

Section 29 - The pipe transport

29. The pipe transport

The pipe transport is used to deliver messages via a pipe to a command running in another process. One example is the use of pipe as a pseudo-remote transport for passing messages to some other delivery mechanism (such as UUCP). Another is the use by individual users to automatically process their incoming messages. The pipe transport can be used in one of the following ways:

-

A router routes one address to a transport in the normal way, and the transport is configured as a pipe transport. In this case, `$local_part` contains the local part of the address (as usual), and the command that is run is specified by the `command` option on the transport.

-

If the `batch_max` option is set greater than 1 (the default is 1), the transport can handle more than one address in a single run. In this case, when more than one address is routed to the transport, `$local_part` is not set (because it is not unique). However, the pseudo-variable `$pipe_addresses` (described in section 29.3 below) contains all the addresses that are routed to the transport.

-

A router redirects an address directly to a pipe command (for example, from an alias or forward file). In this case, `$address_pipe` contains the text of the pipe command, and the `command` option on the transport is ignored. If only one address is being transported (`batch_max` is not greater than one, or only one address was redirected to this pipe command), `$local_part` contains the local part that was redirected.

The pipe transport is a non-interactive delivery method. Exim can also deliver messages over pipes using the LMTP interactive protocol. This is implemented by the `lmtp` transport.

In the case when pipe is run as a consequence of an entry in a local user's `.forward` file, the command runs under the uid and gid of that user. In other cases, the uid and gid have to be specified explicitly, either on the transport or on the router that handles the address. Current and `"/home"` directories are also controllable. See chapter 23 for details of the local delivery environment and chapter 25 for a discussion of local delivery batching.

29.1 Concurrent delivery

If two messages arrive at almost the same time, and both are routed to a pipe delivery, the two pipe transports may be run concurrently. You must ensure that any pipe commands you set up are robust against this happening. If the commands write to a file, the `exim_lock` utility might be of use.

29.2 Returned status and data

If the command exits with a non-zero return code, the delivery is deemed to have failed, unless either the `ignore_status` option is set (in which case the return code is treated as zero), or the return code is one of those listed in the `temp_errors` option, which are interpreted as meaning `"try again later"`. In this case, delivery is deferred. Details of a permanent failure are logged, but are not included in the bounce message, which merely contains `"local delivery failed"`.

If the return code is greater than 128 and the command being run is a shell script, it normally means that the script was terminated by a signal whose value is the return code minus 128.

If Exim is unable to run the command (that is, if `execve()` fails), the return code is set to 127. This is the value that a shell returns if it is asked to run a non-existent command. The wording for the log line suggests that a non-existent command may be the problem.

The `return_output` option can affect the result of a pipe delivery. If it is set and the command produces any output on its standard output or standard error streams, the command is considered to have failed, even if it gave a zero return code or if `ignore_status` is set. The output from the command is included as part of the bounce message. The `return_fail_output` option is similar, except that output is returned only when the command exits with a failure return code, that is, a value other than zero or a code that matches `temp_errors`.

29.3 How the command is run

The command line is (by default) broken down into a command name and arguments by the pipe transport itself. The `allow_commands` and `restrict_to_path` options can be used to restrict the commands that may be run.

Unquoted arguments are delimited by white space. If an argument appears in double quotes, backslash is interpreted as an escape character in the usual way. If an argument appears in single quotes, no escaping is done.

String expansion is applied to the command line except when it comes from a traditional `.forward` file (commands from a filter file are expanded). The expansion is applied to each argument in turn rather than to the whole line. For this reason, any string expansion item that contains white space must be quoted so as to be contained within a single argument. A setting such as `command = /some/path ${if eq{$local_part}{postmaster}{xx}{yy}}`

will not work, because the expansion item gets split between several arguments. You have to write `command = /some/path "${if eq{$local_part}{postmaster}{xx}{yy}}"`

to ensure that it is all in one argument. The expansion is done in this way, argument by argument, so that the number of arguments cannot be changed as a result of expansion, and quotes or backslashes in inserted variables do not interact with external quoting. However, this leads to problems if you want to generate multiple arguments (or the command name plus arguments) from a single expansion. In this situation, the simplest solution is to use a shell. For example: `command = /bin/sh -c ${lookup{$local_part}!search{/some/file}}`

Special handling takes place when an argument consists of precisely the text `$pipe_addresses`. This is not a general expansion variable; the only place this string is recognized is when it appears as an argument for a pipe or transport filter command. It causes each address that is being handled to be inserted in the argument list at that point as a separate argument. This avoids any problems with spaces or shell metacharacters, and is of use when a pipe transport is handling groups of addresses in a batch.

