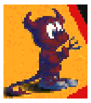
UnixWorld Online: Tutorial: Article No. 008



Sendmail V8: A (Smoother) Engine Powers Network Email

Sendmail, with its cryptic single-character option tags and notorious rewriting rule sets, has nevertheless always been the premier Internet mail transfer agent. But the latest releases, with macro-based configuration and spelled-out options, add some ease-of-use and sophistication to the program's traditional power.

by Richard Reich

Table of Contents

Sendmail is the most common SMTP mail transfer agent on the thousands of mostly Unix-based Internet hosts that handle mail routing and serve as post offices. Millions of e-mail messages are handled by Sendmail every day. Although it is very popular, Sendmail has been obscure and difficult to configure during much of its long history. Recent versions of Sendmail, however, have a much improved configuration system, based on the m4 macro processor and a large set of predefined m4 macros.

This tutorial does not pretend to be a complete treatment of Sendmail. But it will try to show the average system administrator that Sendmail, with macro-based configuration, can be set up usefully with a reasonable amount of study and attention.

The focus will be on Berkeley Sendmail version 8.7, the freely distributed version maintained and improved by its original author, Eric Allman. Most Unix system vendors include ‘‘Sendmail,’’ but often these are old versions, almost always lacking the m4-based configuration environment and other improvements. In addition, older versions of Sendmail have well-known security problems that are repaired in the later versions available from Berkeley. Although there are generally valid arguments against early adoption of new versions of critical software, Sendmail may be an exception to the rule.

This tutorial first describes Internet mail basics and a common strategy for SMTP mail handling on an Internet-connected local network. Sendmail configuration is treated in the context of implementing the example mail strategy. Sendmail’s UUCP capabilities, perhaps less relevant than they were a few years ago, are outside the scope of this presentation. (Sendmail, even with tractable configuration tools, is too large a topic to present in the abstract in this limited space.)

Quick Overview of Internet mail

The rules that permit heterogeneous computer systems to interoperate smoothly on the global Internet are set forth in documents called Requests For Comments, or RFCs. The format of Internet mail messages is defined by RFC 822 (see Reference 6). Thus, Internet e-mail is often called ‘‘RFC 822’’ mail. The protocol used to send RFC-822 e-mail between host computers is referred to as the Simple Mail Transfer Protocol, or SMTP, and is defined in RFC 821 (see Reference 5).

RFC-822 Mail Format

The format of Internet mail is fundamentally very simple: various required and optional message attributes come first in a ‘‘header,’’ followed by a blank line, then the ‘‘body’’ of the message. The header fields predominate in the short example message shown here:

Editor’s Note: The long ‘‘received’’ lines were wrapped then indented so they’ll fit in the average window.

```
Return-Path: pete@maclean.com
Received: from tempo.maclean.com (tempo.maclean.com [204.182.19.66])
    by goldengate.reich.com (8.7.1/8.7.1/FultonSt-gg0916) with ESMTP
    id WAA01451 for ; Sun, 15 Oct 1995 22:09:10 -0700
Received: from petewin95.maclean.com (petewin95.maclean.com [204.182.19.95])
    by tempo.maclean.com (8.7.Beta.10/8.7.Beta.10/FultonSt-tempo0806) with
SMTP
    id WAA12144 for ; Sun, 15 Oct 1995 22:09:08 -0700
Message-Id: <199510160509.WAA12144@tempo.maclean.com>
X-Sender: pete@tempo.maclean.com
X-Mailer: Windows Eudora Pro Version 2.1.2
Mime-Version: 1.0
Content-Type: text/plain; charset="us-ascii"
Date: Sun, 15 Oct 1995 22:06:05 -0700
To: richard@reich.com
From: Pete Maclean
Subject: A question...

Just wondered if this message will appear in your Sendmail article?

-p
```

The blank line after the ‘‘Subject’’ line divides the header from the message body that follows. Any subsequent blank line is part of the message body and has no structural significance. Most header fields are brief and have an intuitively obvious meaning (`Subject: A question...`) while some others are lengthy and not readily understood (`Received: from...`). For a good explanation of many standard as well as non-standard header fields, see Chapter 31 of *Sendmail* (Reference 1).

Each header line consists of a ‘‘keyword-value pair’’ that declares one specific characteristic of the message. For instance, the required line that specifies the recipient of the message consists of the keyword ‘‘To:’’, one or more space or tab (white space) characters, followed by the value that specifies the mailing address of the recipient, here ‘‘richard@reich.com’’.

