

Administering, Securing & Spam-Fighting

sendmail Cookbook



O'REILLY®

Craig Hunt

Delivery and Forwarding

2.0 Introduction

Inbound mail is either delivered directly to the addressee or relayed to another mail host for delivery. Mail is directly delivered only if it is destined for the local host; mail destined for any other host is relayed. In this chapter, we look at ways to properly configure sendmail to deliver mail locally and forward it to other systems.

Delivery is a multistep process. First, sendmail must process the host portion of the delivery address and recognize that the mail is, in fact, addressed to the local host. If it isn't addressed to the local host, it is relayed as described in Chapter 3. If it is addressed to the local host, the user portion of the address is processed against the *aliases* file to determine the proper delivery address. If the *aliases* file returns an external address, the mail is forwarded to the external host for delivery. If it returns the address of a local mailbox, sendmail checks for a *.forward* file. If the *.forward* file exists, the mail is delivered as specified by that file. Otherwise, the mail is delivered to the local mailbox. Figure 2-1 illustrates this delivery flow.*

sendmail processes each delivery address through the *canonify* ruleset and through the *parse* ruleset—rulesets 3 and 0. Aliasing starts when the result of that process tells sendmail to deliver the mail through a mailer that has the A flag set.† If ruleset 0 returns the name of a mailer that does not have the A flag set, aliasing is not done. While the A flag can be set for any mailer, only the *cyrus* mailers and the *local* mailer set the A flag by default. For our discussion of aliasing, we'll use the *local* mailer as an example.

Aliasing first looks up the delivery address in the *aliases* database. If the lookup returns a different email address, the new address is processed. If the mailer used for the new address has the A flag set, that address is looked up in the *aliases* database.

* This is a simplified description based on the default configuration. sendmail flags and options can change the way mail delivery is handled.

† Flags are set through the F parameter in each *sendmail.cf* mailer definition.

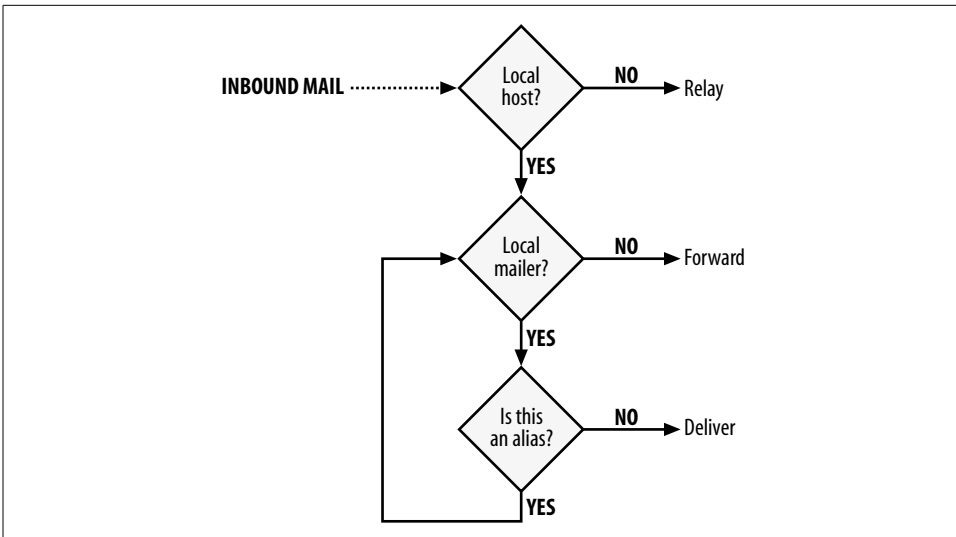


Figure 2-1. The multistep delivery process

This process continues until no new address is returned by the alias lookup. If the mailer used for the final address returned by the *aliases* database has the *w* flag set, *sendmail* looks for a delivery address in the user's *.forward* file. By default, only the local mailer has the *w* flag set. If a *.forward* file is found, delivery is made based on the delivery address that it contains.

Although Figure 2-1 portrays a simplified version of the actual delivery process, it does highlight areas of the configuration that relate to local delivery and forwarding. Class `$$=w` must be properly configured. The *aliases* file must be created and made into a database, and, if used, the *.forward* files must be properly configured.

sendmail only accepts mail for local delivery that is addressed to the local host. All other mail is relayed. Class `$$=w` contains all of the valid hostnames for the local host. If the hostname from the delivery address is found in class `$$=w`, the mail is accepted for local delivery. In fact, ruleset 0 uses class `$$=w` when selecting the local mailer.

Mail delivered by the local mailer must be addressed to a user account that exists on the local host or to an alias that resolves to a valid delivery address. The *aliases* database frees the local mailer from the limitation of having all local mail addressed to actual usernames. It makes *sendmail* a more flexible system by allowing mail addressed to a single alias to be sent to multiple recipients or mail addressed to several different names to be routed to a single recipient. The *aliases* database is also used to forward mail to other computers, programs, and files.

The *aliases* database is so essential to the functioning of a *sendmail* system that *sendmail* complains if the database does not exist. Some users and programs that send out email alerts may also complain because they assume that certain aliases exist.

Additionally, if neither the *aliases* database nor the *aliases* text file is found, sendmail will not apply the user's *.forward* file.

When sendmail looks up a local address in the *aliases* database and no new recipient address is returned by the lookup, sendmail checks to see if the user identified by the local address has a *.forward* file. The possible locations of the user's *.forward* file are defined in the *sendmail.cf* file by the `ForwardPath` option:

```
$ grep ForwardPath sendmail.cf
0 ForwardPath=$z/.forward.$w+$h:$z/.forward+$h:$z/.forward.$w:$z/.forward
```

The path shown above was configured by the `confFORWARD_PATH` define in the *domain/generic.m4* file used in the sample configuration described in Recipe 1.8. The `ForwardPath` in this example includes three *sendmail.cf* macros. The `$z` macro holds the path of the recipient's home directory, as specified in */etc/passwd*. The `$w` macro holds the primary name of the local host. The `$h` macro normally holds the name of the recipient host when ruleset 0 constructs the mailer/host/user delivery triple. In this case, however, the mailer is the local mailer so the host is always the local host. This makes the `$h` macro available for other uses. *procmail* provides one example of how the `$h` macro can be used in other ways. For example, when the *user+detail* addressing syntax is used and *procmail* is used as the local mailer, the `$h` macro contains the *detail* value.*

Given this specific `ForwardPath`, if the local mailer is *procmail*, the hostname is *chef*, and the mail is addressed to *kathy+cookbook*, the following *.forward* files are searched: */home/kathy/.forward.chef+cookbook*, */home/kathy/.forward+cookbook*, */home/kathy/.forward.chef*, and */home/kathy/.forward*. Each file is looked for in order, and the search stops as soon as a file with the specified name is found.

Both the *aliases* database and *.forward* files are capable of forwarding mail. Forwarding and relaying are often confused, particularly when mail is forwarded to an external host. When it is, the effect is essentially the same as relaying to that host—mail is passed to an external system for delivery. The difference is this: mail is forwarded only after it has been accepted for local delivery; mail is relayed when it is not accepted for local delivery. Relaying is covered in Chapter 3.

2.1 Accepting Mail for Other Hosts

Problem

You must configure sendmail to accept mail for local delivery that is addressed to other hosts. A common reason for doing this is to configure a mail exchanger to accept mail for a domain or for individual hosts.

* In the configuration created in Recipe 1.8, *procmail* is used as the local mailer because the *linux.m4* file used in that configuration contains the *local_procmail* feature.

Solution

Create an `/etc/mail/local-host-names` file. Put into that file the hostnames and domain names for which sendmail should accept mail for local delivery. Enter the names with one hostname or domain name per line.

Add the `use_cw_file` feature to the sendmail configuration to make use of the newly created `local-host-names` file. Here are sample lines that could be added to the configuration, if it does not already contain the `use_cw_file` feature:

```
dn1 Load class $=w with other names for the local host
FEATURE(`use_cw_file')
```

Finally, rebuild the `sendmail.cf` file, copy the new `sendmail.cf` file to `/etc/mail`, and restart sendmail, as shown in Recipe 1.8.

Discussion

Mail addressed to one host can be routed to another host for a variety of reasons: forwarding, relaying, `mailertable` entries, and so on. One common reason for routing mail in this manner is because DNS says to do so. A system sending mail routes the mail based on information obtained from DNS. Mail is addressed to some hostname. The remote system takes that hostname and asks DNS if it has mail exchange (MX) records for that host. If no MX record is found for a given host, the address record of the host is obtained from DNS, and the mail is sent directly to the host. If DNS returns MX records, the remote system sends the mail to the system with the lowest preference number listed on the MX record.

Regardless of why the mail is routed to your mail host, the sendmail system must be configured to accept the mail or it will generate a *Relaying denied* error. This is an interesting error because the remote system is not necessarily trying to relay mail; it may be attempting to deliver the mail to the mail exchanger as directed by DNS. However, sendmail only accepts inbound mail for delivery that is addressed to the local system. All other mail is relayed. For local delivery, sendmail accepts mail that is addressed to another host only if it finds the name of that host in class `=$w`. Class `=$w` is an array that contains all of the names that sendmail considers valid for local mail delivery.

There are three ways to load names into class `=$w`. The recommended method for solving this problem is to use the `use_cw_file` feature. Two alternatives are also discussed: placing local hostnames directly in `sendmail.cf` itself or using the `bestmx_is_local` feature.

