

Section 46 - Some common configuration settings

46. Some common configuration settings

This chapter discusses some configuration settings that seem to be fairly common. More examples and discussion can be found in the Exim book. 46.1 Sending mail to a smart host

If you want to send all mail for non-local domains to a "smart host", you should replace the default dnslookup router with a router which does the routing explicitly: `send_to_smart_host`:

```
driver = manualroute
route_list = !+local_domains smart.host.name
transport = remote_smtp
```

You can use the smart host's IP address instead of the name if you wish. If you are using Exim only to submit messages to a smart host, and not for receiving incoming messages, you can arrange for it to do the submission synchronously by setting the `mua_wrapper` option (see chapter 47). 46.2 Using Exim to handle mailing lists

Exim can be used to run simple mailing lists, but for large and/or complicated requirements, the use of additional specialized mailing list software such as Majordomo or Mailman is recommended.

The redirect router can be used to handle mailing lists where each list is maintained in a separate file, which can therefore be managed by an independent manager. The domains router option can be used to run these lists in a separate domain from normal mail. For example: lists:

```
driver = redirect
domains = lists.example
file = /usr/lists/$local_part
forbid_pipe
forbid_file
errors_to = $local_part-request@lists.example
no_more
```

This router is skipped for domains other than lists.example. For addresses in that domain, it looks for a file that matches the local part. If there is no such file, the router declines, but because `no_more` is set, no subsequent routers are tried, and so the whole delivery fails.

The `forbid_pipe` and `forbid_file` options prevent a local part from being expanded into a file name or a pipe delivery, which is usually inappropriate in a mailing list.

The `errors_to` option specifies that any delivery errors caused by addresses taken from a mailing list are to be sent to the given address rather than the original sender of the message. However, before acting on this, Exim verifies the error address, and ignores it if verification fails.

For example, using the configuration above, mail sent to `dicts@lists.example` is passed on to those addresses contained in `/usr/lists/dicts`, with error reports directed to `dicts-request@lists.example`, provided that this address can be verified. There could be a file called `/usr/lists/dicts-request` containing the address(es) of this particular list's manager(s), but other approaches, such as setting up an earlier router (possibly using the `local_part_prefix` or `local_part_suffix` options) to handle addresses of the form `owner-xxx` or `xxx-request`, are also possible. 46.3 Syntax errors in mailing lists

If an entry in redirection data contains a syntax error, Exim normally defers delivery of the original address. That means that a syntax error in a mailing list holds up all deliveries to the list. This may not be appropriate when a list is being maintained automatically from data supplied by users, and the addresses are not rigorously checked.

If the `skip_syntax_errors` option is set, the redirect router just skips entries that fail to parse, noting the incident in the log. If in addition `syntax_errors_to` is set to a verifiable address, a message is sent to it whenever a broken address is skipped. It is usually appropriate to set `syntax_errors_to` to the same address as `errors_to`. 46.4 Re-expansion of mailing lists

Exim remembers every individual address to which a message has been delivered, in order to avoid duplication, but it normally stores only the original recipient addresses with a message. If all the deliveries to a mailing list cannot be done at the first attempt, the mailing list is re-expanded when the delivery is next tried. This means that alterations to the list are taken into account at each delivery attempt, so addresses that have been added to the list since the message arrived

will therefore receive a copy of the message, even though it pre-dates their subscription.

If this behaviour is felt to be undesirable, the `one_time` option can be set on the redirect router. If this is done, any addresses generated by the router that fail to deliver at the first attempt are added to the message as `“top level”` addresses, and the parent address that generated them is marked `“delivered”`. Thus, expansion of the mailing list does not happen again at the subsequent delivery attempts. The disadvantage of this is that if any of the failing addresses are incorrect, correcting them in the file has no effect on pre-existing messages.

The original top-level address is remembered with each of the generated addresses, and is output in any log messages. However, any intermediate parent addresses are not recorded. This makes a difference to the log only if the `all_parents` selector is set, but for mailing lists there is normally only one level of expansion anyway. 46.5 Closed mailing lists

The examples so far have assumed open mailing lists, to which anybody may send mail. It is also possible to set up closed lists, where mail is accepted from specified senders only. This is done by making use of the generic senders option to restrict the router that handles the list.