Simple Mail Transfer Protocol (SMTP)

SMTP is a TCP-based, client-server protocol. Its operation is really quite simple: after a reliable connection is established, the client initiates a brief handshake sequence. Then the client sends one or more messages to the server. Preceding each message, the remote system is given a list of the message’s local recipients as well as the sender’s address. This information is referred to as the message’s ‘‘envelope’’. The natural metaphor of physical letters is instructive: to send a letter to several people at different locations, for each recipient place a copy of the letter in an envelope, which bears both the recipient’s address and the return address of the sender, and post individually to each envelope addressee.

This exchange of information takes place in a formal language of four-character commands and three-digit reply codes, but it is usually replete with human-readable comments that render transcripts of SMTP sessions quite easy to follow. A somewhat improved version of SMTP, Extended SMTP, or ESMTP, is now in wide use. Here’s a real example of an ESMTP mail exchange log. Don’t worry about what everything means, but note the basic simplicity of the conversation.

Editor’s Note: The long lines in this example were wrapped and then indented four spaces so they’ll fit on the average window.

```
$ /usr/sbin/sendmail -v pete@maclean.com < message
pete@maclean.com... Connecting to tempo.maclean.com. via smtp...
220 tempo.maclean.com ESMTP Sendmail 8.7/8.7/FultonSt-tempo0806;
    Sun, 15 Oct 1995 22:47:52 -0700
>>> EHLO goldengate.reich.com
250 tempo.maclean.com Hello richard@goldengate.reich.com [204.182.19.1],
    pleased to meet you
>>> MAIL From: SIZE=10
250 ... Sender ok
>>> RCPT To:
250 Recipient ok
>>> DATA
354 Enter mail, end with "." on a line by itself
>>> .
250 WAA12161 Message accepted for delivery
pete@maclean.com... Sent (WAA12161 Message accepted for delivery)
Closing connection to tempo.maclean.com.
>>> QUIT
221 tempo.maclean.com closing connection
```

Interfaces and Agents

The ‘‘life cycle’’ of an e-mail message involves several distinct stages. Writing a mail message is quite different than sorting envelopes, which in turn, is different than delivering mail. This is true in the realm of electronic mail as well as in the world of surface postal mail (a.k.a. snail mail).

Preparing and reading e-mail is done with a Mail User Agent (MUA). The qualities that people prefer in a MUA vary, as do the platforms on which MUAs are implemented. This leads to a wide variety of different MUAs, catering to different tastes in user interfaces, capabilities and platforms. Some examples include: Eudora, elm, pine, mh, exmh, xmail, mailx, Mail, mail, etc.

Delivering e-mail is generally handled by programs (Mail Delivery Agents, or MTAs) that do one specific type of delivery. For example, putting mail into a local mailbox file on Unix systems is often handled by so called ‘‘bin’’ mail (probably named because it classically had path name `/bin/mail`), sending mail via UUCP is done by `uux`, while forwarding mail via SMTP is done by Sendmail itself.

Mail Transfer Agents, like Sendmail, handle everything else. An MTA determines how a message has to be routed to get to a recipient. It accepts mail from another transfer agent and relays it to an agent closer to the ultimate recipient. It handles the interpretation of address aliases. It transforms addresses so that the panoply of incompatible delivery agents can deal with them properly. It handles special actions required by certain header fields (for instance, ‘‘Bcc:’’ for blind-carbon copy, and ‘‘Return-Receipt-To:’’ to verify delivery). It queues messages when delivery can’t be done immediately and handles them later. It recognizes bad addresses and other errors and reroutes or bounces mail as needed. And more.

What Sendmail Does

Let’s follow the path a message might take, starting after it’s been composed and is handed to Sendmail by a Mail User Agent.

MUA Sends a Message. We’ve composed this simple message:

```
From: Richard Reich <richard@reich.com>
To: pete@maclean.com
Bcc: me
Subject: My Sendmail article


Please read the draft of my Sendmail article.  It will be in the
usual place by tonight.  Thanks.

-r
```

This simple note is intended for a single recipient, with a blind carbon copy for my records. The MUA that composed it will start Sendmail and give it the message and the list of recipients.

Aliases. An alias is a convenient abbreviation for one or more full mailing addresses. That is, an alias can just be a nickname for an address or it can be the name of a list of recipients. Aliases can be maintained and expanded by a MUA or by Sendmail. Most MUAs keep alias information in their own version of an alias file. So, if you use, say, elm ordinarily, its alias file will not be available to Netscape when you use Netscape’s ‘‘Mail Document’’ function. However, aliases maintained centrally by Sendmail will be recognized and expanded regardless of which MUA is used to compose a message.

There is an alias among the recipients of our example message: me. Sendmail will expand it and discover that the full address for ‘‘me’’ is the local mailbox ‘‘richard’’.

Handling Mail. The two recipient addresses are examined by Sendmail. One is local (me) and the other (pete@maclean.com) is an address at a remote host.

