

Procmail FAQ

This is a FAQ for Procmail, the mail processing utility for Unix.

This Procmail FAQ is an attempt at answering the most often asked questions and straightening out the most frequent misconceptions about Procmail. This is no substitute for the manuals, and indeed presupposes some familiarity with the program's regular documentation.

If you feel you have trouble understanding the tips in this FAQ and/or the manual pages, please be invited to check out the newbie links on the companion link page, which also has a lot of other Procmail-related links for you to investigate, including links to several tutorials which provide a sort of "quick start" documentation.

On the other hand, if you are more comfortable reading crib sheets than real prose documentation, you might want to look at the quick reference (still in beta).

The official URL of this page is <http://www.iki.fi/era/procmail/mini-faq.html> -- this is a "virtual" URL which resolves to a different host. Please use this "virtual" URL rather than whatever your browser thinks it is you are currently looking at. This site can and will move in the relatively near future.

The following mirror sites are available:

United States

<http://mirror.ncsa.uiuc.edu/procmail-faq/>

<http://www.zer0.org/procmail/>

<http://www.moongroup.com/unix/procmail/>

United kingdom

<http://www.dcs.ed.ac.uk/~procmail/faq/>

Holland

<http://www.xs4all.nl/~sister/mirror/procmail/>

Please use one of the mirrors if you can.

The author wanted to call this a "Mini-FAQ" but it keeps getting bigger. There are plans to rename it the "Bronto-FAQ."

As this document has changed and grown (it is currently more than twelve times the size of the arguably more elegant original version 1.0) it has become a bit hard to know where exactly to expect information about some things. I apologize for this. In the Contents, I try to include a mention of all very frequently asked questions, even if they're in a subsection of a subsection (further adding to the bloat, I'm afraid). The below table of contents is an abridged "best of" instead of a full TOC. (This makes little sense on the web page but is currently made to fit into various versions. I'll fix it, someday. I think.)

The author's contact information is at the bottom of this page.

Contents (abridged "best of")

- What is Procmail? -- Description, availability, and installation

- Is there a Procmail for Windows NT?
- How can I run an arbitrary Perl or shell script on all or selected incoming mail?
- And among other things, a pointer to a local copy of Stephen's original FAQ
- How to get answers to questions which this FAQ doesn't cover
- Why am I getting spam from the Procmail list server?
- I subscribed to the Procmail-L mailing list earlier but haven't seen any messages for a while. Did the listserv crash?
- How do I use wildcards in Procmail? Explain file locking, please.
... and other syntax issues, including:
 - How can I forward a message but leave a copy on the server?
 - How can I forward to many addresses?
 - Generally speaking, how do I do more than one action on a message?
 - I get these `procmail: Skipped "(something)" errors ...`
 - I know how to forward a message, but how do I forward a modified message?
 - How can I change the contents of a message but otherwise proceed through my `.procmailrc` as usual?
 - I want to pass some information to a script I run from within Procmail
 - What's a "folder"?
 - What does the second colon in `:0:` mean?
 - What does "Couldn't determine implicit lockfile" mean?
 - What's this `From_` header?
 - How can I do a logical OR of two conditions?
 - Can I list several actions under a common condition? How?
 - How can I test the value of a variable or argument?
 - Efficiency tips
 - Short example recipes sprinkled throughout the text
- Help, I get this error message ... -- Troubleshooting tips
Some highlights:
 - An example debugging `.rc` file
 - Known bugs, common gotchas, and funny quirks
... including, but not limited to:
 - Memory handling with huge messages / on FreeBSD
 - Upgrade to 3.12/3.13 breaks Procmail on systems with group-writable directories (e.g. Red Hat Linux)
 - The regex engine is not `egrep` compatible
 - There is no `^FROM` macro
 - Backslash parsing is sometimes counter-intuitive
 - Always include a `SHELL=` definition
 - Getting the thing to run in the first place
 - What goes in your `.forward` file
 - Yikes! Where did my mail go??
 - Why is Procmail writing to a file called `*`?
 - Why is Procmail writing to the console? Can I stop that?
 - What are these fields that get written to the log?
 - Why does formail fail when looking for duplicates?
 - What's this about "rescued data" from a filtering recipe?
 - Why won't `biff` work right for my own folders?
- How do I ...? -- Running Procmail
 - ... match on the `BCC` header?
 - ... implement a virtual domain? Or, how can I let several local users share the same POP mailbox at the upstream? Or, what is the most frequently made wrong assumption about