Using the `use_cw_file` feature

The `use_cw_file` feature directs sendmail to load the `/etc/mail/local-host-names` file into class `$=w`. It does this by placing the following F command in the `sendmail.cf` file:*

```
Fw/etc/mail/local-host-names
```

Once the `use_cw_file` feature is added to the configuration, sendmail expects to find the `local-host-names` file and displays a nonfatal error message when it doesn't. It is possible to configure sendmail to treat `local-host-names` as an optional file by placing the following define in the configuration *before* the `use_cw_file` feature:

```
define(`confCW_FILE', `-o /etc/mail/local-host-names')
```

The `-o` flag makes the file optional, which stops sendmail from complaining if the file is not found. The `confCW_FILE` define can also be used to set a different pathname for the `local-host-names` file, if you decide to place the file in another directory or give it another name. Don't change the pathname unless you absolutely must. Use default pathnames for files whenever possible; it makes it easier for others to find these files when maintaining your system.

Frankly, there is little reason to make the `local-host-names` file optional; in fact, making it optional can hide real errors, such as an error opening the file. If you're not ready to add hostnames to the file, simply create an empty file. But, in all likelihood, if you add the `use_cw_file` feature to your configuration, it is because you have hostnames that you want to add to the `local-host-names` file. As an example, let's create a `local-host-names` file for our sample system that contains the following lines:

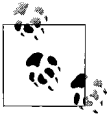
```
wrotethebook.com
horseshoe.wrotethebook.com
```

After creating the `local-host-names` file and building the new `sendmail.cf` file, use the `-bt` option to test the new configuration. First, run `sendmail -bt` to examine the contents of the class `$=w` array:

```
# sendmail -bt -C./sendmail.cf
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> $=w
chef
localhost.localdomain
localhost
[192.168.0.8]
[127.0.0.1]
[chef.wrotethebook.com]
horseshoe.wrotethebook.com
wrotethebook.com
chef.wrotethebook.com
> /quit
```

* The generic Linux configuration created in Recipe 1.8 specifies the `use_cw_file` feature in the `domain/generic.m4` file. If the feature is already defined, it does not need to be added.

This test shows that the *horseshoe.wrotethebook.com* and the *wrotethebook.com* values stored in the *local-host-names* file are added to class `$=w`. It also shows that several other values are stored in this array. These values are all of the hostnames, hostname aliases, and IP addresses assigned to this host that sendmail discovered by probing the various network interfaces. The hostnames associated with the interface addresses are obtained from both the */etc/hosts* file and from DNS.*



Limit the interface probing done by sendmail by adding the following define to the sendmail configuration:

```
define(`confDONT_PROBE_INTERFACES', `true')
```

When this define is used, only the hostnames assigned to the system in the */etc/hosts* file and in DNS, and the system's primary IP address are used as starting values for class `$=w`. This define is not used when you want to accept mail addressed to any of your server's interfaces as local mail. However, the `confDONT_PROBE_INTERFACES` define is very useful when probing the interfaces gives sendmail erroneous information or when a large number of virtual interfaces are used.

Mail addressed to any hostname defined in class `$=w` is delivered by the local mailer. The `$=w` output shown above tells us that this system will deliver mail addressed to *chef*, *localhost*, *horseshoe*, or *wrotethebook.com* using the local mailer. A second test, this time using `sendmail -bv`, verifies this:

```
# hostname
chef
# sendmail -bv alana@crab.wrotethebook.com
alana@crab.wrotethebook.com... deliverable: mailer esmtp, host crab.wrotethebook.com.
, user alana@crab.wrotethebook.com
# sendmail -bv payton@horseshoe.wrotethebook.com
payton@horseshoe.wrotethebook.com... deliverable: mailer local, user payton
```

The `hostname` command shows that the local host is *chef* to illustrate that none of this mail is actually addressed to the local host. The `sendmail -bv` command verifies a delivery address and shows the mailer that will be used to deliver the mail. The first `-bv` test shows that mail addresses to *crab* will be delivered over the network using the `esmtp` mailer. This is just what you would expect because *crab* is an external system. Yet the second `-bv` test shows that mail addressed to *horseshoe*, another external system, will be delivered using the local mailer. Mail addressed to *horseshoe* is treated as if it were addressed to *chef*. This is because *horseshoe* is defined in the *local-host-names* file, and the configuration uses the *use_cw_file* feature to add hostnames from that file to class `$=w`. This test shows that the system is properly configured to handle mail addressed to *horseshoe* as if it were local mail.

* The hostname *localhost.localdomain* is a value that Red Hat puts in the */etc/hosts* file for the loopback address (127.0.0.1).

Using `sendmail.cf` directly

The `use_cw_file` feature works well, and it is the recommended solution for this problem, but there are alternative solutions. For example, placing local hostnames in `sendmail.cf` itself or using the `bestmx_is_local` feature. The sample `local-host-names` file created earlier in this section contained only two lines. Such a small number of names could be placed directly inside of `sendmail.cf` by using the following two lines instead of the `use_cw_file` feature:

```
LOCAL_DOMAIN('wrotethebook.com')
LOCAL_DOMAIN('horseshoe.wrotethebook.com')
```

This alternative solution, however, is more difficult to maintain as the list of clients grows. Every time a `LOCAL_DOMAIN` macro is added to the configuration, the `sendmail.cf` file must be rebuilt, tested, and moved to the `/etc/mail` directory. When the `local-host-names` file is used, there is no need to rebuild `sendmail.cf` just because the `local-host-names` file has been edited.

Using the `bestmx_is_local` feature

The `bestmx_is_local` feature works well if the only reason that hostnames are being added to the `local-host-names` file is because the local host is a mail exchanger. Mail addressed to any system that lists the local host as its preferred mail exchanger is accepted as local mail when the `bestmx_is_local` feature is used. To use this approach, put the following line in the configuration:

```
FEATURE(`bestmx_is_local', `wrotethebook.com')
```

The great advantage of the `bestmx_is_local` feature is that it is easy—there is no `local-host-names` file to maintain. However, that simplicity is also a disadvantage because control over what mail is accepted as local by your system is given to someone else—the domain administrator. The potential problems caused by this lack of control are limited by using an optional domain list in the sample command. Adding the ``wrotethebook.com'` argument to the `bestmx_is_local` feature means that only MX records from that domain are accepted as proof that a host is a valid client. If the optional domain list is not specified for the `bestmx_is_local` feature, any domain in the world can control the mail your system accepts as local simply by putting in an MX record that points to your host, which creates an opportunity for abuse.

Another potential problem with the `bestmx_is_local` solution is that it increases the processing overhead for each piece of mail. This would not be a problem for our small sample system, but it could be a problem for any system that deals with a high volume of mail.

One other limitation of the `bestmx_is_local` solution is that it depends completely on MX records. `bestmx_is_local` works for a mail exchanger because all of the hostnames it handles have MX records; however, it is possible to have other reasons to accept mail as local mail. The `local-host-names` file can store any hostnames that you

wish; it is not limited to hosts that define your system as their mail exchanger. *local-host-names* is fast, flexible, and completely under your control. For those reasons, we chose the *use_cw_file* feature as the preferred solution for this problem.

See Also

The *sendmail* book covers *use_cw_file*, `confCW_FILE`, and `LOCAL_DOMAIN` in 4.8.48; *bestmx_is_local* in 4.8.7; and `confDONT_PROBE_INTERFACES` in 24.9.39.

2.2 Fixing the Alias0 Missing Map Error and Creating Simple Aliases

Problem

sendmail displays the following error message:

```
hash map "Alias0": missing map file /etc/mail/aliases.db: No such file or directory
```

Solution

Create a text file named */etc/mail/aliases* and populate that file with entries in the form of:

```
alias: recipient
```

where *alias* is an alias username and *recipient* is the address to which mail addressed to the alias is delivered.

Run *newaliases* to process the text file and build the *aliases* database file required by sendmail:

```
# newaliases
```

Discussion

No *m4* configuration commands are needed to access the *aliases* database. Any basic sendmail configuration should work.

Most systems come with an *aliases* database. All you need to do is add the aliases that you desire to the existing *aliases* text file and run the *newaliases* command. However, if your system displays the “missing map file” error shown in the problem description, you either do not have an *aliases* database or the one you have is placed in the wrong directory. In either case, you need to put a valid *aliases* database in the proper path. If your system does not have an *aliases* file, use one from a similar system or the sample *aliases* file from the *sendmail* directory of the sendmail distribution as a starting point. If the file you have is in the wrong path, move it or change the sendmail configuration to point to the place where you keep the *aliases* database.

Some older systems store the *aliases* database in the */etc* directory. To keep it in that directory, add the following `ALIAS_FILE` define to the *sendmail.mc* file:

```
define(`ALIAS_FILE', `/etc/aliases')
```

The `ALIAS_FILE` define can accept various flags. You can make the *aliases* database optional using the `-o` flag exactly as it was used in Recipe 2.1 with the `confCW_FILE` define. However, this is not a good idea. The `-o` flag prevents sendmail from displaying the “missing map file” error, but it does not stop sendmail from turning off aliasing when it cannot find an *aliases* file (i.e., it hides the error but does not fix the problem).