The message must be transformed slightly to handle the ‘‘Bcc:’’ header properly. Blind copying requires that the primary recipients not be informed of the blind copy recipient. So Sendmail, after having added my address to its internal list of recipients, deletes the ‘‘Bcc:’’ header field from the message.

Local Delivery. Assuming Sendmail has been configured to use ‘‘bin mail’’ for local delivery, it directs this program to save a copy of the message in my mailbox.

Local delivery is not always so dull, however. A user can keep private aliases in the `.forward` file in their home directory. Mail intended for delivery to a user with such a file will go instead to the addresses listed in that file. Mail can even be delivered (as standard input) to a program you specify in the `.forward` file. (That’s how automatic mail response is implemented sometimes.)

Remote Delivery. Returning to our example, Sendmail now has an address that it determines--by examining its format--is probably intended for a remote Internet recipient. For each remote recipient, Sendmail will call upon its domain name resolver to find out the Internet host to which the message should be sent (that is, a mail exchanger--MX--query will be made). Then, to actually transfer the message, an SMTP session will be initiated with the MTA (perhaps another Sendmail) at each remote mail handling host. A failed transfer will result in the message being queued for later delivery.

The Sendmail daemon. Our sample message will be accepted by a Sendmail daemon on a remote host (the mail exchanger for maclean.com). The message will then go through Sendmail’s handling process on the remote system (assuming it’s running Sendmail). Presumably, the message will be delivered locally to the mailbox of the intended recipient. Alias processing, forwarding, or other kinds of required relaying, however, might result in the message being passed to still another transfer agent.

The Role of DNS

Sendmail uses the Domain Name System to help it deliver mail. Proper implementation of a domain’s mail handling strategy requires that the configurations of both Sendmail and DNS be accurate and coordinated. If a message is to be sent to a non-local recipient, the domain name portion of the recipient’s address must be examined to determine the host where the message should be sent.

First, Sendmail queries the local DNS resolver to find so-called ‘‘Mail Exchange or MX records’’ for the recipient’s domain. For example, to decide where to send a message addressed to pete@maclean.com, Sendmail will look for MX records for the domain name maclean.com. The DNS resolver will return any MX records it finds, often more than one. In the event that the recipient domain has no MX records defined, Sendmail will query DNS for CNAME or A records to arrive at a possible mail exchanger host. Multiple MX records--each specifying an alternative mail-handling host--can be defined for a domain name. An MX record contains a preference field that ranks its mail exchanger host relative to others for the same domain name. (The preference field’s value is like a golf score: lower numbers are preferred, with zero the best. The maximum value is 65535). A mail transfer agent is required to choose the most preferred mail-exchange host among those that are currently functioning.

Given a choice among several equally preferred hosts, Sendmail will choose one at random.

Continuing with our example (sending a message to pete@maclean.com), the DNS resolver might return to Sendmail MX records for maclean.com like the following (rendered here in the textual form used by BIND's configuration file):

```
maclean.com.    85676   MX      10 tempo.maclean.com.
maclean.com.    85676   MX      20 goldengate.reich.com.
```

The fields are the recipient domain name, the TTL (time-to-live value in seconds), the record type, the preference value and, finally, the mail exchange host.

The records define a preferred mail exchanger at tempo.maclean.com and a less preferred one at goldengate.reich.com. This means that Sendmail will try to send the message to tempo.maclean.com--and failing that--goldengate.reich.com. If it gets to goldengate, the Sendmail daemon there will make its own attempt to deliver the message. Unless tempo has recovered in time, goldengate also fails to relay the mail as we'll see below.

Then a crucial bit of special handling is invoked to avoid sending mail about pointlessly. Sendmail will not relay mail to a mail exchanger that has an equal or greater preference value than its own. As long as tempo is unreachable, goldengate won't be able to relay the message because it can't find any other acceptable host. It will queue the message to disk and try to deliver it later.

Thus it's crucial to get the MX records right. If your domain has an erroneous MX record in its DNS server configuration, your perfectly configured Sendmail daemon may never see an incoming message. Remote Sendmails (or other transfer agents) may not find out that your host handles mail at all!

Installing Sendmail and Friends

To get the very latest version of Sendmail, you may want to download the source package from its home at Berkeley. Compilation and installation of the Berkeley distribution is a relatively smooth operation. The source package includes make-description files tailored for many different systems and a ''build'' script that automatically chooses the correct one. Often one or two simple changes are necessary to the appropriate make-description file to match the configuration of a particular system, but these are usually quite obvious. (See below for where Sendmail and its helpers can be found.)

Berkeley ''db'' is a library for manipulation of indexed data records, such as the aliases file. Sendmail can get by with weaker data management packages (for instance, ndbm) or with none at all. But db does enhance Sendmail's efficiency and robustness.

