Pure Postal Power

By Jerry Peek

Last month's feature, "Personal Post," (available online at http://www.linuxmagazine.com/2003-07/email_01.html) explained how email is delivered, and also described a variety of power tools ideal for untangling a morass of messages.

One of the tools mentioned last month was *fetchmail*, a great Linux utility that can centralize your email and automate tasks that would otherwise be spread across the network on multiple mail systems on multiple hosts. This month, let's look at *fetchmail* in more detail, and apply it to some difficult email problems.

Setting up *fetchmail* to work well takes some time and requires some familiarity with email transport. Let's quickly review how email works before we go any further. Here are the basics:

- Users compose, send, and read email with a *Mail User Agent* (or MUA). *mutt*, Outlook Express, and Mozilla are examples of an MUA.

- Email is sent from user to user by a *Mail Transfer Agent* (MTA), a program like *sendmail* or *qmail*.

- The *header* of an email message is composed of a series of fields with names like `Received`, `Message-ID`, and `Content-Type`.

- It's important to understand that the addresses in header fields like `From` and `To` are *not* necessarily where the message originated from or where the message was delivered to! Instead, the message *envelope* contains the actual addresses of the sender and recipients — in the same way a paper envelope holds "snail mail" for delivery.

- Years ago, the *body* of a message was limited to plain ASCII text. But, more recently, the MIME standard provides for reliable delivery of messages with non-ASCII text and data.

- The *procmail* utility is a delivery agent that processes incoming messages. *procmail* can file, forward, and possibly edit messages as they arrive. *procmail* rules are called *recipes*, and recipes can be fairly elaborate — it's not unusual for a recipe to invoke other Linux utilities such as *formail*, *sed*, and your local MTA.

- Mail messages are typically stored in a *folder* or other *message store*. If it's on your local machine, your MUA can read that folder directly. If it's on a remote machine, a network protocol like POP can copy the messages to your local host, or IMAP can access that folder directly.

- The *POPfile* program retrieves messages from a POP server, and analyzes them dynamically, marking the header of each message to help you categorize your incoming email.

- The *fetchmail* program retrieves messages from remote servers and gives them to your local MTA to deliver.

If you're familiar with all of these concepts, you're already well-ahead on the power curve. Now, let's dig in to what you can do with *fetchmail!*

(Some of) What Fetchmail Does

fetchmail can retrieve email messages that are waiting for you or multiple users, on multiple remote servers — and deliver them to one or more other mailboxes. It can do this on-demand (getting new mail when you run *fetchmail* from an hourly *cron* job or from the command line) or as a *daemon* (a continuously-running "background" process that you start just once — for instance, each time your system reboots).

What good are all of those features? In the simplest case, *fetchmail* lets you manage multiple mailboxes from one central host. *fetchmail* can talk to many remote servers, get your email, and deliver it reliably to your local MTA. All of the details are handled for you: the messages simply appear in your local mailbox. *fetchmail* can also "mark" message headers so you'll know which message came from which mailbox, letting you separate incoming messages automatically with a tool like *procmail*.

fetchmail understands multiple protocols, including obscure ones that most MUAs don't. It supports SSH and other methods to encrypt and verify mail transfer from remote hosts. It can split huge message deliveries into smaller parts, retry if the connection fails, and much more. This means that your MUA doesn't need to be super-sophisticated and reliable at retrieving remote mail. In fact, this lets you use *any* MUA — even one that can't get email from remote mailboxes.

First-time Setup

The *fetchmail* home page, http://www.catb.org/~esr/fetchmail, has all the links you need — including lots of documentation on installing and using the utility. It's worth looking through the long list of *fetchmail* features so you'll be ready to take advantage of all its capabilities. That's even true if you use *fetchmailconf*, a graphical configurator (written in Python). *fetchmailconf*'s short descriptions make setup easy, but it helps to have some background so you'll understand exactly what the settings are doing. There's a screen shot of one panel from *fetchmailconf* — the Expert-mode Mail Server Options — in *Figure One*.

In the user interface for *fetchmailconf*, the "Probe for supported protocols" button is handy if you don't know a mail server well. For example, the probe might suggest that a POP server also supports APOP — which, if you have *popauth* set up, will let you avoid sending your password in the clear.

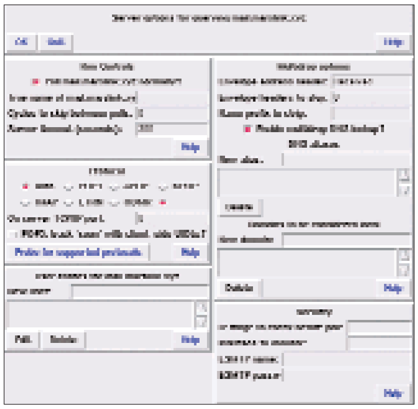
FIGURE ONE: *fetchmailconf*'s "Expert Mail Server Options" panel.