The following example uses the same file as a list of recipients and as a list of permitted senders. It requires three routers: `lists_request`:

```
driver = redirect
domains = lists.example
local_part_suffix = -request
file = /usr/lists/$local_part$local_part_suffix
no_more
```

`lists_post`:

```
driver = redirect
domains = lists.example
senders = ${if exists {/usr/lists/$local_part}\
    {lsearch;/usr/lists/$local_part}{*}}
file = /usr/lists/$local_part
forbid_pipe
forbid_file
errors_to = $local_part-request@lists.example
no_more
```

`lists_closed`:

```
driver = redirect
domains = lists.example
allow_fail
data = :fail: $local_part@lists.example is a closed mailing list
```

All three routers have the same domains setting, so for any other domains, they are all skipped. The first router runs only if the local part ends in `-request`. It handles messages to the list manager(s) by means of an open mailing list.

The second router runs only if the senders precondition is satisfied. It checks for the existence of a list that corresponds to the local part, and then checks that the sender is on the list by means of a linear search. It is necessary to check for the existence of the file before trying to search it, because otherwise Exim thinks there is a configuration error. If the file does not exist, the expansion of senders is `*`, which matches all senders. This means that the router runs, but because there is no list, declines, and `no_more` ensures that no further routers are run. The address fails with an `“unrouteable address”` error.

The third router runs only if the second router is skipped, which happens when a mailing list exists, but the sender is not on it. This router forcibly fails the address, giving a suitable error message. 46.6 Variable Envelope Return Paths (VERP)

Variable Envelope Return Paths – see <http://cr.yip.to/proto/verp.txt> – are a way of helping mailing list administrators discover which subscription address is the cause of a particular delivery failure. The idea is to encode the original recipient address in the outgoing envelope sender address, so that if the message is forwarded by another host and then subsequently bounces, the original recipient can be extracted from the recipient address of the bounce.

Envelope sender addresses can be modified by Exim using two different facilities: the `errors_to` option on a router (as shown in previous mailing list examples), or the `return_path` option on a transport. The second of these is effective only if the message is successfully delivered to another host; it is not used for errors detected on the local host (see the description of `return_path` in chapter 24). Here is an example of the use of `return_path` to implement VERP on an smtp

```
transport: verp_smtp:
  driver = smtp
  max_rcpt = 1
  return_path = \
    ${if match {$return_path}{^(.+?)-request@your.dom.example\$}}\
    {$1-request=$local_part%$domain@your.dom.example}fail}
```

This has the effect of rewriting the return path (envelope sender) on outgoing SMTP messages, if the local part of the original return path ends in “-request”, and the domain is your.dom.example. The rewriting inserts the local part and domain of the recipient into the return path. Suppose, for example, that a message whose return path has been set to somelist-request@your.dom.example is sent to subscriber@other.dom.example. In the transport, the return path is rewritten as somelist-request=subscriber%other.dom.example@your.dom.example

For this to work, you must also arrange for outgoing messages that have “-request” in their return paths to have just a single recipient. That is achieved by setting max_rcpt to 1. Without this, a single copy of a message might be sent to several different recipients in the same domain, in which case \$local_part is not available in the transport, because it is not unique.

Unless your host is doing nothing but mailing list deliveries, you should probably use a separate transport for the VERP deliveries, so as not to use extra resources for the others. This can easily be done by expanding the transport option in the router: dnslookup:

```
driver = dnslookup
domains = ! +local_domains
transport = \
  ${if match {$return_path}{^(.+?)-request@your.dom.example\$}}\
  {verp_smtp}{remote_smtp}}
no_more
```

If you want to change the return path using errors_to in a router instead of using return_path in the transport, you need to set errors_to on all routers that handle mailing list addresses. This will ensure that all delivery errors, including those detected on the local host, are sent to the VERP address.

On a host that does no local deliveries and has no manual routing, only the dnslookup router needs to be changed. A special transport is not needed for SMTP deliveries. Every mailing list recipient has its own return path value, and so Exim must hand them to the transport one at a time. Here is an example of a dnslookup router that implements VERP:

```
verp_dnslookup:
  driver = dnslookup
  domains = ! +local_domains
  transport = remote_smtp
  errors_to = \
    ${if match {$return_path}{^(.+?)-request@your.dom.example\$}}\
    {$1-request=$local_part%$domain@your.dom.example}fail}
no_more
```

Before you start sending out messages with VERPed return paths, you must also configure Exim to accept the bounce messages that come back to those paths. Typically this is done by setting a local_part_suffix option for a router, and using this to route the messages to wherever you want to handle them.