- mail delivery?
- ... figure out who to Cc: when there are several recipients?
- ... run Procmail on a file of messages?
- ... do different actions depending on the time of day?
- ... trim down the From: field to just user@host?
- ... know what EXITCODE to use?
- ... prevent my .forward from showing in bounces?
- ... extract (or kill) MIME parts from messages?
- ... write a "vacation" program? An autoresponder?
- Where can I learn more? -- A small links collection
- Appendices
 - Appendix A -- Folder Formats
 - Appendix B -- Figuring Out the Mail Flow

Version information:

\$Id: mini-faq.prep,v 1.306 1999/06/23 15:02:41 era Exp \$

The version history details recent developments.

What is Procmail?

Description, availability, and installation

Procmail is a mail processing utility, which can help you filter your mail; sort incoming mail according to sender, Subject line, length of message, keywords in the message, etc; implement an ftp-by-mail server, and much more.

Procmail is also a complete drop-in replacement for your MDA. (If this doesn't mean anything to you, you don't want to know.)

Procmail runs under Unix. See Infinite Ink's Mail Filtering and Robots page for information about related utilities for various other platforms, and competing Unix programs, too (there aren't that many of either).

You can download Procmail from the main Procmail site or a number of mirrors. The Links page has a listing of well-established mirror sites.

In ancient times, then-current versions were posted to `comp.sources.misc` (vol 43, July 1994, is the newest I could find).

The recommended version of Procmail to install is **3.11pre7**. [The previous stable version was 3.11pre4. Don't use pre5 or pre6. You should also avoid the ancient version 3.10.]

3.12 is in the works. Check out <http://www.procmail.org/procmail-snapshot.tar.gz> for a snapshot of the current development version, and/or Philip Guenther's "to do" list at <http://www.procmail.org/todo.html>

News / Apr 13 1999: Version 3.13.1 is out now!

Please note that some of the tips in this FAQ (or elsewhere) might not work for versions older than 3.11pre4. In particular, a great number of sites seem to be stuck on 3.10, which is a bad choice for a

number of reasons. There should be plenty of incentive to upgrade to 3.11pre7 (or pre4, if you're chicken -- this was the recommended version for a good one and a half years before the issue of pre7. And don't let that "pre" scare you).

The installation procedure is fairly straightforward but probably not the first thing you should attempt after you get a Unix account.

If you feel adventurous, and have a friend with a working copy of Procmail for your type of operating system and hardware, you can just snatch her/his binary. However, you need to be aware that this defeats some checks which the installation program performs, such as determine where your mail spool is, what kinds of file locking should be employed, etc. Be particularly wary if you use NFS-mounted mail spool directories.

The distribution comes with a simple FAQ (locally produced HTML version -- the text-only original is also available) which covers some issues faced when first getting acquainted with Procmail, such as how to view the manual pages, but it primarily addresses various installation problems. There's also answers to some very frequently asked questions, some of which are not dealt with in the document you're reading now. Please look at least at the TOC of the "original" FAQ as well.

(An abridged table of contents of the "original" Procmail FAQ is in the "Quick Questions" section below. Various files in the Procmail distribution package also contain nuggets of very valuable information, especially for the administrator setting up Procmail for the first time.)

If you can't find the answer to your question in either of these FAQs, please take a look at the Links section towards the end of this document -- but first, make sure you have found all of Procmail's manual pages; there are several.

If you are new to Unix, you should probably read up on regular expressions (grep/sed/awk/perl etc) and a little on mail handling before attempting to tackle the Procmail manual pages.

Related quick questions

Q: Is there a Procmail for Windows NT?

A: No, and it's somewhat unlikely that anybody would undertake a port. Read Bart Schaefer's excellent summary of the problems. Excerpt: "I've seriously looked at porting it to NT, yes. The problems are pretty severe."

Q: How can I run an arbitrary Perl or shell script on all or selected incoming mail?