The expert configurator can probably get you a working setup in a flash (if you browsed the documentation first), but you may be able to optimize it. Let's take a look at one sample setup, then look at a few (perhaps unusual) ways to integrate *fetchmail* into a personal setup on your workstation.

A Sample *.fetchmailrc* File

Let's say that you get mail from four mailboxes on other hosts. You always want to monitor your office mailbox at *pop.xyz.com*. You also get mail from your office "support" mailbox from the same host, but only while you're on support duty from noon to 1 PM weekdays. Third, you're testing POPfile on your workstation, running its POP server on port 9999, constantly getting copies of "marketing" mail to see how well POPfile can classify them; you're sending those messages to the local address *alexa*. Fourth, you sometimes want to get copies of your family's email from *mail.marslink.net* — but also leave the messages on that server for your family to read from home.

Listing One shows part of the *.fetchmailrc* file you'd put in your home directory (the `set` *`option-name`* lines have been omitted). The *fetchmail* manual page and FAQ have details, but let's see the highlights:

- The `defaults` line defines options used in all of the other entries. The first line specifies that all servers use the POP3 protocol; the `tracepolls` option marks message headers with the remote mailbox name; and *fetchmail* delivers most remote messages (except the *marketing* mail) to the local user `zoe`.

- The other entries describe four remote mailboxes. The two starting with `poll` are checked every time *fetchmail* runs — unless you specify particular mailboxes on the command line. Conversely, the two entries starting with `skip` aren't polled unless you specify them on the command line.

- The password entries are needed for *fetchmail*'s daemon mode. You can also use them if you aren't concerned about someone reading the passwords from your *.fetchmailrc* file. Without them, *fetchmail* will prompt you for each POP3 server password. However, there are other ways to authenticate; see the *man* page for *fetchmail*.

- `via` is used in three of the entries to identify three remote mailboxes by names that we choose: `office`, `support`, and `home`. Otherwise, *fetchmail* identifies remote mailboxes by the server's hostname or IP address, as we have with the local POPfile server. Here, `via` is necessary because we've got two different mailboxes with different configurations on the same server *pop.xyz.com*, and we don't want to poll both of them every time.

- A local POPfile daemon is running at port 9999 on your workstation, overriding the default POP port 110. *fetchmail* accesses it through the local IP address `127.0.0.1`, getting `marketing` mail from the `pop.xyz.com` mail server.

- The `keep` option on the `home` entry ensures that messages won't be deleted from the remote mailbox.

There are many other options. Also, you can override the *.fetchmailrc* configuration when you call *fetchmail* from the command line. For instance, the command `fetchmail -K home` would get mail from the *home* mailbox and delete messages after downloading them.

LISTING ONE: Part of a *.fetchmailrc* file

```
defaults proto pop3 tracepolls user 'zoe' here
poll office via 'pop.xyz.com'
  user 'zsmith' there with password 'X'
skip support via 'pop.xyz.com'
  user 'support' there with password 'X'
poll 127.0.0.1 port 9999
  user 'pop.xyz.com:marketing' there
  with password 'X' is 'alexa' here
skip home via 'mail.marslink.net'
  user 'sm2736' there with password 'X' keep
```

Poll Control

How often should *fetchmail* poll remote mailboxes? To get mail from all entries that start with *poll,* daemon mode can do the job: it puts *fetchmail* in the background and polls at an interval (in seconds) that you set. For example, `fetchmail -d 360` causes *fetchmail* to poll every six minutes.

(Tip: if you configure *fetchmail* to poll an external mail server often, your local machine may not even *need* an incoming SMTP server accessible to the outside world! This "pull-only" operation can be a security win.)

Other than its interval option for individual servers, though, *fetchmail*'s daemon mode doesn't give much control over the times that *particular* servers are polled. For more control over your own *fetchmail* process, you can run it from the command line, from a GUI button on your desktop, from a *cron* job, and so on.

For instance, you can add a desktop button that runs `fetchmail` to poll all servers, or runs `fetchmail mail.marslink.xyz` to poll only that one server. You can also write a shell alias that checks for email on your home ISP's server when you type *hm* at a shell prompt. Or, you can set up *cron* jobs to do the right thing, polling your office and marketing mailboxes 24 hours a day, polling the support mailbox every five minutes from 1200 to 1300 hours, and polling your home account only after 6 PM. (The February 2003 "Power Tools" column, "Running Jobs Unattended," available online at http://www.linux-mag.com/2003-02/power_01.html, covers *cron*.)

If you want multiple configurations, you can name a specific configuration file on the command line, and you can even feed a configuration setup to *fetchmail*'s standard input. You, or a *cron* job, can also simply update the *.fetchmailrc* control file when you want its configuration to change; the *fetchmail* daemon will notice the change and automatically restart itself.