The overhead incurred in using VERP depends very much on the size of the message, the number of recipient addresses that resolve to the same remote host, and the speed of the connection over which the message is being sent. If a lot of addresses resolve to the same host and the connection is slow, sending a separate copy of the message for each address may take substantially longer than sending a single copy with many recipients (for which VERP cannot be used). 46.7 Virtual domains

The phrase virtual domain is unfortunately used with two rather different meanings:

-

A domain for which there are no real mailboxes; all valid local parts are aliases for other email addresses. Common examples are organizational top-level domains and “vanity” domains.

-

One of a number of independent domains that are all handled by the same host, with mailboxes on that host, but where the mailbox owners do not necessarily have login accounts on that host.

The first usage is probably more common, and does seem more “virtual” than the second. This kind of domain can be handled in Exim with a straightforward aliasing router. One approach is to create a separate alias file for each virtual domain. Exim can test for the existence of the alias file to determine whether the domain exists. The `dsearch` lookup type is useful here, leading to a router of this form: `virtual`:

```
driver = redirect
domains = dsearch;/etc/mail/virtual
data = ${lookup{$local_part}lsearch{/etc/mail/virtual/$domain}}
no_more
```

The `domains` option specifies that the router is to be skipped, unless there is a file in the `/etc/mail/virtual` directory whose name is the same as the domain that is being processed. When the router runs, it looks up the local part in the file to find a new address (or list of addresses). The `no_more` setting ensures that if the lookup fails (leading to data being an empty string), Exim gives up on the address without trying any subsequent routers.

This one router can handle all the virtual domains because the alias file names follow a fixed pattern. Permissions can be arranged so that appropriate people can edit the different alias files. A successful aliasing operation results in a new envelope recipient address, which is then routed from scratch.

The other kind of “virtual” domain can also be handled in a straightforward way. One approach is to create a file for each domain containing a list of valid local parts, and use it in a router like this: `my_domains`:

```
driver = accept
domains = dsearch;/etc/mail/domains
local_parts = lsearch;/etc/mail/domains/$domain
transport = my_mailboxes
```

The address is accepted if there is a file for the domain, and the local part can be found in the file. The `domains` option is used to check for the file’s existence because `domains` is tested before the `local_parts` option (see section 3.12). You cannot use `require_files`, because that option is tested after `local_parts`. The transport is as follows: `my_mailboxes`:

```
driver = appendfile
file = /var/mail/$domain/$local_part
user = mail
```

This uses a directory of mailboxes for each domain. The `user` setting is required, to specify which uid is to be used for writing to the mailboxes.

The configuration shown here is just one example of how you might support this requirement. There are many other ways this kind of configuration can be set up, for example, by using a database instead of separate files to hold all the information about the domains. 46.8 Multiple user mailboxes

Heavy email users often want to operate with multiple mailboxes, into which incoming mail is automatically sorted. A popular way of handling this is to allow users to use multiple sender addresses, so that replies can easily be identified. Users are permitted to add prefixes or suffixes to their local parts for this purpose. The wildcard facility of the generic router options `local_part_prefix` and `local_part_suffix` can be used for this. For example, consider this router: `userforward`:

```
driver = redirect
check_local_user
file = $home/.forward
local_part_suffix = -*
local_part_suffix_optional
allow_filter
```

It runs a user’s `.forward` file for all local parts of the form `username-*`. Within the filter file the user can distinguish different cases by testing the variable `$local_part_suffix`. For example: if `$local_part_suffix` contains `-special` then `save /home/$local_part/Mail/special`
endif

If the filter file does not exist, or does not deal with such addresses, they fall through to subsequent routers, and, assuming no subsequent use of the `local_part_suffix` option is made, they presumably fail. Thus, users have control over

which suffixes are valid.