A: Install Procmail. Read the manual pages (there are several). Thank you.

```
:0
* conditions, if any
| your-script-here
```

The conditions, in their simplest form, are regular expressions to match against the header of each incoming mail message. Correction: Even simpler, you can leave out the condition lines completely if you want to do your action (in this case, run a shell script) unconditionally.

More-complicated conditions can also be exit codes of other shell scripts or programs, or tests against the full body of the message, or against Procmail variables (Procmail's variables are also

exported to the environment of subprocesses, so they are essentially environment variables. There are details about this later in this FAQ.)

Actions can also be to save the message to a folder (appended to a Unix mailbox file, or written to a new file in a directory) or to forward the message to one or more other addresses. Finally, the action can be a nested block of more "recipes," as these condition-action mappings are called in Procmail jargon, to try if the outer condition is met. The `procmailrc(5)` manual page has the full scoop.

Obviously, you are not restricted to Perl or shell scripts. Anything you can run from a Unix command prompt can be run from Procmail, in principle, although running interactive programs doesn't usually make much sense.

Q: I can't read. What's in the "original" Procmail FAQ?

A: Here's an abridged listing.

- How do I go about setting up a mailing list or an archive server?
- I installed procmail but how am I supposed to use it now?
- Various questions about compilations problems
- I compiled OK but I have several lock processes which won't die
- When I send myself a test mail, it bounces
- Where do I look for examples?
- Why should I put my login name in the `.forward` file?
- How do I view the man pages?
- The `From_` line on arriving mail has the wrong time
- I get bounce messages about "Cannot mail directly to programs"
- "User doesn't have a valid shell for mailing to programs."
- Locking problems (with OpenLook, with no `.procmailrc`, etc)

Not every question is a Frequently Asked Question, of course. That's why there is a mailing list, after all.

(Additionally, not all answers in the FAQs are Frequently Wanted and/or Understood. Sigh. ;-)

Q: What if Procmail is already installed by another user on my host?

A: Could be. Ask around. Yes, one installation per site should suffice.

Q: How do I know I have found all the manual pages?

A: You read them; most of them contain useful information. Start with the *procmail(1)* manual page; it contains pointers to the others under the "SEE ALSO" heading.

Q: How do I know which version of Procmail I have?

A: You examine the output of the following command:

```
$ procmail -v
procmail v3.11pre4 1995/10/29 written and created by Stephen R. van den Berg
                                     <srb@cuci.nl>
```

Submit questions/answers to the procmail-related mailinglist by sending to:
<procmail@informatik.rwth-aachen.de>

And of course, subscription and information requests for this list to:
<procmail-request@informatik.rwth-aachen.de>

```
Locking strategies:      dotlocking, fcntl()  
Default rcfile:        $HOME/.procmailrc  
System mailbox:       /var/spool/mail/$LOGNAME
```

For Formail and Lockfile, the companion utilities, it's harder to say as older versions (before 3.12) won't tell you themselves.

Q: Please please tell me the address of the Procmail mailing list!

A: See answer to previous question.

There is also a competing / complementary / whatever list. Rhett 'Jonzy' Jones writes:

I have just created two new mailing lists, which are Procmail/SmartList supported, along with my modifications to prevent UCE/SPAM. These mailing lists are called:

```
smartlist@lists.utah.edu  
procmail@lists.utah.edu
```

To subscribe to these mailing lists send the Subject: subscribe to:

```
smartlist-request@lists.utah.edu  
procmail-request@lists.utah.edu
```

NOTE - You will not be able to post to these mailing until your address has been verified. Furthermore, you will NOT be able to subscribe to either one of these mailing lists if your domain is one of the 2850+ domain which I deny both subscription and postings from.

There is now also a `procmail-dev` list for those interested in developing Procmail, and similarly a `smartlist-dev` for SmartList development. To join, send a subscribe request to `procmail-dev-request@cuci.nl` and/or `smartlist-dev-request@cuci.nl`, respectively.

Q: I have a question which this FAQ doesn't answer.

A: Ask on one of the Procmail mailing lists (see previous questions), or try a newsgroup such as `comp.mail.misc`. The links page has links to various resources, including tutorials and examples.