Instead of changing the default path or hiding the error with an `ALIAS_FILE` define, place the *aliases* text file in the */etc/mail* directory. This is probably the best solution for most versions of Unix because */etc/mail* is the default directory where most administrators expect to find this file. The following example moves the *aliases* text file and runs *newaliases* to solve the “missing map file” problem:

```
# sendmail -bv ftp
hash map "Alias0": missing map file /etc/mail/aliases.db: No such file or directory
ftp... deliverable: mailer local, user ftp
# mv aliases /etc/mail/aliases
# sendmail -bv ftp
hash map "Alias0": missing map file /etc/mail/aliases.db: No such file or directory
root... deliverable: mailer local, user root
# newaliases
/etc/mail/aliases: 40 aliases, longest 10 bytes, 395 bytes total
# sendmail -bv ftp
root... deliverable: mailer local, user root
```

The `mv` command and the *newaliases* command solve the problem. The three `sendmail -bv ftp` lines are used to show the effect of the `mv` and *newaliases* commands. The first `-bv` command shows the error message that we expect, and it shows that mail addressed to *ftp* will be sent to username *ftp* via the local mailer. After the `mv` command is executed, the second `-bv` command again shows the error message because we have not yet built the *aliases* database from the *aliases* text file. However, this time the `-bv` command shows that mail addressed to *ftp* will be delivered to the *root* user account via the local mailer. Clearly something has happened, and that something is aliasing. sendmail used the *aliases* text file to find that the correct delivery address for the *ftp* alias is *root*. Finally, *newaliases* is executed to build the database, and the last `-bv` test is run. This time there is no error, and we are again told the mail will be delivered to *root* via the local mailer. The output of the *newaliases* command says that *newaliases* processed 40 aliases.

The three `sendmail -bv` tests show exactly how sendmail handles the *aliases* files. If sendmail can find the *aliases* database, it uses it. If no *aliases* database is found, sendmail displays an error and checks for the *aliases* text file. If it finds the text file, sendmail uses it. If neither the database nor the text file can be found, sendmail turns off aliasing.

Entries from the sample Red Hat *aliases* file are shown below:

```
# Basic system aliases -- these MUST be present.
mailer-daemon: postmaster
postmaster:    root

# General redirections for pseudo accounts.
bin:           root
daemon:       root
adm:          root
lp:           root
sync:         root
shutdown:     root
halt:         root
mail:         root
news:         root
ftp:          root

# trap decode to catch security attacks
decode:        root

# Person who should get root's mail
#root:         marc
```

Lines that begin with a hash mark (#) are comments and can be ignored. The active entries begin with an alias and a colon that is followed by the recipient address. Notice that the recipient can be another alias. For example, *mailer-daemon* points to *postmaster*, which is really an alias for *root*. A `-bv` test shows the effect of this double alias:

```
$ sendmail -bv mailer-daemon
root... deliverable: mailer local, user root
```

Most of the entries in the sample Red Hat *aliases* file route mail addressed to non-login system accounts to the *root* user. No one actually logs in as *root*, yet all of the mail addressed to system accounts is routed to the *root* account. Notice the last comment in the sample *aliases* file is a commented-out alias for *root*. Remove the hash mark from the beginning of this line and change the recipient field to the user ID of the administrator's login account to route mail addressed to *root* to someplace where it will be read. Here is an example:

```
# Person who should get root's mail
root:          logan
```

In this example, the administrator logs in to the *logan* account to read *root*'s mail. A `sendmail -bv` command shows the effect of this alias:

```
# sendmail -bv bin
logan... deliverable: mailer local, user logan
```

`sendmail` first looks up *bin* and finds that it points to *root*. It then looks up *root* and finds that it points to *logan*. *logan* is not an alias, so delivery is made to the *logan* account. In this case, the aliases were nested two levels deep. By default, `sendmail`

will search up to 10 levels of nested aliases and then display the following error if the alias does not resolve to a delivery address:

```
aliasing/forwarding loop broken (11 aliases deep; 10 max)
aliasing/forwarding loop broken
```

Add the `confMAX_ALIAS_RECURSION` define to the sendmail configuration to change the depth to which sendmail will search nested aliases; for example:

```
define(confMAX_ALIAS_RECURSION, 5)
```

The line shown above reduces recursion to a maximum of five nested aliases. However, there is usually no good reason to change the default provided by sendmail. Reducing the amount of recursion doesn't enhance performance because the only time the maximum amount of recursion is reached is on those rare occasions that you have created an alias loop. You never want more recursion because if you need more than 10 levels of nesting you have designed your aliases incorrectly, which impacts sendmail's performance. As with most sendmail defines, the default value is the best.

You can control alias recursion for individual aliases without using an *m4* macro or changing the *sendmail.cf* file. Force aliasing to stop at a specific alias by placing a backslash in front of the recipient address for that alias. For example:

```
bin:          \root
```

This is a contrived example. Of course, you don't really want to deliver mail to the *root* account. However, this contrived example works well to illustrate the impact of the backslash because the specific recursion of *bin* to *root* to *logan* is described just a few paragraphs back. Rerunning the `sendmail -bv` test after this alias is inserted in the *aliases* database shows the following result:

```
# sendmail -bv bin
\root... deliverable: mailer local, user \root
```

Here the *bin* alias resolves to *root* and aliasing terminates—sendmail does not go on to resolve *root* to *logan*. Usually, this type of recipient address is used in combination with the mail being copied to a file or forwarded to a program, and it is most commonly found in the *.forward* file instead of the *aliases* file. It is used in a *.forward* file in Recipe 2.9.

After creating the *aliases* text file, run *newaliases* to build the *aliases* database. The *aliases* database is actually built by sendmail; *newaliases* is really a symbolic link to the sendmail program and is equivalent to the `sendmail` command with the `-bi` flag.

See Also

Recipe 2.3 is a related recipe that shows how aliases can be read from an LDAP server. The *sendmail* book covers the *aliases* database in Chapter 12.

2.3 Reading Aliases via LDAP

Problem

When your organization stores aliases on an LDAP server, you need to configure sendmail to read aliases from the LDAP server.

Solution

Use `sendmail -bt -d0.1` to check the sendmail compiler options. If sendmail was not compiled with LDAP support, recompile and reinstall sendmail as described in Recipe 1.3.

If the LDAP server has not yet been configured to support sendmail queries, copy the *sendmail.schema* file to the appropriate location on the LDAP server and update the server configuration to use the schema file. Recipe 1.3 covers the necessary LDAP server configuration.

Once the LDAP server is configured to support the sendmail schema, add sendmail aliases to the LDAP database using the format defined by that schema. When OpenLDAP is used, this is done by first creating an LDIF file containing the LDAP records and then running *ldapadd* to add these records to the LDAP database. The Discussion section shows an example of doing this for alias records.

On the sendmail system, add an `ALIAS_FILE` define, containing the string `ldap:`, to the sendmail configuration. Also add a `confLDAP_CLUSTER` define containing the same value as the `sendmailMTACluster` attribute used in the entries added to the LDAP server. Here is an example of these configuration commands:

```
# Set the LDAP cluster value
define(`confLDAP_CLUSTER', `wrotethebook.com')
# Tell sendmail that aliases are available via LDAP
define(`ALIAS_FILE', `ldap:')
```

Build the sendmail configuration file, copy it to */etc/mail/sendmail.cf*, and restart sendmail with the new configuration, as described in Recipe 1.8.

Discussion

This recipe provides instructions for both the sendmail administrator and the LDAP administrator because the LDAP server must be properly installed, configured, and running, and it must include the sendmail schema in order to understand and properly process queries from sendmail. Nothing in this recipe will work without the close cooperation of the LDAP administrator. In fact, the bulk of the configuration takes place on the LDAP server. You should have some experience with LDAP before attempting to use it with sendmail.

To add sendmail aliases to the LDAP database, start by building an LDIF file formatted according to the sendmail schema. Here is an example that adds the *mailer-daemon*, *postmaster*, and *root* aliases from Recipe 2.2 to the LDAP database:

```
# cat > ldap-aliases
dn: sendmailMTAKey=mailer-daemon, dc=wrotethebook, dc=com
objectClass: sendmailMTA
objectClass: sendmailMTAAlias
objectClass: sendmailMTAAliasObject
sendmailMTAAliasGrouping: aliases
sendmailMTACluster: wrotethebook.com
sendmailMTAKey: mailer-daemon
sendmailMTAAliasValue: postmaster

dn: sendmailMTAKey=postmaster, dc=wrotethebook, dc=com
objectClass: sendmailMTA
objectClass: sendmailMTAAlias
objectClass: sendmailMTAAliasObject
sendmailMTAAliasGrouping: aliases
sendmailMTACluster: wrotethebook.com
sendmailMTAKey: postmaster
sendmailMTAAliasValue: root

dn: sendmailMTAKey=root, dc=wrotethebook, dc=com
objectClass: sendmailMTA
objectClass: sendmailMTAAlias
objectClass: sendmailMTAAliasObject
sendmailMTAAliasGrouping: aliases
sendmailMTACluster: wrotethebook.com
sendmailMTAKey: root
sendmailMTAAliasValue: logan
Ctrl-D
# ldapadd -x -D "cn=Manager,dc=wrotethebook,dc=com" \
> -W -f ldap-aliases
Enter LDAP Password: SecretLDAPpassword
adding new entry "sendmailMTAKey=mailer-daemon, dc=wrotethebook, dc=com"
adding new entry "sendmailMTAKey=postmaster, dc=wrotethebook, dc=com"
adding new entry "sendmailMTAKey=root, dc=wrotethebook, dc=com"
```

The example just shown names the LDIF file *ldap-aliases*. Running the *ldapadd* command adds these entries to the LDAP database. A quick check with the *ldapsearch* command shows the newly added records:

```
# ldapsearch -x '(objectclass=sendmailMTAAlias)' \
> sendmailMTAKey sendmailMTAAliasValue
version: 2
#
# filter: (objectclass=sendmailMTAAlias)
# requesting: sendmailMTAKey sendmailMTAAliasValue
#
# mailer-daemon, wrotethebook, com
dn: sendmailMTAKey=mailer-daemon, dc=wrotethebook, dc=com
sendmailMTAKey: mailer-daemon
sendmailMTAAliasValue: postmaster
```

```

# postmaster, wrotethebook, com
dn: sendmailMTAKey=postmaster, dc=wrotethebook, dc=com
sendmailMTAKey: postmaster
sendmailMTAAliasValue: root

# root, wrotethebook, com
dn: sendmailMTAKey=root, dc=wrotethebook, dc=com
sendmailMTAKey: root
sendmailMTAAliasValue: logan

# search result
search: 2
result: 0 Success

# numResponses: 4
# numEntries: 3

```

Notice that this *ldapsearch* command works without either an *-h* or a *-b* argument. (*-h* defines the LDAP server name and *-b* defines the LDAP default base distinguished name.) This test works without these arguments because the correct LDAP server hostname and base distinguished name are defined in the *ldap.conf* file. If those *ldap.conf* values are not correct for the sendmail query, provide the correct *-h* and *-b* values on the *ldapsearch* command line. If you must provide *-h* and *-b* values to *ldapsearch* to successfully run this test, the sendmail administrator must provide the same values to sendmail. Use the `confLDAP_DEFAULT_SPEC` define to set *-h* and *-b* values for sendmail. See Recipe 5.9 for an example of how `confLDAP_DEFAULT_SPEC` is used.

