path args are beginning -- these contain similar information to RSYNC_REQUEST, but with values separated and the module name stripped off.
RSYNC_EXIT_STATUS: (post-xfer only) the server side's exit value. This will be 0 for a successful run, a positive value for an error that the server generated, or a -1 if rsync failed to exit properly. Note that an error that occurs on the client side does not currently get sent to the server side, so this is not the final exit status for the whole transfer.
RSYNC_RAW_STATUS: (post-xfer only) the raw exit value from waitpid().

Even though the commands can be associated with a particular module, they are run using the permissions of the user that started the daemon (not the module's uid/gid setting) without any chroot restrictions.

These settings honor 2 environment variables: use RSYNC_SHELL to set a shell to use when running the command (which otherwise uses your system() call's default shell), and use RSYNC_NO_XFER_EXEC to disable both options completely.

CONFIG DIRECTIVES

There are currently two config directives available that allow a config file to incorporate the contents of other files: &include and &merge. Both allow a reference to either a file or a directory. They differ in how segregated the file's contents are considered to be.

The &include directive treats each file as more distinct, with each one inheriting the defaults of the parent file, starting the parameter parsing as globals/defaults, and leaving the defaults unchanged for the parsing of the rest of the parent file.

The &merge directive, on the other hand, treats the file's contents as if it were simply inserted in place of the directive, and thus it can set parameters in a module started in another file, can affect the defaults for other files, etc.

When an &include or &merge directive refers to a directory, it will read in all the *.conf or *.inc files (respectively) that are contained inside that directory (without any recursive scanning), with the files sorted into alpha order. So, if you have a directory named "rsyncd.d" with the files "foo.conf", "bar.conf", and "baz.conf" inside it, this directive:

```
&include /path/rsyncd.d
```

would be the same as this set of directives:

```
&include /path/rsyncd.d/bar.conf
&include /path/rsyncd.d/baz.conf
&include /path/rsyncd.d/foo.conf
```

except that it adjusts as files are added and removed from the directory.

The advantage of the &include directive is that you can define one or more modules in a separate file without worrying about unintended side-effects between the self-contained module files.

The advantage of the &merge directive is that you can load config snippets that can be included into multiple module definitions, and you can also set global values that will affect connections (such as motd file), or globals that will affect other include files.

For example, this is a useful /etc/rsyncd.conf file:

```
port = 873
log file = /var/log/rsync.log
pid file = /var/lock/rsync.lock

&merge /etc/rsyncd.d
&include /etc/rsyncd.d
```

This would merge any /etc/rsyncd.d/*.inc files (for global values that should stay in effect), and then include any /etc/rsyncd.d/*.conf files (defining modules without any global-value cross-talk).
AUTHENTICATION STRENGTH

The authentication protocol used in rsync is a 128 bit MD4 based challenge response system. This is fairly weak protection, though (with at least one brute-force hash-finding algorithm publicly available), so if you want really top-quality security, then I recommend that you run rsync over ssh. (Yes, a future version of rsync will switch over to a stronger hashing method.)

Also note that the rsync daemon protocol does not currently provide any encryption of the data that is transferred over the connection. Only authentication is provided. Use ssh as the transport if you want encryption.

You can also make use of SSL/TLS encryption if you put rsync behind an SSL proxy.
SSL/TLS Daemon Setup

When setting up an rsync daemon for access via SSL/TLS, you will need to configure a TCP proxy (such as haproxy or nginx) as the front-end that handles the encryption.

You should limit the access to the backend-rsyncd port to only allow the proxy to connect. If it is on the same host as the proxy, then configuring it to only listen on localhost is a good idea.
You should consider turning on the proxy protocol rsync-daemon parameter if your proxy supports sending that information. The examples below assume that this is enabled.

An example haproxy setup is as follows:

```
frontend fe_rsync-ssl
  bind :::874 ssl crt /etc/letsencrypt/example.com/combined.pem
  mode tcp
  use_backend be_rsync

backend be_rsync
  mode tcp
  server local-rsync 127.0.0.1:873 check send-proxy
```

An example nginx proxy setup is as follows:

```
stream {
  server {
    listen 874 ssl;
    listen [::]:874 ssl;

    ssl_certificate /etc/letsencrypt/example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/example.com/privkey.pem;

    proxy_pass localhost:873;
    proxy_protocol on; # Requires rsyncd.conf "proxy protocol = true"
    proxy_timeout 1m;
    proxy_connect_timeout 5s;
  }
}
```

DAEMON CONFIG EXAMPLES

A simple rsyncd.conf file that allow anonymous rsync to a ftp area at /home/ftp would be:

```
[ftp]
        path = /home/ftp
        comment = ftp export area
```

A more sophisticated example would be:

```
uid = nobody
gid = nobody
use chroot = yes
max connections = 4
syslog facility = local5
pid file = /var/run/rsyncd.pid

[ftp]
        path = /var/ftp/./pub
        comment = whole ftp area (approx 6.1 GB)

[sambaftp]
        path = /var/ftp/./pub/samba
        comment = Samba ftp area (approx 300 MB)

[rsyncftp]
        path = /var/ftp/./pub/rsync
        comment = rsync ftp area (approx 6 MB)

[sambawww]
        path = /public_html/samba
        comment = Samba WWW pages (approx 240 MB)

[cvs]
        path = /data/cvs
        comment = CVS repository (requires authentication)
        auth users = tridge, susan
        secrets file = /etc/rsyncd.secrets
```

The /etc/rsyncd.secrets file would look something like this:

```
tridge:mypass
susan:herpass
```

CHROOT

Name

chroot -- change root directory

SYNOPSIS

chroot [-G group[,group ...]] [-g group] [-u user] [-n] newroot [command [arg ...]]

DESCRIPTION

chroot 유틸리티는 현재 디렉터리와 루트 디렉터리를, 제공된 디렉터리 newroot로 변경한 다음 제공된 인수 (제공된 경우) 또는 사용자 로그인 셸의 대화형 복사본과 함께 명령을 실행합니다.

옵션은 다음과 같습니다:

-G group[,group ...]	지정된 그룹의 권한으로 명령을 실행합니다
-g group	지정된 그룹의 권한으로 명령을 실행합니다
-u user	사용자 자격으로 명령을 실행합니다
-n	PROC_NO_NEW_PRIVS_CTL procctl(2) 명령을 사용하여 호출 프로세스와 그 하위 프로세스에 대한 SUID/SGID 비활성화합니다. security.bsd.unprivileged_chroot sysctl이 1로 설정된 경우 1로 설정하면 수퍼유저 권한 없이도 수퍼유저 권한 없이 루팅할 수 있습니다.

ENVIRONMENT

다음 환경 변수는 chroot가 참조합니다:

SHELL: 설정하면 SHELL로 지정된 문자열이 실행할 셸의 이름으로 해석됩니다.

EXAMPLES

Example 1: 새 루트 디렉터리로 루팅하기

다음 명령은 표준 루트 디렉터리로 루팅한 후 csh(1) 셸을 엽니다.

```
# chroot / /bin/csh
```

Example 2: 변경된 루트 디렉터리로 명령 실행하기

다음 명령은 루트 디렉터리를 chroot로 변경한 다음 ls(1)을 실행하여 /sbin의 내용을 나열합니다.

```
# chroot /tmp/testroot ls /sbin
```

번역 : 2024.01.19.

원본 : https://man.freebsd.org/cgi/man.cgi?chroot(8)

rsync

NAME

rsync - 빠르고 다재다능한 원격 (및 로컬) 파일 복사 도구

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.

The online version of this manpage (that includes cross-linking of topics) is available at https://download.samba.org/pub/rsync/rsync.1.

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:

- support for copying links, devices, owners, groups, and permissions
- exclude and exclude-from options similar to GNU tar
- a CVS exclude mode for ignoring the same files that CVS would ignore
- can use any transparent remote shell, including ssh or rsh
- does not require super-user privileges
- pipelining of file transfers to minimize latency costs
- 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.md 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 in the data. Note that the expansion of wildcards on the command-line (*.c) into a list of files is handled by the shell before it runs rsync and not by rsync itself (exactly the same as all other Posix-style programs).

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::

COPYING TO A DIFFERENT NAME

When you want to copy a directory to a different name, use a trailing slash on the source directory to put the contents of the directory into any destination directory you like:

rsync -ai foo/ bar/

Rsync also has the ability to customize a destination file's name when copying a single item. The rules for this are:

- The transfer list must consist of a single item (either a file or an empty directory)

- The final element of the destination path must not exist as a directory

- The destination path must not have been specified with a trailing slash

Under those circumstances, rsync will set the name of the destination's single item to the last element of the destination path. Keep in mind that it is best to only use this idiom when copying a file and use the above trailing-slash idiom when copying a directory.

The following example copies the foo.c file as bar.c in the save dir (assuming that bar.c isn't a directory):

```
rsync -ai src/foo.c save/bar.c
```

The single-item copy rule might accidentally bite you if you unknowingly copy a single item and specify a destination dir that doesn't exist (without using a trailing slash). For example, if src/*.c matches one file and save/dir doesn't exist, this will confuse you by naming the destination file save/dir:

```
rsync -ai src/*.c save/dir
```

To prevent such an accident, either make sure the destination dir exists or specify the destination path with a trailing slash:

```
rsync -ai src/*.c save/dir/
```

SORTED TRANSFER ORDER

Rsync always sorts the specified filenames into its internal transfer list. This handles the merging together of the contents of identically named directories, makes it easy to remove duplicate filenames. It can, however, confuse someone when the files are transferred in a different order than what was given on the command-line.

If you need a particular file to be transferred prior to another, either separate the files into different rsync calls, or consider using --delay-updates (which doesn't affect the sorted transfer order, but does make the final file-updating phase happen much more rapidly).

MULTI-HOST SECURITY

Rsync takes steps to ensure that the file requests that are shared in a transfer are protected against various security issues. Most of the

potential problems arise on the receiving side where rsync takes steps to ensure that the list of files being transferred remains within the bounds of what was requested.

Toward this end, rsync 3.1.2 and later have aborted when a file list contains an absolute or relative path that tries to escape out of the top of the transfer. Also, beginning with version 3.2.5, rsync does two more safety checks of the file list to (1) ensure that no extra source arguments were added into the transfer other than those that the client requested and (2) ensure that the file list obeys the exclude rules that were sent to the sender.

For those that don't yet have a 3.2.5 client rsync (or those that want to be extra careful), it is safest to do a copy into a dedicated destination directory for the remote files when you don't trust the remote host. For example, instead of doing an rsync copy into your home directory:

rsync -aiv host1:dir1 ~

Dedicate a "host1-files" dir to the remote content:

rsync -aiv host1:dir1 ~/host1-files

See the --trust-sender option for additional details.

CAUTION: it is not particularly safe to use rsync to copy files from a case-preserving filesystem to a case-ignoring filesystem. If you must perform such a copy, you should either disable symlinks via --no-links or enable the munging of symlinks via --munge-links (and make sure you use the right local or remote option). This will prevent rsync from doing potentially dangerous things if a symlink name overlaps with a file or directory. It does not, however, ensure that you get a full copy of all the files (since that may not be possible when the names overlap). A potentially better solution is to list all the source files and create a safe list of filenames that you pass to the --files-from option. Any files that conflict in name would need to be copied to different destination directories using more than one copy.

While a copy of a case-ignoring filesystem to a case-ignoring filesystem can work out fairly well, if no --delete-during or --delete-before option is active, rsync can potentially update an existing file on the receiveing side without noticing that the upper-/lower-case of the filename should be changed to match the sender.

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 -aiv host:file1 :file2 host:file{3,4} /dest/
rsync -aiv host::modname/file{1,2} host::modname/extra /dest/
rsync -aiv host::modname/first ::extra-file{1,2} /dest/

Note that a daemon connection only supports accessing one module per copy command, so if the start of a follow-up path doesn't begin with the modname of the first path, it is assumed to be a path in the module (such as the extra-file1 & extra-file2 that are grabbed above).