Please don't assume that the FAQ author is interested in inquiries about free consulting services. I read the Procmail mailing list and answer questions when I can. Several others do that too, so you stand a better chance of getting a decent answer by mailing the list rather than me. Thanks.

Before you send stuff to a mailing list or a newsgroup, you need to know the basics of "netiquette," i.e. how to behave online. In particular:

- Be polite and to the point. Don't waste the time of others. Do your homework before you start asking questions.
- *Don't* send subscribe/unsubscribe messages to mailing lists. (Come to think of it, don't send'em to newsgroups, either. :-) Lists usually have a separate "administrative" address which is different from the address where you'd submit discussion messages. Keep the subscription instructions around, so you know how you got on the list in the first place --

usually getting off the list involves a very similar procedure.

- Procmail-specific instructions:
 - Don't ask questions which are answered in the documentation or in the FAQs. In addition to the time of others, you will be wasting your own, because most of the people who read your message will silently ignore you (or in bad cases mutter to themselves and killfile you for all eternity. Procmail users are particularly well equipped for this).
 - Be specific. Include all information pertinent to your question. Use the Subject header to label your message for the reader (not for yourself). Subjects like "Procmail," "Help," or "Problem" will be the first to be skipped by those who don't have the time to read all messages all the time. Similarly, in the message itself, you should describe your problem in concrete terms. Often a "model message" (trimmed down to a bare minimum; it's probably a good idea to indent it with a space or two and keep too much headers rather than too little) and a description of how you would like Procmail to react on it -- step by step -- can be an efficient way to get your message across. (Don't forget that not all of us are mind readers, either; if "framotz" in the input message magically becomes "sqiggly" in the output, you should probably say something about the mechanics behind this transformation.)
 - Things to try when you have a problem:
 - Did you look at the manual pages? There are several.
 - Did you run Procmail with logging enabled?
 - Did you examine the log?
 - Did you try running with a VERBOSE log?
 - Can you reproduce the problem?

If you did try all of the above, please mention it in your message. At the very least, it tells people you did do your homework. By the way, the debugging section below has some suggestions for additional things to try before you post.

There are various netiquette pages all over the net. Try e.g. <http://www.cs.ubc.ca/spider/edmonds/usenet/ml-etiquette.html> or do your own search.

An additional tip: You should perhaps familiarize yourself with the FAQs for related Usenet newsgroups. Apart from various sections of the `comp.mail` hierarchy, various Unix newsgroups -- `comp.unix.shell` in particular -- should be worth a visit.

Q: Why am I getting spam from the Procmail list server?

A: The list is presently open for posting by anyone, including the spammers. This may have to be changed because of the rising tide of spam, which would be unfortunate, because many people might not want to be bothered to subscribe to the list in order to just ask one question. Anyhow, stay tuned (and tune your filters). (Actually the list's setup was recently changed to filter out the most obvious spam.)

Q: I subscribed to the list at Aachen earlier but haven't seen any messages for a while. Did the listserv crash?

A: Try subscribing yourself anew and see if things start to change. SmartList is rather paranoid and will easily unsubscribe you if it gets bounces from your address.

By the way, the on-line mailing list archive at <http://www.rosat.mpe-garching.mpg.de/mailling-lists/procmail/> should have all recent messages on store. You can check there whether you have received the newest messages.

How do I use wildcards in Procmail? Explain file locking, please.

... and other syntax issues

Procmail uses regular expression syntax, which is more complicated but also vastly more versatile than the "glob" wildcards used in many shells for matching file names (where `pr*mail` would match anything which starts with "pr" and ends with "mail").

Briefly, the regular expression to match anything that begins with "pr" and ends with "mail" is `^pr.*mail$` but for practical purposes, e.g. in a Subject: line, you might want to try the following recipe:

```
:0:
* ^Subject: pr.*mail
procmail-mail
```

This says: Anything that begins with "Subject: pr" and contains the string "mail" (followed by anything, or nothing) somewhere after that is to be saved in the folder `procmail-mail`.

Note that there is no need for a trailing `.*` wildcard (and certainly not a sole `*`) -- the recipe will match regardless.

The neat stuff starts when you want to ignore "pro-mail" and "ProMail" while still looking for anything else that begins with "pro" and also contains "mail", but this example ends here.

