Personal Post

Build a custom email setup that fits your needs

By Jerry Peek

If you're like most IT professionals, you get floods of email. Some is probably worthless, but you need to read, organize, and store. All email programs — sometimes referred to as *Mail User Agents* or MUAs — let you read and handle incoming mail and send new messages. Most modern MUAs have lots of handy fea tures, but most of them are also crippled by their "one-size-fits-all" interface, with c the features that the MUA's developers put there. (And if Microsoft wrote the MUA even more restricted by a proprietary storage format and vulnerabilities that you can't fix yourself.)

There's no one "perfect" solution for every email user, from casual to pro. So, in this article, we won't try to choose one. Instead, we'll help you make a system that fits *you*. By learning the fundamentals of how email is transmitted, received, edited, and stored, you can build your own custom solution. Or, if you use an "off-the-shelf" package, you'll still be able to understand what's going on "under the hood." Let's dig in!

What's in a Message?

To automate email processing, it helps to know how a message is constructed and delivered. Armed with that information, you can then parse messages, search through them, and edit them with a script or a mail-filtering system.

An email message is a series of lines of characters. It has two parts, the *header* and the *body*, which are separated by an empty line. The entire message is transmitted in an *envelope*, typically via the SMTP or ESMTP protocol.

An email message header has meta-information about the message and it's a series of *header fields*. *Listing One* shows some fields from a typical header. The email header begins on the first line of the message. Header fields start in the first character position of a line, and long fields can be split across lines as long as continuation lines are indented with space or tab characters — as the `Received:` `cc:` and `Content-Type` fields are in *Listing One*. The header ends with an empty line (no whitespace allowed). (If you're wondering why there's a plus sign `+` in the `cc:` header, see the sidebar "Power Tip for Mail Administrators.")

The other part of a message is the *body*. Originally, a message body was just a series of lines of plain seven-bit ASCII text. The MIME standard extended the seven-bit email world to carry non-English languages and data reliably. If a message header has a `MIME-Version` field, the body is in MIME format; `Content-Type`describes the body format.

describes the body format.

Mail Folders in *mbox* Format

Most email systems store multiple messages in a *folder* or *mailbox* — which, on many systems, is actually a plain-text file in *mbox* format. Each message begins with a *separator* line composed of the word `From` followed by a space, the envelope sender address, and the date received. For example, `From jpeek  Thu Jun 26 08:15:09 2003` is a separator line. Messages often end with an empty line, but this isn't required. *Listing Two* shows a typical folder in *mbox* format with three (short) messages.

Notice that the last line of the second message starts with a greater-than (`>`) character. This line was *escaped* upon delivery to the folder because the original message line started with `From` and a space; left unchanged, this line would (wrongly) act as a message separator. This is one problem with the *mbox* format. The sidebar "Mail, MH-style" explains one solution.

Sending Messages

When you send a message to someone on another host, the message probably follows a path something like *Figure One*.

We've discussed all but two pieces of that path. First, the *Mail Transfer Agent* (MTA) is a program that sends and receives users' messages. Two popular MTAs are *sendmail* and Postfix. Second, a mail spool file is also called a "mailbox" or "inbox" file. It's typically named for its user and stored in a directory like */var/spool/mail* or */usr/mail*. On many systems, it uses the *mbox* format we saw in *Listing Two*. Messages wait in the mailbox until the recipient uses her MUA to read "new mail." That's when she can delete messages and move messages to other folders. She can also leave the messages in the spool file — so, for instance, a POP or IMAP server can serve them over a network to wherever she is.

Almost all of this can be customized with tools like *procmail* and *fetchmail*, which we'll cover later in this article. For more about email administration, see "Administering Email" in the May 2001 issue of Linux Magazine, available online at http://www.linux-mag.com/2001-05/guru_01.html.

One important piece of this puzzle isn't shown in *Figure Two:* how a message is addressed and delivered. Let's see how that works next.

The Message Envelope

A message has a sender and one or more recipients. Where are their addresses? Are they in the `From:`, `To:` and `cc:` header fields? (Look back at *Listing One* for reference.) Although they *may* be, they often *aren't!*

For example, our sample message in *Listing One* was addressed to *jpeek@foo.xyz*, but — as it turns out — this particular message was delivered to *jpeek@jpeek.com*. This delivery address doesn't appear in the header. (Actually, in this case, the MTA wrote that address into a `Received` field — which is handy but not required.)