So, for example, here's a *cron* job that would use your daytime setup between 8 AM and 5 PM weekdays, and your nighttime setup the rest of the time. (We're using relative pathnames to the files; a personal *cron* job typically runs from your home directory.)

```
0 8 * * 1-5 cp lib/fetchmail.day .fetchmailrc
0 17 * * 1-5 cp lib/fetchmail.nt .fetchmailrc
```

If you run *cron* jobs like the ones mentioned, what happens if you aren't at work? Will *cron* keep pulling in the "support" mail while you aren't there? A simple personal crontab entry to fix that would look like this (which you should enter on a single *crontab* line):

```
* 12 * * 1-5  {who | grep "^$USER "
  >/dev/null ; } && fetchmail support
```

Every minute between 12 Noon and 12:59 PM, that command runs *who* to get a list of logged-on users. The *grep* command looks for your username (you may need `$LOGNAME` instead) at the start of a line, with a space after it (to be sure it's unique). *grep* output is redirected to the Linux "bit bucket" */dev/null*, but *grep* also returns an exit status. If the status is 0 (true), because *grep* found your account logged on, then the shell's `&&` operator executes `fetchmail support`. If you aren't logged on, *cron*'s shell won't run *fetchmail*.

Got Mail?

When *fetchmail* polls, how can you know if it retrieved new mail? If you're running it from the command line and don't have the `syslog` option set, you'll see progress messages that tell you what's happening. Otherwise, you might set your MUA, or an X utility like *xbiff,* to ring a bell or do something else when new mail arrives in your local mailbox.

And here's another idea: if you're running *fetchmail* from a command line (a button, a *cron* job, and so on), you can test its exit status. Fetchmail returns a zero exit status if it's delivered new mail, or a non-zero status otherwise. (See "Exit Codes" in the *man* page.)

If you're using the X Window System and you want to be notified right away when mail comes in, try the *getmail* script shown in *Listing Two*. It runs *fetchmail,* passing it any command line arguments that you specify through the shell's special parameter `$@`. For instance, to check *support* and *home* mail, you'd run `getmail support home`. The shell's `if` statement checks the *fetchmail* exit status; if it's zero, a yellow *xmessage* window opens near your mouse cursor.

While *getmail* is useful, you can't run it from a *cron* job. Why? A *cron* job usually doesn't run under the control of your window system, so it doesn't have access to your X display.

See POWER TOOLS, *pg. 64*

LISTING TWO: *getmail* script, version one

```
#!/bin/sh
if fetchmail "$@"
then xmessage ?near ?bg yellow "NEW MAIL"
fi
```

LISTING THREE: *getmail* script, version two

```
#!/bin/sh
fetchmail support &&
test -s /home/zoe/mail/support &&
xmessage -near -bg yellow "NEW MAIL"
```

POWER TOOLS, *from pg. 36*

And there could be another problem with *getmail*. If your setup does some mail filtering after *fetchmail* runs — for example, POPfile looking for spam or *procmail* tossing it — it's possible that the "NEW MAIL" window might pop up when there's no new email that you'd want to see.

The Daemon's in the Details

The second version of *getmail*, in *Listing Three*, runs *fetchmail* on the remote *support* mailbox, then tests the local *support* mail folder. If the folder contains mail, the *xmessage* window opens. (This still isn't optimal because, if there's any old mail in the *support* folder, *xmessage* tells you about it. It's just a simple example of what you can do. A smarter program that watches the mailbox state, like *xbiff*, would avoid false notifications.)

A program like *getmail* doesn't work with *fetchmail*'s daemon mode, of course, because it waits for *fetchmail* to terminate. If you want to run a program after each time the *fetchmail* daemon polls a server, add the program name and command-line arguments to a `postconnect` option in your *.fetchmailrc* file, like this:

```
skip support via 'pop.xyz.com'
  user 'support' there with password 'X'
  postconnect /home/zoe/bin/mail-notify
```

The *fetchmail* daemon won't poll again until your `postconnect` program terminates, so that program should put any long-running processes (like an *xmessage* window) into the background. There's an example script online at http://www.linuxmagazine.com/downloads/2003-08/power/mail-notify.txt.

But it's hard to use an X utility like *xmessage* from *fetchmail*'s daemon mode because, by default, a *fetchmail* daemon detaches from your *tty*. The *fetchmail* `-N` or `--nodetach` command-line option keeps the daemon running in the foreground. So, if you start *fetchmail* from your shell's command line, it will have access to your X display. You could put `fetchmail -N` into the background, from your shell — and then it can access your terminal to send messages and open X windows.

The *fetchmail* design notes at http://www.catb.org/~esr/fetchmail/design-notes.html are worthwhile reading and end with a long list of RFCs that explain more about this tool.

Jerry Peek (jpeek@jpeek.com) is a freelance writer and instructor who has used Unix and Linux for over 20 years.