Any beginners' book about Unix will contain a more detailed tutorial on regular expressions, most likely in conjunction with examples using the `grep` and `sed` programs. For reference, the manual page for `egrep` (an extended `grep`) at your site should contain a concise listing of regular expression operators. (Note that the manual's assertion that Procmail is 100% `egrep` compatible is not strictly speaking true. For one thing, there are many `egrep` implementations.)

One frequently seen Procmailism is this:

```
[      ]*
```

The brackets contain one space and one tab, in any order. This regular expression will catch any sequence of horizontal whitespace (including none at all -- change the `*` to a `+` if you want to make sure there's at least one whitespace character).

Some mailer programs use tabs instead of spaces in some places, others will attempt to "pad" all header fields with spaces to make the headers somewhat more readable. Finally, Subject headers are written by mere humans who sometimes press the space bar twice, perhaps only because they want to see if that will break your autoresponder script. Therefore, your Procmail recipes frequently need to match arbitrary runs of whitespace characters if you want them to work in all situations.

File locking

Under Unix and other multitasking operating systems, several processes can be running at once.

This means several mail messages can be in delivery at the same time. Without Procmail, the default system mailer hopefully handles this just fine when mail is delivered to a system mailbox, but if two Procmails are delivering two messages to the same file more or less at the same time, you end up with problems. (A typical symptom is that the first message gets a `FFrom` in its first line, and the second gets a `rom` without the `F`. This messes up the mailbox format thoroughly.)

Rule of thumb: Use file locking when delivering to a file. Don't use file locking when delivering to `/dev/null` (because then it doesn't matter if the message gets mangled, and you might not have the permission to acquire a lock on a device), forwarding to another address, or piping into a program. A pipeline which ends up appending to a file should still use a lock, of course, since there is the same race condition as when delivering straight to a file.

Rule of index finger: Using an unnecessary extra lock seldom hurts. (When it does *hurt*, you'll notice. :^)

Examples:

```
:0:    # Deliver to a file, let Procmail figure out how to lock it
* ^From scooby
scooby

:0     # Forwarding; no locking required
* ^TO_dogbert
! bofh@dilbert.com

:0:snoopy.lock # Explicitly name a file to use as a lock
* ^Subject:.*snoopy
| $HOME/bin/woodstock-enhancer.pl >>snoopy.mbox
```

The last one might need a little elaboration. When mail with a Subject: line containing the word "snoopy" anywhere in it arrives, Procmail will create a lock file called `snoopy.lock` and hold on to that file while executing the recipe (in this case, piping the mail to a program called `woodstock-enhancer.pl` and appending the program's output to the file `snoopy.mbox`). If a second message matches the same recipe, the second instance of Procmail which attempts to deliver that message will politely wait until the first Procmail lets go of the lock file.

(You don't really need to name the lock file in this isolated case. Procmail would automatically deduce a lock file based on the name of the file you're appending to, `snoopy.mbox`. But you might have several recipes which do separate things which need to be serialized somehow, so that one doesn't occur before the previous one has finished its critical section. Say, maybe you have a second recipe which modifies the program `woodstock-enhancer.pl` itself, or, more realistically, modifies a separate data file used by that program. You can't have that running while this recipe runs, so they have to share a lock, and you have to tell Procmail the name of the lock file to use.)

A (somewhat less common, but still) newbie mistake is to put the name of the file you want to write to as the name of the lock file to use, too. This effectively prevents Procmail from doing anything at all. The meaning of a lockfile is, "if this file exists, you have to wait."

The manual really does explain most of this, although it's not very explicit about why you would or wouldn't want to use locking in the first place.

A typical newbie problem is placing a redundant but harmless lock on a forwarding recipe. Here's an example:

```
:0:
* ^TO_johnny@lunatix\.com
! jjasmith@ppp.home.in.isp.net
```

This will forward to the address `jjasmith` any mail addressed to `johnny`. The trailing colon says to use a lock file, but Procmail can't figure out what file you want to lock because there is no file involved anywhere. Indeed, locking doesn't make any sense here, and you'll get warning messages in the log stating that Procmail "couldn't determine implicit lockfile." Remove the second colon on the first line and all will be fine.

(If you think you want to use a variant of this recipe, don't. Read the BCC question below.)