Really old versions of rsync (2.6.9 and before) only allowed specifying one remote-source arg, so some people have instead relied on the remote-shell performing space splitting to break up an arg into multiple paths. Such unintuitive behavior is no longer supported by default (though you can request it, as described below).

Starting in 3.2.4, filenames are passed to a remote shell in such a way as to preserve the characters you give it. Thus, if you ask for a file with spaces in the name, that's what the remote rsync looks for:

rsync -aiv host:'a simple file.pdf' /dest/

If you use scripts that have been written to manually apply extra quoting to the remote rsync args (or to require remote arg splitting), you can ask rsync to let your script handle the extra escaping. This is done by either adding the --old-args option to the rsync runs in the script (which requires a new rsync) or exporting RSYNC_OLD_ARGS=1 and RSYNC_PROTECT_ARGS=0 (which works with old or new rsync versions).

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:

- Use either double-colon syntax or rsync:// URL syntax instead of the single-colon (remote shell) syntax.

- The first element of the "path" is actually a module name.

o Additional remote source args can use an abbreviated syntax that omits the hostname and/or the module name, as discussed in ADVANCED USAGE.

o The remote daemon may print a "message of the day" when you connect.

o If you specify only the host (with no module or path) then a list of accessible modules on the daemon is output.

o If you specify a remote source path but no destination, a listing of the matching files on the remote daemon is output.

o The --rsh (-e) option must be omitted to avoid changing the connection style from using a socket connection to USING RSYNC-DAEMON FEATURES VIA A REMOTE-SHELL CONNECTION.

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).

Note also that if the RSYNC_SHELL environment variable is set, that program will be used to run the RSYNC_CONNECT_PROG command instead of using the default shell of the system() call.

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".

In this setup, the daemon is started by the ssh command that is accessing the system (which can be forced via the ~/.ssh/authorized_keys file, if desired). However, when accessing a daemon directly, it needs to be started beforehand.

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) manpage -- 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 rsync can be used.

To backup a home directory, which consists of large MS Word files and mail folders, a per-user cron job can be used that runs this each day:

rsync -aiz . bkhost:backup/joe/

To move some files from a remote host to the local host, you could run:

rsync -aiv --remove-source-files rhost:/tmp/{file1,file2}.c ~/src/

OPTION SUMMARY

Here is a short summary of the options available in rsync. Each option also has its own detailed description later in this manpage.

--verbose, -v increase verbosity
--info=FLAGS fine-grained informational verbosity
--debug=FLAGS fine-grained debug verbosity
--stderr=e|a|c change stderr output mode (default: errors)
--quiet, -q suppress non-error messages
--no-motd suppress daemon-mode MOTD
--checksum, -c skip based on checksum, not mod-time & size
--archive, -a archive mode is -rlptgoD (no -A,-X,-U,-N,-H)
--no-OPTION turn off an implied OPTION (e.g. --no-D)
--recursive, -r recurse into directories
--relative, -R use relative path names
--no-implied-dirs don't send implied dirs with --relative
--backup, -b 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)
--update, -u 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
--dirs, -d transfer directories without recursing
--old-dirs, --old-d works like --dirs when talking to old rsync
--mkpath create destination's missing path components
--links, -l copy symlinks as symlinks
--copy-links, -L transform symlink into referent file/dir
--copy-unsafe-links only "unsafe" symlinks are transformed
--safe-links ignore symlinks that point outside the tree
--munge-links munge symlinks to make them safe & unusable
--copy-dirlinks, -k transform symlink to dir into referent dir
--keep-dirlinks, -K treat symlinked dir on receiver as dir
--hard-links, -H preserve hard links
--perms, -p preserve permissions
--fileflags preserve file-flags (aka chflags)
--executability, -E preserve executability
--chmod=CHMOD affect file and/or directory permissions
--acls, -A preserve ACLs (implies --perms)
--xattrs, -X preserve extended attributes
--owner, -o preserve owner (super-user only)
--group, -g preserve group
--devices preserve device files (super-user only)
--copy-devices copy device contents as a regular file
--write-devices write to devices as files (implies --inplace)
--specials preserve special files
-D same as --devices --specials
--times, -t preserve modification times
--atimes, -U preserve access (use) times
--open-noatime avoid changing the atime on opened files
--crtimes, -N preserve create times (newness)
--omit-dir-times, -O omit directories from --times
--omit-link-times, -J omit symlinks from --times
--super receiver attempts super-user activities
--fake-super store/recover privileged attrs using xattrs
--sparse, -S turn sequences of nulls into sparse blocks
--preallocate allocate dest files before writing them
--dry-run, -n perform a trial run with no changes made
--whole-file, -W copy files whole (w/o delta-xfer algorithm)
--checksum-choice=STR choose the checksum algorithm (aka --cc)
--one-file-system, -x don't cross filesystem boundaries
--block-size=SIZE, -B force a fixed checksum block-size
--rsh=COMMAND, -e 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 xfer, not during
--delete-during receiver deletes during the transfer
--delete-delay find deletions during, delete after
--delete-after receiver deletes after transfer, not during
--delete-excluded also delete excluded files from dest dirs
--ignore-missing-args ignore missing source args without error
--delete-missing-args delete missing source args from destination
--ignore-errors delete even if there are I/O errors
--force-delete force deletion of directories even if not empty
--force-change affect user-/system-immutable files/dirs
--force-uchange affect user-immutable files/dirs
--force-schange affect system-immutable files/dirs
--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
--max-alloc=SIZE change a limit relating to memory alloc
--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
--prune-empty-dirs, -m prune empty directory chains from file-list
--numeric-ids don't map uid/gid values by user/group name
--usermap=STRING custom username mapping
--groupmap=STRING custom groupname mapping
--chown=USER:GROUP simple username/groupname mapping
--timeout=SECONDS set I/O timeout in seconds
--contimeout=SECONDS set daemon connection timeout in seconds
--ignore-times, -I don't skip files that match size and time
--size-only skip files that match in size
--modify-window=NUM, -@ set the accuracy for mod-time comparisons
--temp-dir=DIR, -T create temporary files in directory DIR
--fuzzy, -y find similar file for basis if no dest file
--compare-dest=DIR also compare destination files relative to DIR
--copy-dest=DIR ... and include copies of unchanged files
--link-dest=DIR hardlink to files in DIR when unchanged
--compress, -z compress file data during the transfer
--compress-choice=STR choose the compression algorithm (aka --zc)
--compress-level=NUM explicitly set compression level (aka --zl)
--skip-compress=LIST skip compressing files with suffix in LIST
--cvs-exclude, -C auto-ignore files in the same way CVS does
--filter=RULE, -f 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
--from0, -0 all *-from/filter files are delimited by 0s
--old-args disable the modern arg-protection idiom
--secluded-args, -s use the protocol to safely send the args
--trust-sender trust the remote sender's file list
--copy-as=USER[:GROUP] specify user & optional group for the copy
--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
--outbuf=N|L|B set out buffering to None, Line, or Block
--stats give some file-transfer stats
--8-bit-output, -8 leave high-bit chars unescaped in output
--human-readable, -h output numbers in a human-readable format
--progress show progress during transfer
-P same as --partial --progress
--itemize-changes, -i output a change-summary for all updates
--remote-option=OPT, -M send OPTION to the remote side only
--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
--early-input=FILE use FILE for daemon's early exec input
--list-only list the files instead of copying them
--bwlimit=RATE limit socket I/O bandwidth
--stop-after=MINS Stop rsync after MINS minutes have elapsed
--stop-at=y-m-dTh:m Stop rsync at the specified point in time
--fsync fsync every written file
--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)
--ipv4, -4 prefer IPv4
--ipv6, -6 prefer IPv6
--version, -V print the version + other info and exit
--help, -h (*) show this help (* -h is help only on its own)

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=RATE limit socket I/O bandwidth

--config=FILE specify alternate rsyncd.conf file
--dparam=OVERRIDE, -M override global daemon config parameter
--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
--verbose, -v increase verbosity
--ipv4, -4 prefer IPv4
--ipv6, -6 prefer IPv6
--help, -h show this help (when used with --daemon)

OPTIONS

Rsync accepts both long (double-dash + word) and short (single-dash + letter) options. The full list of the available options are described below. If an option can be specified in more than one way, the choices are comma-separated. Some options only have a long variant, not a short.

If the option takes a parameter, the parameter is only listed after the long variant, even though it must also be specified for the short. When specifying a parameter, you can either use the form --option=param, --option param, -o=param, -o param, or -oparam (the latter choices assume that your option has a short variant).

The parameter may need to be quoted in some manner for it to survive the shell's command-line parsing. Also keep in mind that a leading tilde (~) in a pathname is substituted by your shell, so make sure that you separate the option name from the pathname using a space if you want the local shell to expand it.

--help Print a short help page describing the options available in rsync and exit. You can also use -h for --help when it is used without any other options (since it normally means --human-readable).

--version, -V

Print the rsync version plus other info and exit. When repeated, the information is output is a JSON format that is still fairly readable (client side only).

The output includes a list of compiled-in capabilities, a list of optimizations, the default list of checksum algorithms, the default list of compression algorithms, the default list of daemon auth digests, a link to the rsync web site, and a few other items.

--verbose, -v

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.

The end-of-run summary tells you the number of bytes sent to the remote rsync (which is the receiving side on a local copy), the number of bytes received from the remote host, and the average bytes per second of the transferred data computed over the entire length of the rsync run. The second line shows the total size (in bytes), which is the sum of all the file sizes that rsync considered transferring. It also shows a "speedup" value, which is a ratio of the total file size divided by the sum of the sent and received bytes (which is really just a feel-good bigger-is-better number). Note that these byte values can be made more (or less) human-readable by using the --human-readable (or --no-human-readable) options.

In a modern rsync, the -v option is equivalent to the setting of groups of --info and --debug options. You can choose to use these newer options in addition to, or in place of using --verbose, as any fine-grained settings override the implied settings of -v. Both --info and --debug have a way to ask for help that tells you exactly what flags are set for each increase in verbosity.

However, do keep in mind that a daemon's "max verbosity" setting will limit how high of a level the various individual flags can be set on the daemon side. For instance, if the max is 2, then any info and/or debug flag that is set to a higher value than what would be set by -vv will be downgraded to the -vv level in the daemon's logging.

--info=FLAGS

This option lets you have fine-grained control over the information output you want to see. An individual flag name may be followed by a level number, with 0 meaning to silence that output, 1 being the default output level, and higher numbers increasing the output of that flag (for those that support higher levels). Use --info=help to see all the available flag names, what they output, and what flag names are added for each increase in the verbose level. Some examples:

rsync -a --info=progress2 src/ dest/
rsync -avv --info=stats2,misc1,flist0 src/ dest/

Note that --info=name's output is affected by the --out-format and --itemize-changes (-i) options. See those options for more information on what is output and when.

This option was added to 3.1.0, so an older rsync on the server side might reject your attempts at fine-grained control (if one or more flags needed to be send to the server and the server was too old to understand them). See also the "max verbosity" caveat above when dealing with a daemon.

--debug=FLAGS

This option lets you have fine-grained control over the debug output you want to see. An individual flag name may be followed by a level number, with 0 meaning to silence that output, 1 being the default output level, and higher numbers increasing the output of that flag (for those that support higher levels). Use --debug=help to see all the available flag names, what they output, and what flag names are added for each increase in the verbose level. Some examples:

rsync -avvv --debug=none src/ dest/
rsync -avA --del --debug=del2,acl src/ dest/

Note that some debug messages will only be output when the --stderr=all option is specified, especially those pertaining to I/O and buffer debugging.

Beginning in 3.2.0, this option is no longer auto-forwarded to the server side in order to allow you to specify different debug values for each side of the transfer, as well as to specify a new debug option that is only present in one of the rsync versions. If you want to duplicate the same option on both sides, using brace expansion is an easy way to save you some typing. This works in zsh and bash:

rsync -aiv {-M,}--debug=del2 src/ dest/

--stderr=errors|all|client

This option controls which processes output to stderr and if info messages are also changed to stderr. The mode strings can be abbreviated, so feel free to use a single letter value. The 3 possible choices are:

- errors - (the default) causes all the rsync processes to send an error directly to stderr, even if the process is on the remote side of the transfer. Info messages are sent to the client side via the protocol stream. If stderr is not available (i.e. when directly connecting with a daemon via a socket) errors fall back to being sent via the protocol stream.