Sendmail handles SMTP mail transfer directly, but it relies on other programs to handle other kinds of delivery. In particular, Sendmail can be configured to use one of several local mail delivery agents, such as `bin-mail`, `procmail`, `mail.local`, or `deliver`. All these delivery agents have special features and strengths. Although `mail.local` is virtuously simple and `procmail` is robust and powerful, it is easiest at first to use whichever local delivery agent your version of Sendmail is configured for by default.

The Sendmail Configuration File

The Sendmail configuration file, generally named `sendmail.cf`, contains several classes of information that determine the behavior of Sendmail on a host system:

- *Options* determine the values of numerous Sendmail parameters (for instance, file and directory paths, operational control switches, timeout values).
- *Header definitions* are templates used to specify required and optional message headers and their formats.
- *Mailer definitions* specify the programs that will be used to deliver various kinds of mail (for instance, local mailbox delivery, delivery to a file or program) as well as specifying details of Sendmail's interaction with them.
- *Macro and class definitions* provide names for strings and sets of strings (for instance, domain name of this host, set of alternate names) that are used in header definitions and rewriting rules.
- *Rewriting rule sets* are used to parse and transform addresses. In addition to controlling the appearance of addresses and directing special handling of certain classes of addresses, rewriting rules are used by Sendmail to determine, for each message recipient, the final delivery address, the mailer to use and the host system where the message should be delivered (or relayed).
- *Key (map) file declarations* specify the path and other attributes of files that can be used in rewriting rules to lookup and transform elements of addresses.

With very few exceptions, all of these components of the original Sendmail configuration file are hidden by the m4-based configuration macro files (as we'll see below).

For a majority of Sendmail configurations, the m4 macros in the Sendmail distribution package will suffice. For instance, having mail from all local hosts ''masquerade'' as though it comes from *`domain`* is a configuration choice that has been foreseen in the Berkeley release--a one-line macro (`MASQUERADE_AS(`*`domain`*`)`) takes care of numerous details, including adding rewriting rules. However, some of the original configuration elements, like the semantics of Sendmail options and the nuances of rewriting rule sets, must be understood in their full glory if customization is attempted beyond that already anticipated by the existing m4 macros.

Sendmail Options

Sendmail options are set in its configuration file with the single-letter command, capital O. In versions before 8.7, all options had single-letter names. For example, the option A held the path name of the alias file. Beginning with version 8.7, all options can be referred to by full names. For instance, the path name of the alias file is now specified by option `AliasFile`. The old single-letter option names are still recognized for backward compatibility.

To avoid any ambiguity between the older single-letter form and the new full-name form, a space (which may not appear between the O command and the single-letter option being defined) *must* appear between the O command and the full name. For example, to set the name of the alias file in the old style, use:

```
OA/usr/lib/aliases
```

whereas with the new style, employ:

```
O AliasFile=/usr/lib/aliases
```

In a m4 configuration file, you need not worry about defining the alias file name. An operating system specification macro (for example, `OSTYPE(irix4)`) takes care of setting it with the m4 statement:

```
define('ALIAS_FILE', '/usr/lib/aliases')
```

Note in the preceeding example how the arguments of the define command are quoted. First, balanced left and right single-quote marks are used. Second, non-alphabetic characters in a phrase means that the phrase must be quoted.

It's not feasible to explain each of the many global configuration options here that can be set within `sendmail.cf`. For a complete and up-to-date list of these options, consult the BSD *System Manager's Manual* paper entitled ''Sendmail Installation and Operation Guide'' (see Reference 3).

Address Rewriting

At last, the fun part. You can't read too much tutorial material on the subject of Sendmail rewriting rules. So before you leap into my idea of a simple introduction to the subject, you may want to examine Bryan Costales' ''Explosion in a Punctuation Factory'' tutorial article from the UnixWorld Online archives.

Address rewriting rules are the essence of Sendmail's power and its complexity. They can be seen as a simple, quite specialized, text-oriented programming language. Two critical tasks that Sendmail performs--rather than being hard-coded in the Sendmail program itself--are expressed in the language of rewriting rules, making it relatively easy to configure Sendmail's behavior very flexibly, without modifying its internal code.

First, Sendmail must examine each recipient's address to determine which of several mail delivery agents should be used to send the message to--or closer to--the recipient.

Second, Sendmail may transform addresses in both the envelope and the message header to facilitate delivery or reply. (This is probably the moment to address a never-ending controversy that dogs Sendmail: RFC-821, which defines the SMTP protocol, disallows mail transfer agents from modifying message header fields, with a couple of exceptions. Sendmail violates this prohibition. However, if one considers Sendmail to be a mail gateway as well as an MTA, its ''offending'' behavior can be justified as essential to its gateway function. Case closed.)

