By Vikram Vaswani

Table of Contents

The Truth, The Whole Truth, And Nothing But The Truth...

I gotta be honest with you. I love email.

I love the way the email system works. I love the thought of all those messages whizzing around the Internet, jumping from one host to another until they reach their destination. I love the fact that it lets me get in touch with anyone on the planet for less than the cost of a postcard. In short, I think it's the coolest thing since sliced bread.

Managing it, however, really wipes the sunshine from my day.

You see, when I first got hooked up to the Internet, my mail flow was limited to a couple messages per week. No one in cyberspace even knew I existed − and I liked it that way. Then I discovered mailing lists like the Internet Oracle and the Internet Tourbus, and my mailbox started getting bigger. Once people figured out I had email access, I started receiving messages from employers, family and friends in far−off places. I got on the radar of the friendly neighbourhood spammer, who though I might be interested in everything from pet food to oil−pumping machinery and graciously added me to his database...

I'm sure this picture is familiar to you. Before long, the trickle turned into a torrent, and I started spending a good fifty minutes a day wading through the previous night's email. I needed a solution, and I needed it fast.

That's where procmail saved me.

Speaking Geek

Let's start with a very basic question − what is procmail, and how is it going to make your life more fun?

Before I can answer that question, there are three acronyms you need to learn. Here they are:

MTA: An MTA, or mail transfer agent, is a program that routes mail from one host to another on the Internet. An MTA accepts email, looks at the destination address, and either passes the message on to another MTA or hands it off to an MDA for delivery to a local mailbox. Examples of common MTAs are sendmail and qmail.

MDA: An MDA, or mail delivery agent, is the program that accepts email from an MTA and actually delivers it to the recipient's mailbox. When an MDA works locally − that is, delivers mail to user mailboxes local to that host only − it is sometimes referred to as an LDA, or local delivery agent. Examples of common MDAs are procmail and mail.local.

MUA: An MUA, or mail user agent, is the program users interact with to view email messages, reply to or forward them, compose new messages and otherwise manipulate the contents of a mailbox. Examples of common MUAs are pine and mutt.

In a typical *NIX environment, the MTA takes care of accepting email messages and figuring out how to deliver them to their destination. Once the email message actually reaches the destination host, control passes to the LDA, which takes care of delivering the incoming message to the specified user's mailbox (or "mail spool") on that system. The user may then use any MUA to view and manipulate the mailbox and its contents.

Where does procmail fit into this picture? Developed by Stephen R. van den Berg in 1998, procmail is an LDA, commonly used to deliver messages to local mailboxes. Don't think of it as just a delivery boy, though − procmail's built−in constructs allow you to convert it into a primitive mail robot, automatically scanning and sorting received messages according to pre−defined rules.

These procmail rulesets (or "recipes") are very powerful − they allow you to do all kinds of magical things to your email, including automatically forwarding it elsewhere, starting special programs on your system for different types of email, filtering it into different mailboxes or − this is the part I like the most − automatically trashing spam without you ever having to see it.

The end result? Automatic, transparent mail management that allows you to zero in on important email, leaving the chaff for later.

Intrigued? Flip the page and let's get going!

Putting The Pieces Together

The first order of business to install procmail on the your Linux box. Drop by the official procmail Web site at http://www.procmail.org/ and get yourself the latest stable release of the software (this tutorial uses procmail 3.2.2).

Once you've downloaded the source code archive to your Linux box (mine is named "olympus"), log in as "root"

```
[me@olympus] $ su -
Password: ****
```

and extract the source to a temporary directory.

```
[root@olympus] $ cd /tmp
[root@olympus] $ tar -xzvf /home/me/procmail-3.2.2.tar-gz
```

Next, compile and install the package.

```
[root@olympus] $ cd /home/me/procmail-3.2.2
[root@olympus] $ make install
```

Unless you specified a different path, procmail will have been installed to the directory "/usr/bin".

Next, you need to configure your MTA to use procmail for local delivery. You will need to refer to the instructions that ship with your MTA to accomplish this (if you're using sendmail, it should come configured to use procmail out of the box).

Once you've got procmail installed, the next step is to take it for a test drive. Let's do that next.

No Forwarding Address

Let's start with a very simple example − setting up procmail to forward all your incoming mail to a different email address.