- all - causes all rsync messages (info and error) to get written directly to stderr from all (possible) processes. This causes stderr to become line-buffered (instead of raw) and eliminates the ability to divide up the info and error messages by file handle. For those doing debugging or using several levels of verbosity, this option can help to avoid clogging up the transfer stream (which should prevent any chance of a deadlock bug hanging things up). It also allows --debug to enable some extra I/O related messages.

- client - causes all rsync messages to be sent to the client side via the protocol stream. One client process outputs all messages, with errors on stderr and info messages on stdout. This was the default in older rsync versions, but can cause error delays when a lot of transfer data is ahead of the messages. If you're pushing files to an older rsync, you may want to use --stderr=all since that idiom has been around for several releases.

This option was added in rsync 3.2.3. This version also began the forwarding of a non-default setting to the remote side, though rsync uses the backward-compatible options --msgs2stderr and --no-msgs2stderr to represent the all and client settings, respectively. A newer rsync will continue to accept these older option names to maintain compatibility.

--quiet, -q

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.

--ignore-times, -I

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.

This option can be confusing compared to --ignore-existing and --ignore-non-existing in that that they cause rsync to transfer fewer files, while this option causes rsync to transfer more files.

--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=NUM, -@

When comparing two timestamps, rsync treats the timestamps as being equal if they differ by no more than the modify-window value. The default is 0, which matches just integer seconds. If you specify a negative value (and the receiver is at least version 3.1.3) then nanoseconds will also be taken into account. Specifying 1 is useful for copies to/from MS Windows FAT filesystems, because FAT represents times with a 2-second resolution (allowing times to differ from the original by up to 1 second).

If you want all your transfers to default to comparing nanoseconds, you can create a ~/.popt file and put these lines in it:

rsync alias -a -a@-1
rsync alias -t -t@-1

With that as the default, you'd need to specify --modify-window=0 (aka -@0) to override it and ignore nanoseconds, e.g. if you're copying between ext3 and ext4, or if the receiving rsync is older than 3.1.3.

--checksum, -c

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, so this can slow things down significantly (and this is prior to any reading that will be done to transfer changed files)

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.

The checksum used is auto-negotiated between the client and the server, but can be overridden using either the --checksum-choice (--cc) option or an environment variable that is discussed in that option's section.

--archive, -a

This is equivalent to -rlptgoD. It is a quick way of saying you want recursion and want to preserve almost everything. Be aware that it does not include preserving ACLs (-A), xattrs (-X), atimes (-U), crtimes (-N), nor the finding and preserving of hardlinks (-H). It also does not imply --fileflags.

The only exception to the above equivalence is when --files-from is specified, in which case -r is not implied.

--no-OPTION

You may turn off one or more implied options by prefixing the option name with "no-". Not all positive options have a negated opposite, but a lot do, including those that can be used to disable an implied option (e.g. --no-D, --no-perms) or have different defaults in various circumstances (e.g. --no-whole-file, --no-blocking-io, --no-dirs). Every valid negated option ac-

cepts both the short and the long option name after the "no-" prefix (e.g. --no-R is the same as --no-relative).

As an example, if you want to use --archive (-a) but don't want --owner (-o), instead of converting -a into -rlptgD, you can specify -a --no-o (aka --archive --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).

--recursive, -r

This tells rsync to copy directories recursively. See also --dirs (-d) for an option that allows the scanning of a single directory.

See the --inc-recursive option for a discussion of the incremental recursion for creating the list of files to transfer.

--inc-recursive, --i-r

This option explicitly enables on incremental recursion when scanning for files, which is enabled by default when using the --recursive option and both sides of the transfer are running rsync 3.0.0 or newer.

Incremental recursion uses much less memory than non-incremental, while also beginning the transfer more quickly (since it doesn't need to scan the entire transfer hierarchy before it starts transferring files). If no recursion is enabled in the source files, this option has no effect.

Some options require rsync to know the full file list, so these options disable the incremental recursion mode. These include:

- --delete-before (the old default of --delete)
- --delete-after
- --prune-empty-dirs
- --delay-updates

In order to make --delete compatible with incremental recursion, rsync 3.0.0 made --delete-during the default delete mode (which

was first added in 2.6.4).

One side-effect of incremental recursion is that any missing sub-directories inside a recursively-scanned directory are (by default) created prior to recursing into the sub-dirs. This earlier creation point (compared to a non-incremental recursion) allows rsync to then set the modify time of the finished directory right away (without having to delay that until a bunch of recursive copying has finished). However, these early directories don't yet have their completed mode, mtime, or ownership set -- they have more restrictive rights until the subdirectory's copying actually begins. This early-creation idiom can be avoided by using the --omit-dir-times option.

Incremental recursion can be disabled using the --no-inc-recursive (--no-i-r) option.

--no-inc-recursive, --no-i-r

Disables the new incremental recursion algorithm of the --recursive option. This makes rsync scan the full file list before it begins to transfer files. See --inc-recursive for more info.

--relative, -R

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/

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 sym-

link 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.) 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.

--backup, -b

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.

If you don't specify --backup-dir:

1. the --omit-dir-times option will be forced on

2. the use of --delete (without --delete-excluded), causes rsync to add a "protect" filter-rule for the backup suffix to the end of all your existing filters that looks like this: -f "P *~". This rule prevents 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

This implies the --backup option, and 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).

Note that if you specify a relative path, the backup directory will be relative to the destination directory, so you probably want to specify either an absolute path or a path that starts with "../". If an rsync daemon is the receiver, the backup dir cannot go outside the module's path hierarchy, so take extra care not to delete it or copy into it.

--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.

--update, -u
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 dirs, 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, so don't expect any exclude side effects.

A caution for those that choose to combine --inplace with --update: an interrupted transfer will leave behind a partial file on the receiving side that has a very recent modified time, so re-running the transfer will probably not continue the interrupted file. As such, it is usually best to avoid combining this with --inplace unless you have implemented manual steps to handle any interrupted in-progress files.

--inplace
This option changes how rsync transfers a file when its 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:

o Hard links are not broken. This means the new data will be visible through other hard links to the destination file. Moreover, attempts to copy differing source files onto a multiply-linked destination file will result in a "tug of war" with the destination data changing back and

forth.

- 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).

- The file's data will be in an inconsistent state during the transfer and will be left that way if the transfer is interrupted or if an update fails.

- A file that rsync cannot write to cannot be updated. While a super user can update any file, a normal user needs to be granted write permission for the open of the file for writing to be successful.

- 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. This does not apply if you use --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 transferring large files with block-based changes or appended data, and also on systems that are disk bound, not network bound. It can also help keep a copy-on-write filesystem snapshot from diverging the entire contents of a file that only has minor changes.

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 special copy mode only works to efficiently update files that are known to be growing larger where any existing content on the receiving side is also known to be the same as the content on the sender. The use of --append can be dangerous if you aren't 100% sure that all the files in the transfer are shared, growing files. You should thus use filter rules to ensure that you weed out any files that do not fit this criteria.

Rsync updates these growing file in-place without verifying any of the existing content in the file (it only verifies the content that it is appending). Rsync skips any files that exist on the receiving side that are not shorter than the associated file on the sending side (which means that new files are transferred). It also skips any files whose size on the sending side gets shorter during the send negotiations (rsync warns about a "diminished" file when this happens).

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 directories or non-regular files.

--append-verify

This special copy mode works like --append except that all the data in the file is included in the checksum verification (making it less efficient but also potentially safer). This option can be dangerous if you aren't 100% sure that all the files in the transfer are shared, growing files. See the --append option for more details.

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.

--dirs, -d

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 (--old-d) that tells rsync to use a hack of -r --exclude='/*/*' to get an older rsync to list a single directory without recurs-

ing.

--mkpath
Create all missing path components of the destination path.

By default, rsync allows only the final component of the destination path to not exist, which is an attempt to help you to validate your destination path. With this option, rsync creates all the missing destination-path components, just as if mkdir -p $DEST_PATH had been run on the receiving side.

When specifying a destination path, including a trailing slash ensures that the whole path is treated as directory names to be created, even when the file list has a single item. See the COPYING TO A DIFFERENT NAME section for full details on how rsync decides if a final destination-path component should be created as a directory or not.

If you would like the newly-created destination dirs to match the dirs on the sending side, you should be using --relative (-R) instead of --mkpath. For instance, the following two commands result in the same destination tree, but only the second command ensures that the "some/extra/path" components match the dirs on the sending side:

rsync -ai --mkpath host:some/extra/path/*.c some/extra/path/
rsync -aiR host:some/extra/path/*.c ./

--links, -l
Add symlinks to the transferred files instead of noisily ignoring them with a "non-regular file" warning for each symlink encountered. You can alternately silence the warning by specifying --info=nonreg0.

The default handling of symlinks is to recreate each symlink's unchanged value on the receiving side.

See the SYMBOLIC LINKS section for multi-option info.

--copy-links, -L
The sender transforms each symlink encountered in the transfer into the referent item, following the symlink chain to the file or directory that it references. If a symlink chain is broken, an error is output and the file is dropped from the transfer.

This option supersedes any other options that affect symlinks in the transfer, since there are no symlinks left in the transfer.

This option does not change the handling of existing symlinks on the receiving side, unlike versions of rsync prior to 2.6.3 which had the side-effect of telling the receiving side to also follow symlinks. A modern rsync won't forward this option to a remote receiver (since only the sender needs to know about it), so this caveat should only affect someone using an rsync client older than 2.6.7 (which is when -L stopped being forwarded to the receiver).

See the --keep-dirlinks (-K) if you need a symlink to a directory to be treated as a real directory on the receiving side.

See the SYMBOLIC LINKS section for multi-option info.

--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.

Note that the cut-off point is the top of the transfer, which is the part of the path that rsync isn't mentioning in the verbose output. If you copy "/src/subdir" to "/dest/" then the "subdir" directory is a name inside the transfer tree, not the top of the transfer (which is /src) so it is legal for created relative symlinks to refer to other names inside the /src and /dest directories. If you instead copy "/src/subdir/" (with a trailing slash) to "/dest/subdir" that would not allow symlinks to any files outside of "subdir".

Note that safe symlinks are only copied if --links was also specified or implied. The --copy-unsafe-links option has no extra effect when combined with --copy-links.

See the SYMBOLIC LINKS section for multi-option info.

--safe-links

This tells the receiving rsync to ignore any symbolic links in the transfer which point outside the copied tree. All absolute symlinks are also ignored.

Since this ignoring is happening on the receiving side, it will still be effective even when the sending side has munged symlinks (when it is using --munge-links). It also affects deletions, since the file being present in the transfer prevents any matching file on the receiver from being deleted when the symlink is deemed to be unsafe and is skipped.

This option must be combined with --links (or --archive) to have any symlinks in the transfer to conditionally ignore. Its effect is superseded by --copy-unsafe-links.

Using this option in conjunction with --relative may give unexpected results.

See the SYMBOLIC LINKS section for multi-option info.

--munge-links

This option affects just one side of the transfer and tells rsync to munge symlink values when it is receiving files or unmunge symlink values when it is sending files. The munged values make the symlinks unusable on disk but allows the original contents of the symlinks to be recovered.

The server-side rsync often enables this option without the client's knowledge, such as in an rsync daemon's configuration file or by an option given to the rrsync (restricted rsync) script. When specified on the client side, specify the option normally if it is the client side that has/needs the munged symlinks, or use -M--munge-links to give the option to the server when it has/needs the munged symlinks. Note that on a local transfer, the client is the sender, so specifying the option directly unmunges symlinks while specifying it as a remote option munges symlinks.

This option has no effect when sent to a daemon via --remote-option because the daemon configures whether it wants munged symlinks via its "munge symlinks" parameter.