After splitting up into arguments and expansion, the resulting command is run in a subprocess directly from the transport, not under a shell. The message that is being delivered is supplied on the standard input, and the standard output and standard error are both connected to a single pipe that is read by Exim. The `max_output` option controls how much output the command may produce, and the `return_output` and `return_fail_output` options control what is done with it.

Not running the command under a shell (by default) lessens the security risks in cases when a command from a user's filter file is built out of data that was taken from an incoming message. If a shell is required, it can of course be explicitly specified as the command to be run. However, there are circumstances where existing commands (for example, in `.forward` files) expect to be run under a shell and cannot easily be modified. To allow for these cases, there is an option called `use_shell`, which changes the way the pipe transport works. Instead of breaking up the command line as just described, it expands it as a single string and passes the result to `/bin/sh`. The `restrict_to_path` option and the `$pipe_addresses` facility cannot be used with `use_shell`, and the whole mechanism is inherently less secure.

29.4 Environment variables

The environment variables listed below are set up when the command is invoked. This list is a compromise for maximum compatibility with other MTAs. Note that the environment option can be used to add additional variables to this environment.

<code>DOMAIN</code>	the domain of the address
<code>HOME</code>	the home directory, if set
<code>HOST</code>	the host name when called from a router (see below)
<code>LOCAL_PART</code>	see below
<code>LOCAL_PART_PREFIX</code>	see below
<code>LOCAL_PART_SUFFIX</code>	see below
<code>LOGNAME</code>	see below
<code>MESSAGE_ID</code>	Exim's local ID for the message
<code>PATH</code>	as specified by the path option below
<code>QUALIFY_DOMAIN</code>	the sender qualification domain
<code>RECIPIENT</code>	the complete recipient address
<code>SENDER</code>	the sender of the message (empty if a bounce)
<code>SHELL</code>	<code>/bin/sh</code>
<code>TZ</code>	the value of the timezone option, if set
<code>USER</code>	see below

When a pipe transport is called directly from (for example) an accept router, `LOCAL_PART` is set to the local part of the address. When it is called as a result of a forward or alias expansion, `LOCAL_PART` is set to the local part of the address that was expanded. In both cases, any affixes are removed from the local part, and made available in `LOCAL_PART_PREFIX` and `LOCAL_PART_SUFFIX`, respectively. `LOGNAME` and `USER` are set to the same value as `LOCAL_PART` for compatibility with other MTAs.

`HOST` is set only when a pipe transport is called from a router that associates hosts with an address, typically when using pipe as a pseudo-remote transport. `HOST` is set to the first host name specified by the router.

If the transport's generic `home_directory` option is set, its value is used for the HOME environment variable. Otherwise, a home directory may be set by the router's `transport_home_directory` option, which defaults to the user's home directory if `check_local_user` is set.

29.5 Private options for pipe

`allow_commandsUse: pipeType: string list†Default: unset`

The string is expanded, and is then interpreted as a colon-separated list of permitted commands. If `restrict_to_path` is not set, the only commands permitted are those in the `allow_commands` list. They need not be absolute paths; the path option is still used for relative paths. If `restrict_to_path` is set with `allow_commands`, the command must either be in the `allow_commands` list, or a name without any slashes that is found on the path. In other words, if neither `allow_commands` nor `restrict_to_path` is set, there is no restriction on the command, but otherwise only commands that are permitted by one or the other are allowed. For example, if `allow_commands = /usr/bin/vacation`

and `restrict_to_path` is not set, the only permitted command is `/usr/bin/vacation`. The `allow_commands` option may not be set if `use_shell` is set.

`batch_idUse: pipeType: string†Default: unset`

See the description of local delivery batching in chapter 25.

`batch_maxUse: pipeType: integerDefault: 1`

This limits the number of addresses that can be handled in a single delivery. See the description of local delivery batching in chapter 25.

`check_stringUse: pipeType: stringDefault: unset`

As pipe writes the message, the start of each line is tested for matching `check_string`, and if it does, the initial matching characters are replaced by the contents of `escape_string`, provided both are set. The value of `check_string` is a literal string, not a regular expression, and the case of any letters it contains is significant. When `use_bsmtplib` is set, the contents of `check_string` and `escape_string` are forced to values that implement the SMTP escaping protocol. Any settings made in the configuration file are ignored.

`commandUse: pipeType: string†Default: unset`

This option need not be set when pipe is being used to deliver to pipes obtained directly from address redirections. In other cases, the option must be set, to provide a command to be run. It need not yield an absolute path (see the path option below). The command is split up into separate arguments by Exim, and each argument is separately expanded, as described in section 29.3 above.

`environmentUse: pipeType: string†Default: unset`

This option is used to add additional variables to the environment in which the command runs (see section 29.4 for the default list). Its value is a string which is expanded, and then interpreted as a colon-separated list of environment settings of the form `<name>=<value>`.