Related quick questions

Q: "Regular expressions?" What's that? Wouldn't it be much simpler to just use glob-like patterns? Does this mean I have to learn something new again? Augh.

A: Yes. Don't bother with Procmail if you want something simple and stupid. (But getting started really doesn't take that much, all things counted.)

Q: How can I forward a message but still leave a copy on the server?

A: Rephrase: Forward a copy, then proceed with normal delivery (which ends up saving to your normal inbox on the server, unless other subsequent Procmail recipes in your `.procmailrc` are triggered, and end up delivering the message elsewhere).

```
:0c                                # That's colon, zero, lowercase cee
! self@other-place.net             # That's exclamation mark, address to forward to
```

To do this conditionally, add conditions. To not save a copy on the server, omit the `c` from `:0c`

Q: How can I forward to many addresses?

A: In the simple case, just add more addresses to the action line:

```
:0
* ^Subject: result from cgi-bin/www-feedback$
! first@one.com second@two.net third@three.org
```

If you have a largish list of recipients, you might prefer to store the addresses in an external file you can edit without mucking with your Procmail filters:

```
# The file $MAILDIR/addresses.txt contains the recipients, one per line
:0
* ^Subject: result from cgi-bin/www-feedback$
! `cat addresses.txt`
# ^ Make sure those ^ are backticks, BTW (ASCII 96)
```

This will break as soon as the output of the backticks is too large for your shell to handle. You can use more trickery to get around that, but why not resort to a ready-made solution instead? A mailing list manager will scale to thousands of recipients and have some provisions for handling bounces, preventing mail loops, and automatically adding and deleting subscribers. (You might want to look at SmartList, which runs on top of Procmail.)

By the way, if mail to one address should always be redirected to one or more other addresses, you should read the "virtual domain" section below. (Don't use Procmail for this. The universe will implode if you do.)

Q: Okay, then I have a different but related question: How do I do more than one action on a message, generally speaking?

A: You "clone" it with the `:c` flag.

```
# First, we forward a copy to Don
:c
! gillette@hedgehogs.com

# Then, we stash it in a folder
:0:
copied-to-don
```

See also the answer to the brace question.

If you have both the `:f` flag and the `:c` flag on the same recipe, you are probably very confused.

Q: I get these `procmail: Skipped "(something)"` messages in the log, what do they mean?

A: They're Procmail's way of saying you have a syntax error in your recipes.

One frequent misunderstanding occurs when you have several actions you would like to do in response to a matched condition. There *must* be one "colon" line for each action, and each action *must* have a colon line. See examples above and below, especially the question whose answer is "use braces".

Q: I know how to forward a message using an `!` action, but that doesn't let me modify the message I forward. Is there a way to do that?

A: Typically, you want to add or change a header. This sounds like `formail`. The only thing that remains then is to actually send it off. You can of course filter first and then send (see next question), but you might as well do both in one fell swoop (unless you also want the modified message in your normal mail stream; again, see the next question for more).

```
:0c
* ^TO_sales@pizzazz\.tm\>
* ! ^X-Loop: sales@mundane\.domain\.net
| formail -k -X "From:" -X "Subject:" \
    -I "To: sales@mundane.domain.net" -X "To:" \
    -I "X-Loop: sales@mundane.domain.net" -X "X-Loop:" \
| $SENDMAIL $SENDMAILFLAGS -t
```

This is almost like a real-world example. It will (*a*) trim down the headers considerably, sparing only the `From:` and `Subject:` of the original. Then (*b*) we add some headers of our own (remember to extract them with `-X` too!), and (*c*) the results are handed to Sendmail. The `-t` option means the `To:` (and `Cc:` etc) lines in the actual message contain the recipient's address.

If `formail` won't do the modifications you want, you are of course to replace it with whatever you fancy. The basic pattern is the same, anyway: pipe to the program which "fixes" the message, then pipe the results to Sendmail.

If the results don't contain suitable headers, or might contain e.g. your own address, you should take care to tell Sendmail explicitly who to send it to, rather than rely on `sendmail -t`.

Q: How can I change the contents of a message but otherwise proceed through my `.procmailrc` as usual?