This particular message was from the person in the `From` field — *joe@foo.xyz* — but that's not always true. Think of messages from spammers, for instance, who almost always use bogus addresses. (Actually, Joe sent this from his workstation, where his address is *joe@orange.foo.xyz* — but he or his MTA set his canonical address to: *joe@foo.xyz* instead.)

So where are the sender and recipients? In the *message envelope* — as part of the transaction that delivered the message over the network. This is a fundamental and important part of email delivery that's often forgotten. *Figure Two* (pg. XX) shows the situation for one of this message's many recipients, *auser@somewhere.abc*.

How does the MTA get the sender's and recipients' addresses? It may parse the message header (`To:`, `cc:`, etc.), but you can also pass addresses to the MTA in other ways — on its command line, for instance.

However the MTA gets the addresses, it looks up each recipient's mail server (with a DNS lookup) and opens an SMTP (Simple Mail Transfer Protocol) connection to it. The conversation between MTA's looks something like *Listing Three*. (We've added *send:* and *ack:* to label lines from the sender's MTA and acknowledgements from the remote MTA.)

This message had two recipients served by *smtp.somewhere.abc*. The server rejected one of them as unknown, so the envelope sender *joe@foo.xyz* (and not *staff@foo.xyz*) will

LISTING ONE: A typical email message header

```
Received: from mail.foo.xyz ([192.168.2.109])
  by mail.jpeek.com (Postfix) with ESMTP
  id 661467E4C for <jpeek@jpeek.com>;
  Thu, 26 Jun 2003 05:03:59 -0700
Received: (from joe@orange.foo.xyz)
  by mail.foo.xyz (8.12.3/8.12.3) for
  <jpeek@foo.xyz>; Wed, 25 Jun 2003 21:13:29
Message-ID: <44918.1801.JavaMail.root@accu21>
Date: Wed, 25 June 2003 23:13:22 -0400 (EDT)
From: "Joe D. Oakes" <joe@foo.xyz>
To: BigProj testers <bigproj@foo.xyz>
cc: Jerry Peek <jpeek@foo.xyz>,
  Bigproj archive <archive+bigproj@foo.xyz>
Subject: Big Project Update DRAFT, June 2003
MIME-Version: 1.0
Content-Type: multipart/mixed;
  boundary="——-080601060509020103040709"
X-Loop: jpeek@jpeek.com
```

LISTING TWO: Three messages in an *mbox*-format folder

```
From joe@foo.xyz  Thu Jun 26 05:03:59 2003
Received: from mail.foo.xyz ([2.75.12.109])
  by mail.jpeek.com (Postfix) with ESMTP
  I<...most of header and body omitted...>
AAAAA82kBC
————————080601060509020103040709—

From jpeek  Thu Jun 26 08:15:09 2003
Received: from jpeek.com (kumquat.jpeek.com
   ...most of header and body omitted...
and thanks for the great report, Joe.
>From now on, let's do them just like this!

From lucky1@bork.pe  Thu Jun 26 08:22:33 2003
Received: from msater.ru ([212.22.234.7])
   ...most of header and body omitted...
to get rich <EM>fast</EM>!!!!!!
</BODY>
</HTML>
```

FIGURE ONE: Sending a message from here to there

message —> sender's MUA —> sender's MTA —-network—->
recipient's MTA —> recipient's mail spool file —> recipient's MUA

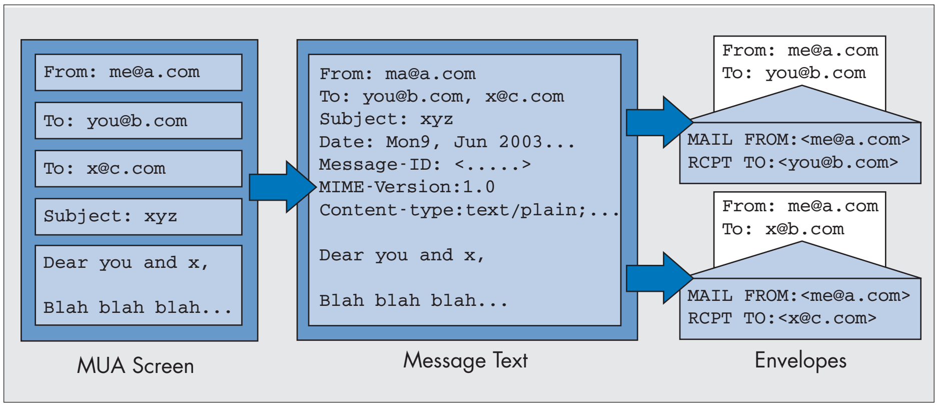

FIGURE TWO: A message and its envelope

get an emailed bounce notice from the MTA at *mail.foo.xyz*.