The symlink value is munged/unmunged once it is in the transfer, so any option that transforms symlinks into non-symlinks occurs prior to the munging/unmunging except for --safe-links, which is a choice that the receiver makes, so it bases its decision on the munged/unmunged value. This does mean that if a receiver has munging enabled, that using --safe-links will cause all symlinks to be ignored (since they are all absolute).

The method that rsync uses to munge the symlinks is to prefix each one's value with the string "/rsyncd-munged/". This prevents the links from being used as long as the directory does not exist. When this option is enabled, rsync will refuse to run if that path is a directory or a symlink to a directory (though it only checks at startup). See also the "munge-symlinks" python script in the support directory of the source code

for a way to munge/unmunge one or more symlinks in-place.

--copy-dirlinks, -k

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-delete or --delete is in effect).

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

--copy-dirlinks applies to all symlinks to directories in the source. If you want to follow only a few specified symlinks, a trick you can use is to pass them as additional source args with a trailing slash, using --relative to make the paths match up right. For example:

rsync -r --relative src/./ src/./follow-me/ dest/

This works because rsync calls lstat(2) on the source arg as given, and the trailing slash makes lstat(2) follow the symlink, giving rise to a directory in the file-list which overrides the symlink found during the scan of "src/./".

See the SYMBOLIC LINKS section for multi-option info.

--keep-dirlinks, -K

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 or enable the --munge-links option on the receiving side! If it is possible for an untrusted user to create their own symlink to any real 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.

See the SYMBOLIC LINKS section for multi-option info.

--hard-links, -H

This tells rsync to look for hard-linked files in the source and link together the corresponding files on the destination. Without this option, hard-linked files in the source are treated as though they were separate files.

This option does NOT necessarily ensure that the pattern of hard links on the destination exactly matches that on the source. Cases in which the destination may end up with extra hard links include the following:

- If the destination contains extraneous hard-links (more linking than what is present in the source file list), the copying algorithm will not break them explicitly. However, if one or more of the paths have content differences, the normal file-update process will break those extra links (unless you are using the --inplace option).

- If you specify a --link-dest directory that contains hard links, the linking of the destination files against the --link-dest files can cause some paths in the destination to become linked together due to the --link-dest associations.

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 --inc-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 (i.e. which files are hard-linked together), just its efficiency (i.e. copying the data for a new, early copy of a hard-linked file that could have been found later in the transfer in another member of the hard-linked set of files). One way to avoid this inefficiency is to disable incremental recursion using the --no-inc-recursive option.

--perms, -p

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.)

--executability, -E

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.

--acls, -A

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.

--xattrs, -X

This option causes rsync to update the destination extended attributes to be the same as the source 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.

The above name filtering can be overridden by using one or more filter options with the x modifier. When you specify an xattr-affecting filter rule, rsync requires that you do your own system/user filtering, as well as any additional filtering for what xattr names are copied and what names are allowed to be deleted. For example, to skip the system namespace, you could specify:

--filter='-x system.*'

To skip all namespaces except the user namespace, you could specify a negated-user match:

--filter='-x! user.*'

To prevent any attributes from being deleted, you could specify a receiver-only rule that excludes all names:

--filter='-xr *'

Note that the -X option does not copy rsync's special xattr values (e.g. those used by --fake-super) unless you repeat the option (e.g. -XX). This "copy all xattrs" mode cannot be used with --fake-super.

--fileflags

This option causes rsync to update the file-flags to be the same as the source files and directories (if your OS supports the chflags(2) system call). Some flags can only be altered by the super-user and some might only be unset below a certain secure-level (usually single-user mode). It will not make files alterable that are set to immutable on the receiver. To do that, see --force-change, --force-uchange, and --force-schange.

--force-change
This option causes rsync to disable both user-immutable and system-immutable flags on files and directories that are being updated or deleted on the receiving side. This option overrides --force-uchange and --force-schange.

--force-uchange
This option causes rsync to disable user-immutable flags on files and directories that are being updated or deleted on the receiving side. It does not try to affect system flags. This option overrides --force-change and --force-schange.

--force-schange
This option causes rsync to disable system-immutable flags on files and directories that are being updated or deleted on the receiving side. It does not try to affect user flags. This option overrides --force-change and --force-uchange.

--chmod=CHMOD
This option tells rsync to apply one or more comma-separated "chmod" modes to the permission of the files in the transfer. The resulting value is treated as though it were 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, the following will ensure that all directories get marked set-gid, that no files are other-writable, that both are user-writable and group-writable, and that both have consistent executability across all bits:

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

Using octal mode numbers is also allowed:

--chmod=D2775,F664

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 trans-

fer.

--owner, -o

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).

--group, -g

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. If the receiving rsync is not being run as the super-user, rsync silently skips creating the device files (see also the --super and --fake-super options).

By default, rsync generates a "non-regular file" warning for each device file encountered when this option is not set. You can silence the warning by specifying --info=nonreg0.

--specials

This option causes rsync to transfer special files, such as named sockets and fifos. If the receiving rsync is not being run as the super-user, rsync silently skips creating the special files (see also the --super and --fake-super options).

By default, rsync generates a "non-regular file" warning for each special file encountered when this option is not set. You can silence the warning by specifying --info=nonreg0.

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

--copy-devices
This tells rsync to treat a device on the sending side as a regular file, allowing it to be copied to a normal destination file (or another device if --write-devices was also specified).

This option is refused by default by an rsync daemon.

--write-devices
This tells rsync to treat a device on the receiving side as a regular file, allowing the writing of file data into a device.

This option implies the --inplace option.

Be careful using this, as you should know what devices are present on the receiving side of the transfer, especially when running rsync as root.

This option is refused by default by an rsync daemon.

--times, -t
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 --ignore-times (-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).

A modern rsync that is using transfer protocol 30 or 31 conveys a modify time using up to 8-bytes. If rsync is forced to speak an older protocol (perhaps due to the remote rsync being older than 3.0.0) a modify time is conveyed using 4-bytes. Prior to 3.2.7, these shorter values could convey a date range of 13-Dec-1901 to 19-Jan-2038. Beginning with 3.2.7, these 4-byte values now convey a date range of 1-Jan-1970 to 7-Feb-2106. If you have files dated older than 1970, make sure your rsync executables are upgraded so that the full range of dates can be conveyed.

--atimes, -U

This tells rsync to set the access (use) times of the destination files to the same value as the source files.

If repeated, it also sets the --open-noatime option, which can help you to make the sending and receiving systems have the same access times on the transferred files without needing to run rsync an extra time after a file is transferred.

Note that some older rsync versions (prior to 3.2.0) may have been built with a pre-release --atimes patch that does not imply --open-noatime when this option is repeated.

--open-noatime

This tells rsync to open files with the O_NOATIME flag (on systems that support it) to avoid changing the access time of the files that are being transferred. If your OS does not support the O_NOATIME flag then rsync will silently ignore this option. Note also that some filesystems are mounted to avoid updating the atime on read access even without the O_NOATIME flag being set.

--crtimes, -N,

This tells rsync to set the create times (newness) of the destination files to the same value as the source files.

--omit-dir-times, -O

This tells rsync to omit directories when it is preserving modification, access, and create 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.

This option also has the side-effect of avoiding early creation of missing sub-directories when incremental recursion is enabled, as discussed in the --inc-recursive section.

--omit-link-times, -J

This tells rsync to omit symlinks when it is preserving modification, access, and create times.

--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 --group option, and copying devices via the --devices op-

tion. 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, use the --remote-option (-M) option:

rsync -av -M--fake-super /src/ host:/dest/

For a local copy, this option affects both the source and the destination. If you wish a local copy to enable this option just for the destination files, specify -M--fake-super. If you wish a local copy to enable this option just for the source files, combine --fake-super with -M--super.

This option is overridden by both --super and --no-super.

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

--sparse, -S

Try to handle sparse files efficiently so they take up less space on the destination. If combined with --inplace the file created might not end up with sparse blocks with some combinations of kernel version and/or filesystem type. If --whole-file is in effect (e.g. for a local copy) then it will always work because rsync truncates the file prior to writing out the up-

dated version.

Note that versions of rsync older than 3.1.3 will reject the combination of --sparse and --inplace.

--preallocate
This tells the receiver to allocate each destination file to its eventual size before writing data to the file. Rsync will only use the real filesystem-level preallocation support provided by Linux's fallocate(2) system call or Cygwin's posix_fallocate(3), not the slow glibc implementation that writes a null byte into each block.

Without this option, larger files may not be entirely contiguous on the filesystem, but with this option rsync will probably copy more slowly. If the destination is not an extent-supporting filesystem (such as ext4, xfs, NTFS, etc.), this option may have no positive effect at all.

If combined with --sparse, the file will only have sparse blocks (as opposed to allocated sequences of null bytes) if the kernel version and filesystem type support creating holes in the allocated data.

--dry-run, -n
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 --verbose (-v) and/or --itemize-changes (-i) 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 should be mostly unchanged, 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 were needed.

--whole-file, -W
This option disables rsync's delta-transfer algorithm, which causes all transferred files to be sent whole. 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, but only if no batch-writing option is in effect.

--no-whole-file, --no-W
Disable whole-file updating when it is enabled by default for a local transfer. This usually slows rsync down, but it can be useful if you are trying to minimize the writes to the destination file (if combined with --inplace) or for testing the checksum-based update algorithm.

See also the --whole-file option.

--checksum-choice=STR, --cc=STR
This option overrides the checksum algorithms. If one algorithm name is specified, it is used for both the transfer checksums and (assuming --checksum is specified) the pre-transfer checksums. If two comma-separated names are supplied, the first name affects the transfer checksums, and the second name affects the pre-transfer checksums (-c).

The checksum options that you may be able to use are:

- auto (the default automatic choice)
- xxh128
- xxh3
- xxh64 (aka xxhash)
- md5
- md4
- sha1
- none

Run rsync --version to see the default checksum list compiled into your version (which may differ from the list above).

If "none" is specified for the first (or only) name, the --whole-file option is forced on and no checksum verification is performed on the transferred data. If "none" is specified for the second (or only) name, the --checksum option cannot be used.

The "auto" option is the default, where rsync bases its algorithm choice on a negotiation between the client and the server as follows:

When both sides of the transfer are at least 3.2.0, rsync chooses the first algorithm in the client's list of choices that is also in the server's list of choices. If no common checksum choice is found, rsync exits with an error. If the remote rsync is too old to support checksum negotiation, a value is chosen based on the protocol version (which chooses between MD5 and various flavors of MD4 based on protocol age).

The default order can be customized by setting the environment variable RSYNC_CHECKSUM_LIST to a space-separated list of acceptable checksum names. If the string contains a "&" character, it is separated into the "client string & server string", otherwise the same string applies to both. If the string (or string portion) contains no non-whitespace characters, the default checksum list is used. This method does not allow you to specify the transfer checksum separately from the pre-transfer checksum, and it discards "auto" and all unknown checksum names. A list with only invalid names results in a failed negotiation.

The use of the --checksum-choice option overrides this environment list.

--one-file-system, -x

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.

--ignore-non-existing, --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, so don't expect any exclude side effects.

--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 --ignore-non-existing.

This option is a TRANSFER RULE, so don't expect any exclude side effects.

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.

When --info=skip2 is used rsync will output "FILENAME exists (INFO)" messages where the INFO indicates one of "type change", "sum change" (requires -c), "file change" (based on the quick check), "attr change", or "uptodate". Using --info=skip1 (which is also implied by 2 -v options) outputs the exists message without the INFO suffix.

--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.

Note that you should only use this option on source files that are quiescent. If you are using this to move files that show up in a particular directory over to another host, make sure that the finished files get renamed into the source directory, not directly written into it, so that rsync can't possibly transfer a file that is not yet fully written. If you can't first write the files into a different directory, you should use a naming

idiom that lets rsync avoid transferring files that are not yet finished (e.g. name the file "foo.new" when it is written, rename it to "foo" when it is done, and then use the option --exclude='*.new' for the rsync transfer).

Starting with 3.1.0, rsync will skip the sender-side removal (and output an error) if the file's size or modify time has not stayed unchanged.