When Sendmail is presented with a message it examines the addresses in the envelope and the header fields (''From:'', ''To:'', ''Sender:'', and so forth). Each address is placed in a area called the ''workspace'', and--depending on whether the address is for a sender or a recipient and whether it came from the envelope or a message header field--certain rule sets are applied to the address in a prescribed order. Also, once the appropriate mail delivery agent is determined for a particular message, an associated rule set is applied.

Rewriting rules are organized into rule sets. A rule set is like a small program consisting of an ordered sequence of rules. The program acts on the address in the workspace, applying each rewriting rule as long as its matching clause matches the address in the workspace. When it does not, the next rule in sequence is tried. (This flow-of-control, such as it is, can be modified very slightly, as explained below.) Viewed this way, a rule set is a function acting on an address, yielding an address.

Rule sets are identified by number, each new rule set beginning with an S followed by its identifying number. Each rule in the set follows. Rules always begin with the letter R. The rule set is terminated when a non-R command is encountered. For example:

```
S17
R$* < @ $=w >       $: $1 < @ ourco.com >

------------- ----- ---------------------
      |         |              |
     lhs   one or more        rhs
              tabs
```

Rewriting rules appear cryptic, but they are actually conceptually simple (as well as being cryptic!). A rule contains a ‘‘left-hand side’’ (lhs), a ‘‘right-hand side’’ (rhs) and, optionally, a ‘‘comment,’’ separated from each other by one or more tabs. Note that space characters (which can be used to separate tokens for readbility) are *not* valid rule-part separators.

When a rule is applied to the address in the workspace, the left-hand side is compared to the address as a pattern. If the pattern matches, the address in the workspace is replaced by the rule’s right-hand side.

The pattern-matching proceeds simply. Ordinary words are matched literally. Operators, which begin with a dollar sign (`$`), have the following meanings on the left-hand side:

```
$*   Match zero or more tokens
$+   Match one or more tokens
$-   Match exactly one token
$=x  Match any phrase in class x
$^x  Match any word not in class x
```

If an operator matches part of the address in the workspace, then the matched token(s) are assigned to the positional operator `$n`, where *n* is 1 for the first match, 2 for the second such match, and so forth. For example, applying the left-hand side `$- @ $*` to `fred@athena.mit.edu` we have a match that assigns ‘‘fred’’ to `$1` and ‘‘athena.mit.edu’’ to `$2`.

When a left-hand side pattern match succeeds, the workspace is replaced with the contents of the rule’s right-hand side. Analogous to matching, the replacement process copies literal tokens from the left-hand side to the workspace and gives a special interpretation to operators. Some of the recognized right-hand side operators include:

```
$n           Substitute the nth matched
             token from the lhs
$>n          Call rule set n
$#mailer     Specify delivery agent, mailer
$@host       Specify host
$:user       Specify user
$( token $)  Look up token in a database
```

Continuing with our example, if the workspace address is ‘‘fred@athena.mit.edu’’ and the current rule is:

```
R$-@$* phil@$2
```

then the workspace will be rewritten as ‘‘phil@athena.mit.edu’’.

The `$>`*n* symbol tells Sendmail to go to rule set *n* after the current rewrite rule has been processed. This mechanism acts like a subroutine facility. For example, it is sometimes necessary to make sure an address is in the standard form Sendmail expects when applying rewriting rules. This operation of ‘‘canonicalization’’ is done by rule set 3. A rule to invoke rule set 3 looks like:

```
R$*    $: $>3 $1
```

(The $: at the beginning of the right-hand side in this example is the ‘‘one-time only’’ prefix. (See below, too). It stops Sendmail from applying the rule over and over, which it would do if not restrained. The left-hand side--`$*`--means ‘‘match anything’’.)

The mailer, host, and user specification symbols are used to resolve envelope-recipient addresses. These constructs appear only in rule set 0 (or rule sets called by rule set 0), which uses rewrite rules to parse and resolve recipient addresses. For example, after some involved application of rule set 0, Sendmail will at last decide that an address is local and resolve the host (this one), the user (the addressee) and the mailer (whatever local mailer has been configured) with this rule:

```
R$+    $#local $: $1
```

In this example the address in the workspace, which consists of one or more tokens (`$+`), is the name of a local user (`$: $1`) and should be delivered by the local mail delivery agent (`$#local`).

The complex token-lookup function (`$( ... $)`) permits substitution of text in an address based on mapping files, that is, ndbm or db databases. The argument to the lookup is always constructed from things that have matched on the left-hand side. For example, if the left-hand side has parsed a user name into `$1` and a domain name into `$2`, the token lookup function on the right-hand side might translate a ‘‘user@domain’’ phrase into a replacement phrase by using `$1@$2` as its argument. If the mapping file consulted by the lookup function contains a map for the argument, it is returned and replaces the entire lookup function as the workspace is rewritten. See the major configuration example below for a practical example.