The MTA at *mail.foo.xyz* may need to repeat the process in *Listing Three* many times to deliver this same message to each of its recipients. *The message header and body don't change;* each recipient gets the same message, but *the envelope is different* for each recipient — just as it would be if you were sending a photocopied, "snail-mail" letter to many different people.

Notice again that the header contains neither the recipients' addresses or the actual envelope sender. (The sender's or recipient's MTA may add these to a header field like `Received`, but that's not required.) Understanding these concepts will help you do reliable automated mail processing. (For instance, what happens if some message from a *procmail* setup is undeliverable? Does it start an endless mail loop?)

You can control the message envelope by calling the MTA directly. We'll see this important technique soon.

MAIL, MH-STYLE

mbox folder format, and the giant monolithic MUAs that have grown up to handle those folders, have disadvantages. Folders with large messages, or with lots of messages, become huge files that take time to parse and rearrange. Message lines starting with "`From`" must be escaped. Meta-information must be kept somewhere else. For instance, IMAP servers create a first message with a subject like "`Do not remove this message`" and store folder information in that message.

An alternative with a lot of advantages is the MH format. MH stores each message in a separate file, so there's no separator line needed. An MH message folder is simply a Linux directory full of message files, where the filename (*1, 2,* etc.) is the message number. Using MH format, removing and reordering messages can be done with simple Linux utilities like *rm* and *mv;* you can also use MH-specific utilities to delete and renumber individual messages.

Just as directories can have subdirectories, an MH folder can have subfolders to an unlimited depth. Thanks to Linux filesystem hard links, MH can also store the same message in multiple folders, letting you organize messages into various orderings without using any additional disk space.

MH is so different that all of its uses aren't obvious at first: you have to "break the mold," the mindset of using other MUAs. It's also 25 years old, and it's showing its age in some ways. Still, if you have a lot of email, MH is worth careful consideration. There's more at http://www.jpeek.com/email/mh.

Sending Mail With a Simple MUA

An easy way to email "just text" (no graphics, no multipart messages) — a text file or program output, for instance — is with *mail*, a simple Linux MUA. Its command-line arguments build the message header. The `-s` option adds a `Subject` field, and `-c` adds `cc` addresses. (Some older versions don't support those options.) Other arguments are addresses for the `To` field. It's safest to use the simple *user@host* syntax — no names or characters like `"` (double-quote) or `<>`. *mail* reads the message body from its standard input.

The first example below passes *mail* a subject field and the two addresses *jo@a.b* and *ed@c.d*. It uses the shell operator `<` to redirect the file *log* to the standard input of *mail*.

The second example mails the standard output of the program *prog* to yourself; your address comes from the `$USER` environment variable (although you may need `$LOGNAME` instead).

```
% mail -s "build log" jo@a.b ed@c.d < log
% prog | mail -s "prog output" $USER
```

Sending Mail Without an MUA

mail can forward existing messages, but it adds a new header. To send the original body and header, invoke an MTA directly, like an MUA might. This gives you complete control of the message header, body, and envelope. Give the MTA a complete message — header and body — for delivery. This can be useful from the command line — for instance, to resend a message to other addresses without adding a new header. It's especially useful for automated mail processing from a program or from *procmail*.

Listing Four has an example with the *sendmail* MTA. (It also works with *sendmail*-type interfaces, such as the Postfix *sendmail* command.) We're sending this little message to *me@local.xyz*. The envelope sender is set to *me@remote.xyz*. Setting the envelope sender to another host, like this, could prevent mail loops as we test because any bounces should be sent there.