Starting with 3.2.6, a local rsync copy will ensure that the sender does not remove a file the receiver just verified, such as when the user accidentally makes the source and destination directory the same path.

--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 (-n) option 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 from 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, or 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.

See also the --delete-delay option that might be a faster choice for those that just want the deletions to occur at the end of the transfer.

--delete-excluded
This option turns any unqualified exclude/include rules into server-side rules that do not affect the receiver's deletions.

By default, an exclude or include has both a server-side effect (to "hide" and "show" files when building the server's file list) and a receiver-side effect (to "protect" and "risk" files when deletions are occurring). Any rule that has no modifier to specify what sides it is executed on will be instead treated as if it were a server-side rule only, avoiding any "protect" effects of the rules.

A rule can still apply to both sides even with this option specified if the rule is given both the sender & receiver modifier letters (e.g., -f'-sr foo'). Receiver-side protect/risk rules can also be explicitly specified to limit the deletions. This saves you from having to edit a bunch of -f'- foo' rules into -f'-s foo' (aka -f'H foo') rules (not to mention the corresponding includes).

See the FILTER RULES section for more information. See --delete (which is implied) for more details on deletion.

--ignore-missing-args
When rsync is first processing the explicitly requested source files (e.g. command-line arguments or --files-from entries), it is normally an error if the file cannot be found. This option suppresses that error, and does not try to transfer the file. This does not affect subsequent vanished-file errors if a file was initially found to be present and later is no longer there.

--delete-missing-args
This option takes the behavior of the (implied) --ignore-missing-args option a step farther: each missing arg will become a deletion request of the corresponding destination file on the receiving side (should it exist). If the destination file is a non-empty directory, it will only be successfully deleted if --force-delete or --delete are in effect. Other than that, this option is independent of any other type of delete processing.

The missing source files are represented by special file-list entries which display as a "*missing" entry in the --list-only output.

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

--force-delete, --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 that some older rsync versions used to require --force 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, all further deletions are skipped through the end of the transfer. At the end, rsync outputs a warning (including a count of the skipped deletions) and exits with an error code of 25 (unless some more important error condition also occurred).

Beginning with 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 really old 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. A numeric value can be suffixed with a string to indicate the numeric units or left unqualified to

specify bytes. Feel free to use a fractional value along with the units, such as --max-size=1.5m.

This option is a TRANSFER RULE, so don't expect any exclude side effects.

The first letter of a units string can be B (bytes), K (kilo), M (mega), G (giga), T (tera), or P (peta). If the string is a single char or has "ib" added to it (e.g. "G" or "GiB") then the units are multiples of 1024. If you use a two-letter suffix that ends with a "B" (e.g. "kb") then you get units that are multiples of 1000. The string's letters can be any mix of upper and lower-case that you want to use.

Finally, if the string ends with either "+1" or "-1", it is offset by one byte in the indicated direction. The largest possible value is usually 8192P-1.

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

Note that rsync versions prior to 3.1.0 did not allow --max-size=0.

--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 info.

Note that rsync versions prior to 3.1.0 did not allow --min-size=0.

--max-alloc=SIZE

By default rsync limits an individual malloc/realloc to about 1GB in size. For most people this limit works just fine and prevents a protocol error causing rsync to request massive amounts of memory. However, if you have many millions of files in a transfer, a large amount of server memory, and you don't want to split up your transfer into multiple parts, you can increase the per-allocation limit to something larger and rsync will consume more memory.

Keep in mind that this is not a limit on the total size of allocated memory. It is a sanity-check value for each individual allocation.

See the --max-size option for a description of how SIZE can be specified. The default suffix if none is given is bytes.

Beginning in 3.2.3, a value of 0 specifies no limit.

You can set a default value using the environment variable RSYNC_MAX_ALLOC using the same SIZE values as supported by this option. If the remote rsync doesn't understand the --max-alloc option, you can override an environmental value by specifying --max-alloc=1g, which will make rsync avoid sending the option to the remote side (because "1G" is the default).

--block-size=SIZE, -B

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.

Beginning in 3.2.3 the SIZE can be specified with a suffix as detailed in the --max-size option. Older versions only accepted a byte count.

--rsh=COMMAND, -e

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 USING RSYNC-DAEMON FEATURES VIA A REMOTE-SHELL CONNECTION section above.

Beginning with rsync 3.2.0, the RSYNC_PORT environment variable will be set when a daemon connection is being made via a remote-shell connection. It is set to 0 if the default daemon port is being assumed, or it is set to the value of the rsync port that was specified via either the --port option or a non-empty port value in an rsync:// URL. This allows the script to discern if a non-default port is being requested, allowing for things such as an SSL or stunnel helper script to connect to a default or alternate port.

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 back-slashes). 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/

--remote-option=OPTION, -M
This option is used for more advanced situations where you want certain effects to be limited to one side of the transfer only. For instance, if you want to pass --log-file=FILE and --fake-super to the remote system, specify it like this:

rsync -av -M --log-file=foo -M--fake-super src/ dest/

If you want to have an option affect only the local side of a transfer when it normally affects both sides, send its negation to the remote side. Like this:

rsync -av -x -M--no-x src/ dest/

Be cautious using this, as it is possible to toggle an option that will cause rsync to have a different idea about what data to expect next over the socket, and that will make it fail in a cryptic fashion.

Note that you should use a separate -M option for each remote option you want to pass. On older rsync versions, the presence of any spaces in the remote-option arg could cause it to be split into separate remote args, but this requires the use of --old-args in a modern rsync.

When performing a local transfer, the "local" side is the sender and the "remote" side is the receiver.

Note some versions of the popt option-parsing library have a bug in them that prevents you from using an adjacent arg with an equal in it next to a short option letter (e.g. -M--log-file=/tmp/foo). If this bug affects your version of popt, you can use the version of popt that is included with rsync.

--cvs-exclude, -C

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/ .hg/ .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.

--filter=RULE, -f

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 specifies an exclude rule and does not allow the full rule-parsing syntax of normal filter rules. This is equivalent to specifying -f'- PATTERN'.

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 are ignored, as are whole-line comments that start with ';' or '#' (filename rules that contain those characters are unaffected).

If a line begins with "- " (dash, space) or "+ " (plus, space), then the type of rule is being explicitly specified as an exclude or an include (respectively). Any rules without such a prefix are taken to be an exclude.

If a line consists of just "!", then the current filter rules are cleared before adding any further rules.

If FILE is '-', the list will be read from standard input.

--include=PATTERN
This option is a simplified form of the --filter option that specifies an include rule and does not allow the full rule-parsing syntax of normal filter rules. This is equivalent to specifying -f'+ PATTERN'.

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 are ignored, as are whole-line comments that start with ';' or '#' (filename rules that contain those characters are unaffected).

If a line begins with "- " (dash, space) or "+ " (plus, space), then the type of rule is being explicitly specified as an ex-

clude or an include (respectively). Any rules without such a prefix are taken to be an include.

If a line consists of just "!", then the current filter rules are cleared before adding any further rules.

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) -r 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.

If the --iconv and --secluded-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.

NOTE: sorting the list of files in the --files-from input helps rsync to be more efficient, as it will avoid re-visiting the path elements that are shared between adjacent entries. If the input is not sorted, some path elements (implied directories) may end up being scanned multiple times, and rsync will eventually unduplicate them after they get turned into file-list elements.

--from0, -0

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).

--old-args

This option tells rsync to stop trying to protect the arg values on the remote side from unintended word-splitting or other misinterpretation. It also allows the client to treat an empty arg as a "." instead of generating an error.

The default in a modern rsync is for "shell-active" characters (including spaces) to be backslash-escaped in the args that are sent to the remote shell. The wildcard characters *, ?, [, &]

are not escaped in filename args (allowing them to expand into multiple filenames) while being protected in option args, such as --usermap.

If you have a script that wants to use old-style arg splitting in its filenames, specify this option once. If the remote shell has a problem with any backslash escapes at all, specify this option twice.

You may also control this setting via the RSYNC_OLD_ARGS environment variable. If it has the value "1", rsync will default to a single-option setting. If it has the value "2" (or more), rsync will default to a repeated-option setting. If it is "0", you'll get the default escaping behavior. The environment is always overridden by manually specified positive or negative options (the negative is --no-old-args).

Note that this option also disables the extra safety check added in 3.2.5 that ensures that a remote sender isn't including extra top-level items in the file-list that you didn't request. This side-effect is necessary because we can't know for sure what names to expect when the remote shell is interpreting the args.

This option conflicts with the --secluded-args option.

--secluded-args, -s

This option sends all filenames and most options to the remote rsync via the protocol (not the remote shell command line) which avoids letting the remote shell modify them. Wildcards are expanded on the remote host by rsync instead of a shell.

This is similar to the default backslash-escaping of args that was added in 3.2.4 (see --old-args) in that it prevents things like space splitting and unwanted special-character side-effects. However, it has the drawbacks of being incompatible with older rsync versions (prior to 3.0.0) and of being refused by restricted shells that want to be able to inspect all the option values for safety.

This option is useful for those times that you need the argument's character set to be converted for the remote host, if the remote shell is incompatible with the default backslash-escpaing method, or there is some other reason that you want the majority of the options and arguments to bypass the command-line of the remote shell.

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

You may also control this setting via the RSYNC_PROTECT_ARGS environment variable. If it has a non-zero value, this setting will be enabled by default, otherwise it will be disabled by default. Either state is overridden by a manually specified positive or negative version of this option (note that --no-s and --no-secluded-args are the negative versions). This environment variable is also superseded by a non-zero RSYNC_OLD_ARGS export.

This option conflicts with the --old-args option.

This option used to be called --protect-args (before 3.2.6) and that older name can still be used (though specifying it as -s is always the easiest and most compatible choice).

--trust-sender

This option disables two extra validation checks that a local client performs on the file list generated by a remote sender. This option should only be used if you trust the sender to not put something malicious in the file list (something that could possibly be done via a modified rsync, a modified shell, or some other similar manipulation).

Normally, the rsync client (as of version 3.2.5) runs two extra validation checks when pulling files from a remote rsync:

- It verifies that additional arg items didn't get added at the top of the transfer.

- It verifies that none of the items in the file list are names that should have been excluded (if filter rules were specified).

Note that various options can turn off one or both of these checks if the option interferes with the validation. For instance:

- Using a per-directory filter file reads filter rules that only the server knows about, so the filter checking is disabled.

- Using the --old-args option allows the sender to manipulate the requested args, so the arg checking is disabled.

o Reading the files-from list from the server side means that the client doesn't know the arg list, so the arg checking is disabled.

o Using --read-batch disables both checks since the batch file's contents will have been verified when it was created.

This option may help an under-powered client server if the extra pattern matching is slowing things down on a huge transfer. It can also be used to work around a currently-unknown bug in the verification logic for a transfer from a trusted sender.

When using this option it is a good idea to specify a dedicated destination directory, as discussed in the MULTI-HOST SECURITY section.

--copy-as=USER[:GROUP]

This option instructs rsync to use the USER and (if specified after a colon) the GROUP for the copy operations. This only works if the user that is running rsync has the ability to change users. If the group is not specified then the user's default groups are used.

This option can help to reduce the risk of an rsync being run as root into or out of a directory that might have live changes happening to it and you want to make sure that root-level read or write actions of system files are not possible. While you could alternatively run all of rsync as the specified user, sometimes you need the root-level host-access credentials to be used, so this allows rsync to drop root for the copying part of the operation after the remote-shell or daemon connection is established.

The option only affects one side of the transfer unless the transfer is local, in which case it affects both sides. Use the --remote-option to affect the remote side, such as -M--copy-as=joe. For a local transfer, the lsh (or lsh.sh) support file provides a local-shell helper script that can be used to allow a "localhost:" or "lh:" host-spec to be specified without needing to setup any remote shells, allowing you to specify remote options that affect the side of the transfer that is using the host-spec (and using hostname "lh" avoids the overriding of the remote directory to the user's home dir).