A: This is what the `:f` flag is for.

```
# Add an "X-Likely-Spam: ugh" header to all messages
# which have an X-UIDL header
:0fhw
* ^X-UIDL:
| formail -I "X-Likely-Spam: ugh"

# Continues here with the slightly modified message ...
```

If you have both the `:f` flag and the `:c` flag on the same recipe, you are probably very confused.

(An X-UIDL is not necessarily a sign that a message is spam. This header is added by some POP programs. If you don't use POP, or your POP doesn't add this header, it's fairly likely to be spam, but you can never be too sure. Somebody who uses POP could have resent a message to you, for instance, X-UIDL header and all.)

Q: I want to pass some information (e.g. the contents of the Subject header) to a script I run from within Procmail. How can I do that?

A: Procmail's variables are exported to the environment of any process you run from within Procmail (with a few exceptions which, typically, you don't need to worry about). (This also means that if the script is yours to modify, you might not even need to pass in anything explicitly -- just make the script read the appropriate environment variable. However, this is usually a bad solution because you don't want your scripts to depend on environment variables too much.)

In the following example, we are grabbing the contents of the `Subject:` header into the variable `SUBJECT`, and then pass that in as the `-s` option of `our-script`.

```
SUBJECT=`formail -zxSubject:`
:0w:
| our-script -s "$SUBJECT" >>output
```

This construct is a good general solution, but in this particular case, when the information the script needs is simply the value of a field, we should avoid the overhead of the external process call (here, `formail`).

There are two ways to accomplish this savings: (a) since the script gets the message on standard input, it would probably be fairly straightforward to modify the script so it gleans the required information directly from the message's headers; or (b) you can use the `MATCH` construct to do exactly the same kind of grab you get with `formail`:

```
:0w:
* ^Subject:[    ]\^[^    ].*
| our-script -s "$MATCH" >>output
```

Do note that the condition, which we use for its `MATCH`-grabbing side effect, still has to match, i.e. the script will only be run on messages with a non-empty Subject header.

The whitespace between the `[]` brackets consists of a space and a tab, as usual.

Q: I'm trying to read the manual but there's one thing I'm not sure of. It keeps talking about "folders" all the time. How is this different from "mbox file," "file in directory," or "directory," or does it actually mean one (or more) of those?

A: Procmal can save messages to various formats, some of which involve individual files in named directories, others are flat files to the end of which Procmal appends new messages, etcetera. Appendix A contains some helpful descriptions of various folder formats.

Q: What does the second colon in `:0:` mean? Should I worry?

A: You just missed it. It tells Procmal to use locking on this recipe. Go back and read the above snippet about file locking. See the next question.

Q: What does "Couldn't determine implicit lockfile" mean?

A: Briefly, that you have `:0:` where you should have either a named lock file or just `:0` (ignoring any possible flags here). See previous question. Hope this helps.

Q: What does "Extraneous locallockfile ignored" mean?

A: See previous question. Hope this helps.

Q: What does "Extraneous filter-flag ignored" mean?

A: Briefly, that you have `:0f` where you should have `:0` (ignoring any other flags and possible lock file you might have). The `f` flag is perhaps a bit poorly explained in the manual, seeing as lots of people are using it where they shouldn't (probably "filtering" sounds too enticing in the manual for a filtering program -- *of course* you want to filter your mail, dear).

Q: Okay, then what is the significance of the number zero on the `:0` line?

A: Nowadays, none really. In old versions of Procmal, you had to tell it how many condition lines your recipe contained (yes, "bletch"). The syntax was later extended and the number zero special-cased to mean, all the following lines which begin with asterisks are condition lines. (The asterisk at the beginning of each condition is obviously also part of this extended syntax.)

Q: What's this `From_` header everyone seems to be talking about? I can't find a header like that on a single one of all the messages I have on store.