LISTING THREE: Sending a message from *foo.xyz* to *somewhere.abc*

```
ack:  220 I am smtp.somewhere.abc
send: HELO mail.foo.xyz
ack:  250 So happy to meet you
send: MAIL FROM:<joe@foo.xyz>
ack:  250 Sender OK
send: RCPT TO:<auser@somewhere.abc>
ack:  250 Recipient OK (local)
send: RCPT TO:<buser@somewhere.abc>
ack:  550 <buser@somewhere.abc> unknown
send: DATA
ack:  354 I am ready for the message
send: Received: (from joe@orange.foo.xyz)
send:    by mail.foo.xyz ...
send: From: FooCorp Staff <staff@foo.xyz>
send: To: BigProj Testers <bigproj@foo.xyz>
send:    ...
send: .
ack:  250 Received and stored 9373 octets.
send: QUIT
ack:  221 smtp.somewhere.abc signing off.
```

LISTING FOUR: Sending a simple message with an MTA

```
$ cat msg
From: "Joe D. Oakes" <joe@foo.xyz>
To: Some test user <myfriend@foo.xyz>
cc: Other test user <someone@foo.xyz>,
  Jerry Peek <jpeek@foo.xyz>
Subject: test message

This is a test message.
$ /usr/sbin/sendmail -f me@remote.xyz me@local.xyz < msg
```

(If you haven't worked with MTAs before, we suggest practicing before you try this example. There's an MTA tutorial in the online companion to this feature at http://www.linuxmagazine.com/downloads/2003-07/power/0307_webpage_1.html.)

As *cat* shows, the *msg* file has a bare-bones message. This new message has no `Message-ID` or `Date` fields.

After you send the message, read your *local.xyz* spool file (if possible) with a read-only file browser — for example, `less /usr/mail/me`. Compare the original *msg* file to the message you received. The MTA should have added a `Message-ID` field. The sidebar "Mail Message Message-IDs" has more about that.

Header Editing with *sed*

If you process mail automatically — with *procmail*, for instance — you may need to edit message headers. (There's a *procmail* tutorial in the July,2001 issue, online at http://www.linux-magazine.com/2001-07/guru_0 .html.) A simple way to do this is with the stream editor *sed*. *sed* reads standard input and writes to standard output; editing commands come from its command line or a script file. (Why *sed* instead of, say, Perl? *sed* is a small editing program that's quick to start. This can speed up *procmail* on busy hosts.) Try the following two commands interactively (at a shell prompt) to edit your *msg* file from *Listing Four:*

```
% sed 's/test/TEST/' < msg
% sed '1,/^$/s/test/TEST/' < msg
```

The first `sed` command changes the first `test` on every line — both header and body — to `TEST`. To edit only the header, we added the address range `1,/^$/` in the second command. This tells *sed* to start editing at line `1` and end at the first line that matches the regular expression `^$` — which is an empty line. A header always ends with an empty line.

Let's see an example. You subscribe to a mailing list from *bigcorp.za*. One of the experts on that list is *ImanXprt@somewhere.com.au*, who's working on a project like yours. You'd like to forward his list messages (but not any personal messages he might send you) to three of your staff. You modify the message subject by adding `FWD:` to the start. That's all done with this recipe in your *.procmailrc* file:

```
:0 c
* ^x-mailing-list: mfgproc@bigcorp.za
* ^from: .*ImanXprt@somewhere.com.au
| sed '1,/^$/s/^Subject: /Subject: FWD: /' | \
  /usr/sbin/sendmail -oi groucho harpo chico
```

After *sed* filters the message header and edits the subject, the shell pipes the entire message — header and body — to *sendmail*. The `-oi` option sets the *sendmail IgnoreDots* option, which keeps message lines with a single dot (`.`) from ending the SMTP DATA transfer before the whole message has been sent.

When *groucho*, *harpo*, and *chico* see their email, will their addresses be in the `To` or `cc` fields? No! Our script didn't edit those two fields. The recipients' addresses are in the message envelope as the MTA delivers the message.

(Why do it this way — instead of, say, forwardng the messages from your MUA? One reason is that your MUA would probably add a new header — so the forwarding messages would come from *your* address, preventing your staff members from sending a direct reply to the expert. The expert also couldn't follow the message thread because your staff members' replies wouldn't refer to his original `Message-ID`.)

sed is great for transformations that depend on context; changing part of an individual field or editing based on where the text is in the header. You can also put more complex scripts in files and call them with `sed -f`. There are more *sed* examples online at http://www.linuxmagazine.com/downloads/2003-07/power/0307_webpage_3.html. Often, though, there's a better header-editor than *sed*.

