Rocky Enterprise Linux 9.2 Manual Pages on command 'rsyncd.conf.5'

$ man rsyncd.conf.5

rsyncd.conf(5) User Commands rsyncd.conf(5)

NAME

rsyncd.conf - configuration file for rsync in daemon mode

SYNOPSIS

rsyncd.conf

DESCRIPTION

The rsyncd.conf file is the runtime configuration file for rsync when run as an rsync daemon.

The rsyncd.conf file controls authentication, access, logging and available modules.

FILE FORMAT

The file consists of modules and parameters. A module begins with the name of the module in square brackets and continues until the next mod? ule begins. Modules contain parameters of the form name = value.

The file is line-based -- that is, each newline-terminated line repre? sents either a comment, a module name or a parameter.

Only the first equals sign in a parameter is significant. Whitespace before or after the first equals sign is discarded. Leading, trailing

and internal whitespace in module and parameter names is irrelevant. Leading and trailing whitespace in a parameter value is discarded. In? ternal whitespace within a parameter value is retained verbatim.

Any line beginning with a hash (#) is ignored, as are lines containing only whitespace. (If a hash occurs after anything other than leading whitespace, it is considered a part of the line's content.)

Any line ending in a \ is "continued" on the next line in the customary UNIX fashion.

The values following the equals sign in parameters are all either a string (no quotes needed) or a boolean, which may be given as yes/no, 0/1 or true/false. Case is not significant in boolean values, but is preserved in string values.

LAUNCHING THE RSYNC DAEMON

The rsync daemon is launched by specifying the --daemon option to rsync.

The daemon must run with root privileges if you wish to use chroot, to bind to a port numbered under 1024 (as is the default 873), or to set file ownership. Otherwise, it must just have permission to read and write the appropriate data, log, and lock files.

You can launch it either via inetd, as a stand-alone daemon, or from an rsync client via a remote shell. If run as a stand-alone daemon then just run the command "rsync --daemon" from a suitable startup script.

When run via inetd you should add a line like this to /etc/services:

```
rsync           873/tcp
```

and a single line something like this to /etc/inetd.conf:

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

Replace "/usr/bin/rsync" with the path to where you have rsync in? stalled on your system. You will then need to send inetd a HUP signal to tell it to reread its config file.

Note that you should not send the rsync daemon a HUP signal to force it to reread the rsyncd.conf file. The file is re-read on each client con? nection.

GLOBAL PARAMETERS

The first parameters in the file (before a [module] header) are the global parameters. Rsync also allows for the use of a "[global]" mod?ule name to indicate the start of one or more global-parameter sections (the name must be lower case).

You may also include any module parameters in the global part of the config file in which case the supplied value will override the default for that parameter.

You may use references to environment variables in the values of param?eters. String parameters will have %VAR% references expanded as late as possible (when the string is first used in the program), allowing for the use of variables that rsync sets at connection time, such as RSYNC_USER_NAME. Non-string parameters (such as true/false settings) are expanded when read from the config file. If a variable does not exist in the environment, or if a sequence of characters is not a valid reference (such as an un-paired percent sign), the raw characters are passed through unchanged. This helps with backward compatibility and safety (e.g. expanding a non-existent %VAR% to an empty string in a path could result in a very unsafe path). The safest way to insert a literal % into a value is to use %%.

motd file

This parameter allows you to specify a "message of the day" to display to clients on each connect. This usually contains site information and any legal notices. The default is no motd file.

This can be overridden by the --dparam=motdfile=FILE command-line option when starting the daemon.

pid file

This parameter tells the rsync daemon to write its process ID to that file. The rsync keeps the file locked so that it can know when it is safe to overwrite an existing file.

The filename can be overridden by the --dparam=pidfile=FILE com?mand-line option when starting the daemon.

port You can override the default port the daemon will listen on by specifying this value (defaults to 873). This is ignored if the

daemon is being run by inetd, and is superseded by the --port command-line option.

address

You can override the default IP address the daemon will listen on by specifying this value. This is ignored if the daemon is being run by inetd, and is superseded by the --address command-line option.

socket options

This parameter can provide endless fun for people who like to tune their systems to the utmost degree. You can set all sorts of socket options which may make transfers faster (or slower!). Read the man page for the setsockopt() system call for details on some of the options you may be able to set. By default no special socket options are set. These settings can also be specified via the --sockopts command-line option.

listen backlog

You can override the default backlog value when the daemon lis? tens for connections. It defaults to 5.

MODULE PARAMETERS

After the global parameters you should define a number of modules, each module exports a directory tree as a symbolic name. Modules are ex? ported by specifying a module name in square brackets [module] followed by the parameters for that module. The module name cannot contain a slash or a closing square bracket. If the name contains whitespace, each internal sequence of whitespace will be changed into a single space, while leading or trailing whitespace will be discarded. Also, the name cannot be "global" as that exact name indicates that global parameters follow (see above).

As with GLOBAL PARAMETERS, you may use references to environment vari? ables in the values of parameters. See the GLOBAL PARAMETERS section for more details.

comment

This parameter specifies a description string that is displayed

next to the module name when clients obtain a list of available modules. The default is no comment.

path This parameter specifies the directory in the daemon's filesys? tem to make available in this module. You must specify this pa? rameter for each module in rsyncd.conf.