`escape_stringUse: pipeType: stringDefault: unset`

See `check_string` above.

`freeze_exec_failUse: pipeType: booleanDefault: false`

Failure to exec the command in a pipe transport is by default treated like any other failure while running the command. However, if `freeze_exec_fail` is set, failure to exec is treated specially, and causes the message to be frozen, whatever the setting of `ignore_status`.

`ignore_statusUse: pipeType: booleanDefault: false`

If this option is true, the status returned by the subprocess that is set up to run the command is ignored, and Exim behaves as if zero had been returned. Otherwise, a non-zero status or termination by signal causes an error return from the transport unless the status value is one of those listed in `temp_errors`; these cause the delivery to be deferred and

tried again later.

Note: This option does not apply to timeouts, which do not return a status. See the `timeout_defer` option for how timeouts are handled.

`log_defer_output`Use: pipeType: booleanDefault: false

If this option is set, and the status returned by the command is one of the codes listed in `temp_errors` (that is, delivery was deferred), and any output was produced, the first line of it is written to the main log.

`log_fail_output`Use: pipeType: booleanDefault: false

If this option is set, and the command returns any output, and also ends with a return code that is neither zero nor one of the return codes listed in `temp_errors` (that is, the delivery failed), the first line of output is written to the main log. This option and `log_output` are mutually exclusive. Only one of them may be set.

`log_output`Use: pipeType: booleanDefault: false

If this option is set and the command returns any output, the first line of output is written to the main log, whatever the return code. This option and `log_fail_output` are mutually exclusive. Only one of them may be set.

`max_output`Use: pipeType: integerDefault: 20K

This specifies the maximum amount of output that the command may produce on its standard output and standard error file combined. If the limit is exceeded, the process running the command is killed. This is intended as a safety measure to catch runaway processes. The limit is applied independently of the settings of the options that control what is done with such output (for example, `return_output`). Because of buffering effects, the amount of output may exceed the limit by a small amount before Exim notices.

`message_prefix`Use: pipeType: string†Default: see below

The string specified here is expanded and output at the start of every message. The default is unset if `use_bsmtip` is set. Otherwise it is `message_prefix = \`
`From ${if def:return_path{$return_path}{MAILER-DAEMON}}\`
`${tod_bsdingbox}\n`

This is required by the commonly used `/usr/bin/vacation` program. However, it must not be present if delivery is to the Cyrus IMAP server, or to the `tmail` local delivery agent. The prefix can be suppressed by setting `message_prefix =`

`message_suffix`Use: pipeType: string†Default: see below

The string specified here is expanded and output at the end of every message. The default is unset if `use_bsmtip` is set. Otherwise it is a single newline. The suffix can be suppressed by setting `message_suffix =`

`path`Use: pipeType: stringDefault: `/bin:/usr/bin`

This option specifies the string that is set up in the `PATH` environment variable of the subprocess. If the command option does not yield an absolute path name, the command is sought in the `PATH` directories, in the usual way. Warning: This does not apply to a command specified as a transport filter.

`pipe_as_creator`Use: pipeType: booleanDefault: false

If the generic user option is not set and this option is true, the delivery process is run under the uid that was in force when Exim was originally called to accept the message. If the group id is not otherwise set (via the generic group option), the gid that was in force when Exim was originally called to accept the message is used.

`restrict_to_path`Use: pipeType: booleanDefault: false

When this option is set, any command name not listed in `allow_commands` must contain no slashes. The command is searched for only in the directories listed in the `path` option. This option is intended for use in the case when a pipe command has been generated from a user's `.forward` file. This is usually handled by a pipe transport called `address_pipe`.

`return_fail_output`Use: pipeType: booleanDefault: false

If this option is true, and the command produced any output and ended with a return code other than zero or one of the codes listed in `temp_errors` (that is, the delivery failed), the output is returned in the bounce message. However, if the message has a null sender (that is, it is itself a bounce message), output from the command is discarded. This option and `return_output` are mutually exclusive. Only one of them may be set.

`return_output`Use: pipeType: booleanDefault: false

If this option is true, and the command produced any output, the delivery is deemed to have failed whatever the return code from the command, and the output is returned in the bounce message. Otherwise, the output is just discarded. However, if the message has a null sender (that is, it is a bounce message), output from the command is always discarded, whatever the setting of this option. This option and `return_fail_output` are mutually exclusive. Only one of them may be set.