In addition to the substitution operators, there are two other operators that have special meanings when they appear as the first token on the right-hand side. The `$:` operator instructs Sendmail to apply this rule only once--even if it matches--to prevent infinite looping. Ordinarily, a rule is applied repeatedly, until it fails to match. The `$@` operator directs Sendmail to exit from the rule set with the remainder of the right-hand side as the rule set’s result.

A Configuration Example

The first and most important step in developing a Sendmail configuration is deciding upon a mail-handling strategy. For local networks of reasonable size, a single mail hub system offers centralized administration, coherent e-mail address structure and high levels of reliability, integrity and performance. Even for networks consisting of as few as three or four systems, a mail hub approach makes sense. (In fact, a network consisting of just a single workstation can be viewed as a mail hub and client system rolled into one.)

A Mail Hub Strategy

The mail hub we will configure processes all outgoing messages and acts as the ‘‘post office’’ for mail coming in from outside the network. (It acts as a post office for local, intra-network mail as well.) Every user who wants to receive mail has a mailbox on the mailhub machine. Enforcement of acceptable use is centralized, as is technical administration of such tasks as back up and mail system/DNS coordination. With a combination of aliases and rewriting of sender addresses on outgoing mail at the hub, all users in our example network have Internet e-mail addresses of the form ‘‘Firstname_Lastname@ourco.com’’. Using ‘‘guessable’’ names is desirable (though some disagree), especially in an environment where security or privacy concerns may prohibit open directory services (for example, finger). Using ‘‘domain addressing’’ (‘‘ourco.com’’ instead of ‘‘zippy.research.ourco.com’’) not only hides internal domain structure, but it’s simply more handsome. These policies demand some administrative effort, but without a mail hub the fragmented administrative effort required could be greater still.

Client systems (that is, the non-hub systems on the local network) can be configured in a few different ways, each consistent with the overall mail strategy. A ‘‘smart’’ client can run a Sendmail daemon that handles idiosyncratic alias processing as well as dispatching mail to the hub. A ‘‘null’’ client starts Sendmail locally on a per-message basis, using it simply to pass each message to the Sendmail hub with no local processing. Some hosts, such as those running Eudora on a Macintosh or Windows system, rely on establishing a SMTP connection from the mail user agent (MUA) directly to the mail hub, or use a POP (Post Office Protocol) (see Reference 7) connection with a user’s mailbox on the hub machine.

m4 and Sendmail

The m4 macro processor can be thought of as a translator from a simple Sendmail configuration language to the opaque native configuration used in its configuration file (`sendmail.cf`). An m4 configuration file is rarely more than a few readable lines; the `sendmail.cf` file created by m4 will often be several hundred cryptic lines.

The `cf` directory tree within the Berkeley Sendmail distribution package contains the various Sendmail configuration macros spread among a few descriptively named subdirectories. Macro files refer to one another using relative path names (`` `../m4/cf.m4' ``). The integrity of the interrelated m4 macros depends on the `cf` directory tree’s structure. Don’t disturb it. You can put the `cf` tree anywhere that’s convenient (including leaving it right where you find it) so long as you move all of it.

If you work with a Sendmail older than version 8.7, the m4 configuration file you write (or adapt from prototypes and samples) should be kept in the `cf/cf` directory. It will consist of a few concise calls to various macros and symbol definitions that invoke and control the expansion of large sequences of complex Sendmail configuration language. Indeed, the first line of an m4 Sendmail configuration file must be:

```
include(`../m4/cf.m4')
```

The `cf.m4` macro file contains (or includes) all the definitions of macros we might invoke in an m4 Sendmail configuration. We will touch on a few of the important macros in the examples below.

Compiling a m4 Sendmail configuration is very simple. Just invoke m4 with the m4-format configuration file as its argument. The standard output, which can be redirected to a disk file, will be the desired Sendmail-style configuration file.

```
$ m4 mailhub.mc > mailhub.cf
```

After successful compilation, the ‘‘.cf’’ configuration can be tested using the `-C` Sendmail option (that is, `/usr/sbin/sendmail -C mailhub.cf`) and ultimately copied to to its final destination. If Sendmail is running in daemon mode, kill and restart it to insure that Sendmail reads its new configuration.

As of Sendmail version 8.7, you can place your configuration file anywhere and use parameters on the m4 command line to specify a base include directory. You can also omit the first include line, `include(‘../m4/cf.m4’)`, from the configuration file, specifying `cf.m4` on the m4 command line before your configuration file. For example, if you decide to keep `mailhub.mc` somewhere other than `cf/cf`, you would compile it with something like:

```
$ m4 -I /usr/src/sendmail/cf /usr/src/sendmail/cf/m4/cf.m4 \
mailhub.mc > mailhub.cf
```

A Mail Hub: Command Line and m4 Configuration

Daemon Mode. As a mail hub, Sendmail must be available to handle incoming mail (via SMTP connections) at all times. Sendmail can be invoked at system startup time (or any time) in ‘‘daemon mode.’’. It will listen for and process all incoming SMTP connections, creating subprocesses as necessary to complete the mail transfer work.