Alternatively, a suffix can be used to trigger the use of a different `.forward` file – which is the way a similar facility is implemented in another MTA: `userforward`:

```
driver = redirect
check_local_user
file = $home/.forward$local_part_suffix
local_part_suffix = -*
local_part_suffix_optional
allow_filter
```

If there is no suffix, `.forward` is used; if the suffix is `-special`, for example, `.forward-special` is used. Once again, if the appropriate file does not exist, or does not deal with the address, it is passed on to subsequent routers, which could, if required, look for an unqualified `.forward` file to use as a default. 46.9 Simplified vacation processing

The traditional way of running the vacation program is for a user to set up a pipe command in a `.forward` file (see section 22.6 for syntax details). This is prone to error by inexperienced users. There are two features of Exim that can be used to make this process simpler for users:

-

A local part prefix such as `“vacation”` can be specified on a router which can cause the message to be delivered directly to the vacation program, or alternatively can use Exim's autoreply transport. The contents of a user's `.forward` file are then much simpler. For example: `spqr, vacation-spqr`

-

The `require_files` generic router option can be used to trigger a vacation delivery by checking for the existence of a certain file in the user's home directory. The `unseen` generic option should also be used, to ensure that the original delivery also proceeds. In this case, all the user has to do is to create a file called, say, `.vacation`, containing a vacation message.

Another advantage of both these methods is that they both work even when the use of arbitrary pipes by users is locked out. 46.10 Taking copies of mail

Some installations have policies that require archive copies of all messages to be made. A single copy of each message can easily be taken by an appropriate command in a system filter, which could, for example, use a different file for each day's messages.

There is also a shadow transport mechanism that can be used to take copies of messages that are successfully delivered by local transports, one copy per delivery. This could be used, *inter alia*, to implement automatic notification of delivery by sites that insist on doing such things. 46.11 Intermittently connected hosts

It has become quite common (because it is cheaper) for hosts to connect to the Internet periodically rather than remain connected all the time. The normal arrangement is that mail for such hosts accumulates on a system that is permanently connected.

Exim was designed for use on permanently connected hosts, and so it is not particularly well-suited to use in an intermittently connected environment. Nevertheless there are some features that can be used. 46.12 Exim on the upstream server host

It is tempting to arrange for incoming mail for the intermittently connected host to remain on Exim's queue until the client connects. However, this approach does not scale very well. Two different kinds of waiting message are being mixed up in the same queue – those that cannot be delivered because of some temporary problem, and those that are waiting for their destination host to connect. This makes it hard to manage the queue, as well as wasting resources, because each queue runner scans the entire queue.

A better approach is to separate off those messages that are waiting for an intermittently connected host. This can be done by delivering these messages into local files in batch SMTP, `“mailstore”`, or other envelope-preserving format, from where they are transmitted by other software when their destination connects. This makes it easy to collect all the mail for one host in a single directory, and to apply local timeout rules on a per-message basis if required.

On a very small scale, leaving the mail on Exim's queue can be made to work. If you are doing this, you should configure Exim with a long retry period for the intermittent host. For example: `cheshire.wonderland.fict.example *`

F,5d,24h

This stops a lot of failed delivery attempts from occurring, but Exim remembers which messages it has queued up for that host. Once the intermittent host comes online, forcing delivery of one message (either by using the `-M` or `-R` options, or by using the ETRN SMTP command (see section 44.8) causes all the queued up messages to be delivered, often down a single SMTP connection. While the host remains connected, any new messages get delivered immediately.

If the connecting hosts do not have fixed IP addresses, that is, if a host is issued with a different IP address each time it connects, Exim's retry mechanisms on the holding host get confused, because the IP address is normally used as part of the key string for holding retry information. This can be avoided by unsetting `retry_include_ip_address` on the smtp transport. Since this has disadvantages for permanently connected hosts, it is best to arrange a separate transport for the intermittently connected ones. 46.13 Exim on the intermittently connected client host

The value of `smtp_accept_queue_per_connection` should probably be increased, or even set to zero (that is, disabled) on the intermittently connected host, so that all incoming messages down a single connection get delivered immediately.

Mail waiting to be sent from an intermittently connected host will probably not have been routed, because without a connection DNS lookups are not possible. This means that if a normal queue run is done at connection time, each message is likely to be sent in a separate SMTP session. This can be avoided by starting the queue run with a command line option beginning with `-qq` instead of `-q`. In this case, the queue is scanned twice. In the first pass, routing is done but no deliveries take place. The second pass is a normal queue run; since all the messages have been previously routed, those destined for the same host are likely to get sent as multiple deliveries in a single SMTP connection.