Switch to your home directory and use your favourite text editor to create a file named ".procmailrc". Enter the following lines into this file (replace the second line with the email address your messages should be forwarded to):

```
:0
!my.other.email.address@some.other.host
```

Save the file, and change its permissions so that it is owned by you, writable only by you, and readable by everyone else (procmail may barf if this file is secured in any other manner).

Now, send yourself some email and keep an eye on your mail queue. Once sendmail processes your message and hands it over to procmail for local delivery, procmail will read the instructions in your ".procmailrc" file and forward the message to the specified destination email address.

Simple, huh? Let's look at the engine that made this happen:

```
:0
!me@some.other.host
```

This snippet is a single procmail "recipe". Each recipe must conform to the following format:

```
:0 <flags>
* <pattern>
<action>
```

As you can see, a recipe is simply a regular expression that procmail searches for in the header of incoming email. If a message matches a recipe, procmail looks in the action line of the recipe to figure out what to do with the message; if it doesn't match, procmail proceeds to the next recipe. If no recipes match, procmail delivers the message in the default manner.

The line

```
:0 <flags>
```

marks the beginning of a recipe. It may include one or more flags that alter the way procmail handles the recipe − for example, generating a copy of the message first, performing regular expression matching on the body instead of the header, matching in a case−sensitive manner, and so on.

This is followed by one or more

```
* <pattern>
```

lines, which mark the beginning of a procmail condition. These conditions are typically regular expressions, which procmail uses when scanning the headers of incoming email. A single recipe can contain multiple conditions (note that, in the example above, I've omitted this line altogether, since I want to match *all* incoming email; however, I'll shortly demonstrate how this pattern−matching capability can come in handy for matching specific message types).

Messages that match the specified condition(s) are handled via the last line of the recipe, which specifies the action to be taken.

```
<action>
```

Typically, this <action> is a mailbox name, to which the matched message(s) are to be delivered; however, it may also be another program or (as in the example above) another email address. An exclamation (!) at the beginning of the action line indicates that the message should be forwarded to the email address(es) following it, while a pipe (|) indicates that it should be piped as input to the specified program; anything else is treated as a literal mailbox name.

Once the action line of the recipe is successfully executed, procmail considers its job done, exits, and goes back to waiting for another incoming message.

The example above would work only for *your* mail, since the ".procmailrc" file resides in *your* home directory. If you're a system administrator, you might want to apply a recipe to *all* users on the system − in this case, you should use procmail's global recipe file, usually located in "/etc/procmailrc".

Of Wheat And Chaff

Let's look at another recipe, this one a little more complex. Let's suppose I wanted to forward only email from the domain "melonfire.com" to my second email address. Here's what my recipe would look like:

```
:0
* ^From:.*@melonfire.com
!my.other.email.address@some.other.host
```

In this case, only messages containing the expression "melonfire.com" in the "From" header line would get processed by this recipe and forwarded to my second email address − all other messages would be delivered in the default manner.

Wanna filter it a little more? Let's make sure I only forward messages that are addressed to me directly, and that are less than 25K in size:

```
:0
* ^From:.*@melonfire.com
* ^To: me@melonfire.com
* < 25600
!my.other.email.address@some.other.host
```

I can also send mail to a specific mailbox instead of forwarding it to another address − all I need to do is specify the mailbox name in the recipe's action line.

```
:0
* ^TO_webmaster@melonfire.com
WEBMASTER
```

In this case, all messages directed to "webmaster@melonfire.com" get diverted to a mailbox named "WEBMASTER" in my home directory. The ^TO_ construct is a special expression that tells procmail to scan the "To", "Cc" and other destination headers for a match.

If you're on a lot of mailing lists, it's pretty simple to adapt this recipe so that messages from each mailing list end up in separate folders. Take a look at my sample configuration:

```
:0:
* From:.*redhat-list@redhat.com
LISTS

:0:
```

```
* From:.*php-general@lists.php.net
LISTS

:0:
* From:.*oracle-request@cs.indiana.edu
FUN

:0:
* From:.*list@lockergnome.com
FUN
```

Note the second colon (:) at the beginning of the recipes above − this tells procmail to use a lock file while processing these recipes to ensure that the mailbox to which mail is being delivered does not get mangled if two processes try to access it at the same time.