The *ldapsearch* test shows that the LDAP server is now ready to answer sendmail's queries for aliases. sendmail must also be properly prepared to work with LDAP. sendmail must be compiled with LDAP support, as described in Recipe 1.3, and it must have the correct configuration. In most cases, adding two defines to a basic sendmail configuration is all that is required to configure sendmail to read aliases via LDAP. The two defines are:

```

define(`confLDAP_CLUSTER', `wrotethebook.com')
define(`ALIAS_FILE', `ldap:')

```

`ALIAS_FILE` defines the location of the *aliases* database. Instead of providing a file path as the argument for the `ALIAS_FILE` define, the example provides the string `ldap:`, which tells sendmail to read aliases from the LDAP server using the standard sendmail schema. The `ALIAS_FILE` define just shown is equivalent to:

```

define(`ALIAS_FILE', `ldap: -k (&(objectClass=sendmailMTAAliasObject)
(sendmailMTAAliasGrouping=aliases) (|(sendmailMTACluster=${sendmailMTACluster})
(sendmailMTAHost=${j})) (sendmailMTAKey=%0)) -v sendmailMTAAliasValue')

```

The expanded command shows the search key, which is defined by the *-k* argument, and the return value, which is defined by the *-v* argument. The return value is easy to understand, it is the value stored in the `sendmailMTAAliasValue` attribute. The search key is more complex; however, it is the same search criteria syntax used with the *ldapsearch* command, which is something all LDAP administrators are familiar with.

The tricky part is that the key combines basic LDAP search criteria with sendmail macros. `${sendmailMTACluster}` holds the value defined by `confLDAP_CLUSTER` in the master configuration file. `$j` returns the fully qualified name of the local host. In this case, `%0` is the alias for which sendmail is searching. Put all together, this key searches for a record with:

- An `objectClass` of `sendmailMTAAliasObject`
- A `sendmailMTAAliasGrouping` of aliases
- Either a `sendmailMTACluster` matching the value defined by `confLDAP_CLUSTER` or a `sendmailMTAHost` attribute containing the name of the local host
- A `sendmailMTAKey` matching the desired alias

The effect of the search key can be easily simulated using the `ldapsearch` command:

```
# ldapsearch -LLL -x '(&(objectClass=sendmailMTAAliasObject) \
> (sendmailMTAAliasGrouping=aliases) \
> (|(sendmailMTACluster=wrotethebook.com) \
> (sendmailMTAHost=rodent.wrotethebook.com)) \
> (sendmailMTAKey=postmaster))' \
> sendmailMTAAliasValue
dn: sendmailMTAKey=postmaster, dc=wrotethebook, dc=com
sendmailMTAAliasValue: root
```

This `ldapsearch` command shows the key that sendmail would use to look up the `postmaster` alias when running on a host named `rodent.wrotethebook.com` with `confLDAP_CLUSTER` defined as `wrotethebook.com`. The value returned is `root`.

This recipe uses the `confLDAP_CLUSTER` define to set a value for `${sendmailMTACluster}` because all of the entries added to the LDAP database in this recipe contain a `sendmailMTACluster` attribute. In order to match those LDAP records, sendmail queries must contain the correct `${sendmailMTACluster}` value.

The alternative to using a cluster value is to have LDAP database entries defined for individual hosts. In that case, the LDAP entry does not use the `sendmailMTACluster` attribute. Instead, it uses the `sendmailMTAHost` attribute, and the value assigned to the attribute is the fully qualified hostname of a specific host. If you decide to create LDAP entries for individual hosts, the `sendmailMTAHost` attribute must be specified in each LDAP entry. But no special host value needs to be configured for sendmail because sendmail uses the value in `$j`. When the `sendmailMTACluster` attribute is not used on the LDAP records, the `confLDAP_CLUSTER` define is not required for the sendmail configuration. However, when the `confLDAP_CLUSTER` define is not used, sendmail can retrieve only those LDAP records that contain a `sendmailMTAHost` attribute that matches the value returned by `$j`.

The cluster value provides a way for a group of hosts to share common LDAP data. It is similar to a NIS domain. The sample `confLDAP_CLUSTER` define uses the DNS domain name as the cluster value. However, the cluster value is arbitrary and does not need to be a NIS or DNS domain name.

After configuring the LDAP server and the sendmail system, a test shows that aliases are successfully retrieved from the LDAP server. The effect of the aliases can be seen using the `sendmail -bv` command:

```
# sendmail -bv -Cgeneric-linux.cf mailer-daemon
root... deliverable: mailer local, user root
# sendmail -bv mailer-daemon
logan... deliverable: mailer local, user logan
```

This test shows the LDAP server in action. When the first test is run using the generic configuration, the *mailer-daemon* alias is resolved using the local *aliases* database because the generic configuration does not override the default `ALIAS_FILE` path. The second test uses the sendmail configuration created in this recipe, which points to the LDAP server. The *mailer-daemon* alias is resolved using the three records added to the LDAP database earlier in this section.

See Also

Refer to the other recipes in this chapter for descriptions of the various alias formats—all of which can be read from an LDAP server. For information on LDAP, see *Understanding and Deploying LDAP Directory Services* by Howes, Smith, and Good (Macmillan) and *LDAP System Administration* by Gerald Carter (O’Reilly). The *cf/README* file covers this topic in the *Using LDAP for Aliases, Maps, and Classes* section. The *sendmail* book covers the `ALIAS_FILE` define in section 24.9.1 and the `confLDAP_CLUSTER` define in section 21.9.82.

2.4 Configuring Red Hat 7.3 to Read Aliases from a NIS Server

Problem

You want to use NIS to access the *aliases* database on a Red Hat 7.3 system.

Solution

Change to the `/etc/mail` directory and create a *service.switch* file. Here is an example:

```
# cd /etc/mail
# cat - > service.switch
aliases nis files
hosts dns files
passwd files nis
```

Discussion

No *m4* configuration commands are needed to enable support for the *service.switch* file—it is available by default. The basic sendmail configuration already installed on your system will probably work with this file; the generic configuration discussed in Recipe 1.8 certainly will.

Network Information Service (NIS), developed by Sun Microsystems, makes many system administration databases available over the network. NIS allows an organization to centrally control and maintain important configuration files for all Unix clients. The *aliases* database is one of the files that can be centrally maintained and accessed through NIS.

A service switch file defines the sources for various system administration files and the order in which those sources should be queried. *service.switch* is the default filename that sendmail uses for a service switch file if the system does not have a service switch file that sendmail can use.* When running on a Red Hat Linux system, current versions of sendmail look for *service.switch* in the */etc/mail* directory; earlier versions looked for the file in the */etc* directory. The path to the sendmail service switch file can be changed using the `confSERVICE_SWITCH_FILE` define. However, it is generally easier to place the *service.switch* file in the */etc/mail* directory where sendmail, and most administrators, expect to find it.

The sendmail *service.switch* file is much shorter than the system's service switch file because sendmail is only interested in three types of system administration information: mail aliases, host information, and user information.

The *service.switch* file contains an entry for each type of information identified by the name of the file that traditionally provides the information: *aliases*, *hosts*, or *passwd*. The name is followed by the list of sources for that type of information. The aliases entry in the example is:

```
aliases nis files
```

NIS is listed first, meaning that sendmail will first attempt to resolve an alias via NIS and will only look the alias up in the local *aliases* file if NIS cannot resolve the alias. Thus, aliases that only exist in the local file will still be resolved, but aliases in the NIS map file will take precedence.

Several sources of information can be specified in the *service.switch* file:

files

The local */etc/mail/aliases*, */etc/hosts*, and */etc/passwd* files.

nis

NIS Version 2, which can be used for *aliases*, *hosts*, and *passwd* information.

* Red Hat 7.3 has a service switch file named */etc/nsswitch.conf*. However, as the manpage makes clear, the aliases entry in the *nsswitch.conf* file is ignored.

nisplus

NIS Version 3, which can be used for *aliases*, *hosts*, and *passwd* information.

ldap

LDAP can be used for *aliases*, *hosts*, and *passwd* information. Recipe 2.3 shows the correct way to read aliases from an LDAP server.

hesiod

The Hesiod service can be used for *aliases*, *hosts*, and *passwd* information, but it is primarily used for *passwd* information.

dns

DNS can be used for *hosts* information.

To use any of these services, your system must be able to act as a client for the service and must be properly configured. Configuring these services is beyond the scope of this book.