`temp_errors`Use: pipeType: string listDefault: see below

This option contains either a colon-separated list of numbers, or a single asterisk. If `ignore_status` is false and `return_output` is not set, and the command exits with a non-zero return code, the failure is treated as temporary and the delivery is deferred if the return code matches one of the numbers, or if the setting is a single asterisk. Otherwise, non-zero return codes are treated as permanent errors. The default setting contains the codes defined by `EX_TEMPFAIL` and `EX_CANTCREAT` in `sysexits.h`. If Exim is compiled on a system that does not define these macros, it assumes values of 75 and 73, respectively.

`timeout`Use: pipeType: timeDefault: 1h

If the command fails to complete within this time, it is killed. This normally causes the delivery to fail (but see `timeout_defer`). A zero time interval specifies no timeout. In order to ensure that any subprocesses created by the command are also killed, Exim makes the initial process a process group leader, and kills the whole process group on a timeout. However, this can be defeated if one of the processes starts a new process group.

`timeout_defer`Use: pipeType: booleanDefault: false

A timeout in a pipe transport, either in the command that the transport runs, or in a transport filter that is associated with it, is by default treated as a hard error, and the delivery fails. However, if `timeout_defer` is set true, both kinds of timeout become temporary errors, causing the delivery to be deferred.

`umask`Use: pipeType: octal integerDefault: 022

This specifies the umask setting for the subprocess that runs the command.

`use_bsmtpt`Use: pipeType: booleanDefault: false

If this option is set true, the pipe transport writes messages in “batch SMTP” format, with the envelope sender and recipient(s) included as SMTP commands. If you want to include a leading HELO command with such messages, you can do so by setting the `message_prefix` option. See section 44.10 for details of batch SMTP.

`use_classresources`Use: pipeType: booleanDefault: false

This option is available only when Exim is running on FreeBSD, NetBSD, or BSD/OS. If it is set true, the `setclassresources()` function is used to set resource limits when a pipe transport is run to perform a delivery. The limits for the uid under which the pipe is to run are obtained from the login class database.

`use_crlf`Use: pipeType: booleanDefault: false

This option causes lines to be terminated with the two-character CRLF sequence (carriage return, linefeed) instead of just a linefeed character. In the case of batched SMTP, the byte sequence written to the pipe is then an exact image of what would be sent down a real SMTP connection.

The contents of the `message_prefix` and `message_suffix` options are written verbatim, so must contain their own carriage return characters if these are needed. Since the default values for both `message_prefix` and `message_suffix` end with a single linefeed, their values must be changed to end with `\r\n` if `use_crlf` is set.

`use_shell`Use: pipeType: booleanDefault: false

If this option is set, it causes the command to be passed to `/bin/sh` instead of being run directly from the transport, as described in section 29.3. This is less secure, but is needed in some situations where the command is expected to be run under a shell and cannot easily be modified. The `allow_commands` and `restrict_to_path` options, and the `$pipe_addresses` facility are incompatible with `use_shell`. The command is expanded as a single string, and handed to `/bin/sh` as data for its `-c` option.

29.6 Using an external local delivery agent

The pipe transport can be used to pass all messages that require local delivery to a separate local delivery agent such as `procmail`. When doing this, care must be taken to ensure that the pipe is run under an appropriate `uid` and `gid`. In some configurations one wants this to be a `uid` that is trusted by the delivery agent to supply the correct sender of the message. It may be necessary to recompile or reconfigure the delivery agent so that it trusts an appropriate user. The following is an example transport and router configuration for `procmail`:

```
# transport
procmail_pipe:
  driver = pipe
  command = /usr/local/bin/procmail -d $local_part
  return_path_add
  delivery_date_add
  envelope_to_add
  check_string = "From "
  escape_string = ">From "
  user = $local_part
  group = mail

# router
procmail:
  driver = accept
  check_local_user
  transport = procmail_pipe
```

In this example, the pipe is run as the local user, but with the group set to `mail`. An alternative is to run the pipe as a specific user such as `mail` or `exim`, but in this case you must arrange for `procmail` to trust that user to supply a correct sender address. If you do not specify either a group or a user option, the pipe command is run as the local user. The home directory is the user's home directory by default.

Note: The command that the pipe transport runs does not begin with `IFS=" "`

as shown in some `procmail` documentation, because Exim does not by default use a shell to run pipe commands.

The next example shows a transport and a router for a system where local deliveries are handled by the Cyrus IMAP server.

```
# transport
local_delivery_cyrus:
  driver = pipe
  command = /usr/cyrus/bin/deliver \
    -m ${substr_1:$local_part_suffix} -- $local_part
  user = cyrus
  group = mail
  return_output
  log_output
  message_prefix =
  message_suffix =
```

```
# router
local_user_cyrus:
  driver = accept
  check_local_user
  local_part_suffix = .*
  transport = local_delivery_cyrus
  Note the unsetting of message_prefix and message_suffix, and the use of return_output to cause any text written by
  Cyrus to be returned to the sender.
```