For example, the following rsync writes the local files as user "joe":

sudo rsync -aiv --copy-as=joe host1:backups/joe/ /home/joe/

This makes all files owned by user "joe", limits the groups to those that are available to that user, and makes it impossible for the joe user to do a timed exploit of the path to induce a change to a file that the joe user has no permissions to change.

The following command does a local copy into the "dest/" dir as user "joe" (assuming you've installed support/lsh into a dir on your $PATH):

sudo rsync -aive lsh -M--copy-as=joe src/ lh:dest/

--temp-dir=DIR, -T

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 tempo-rary file in the same directory as the associated destination file. Beginning with rsync 3.1.1, the temp-file names inside the specified DIR will not be prefixed with an extra dot (though they will still have a random suffix added).

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 di-rectory 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 di-rectory, 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-up-dates 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.)

--fuzzy, -y

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.

If the option is repeated, the fuzzy scan will also be done in any matching alternate destination directories that are specified via --compare-dest, --copy-dest, or --link-dest.

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. This option is typically used to copy into an empty (or newly created) directory.

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 DIRs 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.

NOTE: beginning with version 3.1.0, rsync will remove a file from a non-empty destination hierarchy if an exact match is found in one of the compare-dest hierarchies (making the end result more closely match a fresh copy).

--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 DIRs 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 files 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 (there is a limit of 20 such directories). 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 DIRs will be selected to try to speed up the transfer.

This option works best when copying into an empty destination hierarchy, as existing files may get their attributes tweaked, and that can affect alternate destination files via hard-links.

Also, itemizing of changes can get a bit muddled. Note that prior to version 3.1.0, an alternate-directory exact match would never be found (nor linked into the destination) when a destination file already exists.

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 --owner (-o) was specified (or implied). You can work-around this bug by avoiding the -o option (or using --no-o) when sending to an old rsync.

--compress, -z

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.

Rsync supports multiple compression methods and will choose one for you unless you force the choice using the --compress-choice (--zc) option.

Run rsync --version to see the default compress list compiled into your version.

When both sides of the transfer are at least 3.2.0, rsync chooses the first algorithm in the client's list of choices that is also in the server's list of choices. If no common compress choice is found, rsync exits with an error. If the remote rsync is too old to support checksum negotiation, its list is assumed to be "zlib".

The default order can be customized by setting the environment variable RSYNC_COMPRESS_LIST to a space-separated list of acceptable compression names. If the string contains a "&" character, it is separated into the "client string & server string", otherwise the same string applies to both. If the string (or string portion) contains no non-whitespace characters, the default compress list is used. Any unknown compression names are discarded from the list, but a list with only invalid names re-

sults in a failed negotiation.

There are some older rsync versions that were configured to reject a -z option and require the use of -zz because their compression library was not compatible with the default zlib compression method. You can usually ignore this weirdness unless the rsync server complains and tells you to specify -zz.

--compress-choice=STR, --zc=STR

This option can be used to override the automatic negotiation of the compression algorithm that occurs when --compress is used. The option implies --compress unless "none" was specified, which instead implies --no-compress.

The compression options that you may be able to use are:

- zstd
- lz4
- zlibx
- zlib
- none

Run rsync --version to see the default compress list compiled into your version (which may differ from the list above).

Note that if you see an error about an option named --old-compress or --new-compress, this is rsync trying to send the --compress-choice=zlib or --compress-choice=zlibx option in a backward-compatible manner that more rsync versions understand. This error indicates that the older rsync version on the server will not allow you to force the compression type.

Note that the "zlibx" compression algorithm is just the "zlib" algorithm with matched data excluded from the compression stream (to try to make it more compatible with an external zlib implementation).

--compress-level=NUM, --zl=NUM

Explicitly set the compression level to use (see --compress, -z) instead of letting it default. The --compress option is implied as long as the level chosen is not a "don't compress" level for the compression algorithm that is in effect (e.g. zlib compression treats level 0 as "off").

The level values vary depending on the checksum in effect. Because rsync will negotiate a checksum choice by default (when the remote rsync is new enough), it can be good to combine this option with a --compress-choice (--zc) option unless you're sure of the choice in effect. For example:

rsync -aiv --zc=zstd --zl=22 host:src/ dest/

For zlib & zlibx compression the valid values are from 1 to 9 with 6 being the default. Specifying --zl=0 turns compression off, and specifying --zl=-1 chooses the default level of 6.

For zstd compression the valid values are from -131072 to 22 with 3 being the default. Specifying 0 chooses the default of 3.

For lz4 compression there are no levels, so the value is always 0.

If you specify a too-large or too-small value, the number is silently limited to a valid value. This allows you to specify something like --zl=999999999 and be assured that you'll end up with the maximum compression level no matter what algorithm was chosen.

If you want to know the compression level that is in effect, specify --debug=nstr to see the "negotiated string" results. This will report something like "Client compress: zstd (level 3)" (along with the checksum choice in effect).

--skip-compress=LIST

NOTE: no compression method currently supports per-file compression changes, so this option has no effect.

Override the list of file suffixes that will be compressed as little as possible. Rsync sets the compression level on a per-file basis based on the file's suffix. If the compression algorithm has an "off" level, then no compression occurs for those files. Other algorithms that support changing the streaming level on-the-fly will have the level minimized to reduces the CPU usage as much as possible for a matching file.

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 files 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, and '-' has no special meaning).

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 file suffixes in the skip-compress list in this version of rsync are:

3g2 3gp 7z aac ace apk avi bz2 deb dmg ear f4v flac flv gpg gz iso jar jpeg jpg lrz lz lz4 lzma lzo m1a m1v m2a m2ts m2v m4a m4b m4p m4r m4v mka mkv mov mp1 mp2 mp3 mp4 mpa mpeg mpg mpv mts odb odf odg odi odm odp ods odt oga ogg ogm ogv ogx opus otg oth otp ots ott oxt png qt rar rpm rz rzip spx squashfs sxc sxd sxg sxm sxw sz tbz tbz2 tgz tlz ts txz tzo vob war webm webp xz z zip zst

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 use chroot setting in the rsyncd.conf manpage for some comments 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.

--usermap=STRING, --groupmap=STRING

These options allow you to specify users and groups that should be mapped to other values by the receiving side. The STRING is one or more FROM:TO pairs of values separated by commas. Any matching FROM value from the sender is replaced with a TO value from the receiver. You may specify usernames or user IDs for the FROM and TO values, and the FROM value may also be a wild-card string, which will be matched against the sender's names (wild-cards do NOT match against ID numbers, though see below for why a '*' matches everything). You may instead specify a range of ID numbers via an inclusive range: LOW-HIGH. For example:

--usermap=0-99:nobody,wayne:admin,*:normal --groupmap=usr:1,1:usr

The first match in the list is the one that is used. You should specify all your user mappings using a single --usermap option, and/or all your group mappings using a single --groupmap option.

Note that the sender's name for the 0 user and group are not transmitted to the receiver, so you should either match these values using a 0, or use the names in effect on the receiving side (typically "root"). All other FROM names match those in use on the sending side. All TO names match those in use on the receiving side.

Any IDs that do not have a name on the sending side are treated as having an empty name for the purpose of matching. This allows them to be matched via a "*" or using an empty name. For instance:

--usermap=:nobody --groupmap=*:nobody

When the --numeric-ids option is used, the sender does not send any names, so all the IDs are treated as having an empty name. This means that you will need to specify numeric FROM values if you want to map these nameless IDs to different values.

For the --usermap option to work, the receiver will need to be running as a super-user (see also the --super and --fake-super options). For the --groupmap option to work, the receiver will need to have permissions to set that group.

Starting with rsync 3.2.4, the --usermap option implies the --owner (-o) option while the --groupmap option implies the --group (-g) option (since rsync needs to have those options enabled for the mapping options to work).

An older rsync client may need to use -s to avoid a complaint about wildcard characters, but a modern rsync handles this automatically.

--chown=USER:GROUP

This option forces all files to be owned by USER with group GROUP. This is a simpler interface than using --usermap & --groupmap directly, but it is implemented using those options internally so they cannot be mixed. If either the USER or GROUP is empty, no mapping for the omitted user/group will occur. If GROUP is empty, the trailing colon may be omitted, but if USER is empty, a leading colon must be supplied.

If you specify "--chown=foo:bar", this is exactly the same as specifying "--usermap=*:foo --groupmap=*:bar", only easier (and with the same implied --owner and/or --group options).

An older rsync client may need to use -s to avoid a complaint about wildcard characters, but a modern rsync handles this automatically.

--timeout=SECONDS

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=SECONDS

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=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 the daemon version of the --address option.

--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 the daemon version of the --port option.

--sockopts=OPTIONS

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 manpage 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.

See also the daemon version of the --sockopts option.

--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.)

--outbuf=MODE

This sets the output buffering mode. The mode can be None (aka Unbuffered), Line, or Block (aka Full). You may specify as little as a single letter for the mode, and use upper or lower case.

The main use of this option is to change Full buffering to Line buffering when rsync's output is going to a file or pipe.

--itemize-changes, -i

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 YXcstpoguaxf, 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:

o 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 indicate if some attributes of the file have changed, as follows:

- "." - the attribute is unchanged.

- "+" - the file is newly created.

- " " - all the attributes are unchanged (all dots turn to spaces).

- "?" - the change is unknown (when the remote rsync is old).

- A letter indicates an attribute is being updated.

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).

-

  - A u|n|b indicates the following information:

    u means the access (use) time is different and is being updated to the sender's value (requires --atimes)

  - n means the create time (newness) is different and is being updated to the sender's value (requires --crtimes)

  - b means that both the access and create times are being updated

- The a means that the ACL information is being changed.

- The x means that the extended attribute information is being 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 either --info=name or -v is specified (this tells you just 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 implies the --info=name option, which 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 an example command that requests the remote side to log what is happening:

rsync -av --remote-option=--log-file=/tmp/rlog src/ dest/

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

See also the daemon version of the --log-file option.

--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'.

See also the daemon version of the --log-file-format option.

--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. This option is equivalent to --info=stats2 if combined with 0 or 1 -v options, or --info=stats3 if combined with 2 or more -v options.

The current statistics are as follows:

- Number of files is the count of all "files" (in the generic sense), which includes directories, symlinks, etc. The total count will be followed by a list of counts by filetype (if the total is non-zero). For example: "(reg: 5, dir: 3, link: 2, dev: 1, special: 1)" lists the totals for regular files, directories, symlinks, devices, and special files. If any of value is 0, it is completely omitted from the list.

- Number of created files is the count of how many "files" (generic sense) were created (as opposed to updated). The total count will be followed by a list of counts by filetype (if the total is non-zero).

- Number of deleted files is the count of how many "files" (generic sense) were deleted. The total count will be followed by a list of counts by filetype (if the total is non-zero). Note that this line is only output if deletions are in effect, and only if protocol 31 is being used (the default for rsync 3.1.x).

- Number of regular files transferred is the count of normal files that were updated via rsync's delta-transfer algorithm, which does not include dirs, symlinks, etc. Note that rsync 3.1.0 added the word "regular" into this

heading.

- 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-bit-output, -8

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).

--human-readable, -h

Output numbers in a more human-readable format. There are 3 possible levels:

1. output numbers with a separator between each set of 3 digits (either a comma or a period, depending on if the decimal point is represented by a period or a comma).

2. output numbers in units of 1000 (with a character suffix for larger units -- see below).

3. output numbers in units of 1024.

The default is human-readable level 1. Each -h option increases the level by one. You can take the level down to 0 (to output numbers as pure digits) by specifying the --no-human-readable (--no-h) option.

The unit letters that are appended in levels 2 and 3 are: K (kilo), M (mega), G (giga), T (tera), or P (peta). For example, a 1234567-byte file would output as 1.23M in level-2 (assuming that a period is your local decimal point).

Backward compatibility note: versions of rsync prior to 3.1.0 do not support human-readable level 1, and they default to level 0. Thus, specifying one or two -h options will behave in a comparable manner in old and new versions as long as you didn't specify a --no-h option prior to one or more -h options. See the --list-only option for one difference.