Wrapping some tests around our recipe shows the immediate impact of our solution:

```
# praliases bin
bin:craig
# ypmatch bin aliases
root
# sendmail -bv bin
craig... deliverable: mailer local, user craig
# cd /etc/mail
# cat - > service.switch
aliases nis files
hosts dns files
passwd files nis
Ctrl-D
# sendmail -bv bin
root... deliverable: mailer local, user root
# praliases kathy
kathy:kathy@chef.wrotethebook.com
# ypmatch kathy aliases
Can't match key kathy in map mail.aliases. Reason: No such key in map
# sendmail -bv kathy
kathy@chef.wrotethebook.com... deliverable: mailer esmtp, host chef.wrotethebook.com.
, user kathy@chef.wrotethebook.com
```

The first `praliases` command shows the value stored for the *bin* alias in the local *aliases* database, and the first `ypmatch` command shows the value stored for that alias in the NIS database. The local file maps *bin* to *craig* and the NIS database maps *bin* to *root*—a more reasonable value. The first `sendmail -bv` test shows that the delivery address from the local file is being used. Then we prepare our simple recipe. As soon as the *service.switch* file is built, the `sendmail -bv` test is rerun. This time, the delivery address used for *bin* is the one defined on the NIS server. Clearly, our simple recipe causes the local system to use NIS to resolve aliases.

The last three lines in the test illustrate the fact that the local file is still used when NIS cannot resolve an alias. In this case, an alias exists for *kathy* in the local file but not in the NIS map. Still, mail will be delivered to *kathy* using the value from the local file.

Alternatives

An alternative to creating a *service.switch* file is to define the sources of information, and the order in which they are searched, using the `ALIAS_FILE` define in *sendmail.cf*. For example, the following define would do the same thing as the aliases entry from the *service.switch* file in our recipe:

```
define(ALIAS_FILE, `nis:mail.aliases,/etc/mail/aliases')
```

This define tells sendmail that the first source of alias information is the NIS *mail.aliases* map, and the second source of information is the local */etc/mail/aliases* file. To implement this alternative solution, add this line to the *sendmail.mc* file, rebuild the *sendmail.cf* file, copy the *sendmail.cf* file to */etc/mail*, and restart sendmail.

This alternative solution is harder to implement and more difficult to maintain than the *service.switch* file. Changing the search order requires modifying *sendmail.mc*, rebuilding *sendmail.cf*, copying the *sendmail.cf* file to the correct directory, and restarting sendmail. To modify the search order using the *service.switch* file, simply change the aliases line in that file. The change is immediately effective. Time and again, the best sendmail configuration solutions do not involve any changes to the *sendmail.cf* file.

One final note about this alternative. Don't use both solutions at the same time. If we used both the *service.switch* file created in our recipe and the `ALIAS_FILE` define just shown, sendmail would first follow the instructions in the *service.switch* file and lookup the alias via NIS and in the local *aliases* file. Then sendmail would perform any optional processing requested by the `ALIAS_FILE` define, meaning it would lookup the same alias with NIS again. This is not a major problem, but it does add unnecessary overhead.

See Also

Recipe 2.5 provides a solution to a similar configuration problem for Solaris. The *sendmail* book covers the *service.switch* file in 12.1.1, the `confSERVICE_SWITCH_FILE` define in 24.9.100, and the `ALIAS_FILE` define in 24.9.1. *Managing NFS and NIS*, Second Edition, by Stern, Eisler, and Labiaga (O'Reilly), covers NIS in detail.

2.5 Configuring Solaris 8 to Read Aliases from a NIS Server

Problem

You must configure a Solaris 8 system to read aliases from a NIS server.

Solution

Edit the aliases entry in the */etc/nsswitch.conf* file, changing it to the following:

```
aliases: nis files
```

Discussion

No *m4* configuration commands are needed to use the *nsswitch.conf* file on a Solaris system. The basic sendmail configuration already installed on your system should work. If your system is not yet configured to run sendmail, use the *generic-solaris.mc* file as a starting point to get your Solaris server running.

Both the Linux system and the Solaris system have a system-wide service switch file named */etc/nsswitch.conf*. The difference is that the Red Hat Linux 7.3 system does not use the aliases entry in that file, and the Solaris system does. When the system has an active system service switch file, sendmail uses that file and ignores the file identified by the `confSERVICE_SWITCH_FILE` define. The correct solution for the Solaris system is to identify the sources of alias information in the *nsswitch.conf* file. Any source that is compatible with the *nsswitch.conf* syntax and with sendmail can be specified. See the *nsswitch.conf* manpage for information about the syntax supported on your computer.

Recipes 2.3 and 2.4 also show aliases being read from an external server. But Recipe 2.2 and most examples in this book assume the *aliases* database is a local file located directly on the sendmail system. This is a common design, and there are some good reasons why.

Aliases are only searched for mailers that have the A flag set. On most systems, only the local mailer has this flag set, which means that aliases only apply to inbound mail that has been accepted for delivery through the local mailer. A centralized database that includes aliases for all of the users in the organization is needed only by a server that will accept mail for all of the users in the organization. For most Unix workstations, the bulk of the entries in a centralized database are unused—only those users who actually receive their mail at the workstation are looked up.

Placing the *aliases* file directly on the system that needs it improves performance and reduces network overhead. It also enhances security—data that passes over the network is subject to corruption and spoofing, and adding another protocol to the mix,

in this case NIS, makes the system vulnerable to any bugs that might appear in that protocol.

Security, performance, and applicability limit the demand for a centralized *aliases* database accessible through the network. Think hard about why you want to put the *aliases* database on an external NIS or LDAP server before you proceed. However, if you decide to read aliases from an external server, sendmail can be configured to do so.

See Also

Recipe 2.4 provides a solution to the same problem for different operating systems. The *nsswitch.conf* file is covered in the books *TCP/IP Network Administration*, Third Edition, by Craig Hunt (O'Reilly), and *Managing NFS and NIS*, Second Edition, by Stern, Eisler, and Labiaga (O'Reilly).

2.6 Forwarding to an External Address

Problem

Mail addressed to the local host needs to be forwarded to another host for final delivery.

Solution

Add an alias to the *aliases* file for each user whose mail must be forwarded to another system. The recipient field of the alias entry must be a full email address that includes the host part.

After adding the desired aliases, rebuild the *aliases* database file with the *newaliases* command.

Discussion

There are many reasons why mail delivered to your system must then be forwarded to another host. For example, a user may have changed addresses, a user may have several addresses but only read mail at one of them, or your host might be a mail hub that collects mail for several different systems. A mail exchanger is a good example of a system that receives mail that might need to be forwarded to another host for delivery.

A mail exchanger receives mail addressed to other hosts and accepts that mail as local mail. (This is discussed in Recipe 2.1.) It is possible, however, that not every user on every host that routes mail to the mail exchanger has a local account on the mail exchanger. Two `sendmail -bv` tests illustrate this:

```
# sendmail -bv pat@wrotethebook.com
pat@wrotethebook.com... deliverable: mailer local, user pat
```

```
# sendmail -bv andy@wrotethebook.com
andy@wrotethebook.com... User unknown
```

In the first case, *pat* is a user account on the mail exchanger, so the mail can be delivered directly to Pat's local mailbox. In the second case, *andy* is not a local user account, so the mail is bounced with a "User unknown" error. Assume, however, that this mail should not be returned with an error because Andy is a current employee who is allowed to receive mail at the address *andy@wrotethebook.com*. Andy doesn't have an account on the mail exchanger, but he does have a valid account on another system in the *wrotethebook.com* domain, and it is on that system that Andy reads his mail.

The "User unknown" error is solved by creating the correct aliases. For example, on our sample mail exchanger, we might add the following alias for *andy*:

```
andy:                andy@rodent.wrotethebook.com
```

The alias is available as soon as *newaliases* is run. Rerunning the `sendmail -bv test` that previously produced the "User unknown" error shows the following result:

```
# sendmail -bv andy@wrotethebook.com
andy@rodent.wrotethebook.com... deliverable: mailer esmtp, host rodent.wrotethebook.
com., user andy@rodent.wrotethebook.com
```

The error is gone. Now, mail addressed to *andy@wrotethebook.com* is forwarded through the esmtp mailer to *andy@rodent.wrotethebook.com*. Remote users do not even need to know that a host named *rodent* exists. They send mail to *user@wrotethebook.com*, and the mail exchanger forwards it to the correct host for delivery.

See Also

Recipe 2.2 provides instructions on building a basic *aliases* file. The *sendmail* book covers the *aliases* database in Chapter 12.

2.7 Creating Mailing Lists

Problem

You have been asked to create mailing lists.

Solution

Add an entry to the *aliases* text file for each mailing list, where the alias field contains the name of the mailing list, and the recipient field contains a comma-separated list of all of the recipients of the mailing list.

Add the special alias that sendmail uses to deliver error messages concerning the mailing list. This special alias must have a name in the format of *owner-list*, where *owner-* is a required string and *list* is the name of the list for which this owner alias is being declared. For example, if the list is named *admin*, this alias must be named *owner-admin*. The recipient field of the special alias should contain the address where the errors are to be sent.

For the convenience of users, add an alias that can be used to reach the person who maintains the mailing list. The format of the alias field should be *list-request*, where *list* is the name of the list for which this request alias is being declared, and *-request* is a required string. For example, if the list is named *admin*, this alias should be named *admin-request*.

Run *newaliases* to rebuild the *aliases* database file:

```
# newaliases
/etc/mail/aliases: 43 aliases, longest 30 bytes, 592 bytes total
```

Discussion

To illustrate how to create mailing lists, let's assume we have been asked to create three different mailing lists:

- Mail addressed to *root* is to be sent to the normal login accounts of the three people who administer the mail server.
- The boss wants to have a mailing list for the management council, which is composed of the four department chiefs and their scientific advisor, and he wants an archive kept of all the mail sent to that mailing list.
- The boss wants a mailing list for all mail users.