Header Editing with *formail*

The *formail* utility, from the authors of *procmail*, has at least two main uses: parsing *mbox*-format mail files, and editing message header fields. Let's see how to edit a header.

Automated message processing can cause a *mail loop*. For instance, a message is delivered to your system's MTA, which starts your delivery agent (e.g., *procmail*), which does something that generates another message. For instance, as we saw earlier, *procmail* might forward the message to other addresses.

To prevent mail loops, your *procmail* setup can check whether it has already seen a particular message, and if it has, not repeat the recipe that caused the mail loop. One way to do this is by adding an `X-Loop:` header field in the recipe that forwards the message — but first, in the same recipe, testing to be sure that the message doesn't already have that field. The field value should be a unique string, like your email address. Here's a *procmail* recipe to do that:

```
:0 c
* !^X-Loop: jpeek@jpeek.com
* !^from: .*a@b\.c
| formail -A"X-Loop: jpeek@jpeek.com" | \
  sendmail -oi -f bounces@jpeek.com a@b.c
```

➤ This recipe forwards a copy of your mail to the address *a@b.c*. The `c` flag tells *procmail* to pass the message to other recipes below. (You might use this recipe while you're on vacation.)

➤ The second line tests the message header for our `X-Loop:` field; if it exists, this recipe is skipped (thanks to the `!` "don't match" operator). The third line also skips the recipe for messages with a `From:` field containing `a@b.c`; these are probably original messages sent by that user to you (and she doesn't need to see a copy of her own messages).

➤ If neither of those regular expressions match, the fourth line invokes a shell to deliver the message. The first `|` means "pipe this message to a shell with the following command line." The first command uses the *formail* option `-A` to add our `X-Loop:` field. (Notice that there's no space after the `-A`.) Another pipe character (and a backslash, `\`, to continue the line) sends the edited header and body to *sendmail*, which delivers the message to *a@b.c*.

Notice the argument `-f bounces@jpeek.com`. This makes *sendmail* set the envelope sender to a special mailbox for bounced messages; if the copy can't be delivered, this asks the delivering MTA to send the bounce notice to *bounces@jpeek.com* — instead of the default, which could eventually reach this *procmail* recipe again.

formail can do many other useful header edits. Check the options and examples in its *man* page, and try searching the Web for *formail*. There are two more examples — and also more information about lockfiles — online at http://www.linuxmagazine.com/downloads/2003-07/power/0307_webpage_4.html. One example shows a way for *a@b.c* to "catch" the messages that our previous recipe forwards to her. The other is an "auto-reply" recipe that uses *procmail* and `formail -r` to accept a file by email, then email the file, on-demand, to a few specified addresses.

Dividing (and Conquering?) Messages

As we've seen, message content is divided into three parts: header, body, and envelope. If you're searching for particular text — an address, for example — you have to ask yourself what part of a message that text might appear in.

For instance, if you're looking for mail sent to the address *a@b.c*, are you searching only the message headers? Are you checking both the `To` and `cc` fields? Could that address have been only in the envelope? Do you also want to find a message to *a@b.c* that was forwarded to another address (that is, the recipient's address appears in the *body* of another message)?

LISTING FIVE: Trashing a message

```
VIRUSFILE=caught_viruses.gz
:0B :
* > 48293
* < 70000
* (^Content-Type: audio/x-wav; name=Com.bat$)
* (^AAAAAAAA2AAAAA4fug4AtAnNIbgBTM0hVG\+p$)
* (^RE9TIG1vZGUuDQ0KJAAAAAAAAAAYmX3gXPgT$)
| gzip >> $VIRUSFILE ; \
  (echo "  Messages in $VIRUSFILE file:"; \
  zcat $VIRUSFILE | grep "^From ") | \
  mail -s "NOTICE: virus caught" $LOGNAME
```

LISTING SIX: Virus Notice

```
From: Jerry Peek <jpeek@jpeek.com>
Date: Thu, 26 Jun 2003 08:45:13 -0400 (EDT)
To: jpeek@jpeek.com
Subject: NOTICE: virus caught

  Messages in caught_viruses.gz file:
From lucky1@bork.pe  Thu Jun 26 08:22:33 2003
From mom@home.ci  Thu Jun 26 08:45:09 2003
```

Storing messages in the *mbox* format complicates this because you may need tools like *grepmail* (introduced in the May 2003 "Do It Yourself" column, available online at http://www.linux-mag.com/2003-05/diy_01.html) or *formail* to split the folder or to search invididual messages.