--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

This option modifies the behavior of the --partial option while also implying that it be enabled. This enhanced partial-file method puts any partially transferred files into the specified DIR instead of writing the partial file 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 files that are 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, but 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 it is needed, and then remove it again when the partial file is deleted. Note that this directory removal is only done for a relative pathname, as it is expected that an absolute path is to a directory that is reserved for partial-dir work.

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 this "perishable" exclude at the end of any other filter rules: -f '-p .rsync-partial/'

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, in-

stead of using --partial-dir=.rsync-tmp along with --progress, you could set RSYNC_PARTIAL_DIR=.rsync-tmp in your environment and then 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).

When a modern rsync resumes the transfer of a file in the partial-dir, that partial file is now updated in-place instead of creating yet another tmp-file copy (so it maxes out at dest + tmp instead of dest + partial + tmp). This requires both ends of the transfer to be at least version 3.2.0.

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 implies --no-inc-recursive since it needs the full file list in memory in order to be able to iterate over it at the end.

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

--prune-empty-dirs, -m

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.

This option can still leave empty directories on the receiving side if you make use of TRANSFER_RULES.

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. With a modern rsync this is the same as specifying --info=flist2,name,progress, but any user-supplied settings for those info flags takes precedence (e.g. --info=flist0 --progress).

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:

1,238,099 100% 146.38kB/s 0:00:08 (xfr#5, to-chk=169/396)

In this example, the file was 1,238,099 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.

In an incremental recursion scan, rsync won't know the total number of files in the file-list until it reaches the ends of the scan, but since it starts to transfer files during the scan, it will display a line with the text "ir-chk" (for incremental recursion check) instead of "to-chk" until the point that it

knows the full size of the list, at which point it will switch to using "to-chk". Thus, seeing "ir-chk" lets you know that the total count of files in the file list is still going to increase (and each time it does, the count of files left to check will increase by the number of the files added to the 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.

There is also a --info=progress2 option that outputs statistics based on the whole transfer, rather than individual files. Use this flag without outputting a filename (e.g. avoid -v or specify --info=name0) if you want to see how the transfer is doing without scrolling the screen with a lot of names. (You don't need to specify the --progress option in order to use --info=progress2.)

Finally, you can get an instant progress report by sending rsync a signal of either SIGINFO or SIGVTALRM. On BSD systems, a SIGINFO is generated by typing a Ctrl+T (Linux doesn't currently support a SIGINFO signal). When the client-side process receives one of those signals, it sets a flag to output a single progress report which is output when the current file transfer finishes (so it may take a little time if a big file is being handled when the signal arrives). A filename is output (if needed) followed by the --info=progress2 format of progress info. If you don't know which of the 3 rsync processes is the client process, it's OK to signal all of them (since the non-client processes ignore the signal).

CAUTION: sending SIGVTALRM to an older rsync (pre-3.2.0) will kill it.

--password-file=FILE
This option allows you to provide a password for accessing an rsync daemon via a file or via standard input if FILE is -. The file should contain just the password on the first line (all other lines are ignored). Rsync will exit with an error if FILE is world readable or if a root-run rsync command finds a non-root-owned file.

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 ef-

fect after the remote shell finishes its authentication (i.e. if you have also specified a password in the daemon's config file).

--early-input=FILE

This option allows rsync to send up to 5K of data to the "early exec" script on its stdin. One possible use of this data is to give the script a secret that can be used to mount an encrypted filesystem (which you should unmount in the the "post-xfer exec" script).

The daemon must be at least version 3.2.1.

--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 specify a single wild-card arg to try to infer this option. A safe example is:

rsync -av --list-only foo* dest/

This option always uses an output format that looks similar to this:

```
drwxrwxr-x          4,096 2022/09/30 12:53:11    support
-rw-rw-r--             80 2005/01/11 10:37:37    support/Makefile
```

The only option that affects this output style is (as of 3.1.0) the --human-readable (-h) option. The default is to output sizes as byte counts with digit separators (in a 14-character-width column). Specifying at least one -h option makes the sizes output with unit suffixes. If you want old-style byte-count sizes without digit separators (and an 11-character-width column) use --no-h.

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 be-

cause 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=RATE

This option allows you to specify the maximum transfer rate for the data sent over the socket, specified in units per second. The RATE value can be suffixed with a string to indicate a size multiplier, and may be a fractional value (e.g. --bwlimit=1.5m). If no suffix is specified, the value will be assumed to be in units of 1024 bytes (as if "K" or "KiB" had been appended). See the --max-size option for a description of all the available suffixes. A value of 0 specifies no limit.

For backward-compatibility reasons, the rate limit will be rounded to the nearest KiB unit, so no rate smaller than 1024 bytes per second is possible.

Rsync writes data over the socket in blocks, and this option both limits the size of the blocks that rsync writes, and tries to keep the average transfer rate at the requested limit. Some burstiness may be seen where rsync writes out a block of data and then sleeps to bring the average rate into compliance.

Due to the internal buffering of data, the --progress option may not be an accurate reflection on how fast the data is being sent. This is because some files can show up as being rapidly sent when the data is quickly buffered, while other can show up as very slow when the flushing of the output buffer occurs. This may be fixed in a future version.

See also the daemon version of the --bwlimit option.

--stop-after=MINS, (--time-limit=MINS)

This option tells rsync to stop copying when the specified number of minutes has elapsed.

For maximal flexibility, rsync does not communicate this option to the remote rsync since it is usually enough that one side of the connection quits as specified. This allows the option's use even when only one side of the connection supports it. You can tell the remote side about the time limit using --remote-option (-M), should the need arise.

The --time-limit version of this option is deprecated.

--stop-at=y-m-dTh:m
This option tells rsync to stop copying when the specified point in time has been reached. The date & time can be fully specified in a numeric format of year-month-dayThour:minute (e.g. 2000-12-31T23:59) in the local timezone. You may choose to separate the date numbers using slashes instead of dashes.

The value can also be abbreviated in a variety of ways, such as specifying a 2-digit year and/or leaving off various values. In all cases, the value will be taken to be the next possible point in time where the supplied information matches. If the value specifies the current time or a past time, rsync exits with an error.

For example, "1-30" specifies the next January 30th (at midnight local time), "14:00" specifies the next 2 P.M., "1" specifies the next 1st of the month at midnight, "31" specifies the next month where we can stop on its 31st day, and ":59" specifies the next 59th minute after the hour.

For maximal flexibility, rsync does not communicate this option to the remote rsync since it is usually enough that one side of the connection quits as specified. This allows the option's use even when only one side of the connection supports it. You can tell the remote side about the time limit using --remote-option (-M), should the need arise. Do keep in mind that the remote host may have a different default timezone than your local host.

--fsync
Cause the receiving side to fsync each finished file. This may slow down the transfer, but can help to provide peace of mind when updating critical files.

--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.

This option overrides the negotiated checksum & compress lists and always negotiates a choice based on old-school md5/md4/zlib choices. If you want a more modern choice, use the --checksum-choice (--cc) and/or --compress-choice (--zc) options.

--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 --secluded-args (-s) option, 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).

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

Tells rsync to prefer IPv4/IPv6 when creating sockets or running ssh. This affects sockets that rsync has direct control over, such as the outgoing socket when directly contacting an rsync daemon, as well as the forwarding of the -4 or -6 option to ssh when rsync can deduce that ssh is being used as the remote shell. For other remote shells you'll need to specify the "--rsh SHELL -4" option directly (or whatever IPv4/IPv6 hint options it uses).

See also the daemon version of these options.

If rsync was compiled without support for IPv6, the --ipv6 option will have no effect. The rsync --version output will contain "no IPv6" if 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 MD4 file checksum calculation (the more modern MD5 file checksums don't use a seed). 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 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) manpage for more details.

--address=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 and the client version of the --address option.

--bwlimit=RATE

This option allows you to specify the maximum transfer rate for the data the daemon sends over the socket. The client can still specify a smaller --bwlimit value, but no larger value will be allowed.

See the client version of the --bwlimit option 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).

--dparam=OVERRIDE, -M

This option can be used to set a daemon-config parameter when

starting up rsync in daemon mode. It is equivalent to adding the parameter at the end of the global settings prior to the first module's definition. The parameter names can be specified without spaces, if you so desire. For instance:

rsync --daemon -M pidfile=/path/rsync.pid

--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 client version of the --port option and the port global setting 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.

See also the client version of the --log-file option.

--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.

See also the client version of the --log-file-format option.

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

See also the client version of the --sockopts option.

--verbose, -v
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.

See also the client version of the --verbose option.

--ipv4, -4 or --ipv6, -6
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).

See also the client version of these options.

If rsync was compiled without support for IPv6, the --ipv6 option will have no effect. The rsync --version output will contain "no IPv6" if is the case.

--help, -h
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 custom control of several aspects of how files are handled:

- Control which files the sending side puts into the file list that describes the transfer hierarchy

- Control which files the receiving side protects from deletion when the file is not in the sender's file list

- Control which extended attribute names are skipped when copying xattrs

The rules are either directly specified via option arguments or they can be read in from one or more files. The filter-rule files can even be a part of the hierarchy of files being copied, affecting different parts of the tree in different ways.

SIMPLE INCLUDE/EXCLUDE RULES

We will first cover the basics of how include & exclude rules affect what files are transferred, ignoring any deletion side-effects. Filter rules mainly affect the contents of directories that rsync is "recursing" into, but they can also affect a top-level item in the transfer that was specified as a argument.

The default for any unmatched file/dir is for it to be included in the transfer, which puts the file/dir into the sender's file list. The use of an exclude rule causes one or more matching files/dirs to be left out of the sender's file list. An include rule can be used to limit the effect of an exclude rule that is matching too many files.

The order of the rules is important because the first rule that matches is the one that takes effect. Thus, if an early rule excludes a file, no include rule that comes after it can have any effect. This means that you must place any include overrides somewhere prior to the exclude that it is intended to limit.

When a directory is excluded, all its contents and sub-contents are also excluded. The sender doesn't scan through any of it at all, which can save a lot of time when skipping large unneeded sub-trees.

It is also important to understand that the include/exclude rules are applied to every file and directory that the sender is recursing into. Thus, if you want a particular deep file to be included, you have to make sure that none of the directories that must be traversed on the way down to that file are excluded or else the file will never be discovered to be included. As an example, if the directory "a/path" was given as a transfer argument and you want to ensure that the file "a/path/down/deep/wanted.txt" is a part of the transfer, then the sender must not exclude the directories "a/path", "a/path/down", or "a/path/down/deep" as it makes it way scanning through the file tree.

When you are working on the rules, it can be helpful to ask rsync to tell you what is being excluded/included and why. Specifying --debug=FILTER or (when pulling files) -M--debug=FILTER turns on level 1 of the FILTER debug information that will output a message any time that a file or directory is included or excluded and which rule it matched. Beginning in 3.2.4 it will also warn if a filter rule has trailing whitespace, since an exclude of "foo " (with a trailing space) will not exclude a file named "foo".

Exclude and include rules can specify wildcard PATTERN MATCHING RULES (similar to shell wildcards) that allow you to match things like a file suffix or a portion of a filename.

A rule can be limited to only affecting a directory by putting a trailing slash onto the filename.

SIMPLE INCLUDE/EXCLUDE EXAMPLE

With the following file tree created on the sending side:

mkdir x/
touch x/file.txt
mkdir x/y/
touch x/y/file.txt
touch x/y/zzz.txt
mkdir x/z/
touch x/z/file.txt

Then the following rsync command will transfer the file "x/y/file.txt" and the directories needed to hold it, resulting in the path "/tmp/x/y/file.txt" existing on the remote host:

rsync -ai -f'+ x/' -f'+ x/y/' -f'+ x/y/file.txt' -f'- *' x host:/tmp/

Aside: this copy could also have been accomplished using the -R option (though the 2 commands behave differently if deletions are enabled):

rsync -aiR x/y/file.txt host:/tmp/

The following command does not need an include of the "x" directory because it is not a part of the transfer (note the traililng slash). Running this command would copy just "/tmp/x/file.txt" because the "y" and "z" dirs get excluded:

rsync -ai -f'+ file.txt' -f'- *' x/ host:/tmp/x/

This command would omit the zzz.txt file while copying "x" and everything else it contains:

rsync -ai -f'- zzz.txt' x host:/tmp/

FILTER RULES WHEN DELETING

By default the include & exclude filter rules affect both the sender (as it creates its file list) and the receiver (as it creates its file lists for calculating deletions). If no delete option is in effect, the receiver skips creating the delete-related file lists. This two-sided default can be manually overridden so that you are only specifying sender rules or receiver rules, as described in the FILTER RULES IN DEPTH section.