To do this, create an *admin* mailing list for the system administrators, a *chiefs* list for the department heads, and an *employees* mailing list for all mail users, similar to the following:

```
# Mailing lists

admin:          alana, logan, pat
owner-admin:    admin-request
admin-request:  alana

chiefs:         reba, sara, tyler@example.com,
                jane@rodent.wrotethebook.com, kathy,
                /home/payton/chiefs-archive
owner-chiefs:   chiefs-request
chiefs-request: alana

employees:      :include:/etc/mail/allusers
owner-employees: employees-request
employees-request: alana
```

Next, edit any aliases in the file that need to reference the newly created aliases. In this example, we want mail addressed to *root* to be sent to the newly created *admin* alias so we make the following change:

```
# Person who should get root's mail
root:          admin
```

The first mailing list is the simplest. It defines three local recipients (*alana*, *logan*, and *pat*) who will receive copies of mail addressed to *admin*. The address *admin* can be used directly to reach these three users, as the following `sendmail -bv` command shows:

```
# sendmail -bv admin
pat... deliverable: mailer local, user pat
logan... deliverable: mailer local, user logan
alana... deliverable: mailer local, user alana
```

We also want mail addressed to *root* to be sent to the three administrators. To that end, we edited the *aliases* text file to point the *root* alias to the *admin* mailing list. Another `sendmail -bv` command illustrates the impact of that edit:

```
# sendmail -bv bin
pat... deliverable: mailer local, user pat
logan... deliverable: mailer local, user logan
alana... deliverable: mailer local, user alana
```

In this case, `sendmail` looks up the *bin* alias and finds that it points to *root*. It then looks up the *root* alias and finds that it points to *admin*. When it looks up *admin*, it finds that it points to *alana*, *logan*, and *pat*. Each of these are looked up and discovered to be delivery addresses that are deliverable through the local mailer.

Immediately following the *admin* alias is an alias with the name *owner-admin*. This is a special alias that `sendmail` expects to find. `sendmail` uses it to deliver error messages if it encounters any errors delivering mail to the *admin* mailing list.

The next alias, *admin-request*, is required only because it is referenced by the *owner-admin* alias; however, using aliases in this format is common with mailing lists because they provide two benefits. First, many users expect to be able to reach the person who maintains the list at an alias in the format of *list-request*—providing this alias meets these user expectations. A second benefit comes from the recipient field of the *owner-list* alias, which is used to construct the Unix From line and the Return-Path: header. Refer back to our example. If the *owner-admin* alias pointed directly to *alana*, the Return-Path: and the Unix From line for mail from the *admin* mailing list would appear to come from *alana*. End users who poke around in the headers and see the Unix From information may assume the mail was sent to them directly by *alana* instead of coming from the mailing list. Pointing the *owner-admin* alias to *admin-request* means that those users see *admin-request* as the source of the mail, which most users find less confusing. Since *admin-request* resolves to *alana* before final delivery is made, errors still get back to *alana* for analysis. Users are happy, and the purpose of the *owner-admin* alias is fulfilled.

Like most formal mailing lists, the *chiefs* mailing list also has the *owner-list* and *list-request* aliases described above. The *chiefs* mailing list delivers mail to the four department heads and their advisor. Three of the recipients (*reba*, *sara*, and *kathy*) are local email accounts. Two (*tyler* and *jane*) are aliases that point to user accounts on other hosts.

The sixth recipient on the *chiefs* mailing list is not a person—it's a file. Defining the full pathname of a file on the right hand side of an alias causes sendmail to append a copy of the mail to that file.* This is a simple way of creating an archive for a mailing list, and it is adequate for the small *chiefs* mailing list. However, the overhead associated with having sendmail copy mail to a file makes this a less than ideal solution for large, busy mailing lists. A better solution for high-volume mailing lists is to have sendmail pipe the mail to an external program that writes the archive. Recipe 2.9 shows how sendmail pipes mail to a program.

Notice that the six recipients of the *chiefs* mailing list are defined on three different lines. When a line in the *aliases* file begins with whitespace (space or tab characters), the line is treated as a continuation of the previous line. Thus, the recipient list for an alias can span multiple lines.

The *employees* mailing list sends mail to all users. It has the same supporting *owner-list* and *list-request* aliases as the other mailing lists. The most interesting thing about this mailing list is the `:include:` syntax used on the recipient side of the alias. The `:include:` syntax causes sendmail to read a list of recipients from an external file. In the example, sendmail reads a file named */etc/mail/allusers*, which contains the delivery address of everyone served by our sample mail server.

On our small sample system, the *alluser* file contains ten local users and three external users. sendmail includes that file into the *aliases* database as the recipients of the *employees* mailing list. Again, a `sendmail -bv` command demonstrates this:

```
# sendmail -bv employees
craig... deliverable: mailer local, user craig
kathy... deliverable: mailer local, user kathy
pat... deliverable: mailer local, user pat
logan... deliverable: mailer local, user logan
alana... deliverable: mailer local, user alana
payton... deliverable: mailer local, user payton
david... deliverable: mailer local, user david
reba... deliverable: mailer local, user reba
sara... deliverable: mailer local, user sara
jay... deliverable: mailer local, user jay
anna@crab.wrotethebook.com... deliverable: mailer esmtp, host crab.wrotethebook.com.,
user anna@crab.wrotethebook.com
andy@rodent.wrotethebook.com... deliverable: mailer esmtp, host rodent.wrotethebook.
com., user andy@rodent.wrotethebook.com
jane@rodent.wrotethebook.com... deliverable: mailer esmtp, host rodent.wrotethebook.
com., user jane@rodent.wrotethebook.com
```

* The file *must* be identified by a full pathname that starts with the root (*/*).

Ten of the recipients are local users who get mail delivered through the local mailer. Three are external users whose mail is forwarded through the esmtp mailer to other hosts in the local domain.

In each of our sample mailing lists, there were no duplicate recipients. For example, the *admin* list resolved to *alana*, *logan*, and *pat*—three unique user addresses. However, as we saw earlier, multiple aliases can point to the same recipient. In particular, all of the nonlogin system user accounts on our sample system point to *root*. What would happen if we had a mailing list that included several of these accounts? For example:

```
sysusers:      daemon, bin, ftp, root, alana
owner-sysusers: sysusers-request
sysusers-request: alana
```

In this sample mailing list, *daemon*, *bin*, *ftp*, and *root* all resolve to *root*. *sendmail* recognizes this problem and deletes the duplicate recipients. When mail is addressed to this sample mailing list, *root* receives only one copy of the mail. If *sendmail* did not delete duplicates, *root* would receive four copies of exactly the same message.

See Also

The *sendmail* book covers the *aliases* database in Chapter 12 and provides information on mailing lists in Chapter 13.

2.8 Migrating Ex-Users to New Addresses

Problem

You have been told to gracefully migrate ex-users to their new delivery addresses.

Solution

Add the *redirect* feature to the *sendmail* configuration using the following `FEATURE` command:

```
dn1 Notify senders about discontinued addresses
FEATURE(`redirect')
```

Rebuild the *sendmail.cf* file, copy the new *sendmail.cf* file to */etc/mail*, and restart *sendmail*, as described in Recipe 1.8.

Add an entry to the *aliases* text file for each ex-user. The *alias* field of the entry is the local username where the ex-user previously received mail. The *recipient* field is the user's new email address with the string `.REDIRECT` appended to the end of the address.

As always, when the *aliases* file is updated, run the *newaliases* script:

```
# newaliases
/etc/mail/aliases: 45 aliases, longest 30 bytes, 670 bytes total
```

Discussion

The first two steps in the Solution section create a sendmail configuration containing the *redirect* feature. Of course, if your configuration already contains that feature, as does the generic Linux configuration described in Recipe 1.8, you can skip those first two steps and go right to creating the aliases.

As an example, let's assume we want to gracefully handle mail for an ex-employee for six months. Michael recently left the company. Proper customer relations is the first and most important step in transitioning Michael's mail. Jay has taken over all of his customers, and we now want those customers to send their business communications to Jay. During Michael's last week in the office, he took Jay to visit all of his local customers, and used conference calls to introduce Jay to all of his remote customers. The customers have been notified, and the business mail should flow to the right destination. If management drops the ball on customer relations, there is very little that a sendmail administrator can do about it. However, you can make sure that Michael gets his personal mail by adding an alias to forward the mail to his new address:

```
michael:                michael@new.job.ora.com
```

Michael is sent monthly reminders that his mail forwarding account will be closed in six months. Six month after his departure, the *michael* alias is edited as follows:

```
michael:                michael@new.job.ora.com.REDIRECT
```

At first, Michael's mail is forwarded to his new address, as this `-bv` test shows:

```
# sendmail -bv michael@wrotethebook.com
michael@new.job.ora.com... deliverable: mailer esmtp,
host new.job.ora.com, user michael@new.job.ora.com
```

Forwarding, however, should not go on indefinitely. As long as the mail gets through, there are a certain number of senders who will not update their address books. At some point you must cut them off and stop forwarding the mail. In the example, we have a small site and are willing to forward mail for six months. A larger site might not want to do any forwarding at all.

At the end of six months, the `.REDIRECT` pseudodomain is added to the alias to terminate forwarding. The effect of the `.REDIRECT` pseudodomain can be seen in a `sendmail -bv` test:

```
# sendmail -bv michael@wrotethebook.com
User has moved; please try <michael@new.job.ora.com>
```

Mail addressed to *michael* is now returned to the original sender with an error message that tells the sender to try Michael's new address. The mail is not forwarded, but the sender is told how to deliver the mail directly.