You may base the path's value off of an environment variable by surrounding the variable name with percent signs. You can even reference a variable that is set by rsync when the user con? nects. For example, this would use the authorizing user's name in the path:

path = /home/%RSYNC_USER_NAME%

It is fine if the path includes internal spaces -- they will be retained verbatim (which means that you shouldn't try to escape them). If your final directory has a trailing space (and this is somehow not something you wish to fix), append a trailing slash to the path to avoid losing the trailing whitespace.

use chroot

If "use chroot" is true, the rsync daemon will chroot to the "path" before starting the file transfer with the client. This has the advantage of extra protection against possible implemen? tation security holes, but it has the disadvantages of requiring super-user privileges, of not being able to follow symbolic links that are either absolute or outside of the new root path, and of complicating the preservation of users and groups by name (see below).

As an additional safety feature, you can specify a dot-dir in the module's "path" to indicate the point where the chroot should occur. This allows rsync to run in a chroot with a non-"/" path for the top of the transfer hierarchy. Doing this guards against unintended library loading (since those absolute paths will not be inside the transfer hierarchy unless you have used an unwise pathname), and lets you setup libraries for the chroot that are outside of the transfer. For example, specify?

ing "/var/rsync/./module1" will chroot to the "/var/rsync" di?
rectory and set the inside-chroot path to "/module1". If you
had omitted the dot-dir, the chroot would have used the whole
path, and the inside-chroot path would have been "/".
When both "use chroot" and "daemon chroot" are false, OR the in?
side-chroot path of "use chroot" is not "/", rsync will: (1)
munge symlinks by default for security reasons (see "munge sym?
links" for a way to turn this off, but only if you trust your
users), (2) substitute leading slashes in absolute paths with
the module's path (so that options such as --backup-dir, --com?
pare-dest, etc. interpret an absolute path as rooted in the mod?
ule's "path" dir), and (3) trim ".." path elements from args if
rsync believes they would escape the module hierarchy. The de?
fault for "use chroot" is true, and is the safer choice (espe?
cially if the module is not read-only).
When this parameter is enabled and the "name converter" parame?
ter is not set, the "numeric ids" parameter will default to be?
ing 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 nu?
meric 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 "ex?
clude" (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 ch? rooting.

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 authoriza? tion 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 with?

out 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-per? missions 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 hierar? chy). 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 sym? link'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 mod? ule's filenames are stored. If the client uses an --iconv op?

tion, 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 particu? lar 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 si? multaneous 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-mod? ule 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 over?

rides 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, lo? cal2, 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 inher? ited 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 ver? bose 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 --de?
bug=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 ex?
ceeded 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 down?
load 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 de? fault 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 set? tings).

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 con? tain a space. In either case any leading and/or trailing white? space 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 un? set 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 un? set, 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 administra? tive 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 in? cluded 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/in? clude. Only one "exclude" parameter can apply to a given mod? ule. See the "filter" parameter for a description of how ex? cluded files affect the daemon.

include

Use an "include" to override the effects of the "exclude" param? eter. 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 user?

names that will be allowed to connect to this module. The user? names 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 re? sponse authentication protocol is used for this exchange. The plain text usernames and passwords are stored in the file speci? fied 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, spec? ifying "@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 op? tions allow you to "deny" a user or a group, set the access to "ro" (read-only), or set the access to "rw" (read/write). Set? ting 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 exam? ple:

```
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 ac? cess. 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 ex? plains 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 han? dle 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 au? thenticating 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 com? ment 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 mod? ule 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 in? clude, 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 nor? mally not be readable by "other"; see "strict modes". If the file is not found or is rejected, no logins for a "user auth" 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 con? necting client's hostname and IP address. If none of the pat? terns 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 net? mask. 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 con? necting 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 re? sult, 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 tempo? rary 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" pa? rameter.

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 se? quences prefixed with a percent (%) character. An optional nu? meric 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 --hu? man-readable command-line option, though the default is for hu? man-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" parame? ter. (A perl script that will summarize this default log format is included in the rsync source code distribution in the "sup? port" subdirectory: rsyncstats.)

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

o %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 dis? played when that is the case). For the checksum to output for a file, either the --checksum option must be in-ef? fect 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 op? tions. 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 vari? ous delete options:

```
refuse options = c delete
```

The reason the above refuses all delete options is that the op? tions 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, --protect-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 "!refus?ing") 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 affect?ing --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-af?ter:

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

A note on refusing "compress" -- it is better to set the "dont compress" daemon parameter to "*" because that disables compres?sion silently instead of returning an error that forces the client to remove the -z option.

If you are un-refusing the compress option, you probably want to match "!compress*" so that you also accept the --compress-level option.

Note that the "copy-devices" & "write-devices" options are re?fused 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?
- --protect-args, -s: This actually makes transfers safer.
- --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

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 (such as zlib/zlibx) then no compression occurs for those files. Other algorithms have the level minimized to reduces 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 not compressed by default.

Specifying a value for the "dont compress" parameter changes the default when the daemon is the sender.

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 com? mand 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:

o 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 de? faults 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 direc? tory.

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 snip? pets 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 chal? lenge 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 en? cryption.

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 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 parameter if

your proxy supports sending that information. The examples be?

low 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 "proxy protocol = true"
    proxy_timeout 1m;
    proxy_connect_timeout 5s;
  }
}
```

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
```

FILES

/etc/rsyncd.conf or rsyncd.conf

SEE ALSO

rsync(1), rsync-ssl(1)

BUGS

Please report bugs! The rsync bug tracking system is online at https://rsync.samba.org/.

VERSION

This man page is current for version 3.2.3 of rsync.

CREDITS

rsync is distributed under the GNU General Public License. See the file COPYING for details.

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

A web site is available at https://rsync.samba.org/.

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

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

THANKS

Thanks to Warren Stanley for his original idea and patch for the rsync daemon. Thanks to Karsten Thygesen for his many suggestions and docu? mentation!

AUTHOR

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

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