When deleting, an exclude protects a file from being removed on the receiving side while an include overrides that protection (putting the file at risk of deletion). The default is for a file to be at risk -- its safety depends on it matching a corresponding file from the sender.

An example of the two-sided exclude effect can be illustrated by the copying of a C development directory between 2 systems. When doing a

touch-up copy, you might want to skip copying the built executable and the .o files (sender hide) so that the receiving side can build their own and not lose any object files that are already correct (receiver protect). For instance:

rsync -ai --del -f'- *.o' -f'- cmd' src host:/dest/

Note that using -f'-p *.o' is even better than -f'- *.o' if there is a chance that the directory structure may have changed. The "p" modifier is discussed in FILTER RULE MODIFIERS.

One final note, if your shell doesn't mind unexpanded wildcards, you could simplify the typing of the filter options by using an underscore in place of the space and leaving off the quotes. For instance, -f -_*.o -f -_cmd (and similar) could be used instead of the filter options above.

FILTER RULES IN DEPTH

Rsync supports old-style include/exclude rules and new-style filter rules. The older rules are specified using --include and --exclude as well as the --include-from and --exclude-from. These are limited in behavior but they don't require a "-" or "+" prefix. An old-style exclude rule is turned into a "- name" filter rule (with no modifiers) and an old-style include rule is turned into a "+ name" filter rule (with no modifiers).

Rsync builds an ordered list of filter rules as specified on the command-line and/or read-in from files. New style 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 (_). Any additional spaces and/or underscores are considered to be a part of the pattern name. Here are the available rule prefixes:

exclude, '-'
specifies an exclude pattern that (by default) is both a hide and a protect.

include, '+'
specifies an include pattern that (by default) is both a show and a risk.

merge, '.'
specifies a merge-file on the client side to read for more rules.

dir-merge, ':'
specifies a per-directory merge-file. Using this kind of filter rule requires that you trust the sending side's filter checking, so it has the side-effect mentioned under the --trust-sender option.

hide, 'H'
specifies a pattern for hiding files from the transfer. Equivalent to a sender-only exclude, so -f'H foo' could also be specified as -f'-s foo'.

show, 'S'
files that match the pattern are not hidden. Equivalent to a sender-only include, so -f'S foo' could also be specified as -f'+s foo'.

protect, 'P'
specifies a pattern for protecting files from deletion. Equivalent to a receiver-only exclude, so -f'P foo' could also be specified as -f'-r foo'.

risk, 'R'
files that match the pattern are not protected. Equivalent to a receiver-only include, so -f'R foo' could also be specified as -f'+r foo'.

clear, '!'
clears the current include/exclude list (takes no arg)

When rules are being read from a file (using merge or dir-merge), empty lines are ignored, as are whole-line comments that start with a '#' (filename rules that contain a hash character are unaffected).

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.

PATTERN MATCHING RULES

Most of the rules mentioned above take an argument that specifies what the rule should match. If rsync is recursing through a directory hierarchy, keep in mind that each pattern is matched against the name of every directory in the descent path as rsync finds the filenames to

send.

The matching rules for the pattern argument take several forms:

- If a pattern contains a / (not counting a trailing slash) or a "**" (which can match a slash), then the pattern is matched against the full pathname, including any leading directories within the transfer. If the pattern doesn't contain a (non-trailing) / or a "**", then it is matched only against the final component of the filename or pathname. For example, foo means that the final path component must be "foo" while foo/bar would match the last 2 elements of the path (as long as both elements are within the transfer).

- A pattern that ends with a / only matches a directory, not a regular file, symlink, or device.

- A pattern that starts with a / is anchored to the start of the transfer path instead of the end. For example, /foo/** or /foo/bar/** match only leading elements in the path. If the rule is read from a per-directory filter file, the transfer path being matched will begin at the level of the filter file instead of the top of the transfer. 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.

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 single character except a slash (/).

- a '*' matches zero or more non-slash characters.

- a '**' matches zero or more characters, including slashes.

- a '[' introduces a character class, such as [a-z] or [[:alpha:]], that must match one character.

- a trailing *** in the pattern is a shorthand that allows you to match a directory and all its contents using a single rule. For example, specifying "dir_name/***" will match both the "dir_name" directory (as if "dir_name/" had been specified) and everything in the directory (as if "dir_name/**" had been specified).

- a backslash can be used to escape a wildcard character, but it is only interpreted as an escape character if at least one wildcard character is present in the match pattern. For instance, the pattern "foo\bar" matches that single backslash literally, while the pattern "foo\bar*" would need to be changed to "foo\\bar*" to avoid the "\b" becoming just "b".

Here are some examples of exclude/include matching:

- Option -f'- *.o' would exclude all filenames ending with .o

- Option -f'- /foo' would exclude a file (or directory) named foo in the transfer-root directory

- Option -f'- foo/' would exclude any directory named foo

- Option -f'- foo/*/bar' would exclude any file/dir named bar which is at two levels below a directory named foo (if foo is in the transfer)

- Option -f'- /foo/**/bar' would exclude any file/dir named bar that was two or more levels below a top-level directory named foo (note that /foo/bar is not excluded by this)

- Options -f'+ */' -f'+ *.c' -f'- *' would include all directories and .c source files but nothing else

- Options -f'+ foo/' -f'+ foo/bar.c' -f'- *' would include only the foo directory and foo/bar.c (the foo directory must be explicitly included or it would be excluded by the "- *")

FILTER RULE MODIFIERS

The following modifiers are accepted after an include (+) or exclude (-) rule:

- A / specifies that the include/exclude rule should be matched against the absolute pathname of the current item. For example, -f'-/ /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, -f'-! */' 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 affects what files are put into the sender's file list. 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 --cvs-exclude (-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.

- An x indicates that a rule affects xattr names in xattr copy/delete operations (and is thus ignored when matching file/dir names). If no xattr-matching rules are specified, a default xattr filtering rule is used (see the --xattrs option).

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:

o A - specifies that the file should consist of only exclude patterns, with no other rule-parsing except for in-file comments.

o A + specifies that the file should consist of only include patterns, with no other rule-parsing except for in-file comments.

o 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.

o A e will exclude the merge-file name from the transfer; e.g. "dir-merge,e .rules" is like "dir-merge .rules" and "- .rules".

o An n specifies that the rules are not inherited by subdirectories.

o 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).

o 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 (except for the ! modifier, which would not be useful). 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. If the merge rule specifies sides to affect (via the s or r modifier or both), then the rules in the file must not specify sides (via a modifier or a rule prefix such as hide).

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
- foo*
```

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
```

TRANSFER RULES

In addition to the FILTER RULES that affect the recursive file scans that generate the file list on the sending and (when deleting) receiving sides, there are transfer rules. These rules affect which files the

generator decides need to be transferred without the side effects of an exclude filter rule. Transfer rules affect only files and never directories.

Because a transfer rule does not affect what goes into the sender's (and receiver's) file list, it cannot have any effect on which files get deleted on the receiving side. For example, if the file "foo" is present in the sender's list but its size is such that it is omitted due to a transfer rule, the receiving side does not request the file. However, its presence in the file list means that a delete pass will not remove a matching file named "foo" on the receiving side. On the other hand, a server-side exclude (hide) of the file "foo" leaves the file out of the server's file list, and absent a receiver-side exclude (protect) the receiver will remove a matching file named "foo" if deletions are requested.

Given that the files are still in the sender's file list, the --prune-empty-dirs option will not judge a directory as being empty even if it contains only files that the transfer rules omitted.

Similarly, a transfer rule does not have any extra effect on which files are deleted on the receiving side, so setting a maximum file size for the transfer does not prevent big files from being deleted.

Examples of transfer rules include the default "quick check" algorithm (which compares size & modify time), the --update option, the --max-size option, the --ignore-non-existing option, and a few others.

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 added to the transfer (instead of being noisily ignored), and the default handling is to recreate them 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 can also distinguish "safe" and "unsafe" symbolic links. An example where this might be used is a web site mirror that wishes to ensure that the rsync module that is copied 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 by the receiver. (Note that you must specify or imply --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 top of the transfer.

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 and directories (leaving no symlinks in the transfer for any other options to affect).

--copy-dirlinks
  Turn just symlinks to directories into real directories, leaving all other symlinks to be handled as described below.

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

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

--links --safe-links
  The receiver skips creating unsafe symlinks found in the transfer and creates the safe ones.

--links
Create all symlinks.

For the effect of --munge-links, see the discussion in that option's section.

Note that the --keep-dirlinks option does not effect symlinks in the transfer but instead affects how rsync treats a symlink to a directory that already exists on the receiving side. See that option's section for a warning.

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. Either:

an attempt was made to manipulate 64-bit files on a platform that cannot support them

- 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. First supported in 3.0.0.

RSYNC_OLD_ARGS

Specify a "1" if you want the --old-args option to be enabled by

default, a "2" (or more) if you want it to be enabled in the repeated-option state, or a "0" to make sure that it is disabled by default. When this environment variable is set to a non-zero value, it supersedes the RSYNC_PROTECT_ARGS variable.

This variable is ignored if --old-args, --no-old-args, or --secluded-args is specified on the command line.

First supported in 3.2.4.

RSYNC_PROTECT_ARGS
Specify a non-zero numeric value if you want the --secluded-args option to be enabled by default, or a zero value to make sure that it is disabled by default.

This variable is ignored if --secluded-args, --no-secluded-args, or --old-args is specified on the command line.

First supported in 3.1.0. Starting in 3.2.4, this variable is ignored if RSYNC_OLD_ARGS is set to a non-zero value.

RSYNC_RSH
This 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 --rsh (-e) option.

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

RSYNC_PASSWORD
This environment variable allows you to set the password for an rsync daemon connection, which avoids the password prompt. Note that this does not supply a password to a remote shell transport such as ssh (consult its documentation for how to do that).

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". If both are set, USER takes precedence.

RSYNC_PARTIAL_DIR
This environment variable specifies the directory to use for a --partial transfer without implying that partial transfers be

enabled. See the --partial-dir option for full details.

RSYNC_COMPRESS_LIST
This environment variable allows you to customize the negotiation of the compression algorithm by specifying an alternate order or a reduced list of names. Use the command rsync --version to see the available compression names. See the --compress option for full details.

RSYNC_CHECKSUM_LIST
This environment variable allows you to customize the negotiation of the checksum algorithm by specifying an alternate order or a reduced list of names. Use the command rsync --version to see the available checksum names. See the --checksum-choice option for full details.

RSYNC_MAX_ALLOC
This environment variable sets an allocation maximum as if you had used the --max-alloc option.

RSYNC_PORT
This environment variable is not read by rsync, but is instead set in its sub-environment when rsync is running the remote shell in combination with a daemon connection. This allows a script such as rsync-ssl to be able to know the port number that the user specified on the command line.

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

RSYNC_CONNECT_PROG
This environment variable is mainly used in debug setups to set the program to use when making a daemon connection. See CONNECTING TO AN RSYNC DAEMON for full details.

RSYNC_SHELL
This environment variable is mainly used in debug setups to set the program to use to run the program specified by RSYNC_CONNECT_PROG. See CONNECTING TO AN RSYNC DAEMON for full details.

FILES

/etc/rsyncd.conf or rsyncd.conf

SEE ALSO

rsync-ssl(1), rsyncd.conf(5), rrsync(1)

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 https://rsync.samba.org/.

VERSION

This manpage is current for version 3.2.7 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 General Public License. See the file COPYING for details.

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

The rsync github project is https://github.com/WayneD/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

Special 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 https://lists.samba.org/.