To start Sendmail in daemon mode, lines like the following are placed conventionally in a system’s network or multiuser server startup script, which is found in various places with various names depending on the Unix implementation. Here’s an example:

```
# Start the Sendmail daemon...
if [ -x /usr/sbin/sendmail ]; then
    echo "Starting sendmail daemon..."
    /usr/sbin/sendmail -bd -q 15m
fi
```

The `-bd` option specifies daemon-mode operation and `-q 15m` directs Sendmail to attempt to send any queued messages every 15 minutes.

The mailhub.mc File. This configuration file for our mail hub system (mailhub.ourco.com) is not too difficult to understand, yet it fully specifies the behavior of our somewhat customized, powerful mail hub. Let’s examine it line-by-line:

```
include(‘../m4/cf.m4’)
VERSIONID(‘mailhub.mc    Richard Reich   11 AUG 95’)

OSTYPE(linux)
FEATURE(nouucp)
FEATURE(use_cw_file)

MASQUERADE_AS(ourco.com)
```

```
MAILER(local)
MAILER(smtp)

LOCAL_CONFIG
Kuserdb btree -o /etc/userdb.db

LOCAL_RULE_1
R$* < @ ourco.com. > $* $: $( userdb $1 $) < @ ourco.com. > $2
```

The first line (`include(`../m4/cf.m4')`...) causes m4 to read and process `cf.m4`, which defines the Sendmail macros and includes a great deal of native Sendmail configuration language. The second line, the `VERSIONID` macro, adds some useful version information as comments in the output file (`mailhub.cf`).

The `OSTYPE` macro is really just a pretty kind of ‘‘include’’ statement. In our example this statement directs m4 to read and process `../ostype/linux.m4`. There is a wide selection of operating system dependent macro files in the `ostype` subdirectory. In general, these files include definitions that determine which local delivery agent will be used and what oddities, if any, an OS may require with respect to directory or file names. After deciding whether the distributed file that corresponds to your operating system makes sense for your situation, use it in every configuration file you write.

Here, `FEATURE(nouucp)` removes UUCP-related rewrite rules, and so forth, from the resulting Sendmail configuration file.

Sendmail must know the names of all hosts or domains that may receive mail on this system. Otherwise, Sendmail will assume the mail should be routed to another destination system. The names can appear on certain command lines in the configuration file, or they can be read from a separate file. The `FEATURE(use_cw_file)` line instructs Sendmail to read the names from the file `/etc/sendmail.cw`. If our host is the highest priority mail exchanger for a domain name, that name should appear in `/etc/sendmail.cw`. In our case, at least the domain name (ourco.com) must appear in the ‘‘.cw’’ file if domain addressing is to function properly.

Here, `MASQUERADE_AS(ourco.com)` specifies the host pseudonym to be used in sender fields of outgoing mail in place of the full domain name of this host. This is the implementation of ‘‘domain addressing.’’ Mail sent from this host (mailhub.ourco.com), appears to be sent from ‘‘ourco.com.’’ So replies will be directed to *sender*@ourco.com.

The two `MAILER` macros define the types of mail delivery our mailhub will require, local delivery and SMTP-based Internet delivery. (Note that the actual local delivery agent used is often determined by the contents of the macro file named by the `OSTYPE` macro. See above.)

Rewriting Rules: A Very Simple Example

The last four lines of our mailhub configuration file have a potent impact on sender addresses in outgoing mail:

```
LOCAL_CONFIG
Kuserdb btree -o /etc/userdb.db
LOCAL_RULE_1
R$* < @ ourco.com. > $* $: $( userdb $1 $) < @ ourco.com. > $2
```

A complete introduction to address rewriting is beyond the scope of this article, but this simple example may show that the subject is not completely incomprehensible. The macros `LOCAL_CONFIG` and `LOCAL_RULE_1` determine where in the resulting ‘‘.cf’’ file the lines that follow them will end up. Raw Sendmail commands that are not rewriting rules usually belong in the `LOCAL_CONFIG` section. The `LOCAL_RULE_1` specifies that the following rule should be placed in the section that processes sender addresses (for example, the ‘‘From:’’ address). The other two lines are ‘‘raw’’ Sendmail configuration language.

Sendmail configuration statements usually begin with a single upper case letter that specifies a particular Sendmail command. In this case, the ‘‘K’’ command directs Sendmail to open a keyed mapping file (`/etc/userdb.db`) for subsequent use.