`.REDIRECT` is clearly not a valid domain name. In sendmail parlance it is a *pseudodomain*. Pseudodomains are listed in class `$=P`, and sendmail does not attempt a DNS

lookup for any domain listed in class `$=P`. Pseudodomains are used to signal special processing. However, simply adding `REDIRECT` to class `$=P` is not enough. The *redirect* feature must be included in the `sendmail` configuration for the `.REDIRECT` pseudo-domain to be handled in the manner described in this recipe.

Alternatives

Simply removing an ex-employee’s account is a popular and viable alternative for migrating ex-employees off of your mail system. This alternative was rejected for the example problem because Michael is a sales person. We depend on sales for the survival of our business and did not want to alienate and confuse customers with a “User unknown” error when they attempted to reach their sales representative. For this reason, the solution emphasizes customer relations. However, if Michael had been a programmer working on internal projects, removing his account without any gentle migration may have been acceptable.

Another tempting alternative solution is to place the following alias in the *aliases* database:

```
michael:      jay
```

Jay has replaced Michael. This alias would forward all of Michael’s mail to Jay. However, we rejected this alternative. If this alias is used, Jay receives unwanted mail from all of the remote mailing lists that Michael joined as well as Michael’s personal mail, such as the invitation to Michael’s high school reunion. We want to maintain a friendly relationship with Michael, so we can’t just discard his mail. At the same time, Jay should not be obligated to forward Michael’s mail. We chose to use the *redirect* feature. But these choices are a matter of policy that should be reviewed with management.

See Also

The *sendmail* book covers the *redirect* feature in section 4.8.39.

2.9 Delivering Mail to a Program

Problem

You want `sendmail` to pass mail messages to another program.

Solution

Address the mail to the program using the pathname of the program preceded by a pipe character.

Discussion

A recipient email address can contain a pipe character and the name of a program. For example:

```
|/usr/local/bin/rtmproc
```

sendmail uses the prog mailer to deliver mail to a recipient address that begins with the pipe character. The P parameter of the prog mailer definition defines the path to the prog mailer program and the A parameter defines the command used to run the mailer. With the generic sendmail configuration described in Recipe 1.8, the P parameter is P=/bin/sh and the A parameter is A=sh -c \$u. \$u is a sendmail macro that contains the email address of the user to which the mail is being delivered. Using the sample email address shown above, the command executed for the prog mailer would be:

```
/bin/sh -c "/usr/local/bin/rtmproc"
```

When the -c option is used with */bin/sh*, shell commands are read from the string that follows -c. In this case, sendmail causes the shell to execute a program named *rtmproc*. sendmail attaches its output to the standard input of the shell and prints out the mail message, which, in the example, sends the mail message to the *rtmproc* program. sendmail also attaches the standard output and standard error of the shell to its input.

The shell executes any command passed to it. The potential security risks of executing any command that follows the pipe character in a recipient address are obvious, and the fact that users are in control of defining such addresses in their *.forward* files adds to the risk. To reduce these risks, use the Sendmail Restricted Shell (*smrsh*) for the prog mailer instead of */bin/sh*. *smrsh* enhances security by limiting the commands that can be executed, and it is discussed in Chapter 10.

Email addresses that begin with pipe characters can be used in the *aliases* database or in *.forward* files. For example, the system administrator might create the following alias if *rtmproc* is some program created to process mail addressed to *root*:

```
root:      "/usr/local/bin/rtmproc"
```

In another example, Rebecca might create the following *.forward* file to filter her mail through the *slocal* program:

```
"|/usr/lib/nmh/slocal -user reba"
```

Aliases that forward to a program are more commonly used in the *.forward* file than in any other manner. System administrators create the *aliases* database, but users create their own *.forward* files. The structure of the *.forward* file is similar to that of an *:include:* file. Each entry in a *.forward* file defines one or more recipient addresses to which mail addressed to the user is delivered. Anything that can appear in the recipient field of an entry in the *aliases* database can also appear in a *.forward* entry. For example, Jill can forward her mail to an external host:

```
jill@ms.foo.edu.
```

Julie can deliver the mail to her local mailbox and make a backup copy in an archive file:

```
\julie
/mnt/nsf1/sara/mail.archive
```

Kathy can define the following *.forward* file when she goes on vacation:

```
\kathy, "|usr/local/bin/vacation kathy"
```

Notice Kathy's entry. Kathy wants to put a copy of the mail in her mailbox before running the *vacation* program. Her *.forward* file starts with `\kathy`. The `\` causes sendmail to deliver mail to the *kathy* account without further aliasing. The `\` is necessary; without it, placing the address *kathy* in Kathy's *.forward* file would cause a loop.

See Also

The *sendmail* book covers the *aliases* database in Chapter 12.

2.10 Using Program Names in Mailing Lists

Problem

sendmail eliminates duplicates in mailing lists; however, you must ensure that all users listed in a mailing list receive their mail, even when multiple users receive their mail through a pipe to the same program.

Solution

If recipient addresses in the *aliases* file are not unique because they identify the same program, make them unique by inserting a harmless shell comment after the program's pathname.

Discussion

Duplicate recipients are deleted during mailing list processing to prevent delivery of multiple copies of the same piece of mail to the same user. This can cause problems when the duplicate recipients are really separate instantiations of the same program. The following *aliases* file illustrates this:

```
sales:          frank, clark, amanda, jill, jeff
owner-sales:    sales-request
sales-request:  alana
clark:          "|usr/bin/procmail"
jeff:           "|usr/bin/procmail"
jill:           "|usr/bin/procmail"
```

In this case, mail to *clark*, *jeff*, and *jill* all appears to go to the same recipient—`"/usr/bin/procmail`. When duplicates are eliminated, only one of the three users gets a copy of the mail.

Most programs that are designed to work with sendmail are written to require a username as a command-line argument. It is not that the program necessarily needs this value, it is done to avoid duplicates when the program is used as a recipient in the *aliases* file. *procmail*, however, does not require the username as a command-line argument; in fact, it runs very nicely without any command-line arguments.

Make each recipient value unique by adding the username as a shell comment after the *procmail* command. The comment is ignored, so it has no impact on the execution of *procmail*, but it makes each recipient line unique, so there are no problems with duplicate recipients. Here is the corrected mailing list for the *aliases* file:

```
sales:          frank, clark, amanda, jill, jeff
owner-sales:   sales-request
sales-request: alana
clark:         "|/usr/bin/procmail #clark"
jeff:          "|/usr/bin/procmail #jeff"
jill:          "|/usr/bin/procmail #jill"
```

Users do not need to add a username when they pipe to the *procmail* program from their *forward* files. When multiple references to *procmail* come from a single file (e.g., the *aliases* file), sendmail needs help to resolve the duplication. sendmail does not need help resolving the duplication when the entries are retrieved from different *forward* files.

See Also

Recipe 2.7 also discusses the elimination of duplicates in mailing lists.

2.11 Allowing Nonlogin Users to Forward to Programs

Problem

You want to allow users who have not been given a valid login shell to forward mail to programs.

Solution

Add `/SENDMAIL/ANY/SHELL/` to the */etc/shell* file. For example:

```
# echo /SENDMAIL/ANY/SHELL/ >> /etc/shells
```

Discussion

Users' home directories can be located on an NFS file server that is configured to allow the user to mount the home directory but is not configured to allow the user to log in. Therefore, the user is not given a valid login shell. A user needs a valid login shell in order to forward mail to a program. Even when *smrsh* is used, as described in Chapter 10, adding programs to the *smrsh* program directory is not enough to make those programs available to the user if the user does not have a valid login shell.

sendmail considers a valid shell to be any shell listed in the */etc/shells* file. If the system does not have an */etc/shells* file, a default list of shells, defined by the `DefaultUserShells` variable in the sendmail source code, is used. If the shell in the user's */etc/passwd* entry is not a valid shell, sendmail refuses to run a program from the user's *.forward* file.

Some NFS servers are configured to allow mounting of home directories while denying login access. A user's */etc/passwd* entry on such a server contains something like */sbin/nologin* or */bin/false* as the user's login shell. These "nologin" shells should never be listed in */etc/shells*. Thus, sendmail does not find the user's shell in */etc/shells* and refuses to run the program the user has placed in the *.forward* file.

Place the string */SENDMAIL/ANY/SHELL/* in the */etc/shells* file to tell sendmail that it should run the program from the user's *.forward* file, even if the user does not have a valid login shell. This recipe adds the entry to the end of an existing */etc/shells* file. If your system does not have an */etc/shells* file, the *echo* command shown in the Solution section creates one that contains the required string.

When *smrsh* is used, putting */SENDMAIL/ANY/SHELL/* in */etc/shells* doesn't change the fact that only programs found in the *smrsh* program directory will execute. The valid login shell requirement is in addition to the *smrsh* requirement. The */SENDMAIL/ANY/SHELL/* string bypasses the valid login shell requirement; it does not bypass the *smrsh* configuration requirement. Because */SENDMAIL/ANY/SHELL/* bypasses a security check, it should be used only when it is absolutely necessary.

See Also

The *cf/README* file covers the use of */SENDMAIL/ANY/SHELL/* in the */etc/shells* file.

2.12 Fixing a .forward Loop

Problem

Mail addressed to a user is being bounced with the error "too many hops."

Solution

First, check the *aliases* database to make sure that it is not the cause of the problem. If the *aliases* database is not the source of the loop, create an alias for the looping username to bypass the user's *.forward* file and force local delivery. The alias field of the new database entry should be the looping username and the recipient field should be the looping username preceded by a `\` character.

Second, check out the contents of the user's *.forward* file. If the user forwards to a remote system, and you have root access to that system, print out the user's *.forward* file on that system. If you don't have root access, *telnet* to the SMTP port of the remote system and use the SMTP EXPN command to see how that system delivers mail addressed to the looping account.