Finally, here's a wicked little recipe I devised especially to help me filter out my annoying cousin Joe, who insists on forwarding me stale jokes every few days.

```
:0
* ^From: .*joe@annoyances.domain.com
/dev/null
```

Hey, it works on in−laws too. No kidding.

Lies, Sweet Lies

You might remember, from the previous page, that I said that once procmail executes the action line of a recipe, it stops processing further recipes and moves on to the next message.

Well...ummm...that was kinda a little white lie.

You see, it is possible to have procmail process a message as per the action line within a recipe, and then, instead of exiting, continue to process subsequent recipes to see if further matches exist. This is accomplished by adding the "c" flag (for "carbon copy") to the first line of a recipe. When procmail encounters this flag, it will first generate a copy of the email message; this copy is then processed as per the recipe, while the original message is passed on to subsequent recipes.

In order to better understand how this works, consider the following variant of the very first recipe in this article, which not only forwards a copy of every incoming message to another email address, but also backs it up to a mailbox named "BACKUP":

```
:0 c
BACKUP

:0
!my.other.email.address@some.other.host
```

In this case, every message intercepted by procmail first gets transferred to the "BACKUP" mailbox. Since the "c" flag is present on that recipe, procmail will also create a copy of the message and continue processing it. This message copy will then match the second recipe, and will get forwarded to the specified email address.

In case you'd prefer to have incoming mail stored in your local spool and also forwarded to another email address, try the following simple variant:

```
:0 c
!my.other.email.address@some.other.host
```

In this case, incoming messages will get automatically forwarded to the specified email address, and procmail will also generate an extra copy because of the special "c" flag. Since the ".procmailrc" file contains only a single recipe, no further matches will exist and so the copy will be handled in the default manner − that is, delivery to the user's local mail spool. The end result: incoming messages are both stored locally and forwarded out of the system.

There are a number of other flags that can be used to alter procmail's default behaviour − take a look at the procmail manual pages for more information.

Canning The Spam

As you might imagine from the preceding discussion, procmail is a great tool to use if you're concerned about spam clogging your mailbox. The simplest solution here is to add a series of recipes to the beginning of your ".procmailrc" file, which can scan incoming email for known spammer addresses and automatically filter those messages out. As an example, consider the following set of recipes:

```
:0
* ^From:.*angie76@spammer.com
SPAM

:0
* ^From:.*@known-bad-domain.com
SPAM

:0
* ^From:.*clouded_mind_99@hotmail.com
SPAM

:0
* ^Subject:.*make money fast
SPAM
```

As I stated earlier, it's usually a good idea to place these recipes neat the top of your ".procmailrc" file, so that they are processed first.

If you get a large amount of spam, the technique above may seem inconvenient, as your ".procmailrc" file will rapidly grow in size as more and more spammers add your email address to their database. For convenience, you can either place these recipes in a separate file and include them into your ".procmailrc" (this technique is discussed a little further down) or you can create a so-called "black list" of known spammer addresses in a separate file, and have procmail scan through it looking for a match every time it receives email. This second technique is a little processor-intensive, but has the benefit of simplicity - it needs only a single recipe. Take a look:

```
:0
* ? egrep -is -f /home/me/black-list.txt
SPAM
```

The file "black-list.txt" is a simple list of email addresses, like so:

```
angie76@spammer.com
clouded_mind_99@hotmail.com
o@o.com
```

In this example, the "egrep" program is called by procmail to see if the message headers contain a known spammer's email address. If a match is found, the message is moved to the "SPAM" mailbox. This is a much simpler approach than the one described previously, as all you need to do is update your black list on a regular basis.

Another technique of dealing with spam is so−called "reverse filtering", which uses a "white list" of known addresses. In this case, mail is only delivered to your mailbox if it matches an address in the white list; all non−matching email is treated as spam and either transferred to a spam folder for later review, or summarily deleted. The following example demonstrates:

```
:0
* !? egrep −is −f /home/me/white−list.txt
SPAM
```

I, Robot

You've already seen how procmail can be set up to forward incoming email to another email address, or to append it to a mailbox. You might also remember that there's a third option − pipe the mail to an external program for further processing. More often that not, this program is "formail", a companion program that ships with the procmail distribution.