Rewriting rules begin with the command letter ‘‘R’’. The rule in this example means: if a sender address is of the form *`sender`* `@ ourco.com` *`anything-else`*, then look up the sender name in the userdb-keyed file. And if the *`sender`* name is found in the file, replace it with the address for that key.

For example, if the userdb file has this record in it:

```
richard Richard_Reich
```

then the actual local address ‘‘richard@ourco.com’’ in outgoing mail will be rewritten as ‘‘Richard_Reich@ourco.com’’.

The `mailhub.cf` file that results from compiling `mailhub.mc` with m4 is shown here. Note that (according to `wc`) `mailhub.cf` has more than ten times as many words as `mailhub.mc`. This is a very rough measure of relative complexity, but it is indicative of the advantage gained from using the m4-based Sendmail configuration tools.

The last step: POP or NFS. The function of the mail hub is to deliver all mail for the entire local network into recipients’ mailboxes resident on the mail hub system. The last step is to get the mail to the recipients.

One very popular solution consists of a Post Office Protocol (POP) Server on the mail hub system, which retrieves mail when asked by a mail user agent (for example, Eudora, Z-Mail). Some POP servers and clients can negotiate the sending of outgoing mail as well.

Another common way to get users and their hub mailboxes together is to allow client systems to mount the mailbox’s directory via NFS (Network File System). A mail user agent, via soft links or environment pointers, sees its mailbox file as though it were local to its own system. Care must be taken whenever NFS is used, however, to maintain user mailbox privacy and system security.

Null Clients

A null client m4 configuration file consists of the macro `FEATURE(nullclient)` and little or nothing else. It expands to a target ‘‘.cf’’ file that causes this workstation’s Sendmail to forward all mail for delivery to a mail hub system. No additional processing takes place. Sendmail on null clients is normally not run in daemon mode, nor does it maintain a mail queue. Sendmail is initiated by a sending MUA for each piece of outgoing mail, which it immediately sends on to the hub system. Of course, some MUAs (like Netscape) do not start up instances of Sendmail--they make an SMTP connection directly to the hub (or to a local smart client).

Where to get Sendmail, etc.

Sendmail is freely distributed. You may be able to find precompiled versions for your Unix version. For instance, a Linux version is available via anonymous FTP from Sunsite at University of North Carolina at Chapel Hill. However, the authoritative source version is available via anonymous FTP from U.C.Berkeley. It's usually little or no trouble to compile and to get running.

Freely available m4 can be obtained from GNU's anonymous FTP archive. The current version is 1.4, which does not change frequently.

Freely available `db` code is available as a compressed-tar archive via anonymous FTP from U.C. Berkeley.

Where to Find Help

The Sendmail Usenet newsgroup is comp.mail.sendmail. The discussions are lively and participants offer timely help to those with problems. Eric Allman--the author of Sendmail--has frequently contributed answers and announcements of new versions. (Eric, who has left Berkeley for a new job, may not be able to commit as much time and energy to Sendmail as he has in the past.)

References

1. Costales, Bryan with Eric Allman & Neil Rickert. *Sendmail*. Sebastopol, CA.: O'Reilly & Associates, 1993. ISBN: 1-56592-056-2. Huge (792 pages), but essential. O'Reilly's descriptive page.
2. Avolio, Frederick M., and Paul A. Vixie. *Sendmail: Theory and Practice*. Wobrun, MA.: Digital Press/Butterworth-Heinemann, 1995. ISBN: 1-55558-127-7. An excellent guide to all aspects of Sendmail, marred slightly by its focus on the pre-V8 version. Vixie Enterprises' descriptive page.
3. Allman, Eric. ''SENDMAIL Installation and Operation Guide'' from *4.4BSD System Manager's Manual*. Berkeley, CA.: The USENIX Association and Sebastopol, CA.: O'Reilly & Associates, 1994. Also, online as 128K gziped-PostScript file. The most recent, definitive version is included in the U.C. Berkeley Sendmail distribution (as `doc/op/op.ps`) The latest details, amazingly concise, but not the easiest to follow.
4. Hedrick, Charles. ''Subject: a brief tutorial on sendmail rules''. An e-mail note written by Charles Hedrick to explain rewriting rules. A genuine Internet classic, though now dated, a gem of brevity and clarity. Available as a 16K text file.
5. Postel, J. ''Simple Mail Transfer Protocol''. RFC 821 (117K text file), 1982.
6. Crocker, D. ''Standard for the format of ARPA Internet text messages''. RFC 822 (103K text file). 1982.
7. Reynolds, J. K. ''Post Office Protocol''. RFC 918 (10K text file). 1984.

Edited by Becca Thomas / Online Editor / UnixWorld Online / beccat@wcmh.com

Last Modified: Wednesday, 27-Dec-95 09:17:32 PST