If these tests show you the loop, tell the user exactly what is wrong and what needs to be fixed. If you cannot get the necessary information quickly from the remote system, tell the user that he probably has a loop in his *.forward* files, that he needs to fix it, and that his *.forward* file on your system will be ignored until he does.

Discussion

Users create and maintain their own *.forward* files. Sometimes, of course, a user makes a mistake when configuring the *.forward* file, and it is up to you to help him correct that mistake. One of the most common configuration errors is a forwarding loop in which the user configures the *.forward* file to forward mail to another system and then configures the *.forward* file on that system to forward mail back to the original system. Sometimes, more than two systems are involved because users often have login accounts on several systems. Let's look at an example.

Assume the username for which mail is bounced is *norman*. Examine how your system handles the *norman* email address by adding the verbose option (`-v`) to the `sendmail -bv` command, as follows:

```
# sendmail -v -bv norman
/home/norman/.forward: line 1: forwarding to norman@crab.wrotethebook.com
norman@crab.wrotethebook.com... deliverable: mailer esmtp, host crab.wrotethebook.com., user norman@crab.wrotethebook.com
```

Adding `-v` to the `sendmail -bv` command provides additional information on how an address is rewritten on the local system. In this case, there is no alias for *norman*, so the *aliases* database cannot be part of the forwarding loop. (It is always best to make sure that you are not the cause of a problem before you get a user involved.) In fact, the `sendmail -v -bv` command makes clear that the information used to rewrite the *norman* address comes from the `/home/norman/.forward` file.

Add an alias to the *aliases* database to break the loop:

```
norman:          \norman
```

The `\` syntax terminates aliasing before Norman's *.forward* file is read. Mail addressed to *norman* that arrives at this host is no longer forwarded off of this host, which breaks the loop.

If you want to investigate further, you can examine the user's *.forward* file on the remote system or use the SMTP EXPN command to check how the remote system handles the user's mail. The `sendmail -bv` command just shown tells us that Norman forwards his mail to *crab.wrotethebook.com*. The following test shows how *crab* handles Norman's mail:

```
# telnet crab.wrotethebook.com smtp
Trying 192.168.0.15...
Connected to crab.wrotethebook.com.
Escape character is '^]'.
220 crab.wrotethebook.com ESMTP Sendmail 8.12.9/8.12.9; Mon, 11 Aug 2003 10:31:49 -
0400
HELO rodent.wrotethebook.com
250 crab.wrotethebook.com Hello rodent.wrotethebook.com [192.168.0.3], pleased to
meet you
EXPN norman
250 2.1.5 Norman Edwards <norman@wrotethebook.com>
QUIT
221 2.0.0 crab.wrotethebook.com closing connection
Connection closed by foreign host.
```

The local system forwards Norman's mail to *crab.wrotethebook.com*, and *crab* forwards the mail to *wrotethebook.com*. If the local system is the mail exchanger for *wrotethebook.com*, the cause of the loop is obvious. This example uses the SMTP EXPN command to examine forwarding on the remote system. However, many mail hosts do not implement the EXPN command. In that case, an error message similar to the following is displayed in response to the EXPN command:

```
502 5.7.0 Sorry, we do not allow this operation
```

It is not absolutely necessary that you gather information from the local and remote *.forward* files. If it is simple to do, your insights about what is wrong with the configuration of these files may speed a permanent solution. However, the *.forward* file is the user's responsibility. If the *norman* alias you added to the *aliases* database breaks the loop and the information about the *.forward* files cannot be gathered easily, you probably have more productive ways to spend your time than trying to gather forwarding information. Norman can easily print out his *.forward* files and bring the information to you later if he needs help.

See Also

The SMTP commands, including EXPN, are covered in the "Simple Mail Transfer Protocol" section of *TCP/IP Network Administration*, Third Edition, by Craig Hunt (O'Reilly). The *sendmail* book covers EXPN in section 10.3.2.

2.13 Enabling the User Database

Problem

Despite the fact that the sendmail developers discourage its use, you have decided to configure the user database.

Solution

Use the sendmail debug option `-d0.1` to check that sendmail was compiled with the `USERDB` flag. If necessary, recompile sendmail to include user database support.

Create the user database source file and place a balanced pair of `maildrop` and `mailname` entries in the file for each username that will be mapped by the user database.

Use the `makemap` script to convert the completed source file into the required `btree` map type.

Assuming that the database file is created in the `/etc/mail` directory and is named `userdb`, add the following define to the sendmail configuration:

```
define(`confUSERDB_SPEC', `/etc/mail/userdb')
```

Rebuild the `sendmail.cf` file, copy the new file to `/etc/mail`, and restart sendmail as described in Recipe 1.8.

Discussion

Three things are required before sendmail will check for a user database: sendmail must be compiled with the `USERDB` compiler flag, the mailer must have `@` flag set, and the path to the user database must be defined inside the sendmail configuration using the `confUSERDB_SPEC` define. The Solution section assumes the file is named `userdb` and is placed in the `/etc/mail` directory.

The sendmail `-d0.1` debug option displays the compiler flags used to create the sendmail binary. Examine the compiler flags listed after “Compiled with:”. If `USERDB` is not listed, recompile the sendmail source code using either the `NEWDB` or `HESIOD` compiler flag and reinstall sendmail. (When either `NEWDB` or `HESIOD` are used, the `USERDB` flag is automatically set.) Select `NEWDB` or `HESIOD` based on whether the user database information will be stored on a Hesiod server or locally in a database file. Chapter 1 provides several examples of recompiling sendmail and, in particular, Recipe 1.4 shows how to add an optional map type to sendmail.

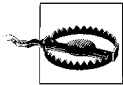
The user database file must be converted to a true database before it can be used by sendmail. Use the `makemap` command to build the database. For example, to create

a database file named *userdb* from a user database source file named */etc/mail/userdb*, use the following *makemap* command:

```
# cd /etc/mail
# makemap btree userdb < userdb
```

The *makemap* program reads the standard input and writes out the specified database of the type selected. The command generally has two arguments: the database type and the name of the database to be written. The user database must be of the *btree* type, and the name of the user database must be the one defined inside the *sendmail* configuration. The *makemap* command is used extensively in this book with a variety of command-line options.

The user database is applied to inbound mail after the *aliases* database and before the *forward* file when the mailer has the *@* flag set.



The user database is not recommended. The *sendmail* developers tell me that it will probably be removed from the distribution sometime in the future. This recipe is shown only because you might have a great idea of how to use it. Just remember that the user database is not recommended for simply mapping outbound sender addresses. If you use it, you should have a special reason for doing so.

The entries in the user database look something like the entries in the *aliases* database except for the addition of a keyword, either *maildrop* or *mailname*. Generally, user database entries are balanced pairs, where each user has both a *maildrop* and a *mailname* entry. A pair of entries for the *andy* user account might be:

```
andy.wright:maildrop andy
andy:mailname andy.wright@wrotethebook.com
```

The entries that use the keyword *maildrop* are almost exactly like entries in the *aliases* database. The value before the colon (*:*) is the user alias and the value after the keyword *maildrop* is the recipient address. The sample *maildrop* entry shown above performs exactly the same function as the following line placed in the *aliases* database:

```
andy.wright: andy
```

Both of these entries take mail addressed to *andy.wright* and deliver it to the *andy* user account. Of course *maildrop* lines are not needed if they replicate lines in the *aliases* file. The *aliases* file has already mapped inbound addresses before the user database is called. Real *maildrop* entries don't duplicate entries already found in the *aliases* database.

The similarity between entries in the *aliases* database and *maildrop* entries in the user database is very strong. The only difference in these entries, other than the addition of the keyword *maildrop*, is that *maildrop* entries cannot point to aliases. The recipient address in a *maildrop* entry must be a real address.

The added feature of the user database is that, unlike the *aliases* and *.forward* files, the user database also applies to outbound mail. *mailname* entries in the user database transform the sender address to create a *reverse alias*. In a *mailname* entry, the value before the colon (:) is the local username, and the value following the keyword *mailname* is the sender address that should be used for mail originating from the user. The *mailname* entry shown above converts the sender address on all mail from the user *andy* to *andy.wright@wrotethebook.com*.

However, the user database is not the only way, or even the recommended way, to rewrite outbound addresses. The sendmail FAQ, in question 3.3, states “the user database is no longer the recommended solution” for rewriting sender addresses. The *genericstable* database is the recommended tool for this task. Chapter 4 covers the *genericstable*.

The *aliases* database and the *genericstable* are preferred alternatives to the user database because:

- The user database is more difficult to use than the alternatives. The *aliases* database is available in all sendmail configurations, but the user database requires *m4* changes to the configuration. The *genericstable* also requires *m4* changes, but in addition to *m4* changes, the user database requires that the sendmail source code be compiled with a special compiler flag.
- The user database replicates, but does not replace, the *aliases* database. For example, the user database cannot point an alias to an alias. Even if you create a user database, you still need an *aliases* database, meaning you now have two files to maintain.
- The user database interacts with other features in ways you might not anticipate, as the sendmail FAQ makes clear in questions 3.3 and 3.4.

The sendmail developers make it clear in the response to FAQ question 3.4 that the user database was developed for a specific configuration requirement at UC Berkeley, and that it may not be applicable to a wide range of configurations. If you still insist on using it for something, this recipe points out the steps necessary to enable the user database.

See Also

Chapter 4 covers the *genericstable*, which is the preferred alternative for writing outbound addresses. The sendmail FAQ provides advice on the user database in questions 3.3 and 3.4. The *sendmail* book covers the user database in 23.7.26.