Formail is a mail formatter, used primarily to extract or manipulate the headers of an email message. Its applications include creating new email messages on the fly, checking the value of a header in a received message, rewriting exiting headers with new data, removing unwanted headers, forcing messages into a standard mailbox format and splitting up a message digest or mailbox into individual messages.

Formail makes it possible to do some very creative things with your email − and one of its most common applications includes creating a simple auto−responder for your email when you're away from your computer. Consider the following recipe, which demonstrates:

```
:0
| (/usr/bin/formail −r ; cat autoresponse.txt) |
/usr/sbin/sendmail −oi
| −t
```

In this case, every message received is piped through formail, which uses the contents of the file "autoresponse.txt" to generate a reply back to the original sender.

It should be noted that the recipe above is purely illustrative and should not be used in a live environment, as it does not include error handling for email loops or messages from mailing lists; refer to the procmail manual for a more comprehensive example.

You can also use procmail in combination with the SpamAssassin spam filter (http://www.spamassassin.org/) to automatically detect and block spam. All you need to do is pipe each message through SpamAssassin, and let it check to see if the message matches any of its spam rulesets. Based on the results of its heuristic tests, SpamAssassin automatically tags each message with an "X−Spam−Status" header indicating whether or not it is spam; this header can then be used by procmail to filter spam out of your regular mail spool into your "SPAM" mailbox, or send it straight to the trash can.

Here's how this might work.

```
:0fw:
| /usr/local/bin/spamassassin

:0:
* ^X−Spam−Status: Yes
SPAM
```

More information on how SpamAssassin can be used with procmail is available at http://www.spamassassin.org/

Tweaking The Engine

In the initial stages of setting up your procmail recipes, you'll need to keep an eye on what's happening so that you don't accidentally lose mail in case one of your recipes is a little off. To assist in the process, procmail comes with powerful logging capabilities, which allow you to see exactly what's happening with your mail messages.

This log is activated via the special "LOGFILE" and "VERBOSE" variables in your ".procmailrc" file, which specify the name of the log and the extent of detail in it, respectively. Consider the following example:

```
LOGFILE=$HOME/procmail.log
VERBOSE = yes
```

You can summarize the contents of this log file using the "mailstat" command, which also ships with the procmail distribution − take a look at the mailstat manual page for information on how to use the procmail logs to build different types of reports.

Procmail typically looks for mailboxes in your home directory. This doesn't usually work for me, since all my mailboxes are in a folder named "mail" under my home directory. If your situation is similar, consider telling procmail to adjust its default mailbox search path via the "MAILDIR" variable.

```
MAILDIR=$HOME/mail
```

Finally, if you find it somewhat unsystematic to keep all your recipes in a single file, you can even split them up into separate files and merge them into your ".procmailrc" file via the "INCLUDERC" variable, as below:

```
INCLUDERC = spam.procmailrc
INCLUDERC = lists.procmailrc
```

If you have a lot of recipes, modularizing them in this manner makes them more manageable, and it also becomes easier to selectively include or exclude them from your ".procmailrc" file.

Closing Time

And that's about all we have time for. In this article, I introduced you to procmail, one of the more powerful mail processors on the *NIX platform, and demonstrated the process of compiling and installing it to your box. I showed you the basics of procmail recipes, explaining how they can be used to filter messages on the basis of specific characteristics, and send them to a mailbox, another email address or an external program. I also explained how procmail's recipes can be used to counter spam, to automatically organize messages from mailing lists into different mailboxes and to create email robots that reformat, reprocess and automatically respond to messages on the fly.

Of course, in all this, I barely scratched the tip of the iceberg − procmail is so versatile that its applications are almost infinite, and listing them all in a single place is well nigh impossible. The following links should provide good starting points, however:

The procmail mailing list, at http://www.rosat.mpe−garching.mpg.de/mailing−lists/procmail/

The procmail FAQ, at http://www.iki.fi/era/procmail/mini−faq.html

Sample recipes, at http://www.math.fu−berlin.de/%7Eguckes/procmail/ and http://www.uwasa.fi/~ts/info/proctips.html

Until next time...happy emailing!

Note: All examples in this article have been tested on Linux/i586 with procmail 3.22. Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or support for the source code described in this article. YMMV!