Then there's the question of sorting incoming mail. Many MUAs have ways to filter mail automatically — for instance, to put all new messages from Joe into a mail folder named *joe*. *procmail* is so flexible that it can out-do almost all of those filters: see our earlier forwarding example, for instance. Understanding the format of a message header and body can help you make *procmail* recipes that do very specific things.

For instance, an email virus is loose in your company and you want to stop it in its tracks. When you inspect the message by reading the mail spool file or a mail folder (using a pager like *less*, and not an MUA, so there's no chance of activating the virus!), you can see that the virus is in a MIME part in the body.

Listing Five has a *procmail* recipe to catch those messages. It searches for a filename from one of the MIME multipart headers and two lines from the MIME-encoded virus body. (We've shortened the lines for printing.)

The shell command starts with *gzip*, which reads the message from its standard input and appends the compressed version to an archive. Then a series of commands find the message-separator lines in the compressed file and use *mail* (explained earlier) to send you an email notice that lists the viruses caught so far. *Listing Six* shows the result.

This technique is useful for sending any sort of notices from a *.procmailrc* recipe, notifying someone that something has happened, for instance. For more details about this recipe, see http://www.linuxmagazine.com/downloads/2003-07/power/0307_webpage_5.html.

Managing Multiple Accounts, Part I: Sending Mail

You might have mailboxes at hosts around your company or all over the Internet. What a tangly mess that can be! This section looks at ways to manage lots of mail accounts.

Let's say that you have a personal account on AOL, but you're currently logged onto your company system. Or you're logged onto your home system, but you want to send email from your company address. There's no Internet "rule" that says you must send email with the address of the host you're currently logged on to. Simply set the `From` field to the address you want to use. If your MTA or email provider won't allow that, set a `Reply-to` field instead; MUAs should check this field, and if it exists, they should send replies to that address instead of to the `From` address. In some MUAs (Mozilla and Netscape, for example), you can set up multiple "accounts", each with a different `From` address. So, if you're at work and you want to send mail from your AOL account, simply use a header that says `From:` *`you`*`@aol.com`. The message doesn't have to be routed through an AOL mail server.

In cases like these, you may want to send a copy of the message back to the account you're emulating. A simple way to do this is by sending yourself a *blind copy*: put your address in the `bcc` header field (like `bcc:` *`you`*`@aol.com>`). What does this do? A "blind" address is added to the message *envelope*, but *not* to the message *header*. Unfortunately, some email providers (like Microsoft Hotmail) treat blind copies (where the recipient's address doesn't appear in the header) as "junk" because spammers often send blind messages. You may need to send yourself a visible copy (the `cc` field) instead.

Managing Multiple Accounts, Part II: Receiving Mail

If you get mail at multiple addresses, you don't have to log in to each of them to "check your mail" everywhere. There are quite a few ways to manage email centrally.

If your MUA supports the Internet Message Access Protocol (IMAP), it can open mail folders stored on a remote host. This lets you maintain mailboxes while you travel, for instance. One disadvantage of IMAP, though, is that it's not as simple to access your mail *without* an MUA. For instance, running a weekly *cron* job to clean out old messages on each host can be tricky.

MAIL MESSAGE IDENTIFIERS

Each new email message you create has its own unique identifier called the *message ID*. Everyone who receives the same header/body pair — where the only lines that differ are fields like `Received` — should have the same `Message-ID` stamped in their copy of your message.

Message IDs are important because your MTA uses the IDs to track and log messages. MUAs also use them to find message threads (the `References` and/or `In-reply-to` fields list the message ID of messages being replied to). You can use message IDs, too — for instance, to write a script or a *procmail* recipe to find duplicate messages.

An MUA or an MTA typically add the `Message-ID` field to new messages, and `Message-ID` fields have a specific format. To read more about message IDs and their format, see http://www.ietf.org/rfc/rfc2822.txt.

POWER TIP FOR MAIL ADMINISTRATORS

The plus sign (+) in the address `archive+bigproj` in the `cc` header field is a handy way to "overload" a single email address — and, in effect, give multiple addresses to any user who wants them. You can find details about this technique in the online companion to this article at http://www.linuxmagazine.com/downloads/2003-07/power/0307_webpage_2.html.)

Another good choice is automatically forwarding some or all mail from your remote mailboxes to a central “mail hub” account, then accessing all your mail by logging onto that one host. If your remote systems support the *.forward* file, create that file and put a complete destination email address into it. You can also use *procmail* to forward some or all of your mail.

For instance, the following single recipe in a *.procmailrc* file on the *a.com* host forwards all incoming mail to *you@b.com:*

```
:0
! you@b.com
```

(You might want to do some other filtering first — loop-stopping, for instance, as shown earlier.)

The handy *fetchmail* utility is designed to retrieve messages from multiple accounts around the Internet. Fetchmail starts a session by POP (Post Office Protocol), IMAP, or one of several other protocols to retrieve waiting email from each remote host you specify. Then it opens an SMTP session to some MTA (typically, your local MTA) and writes the messages there. Fetchmail can start a *daemon* (a continuously-running background process) to do this automatically — or you can run it by hand or from a *cron* job.

fetchmail is very configurable. What *fetchmail* can't do in message retrieval and storage, *procmail* probably can. For instance, maybe you'd like to put messages from a particular remote mailbox into a particular local folder. To make *procmail* work together with *fetchmail,* you'll look at the message headers from that host to see what makes messages from a particular mailbox unique and then add a *procmail* recipe to “catch” those messages.

The *tracepolls* option in Fetchmail versions 5.0 and later makes this easy: it adds the remote host and account name to a `Received` header field, like this:

```
Received: from mail.mars.xyz [63.15.21.7]
    by localhost with POP3 (fetchmail-6.2.0
    polling mail.mars.xyz account jerry)
```

The *procmail* recipe in *Listing Seven* would put those messages into the *mars* folder. This single-line *procmail* pattern matches multi-line header fields. *procmail* concatenates fields while it pattern-matches. If you want to do a similar thing with other tools, try filtering messages through `formail -c`. It makes all header fields single-line.)

LISTING SEVEN: Procmail recipe to grab messages from *fetchmail*

```
:0:
* ^received: .*fetchmail-.* polling
mail\.mars\.xyz account jerry
mars
```

Fetchmail can do much more. The program and the details are at http://www.catb.org/~esr/fetchmail. It comes with a GUI Python script named *fetchmailconf* that simplifies setup and testing.

Classifying Mail with POPfile

POPfile is a proxy, written in Perl, that sits between your MUA and a POP server. (This means it works with basically all MUAs on any platform that has Perl.) It uses Naïve Bayesian filtering to sort your new messages into categories or “buckets.” Unlike our simple *procmail* recipe that searches for specific strings of text, Bayesian filtering “learns” to recognize particular types of mail. For instance, it might learn to recognize meeting minutes, work mail versus family mail, or — what many of POPfile's users use it for — catching spam.

After it classifies a message, POPfile modifies the message header — either adding a word to the `Subject`, like `Subject: [spam]`, or adding a field to the header, like `X-Text-Classification: spam`. Then your MUA — or a delivery agent like *procmail,* if you've wired POPfile to grab mail via *fetchmail* — can decide what to do with each incoming message.

POPfile uses your web browser, with a local HTTP server on port 8080, as a control interface. The most important part of the display is a history of the messages POPfile has classified. POPfile “learns” best by “Training Only on Errors,” or TOE, where you reclassify messages that it got wrong (and it learns the correct bucket). Some users get almost 100% accuracy — although spammers now seem to be hard at work trying to fool this popular spam filter.

The POPfile home page is http://popfile.sourceforge.net.

What Next?

In this article, we've looked at everything from the parts of a message (its header and body) and how it's addressed and delivered (the envelope) to altering message headers with *sed* and *formail,* pulling incoming mail from multiple servers with *fetchmail,* and chosing message destinations (and more) automatically with *procmail.*There's a lot more to study to make your email setup as powerful and efficient as it can be. (A quick web search, or a look through newsgroups in the *comp.mail* hierarchy proves the point.) The fundamentals we've gone through should help you on your way.

In next month's Power Tools column, we'll look at *fetchmail* in more detail.

Jerry Peek is a freelance writer and instructor who has used Unix and Linux for over 20 years. He's happy to hear from readers at jpeek@jpeek.com.