A: This is just a conventional way to talk about the line that starts with the five characters "From " at the very beginning of the headers of a mail message. This is a tricky line for several reasons:

1. It's not really a header, but rather added by your MTA at delivery, so it should properly be referred to as a "pseudo-header." The contents of this pseudo-header are the envelope sender and a time stamp. Some programs will accept basically anything that starts with "From ", others will get confused if e.g. the time stamp is missing.
2. It's very easy to mix up with the `From:` header (note the trailing colon on this one). The underscore is used partially to underscore [sic] the fact that we are talking about this colonless pseudo-header, not the `From:` header proper.
3. Many MUA:s will get extremely confused if a line in the body of a message starts with the

word "From". Often, this is escaped by adding a wedge in front of it, something that in and of itself is a source of confusion (you don't really want anything to muck with the body of your message, ordinarily). Unescaped `From_` lines often lead to messages being split where the unescaped `From` was, because the MUA genuinely thinks this marked the start of a new message, and so the recipient sees two messages, the first ending abruptly in the middle of nowhere and the second continuing where the first left off, only some of the actual message text might show up as rather mysterious-looking headers instead.

The format that uses the `From_` line as message separator is commonly referred to as "Berkeley mbox format." Not all systems use this format, but it's pretty much a de facto standard on Unix. A related format uses the `Content-Length:` header instead, but keeps the `From_` line, leading to utter and hopeless confusion as to which is which.

Q: How can I do a logical OR of two or more conditions?

(For those who skipped the manuals, the default is to AND all the conditions on a recipe. The first one to fail makes Procmail skip right to the next recipe.)

A: The usual reason people are asking this is because the following, which is basically the way to do it, feels clumsy or something.

```
:0
* condition1|condition2|condition3|condition4
{ ... do something about it ... }
```

There are situations where you can't do this, such as when one of the conditions is a negated condition. You can try to fool around with de Morgan's laws to get you somewhere where this doesn't play in, or use scoring:

```
:0
* 1^0 condition1|condition2
* 1^0 ! not condition 3
* 1^0 ? /path/to/external-program whose exit code we want to look at
{ ... now do something about that instead ... }
```

Some side effects are different when you resort to scoring; for instance, if you are also using the `\|` operator to grab stuff into `$MATCH`, scoring will generally grab the last matching line, whereas a straightforward regex OR will stop already at the first one. (You can change the 1 in `1^0` to some really big number to prevent this. See the *procmailsc(5)* manual page if you wonder about the significance of these strange numbers.)

Q: Can I list several actions under a common condition? How?

A: Here you go:

```
:0
* common-condition
{
    :0fbw
    * perhaps an additional condition on this one, even
    | sed -e 's/definatly/definitely/g' -e 's/seperate/separate/g'

    :0c:
    * perhaps an additional condition on this one, too
    action1

    :0
```

```
    ! action2
}
```

- Notice how the "action" of the top-level condition is a set of curly braces containing more recipes.
- Notice also how there is no lock on the top-level recipe.
- And no :c flag either (in the general case. Of course you should have a :c if you want Procmail to branch, and do the braces in one branch, and continue basically as if this recipe hadn't matched in another).
- Every recipe in the braces can have any additional flags and conditions, and even another set of braces as their action, recursively recursively. In other words, they're exactly the same as a recipe in the top level .procmailrc.

See also the description of the :c flag above.

Q: How can I test the value of a variable or an argument? Let's say I manage to pass Procmail a user name using the -a switch, how do I look at the value of \$1 in a recipe?

A: Thusly:

```
ARG="$1"
:0
* ARG ?? regex
action
```

You can test against any Procmail regular expression. \$1 is unusual in that you can't test it directly, you have to assign it to another variable before you can actually compare.

You might want to anchor the beginning and end of the match with the ^^ special anchor (so to match "str" but not "string", "Strauss", "Dnestr", or "tapestry", you'd say ^^str^^. Also note the absence of a dollar sign on the variable name. Of course, this is all in the manual, by the way ;^)

Q: I want to strive for elegance and efficiency in my regular expressions, and ultimately achieve Procmail guru status. What typical newbie mistakes should I try to avoid?

A: Good girl/boy. Here's a partial list.

- Procmail normally ignores case so there's no need to use [Ss][Uu][Bb][Jj][Ee][Cc][Tt] to match the word "Subject" in mixed case, for example. If you actually want case significant matching, however, the D flag does that.
- If you want to do something unconditionally, there's no need to put in an empty "*" line or a bogus "^TO_.*". Just leave out the condition line completely.
- Watch out for embedded carriage returns in the .procmailrc file. See the "Gotchas" section for an example.
- Don't use \/ in a regular expression when you really mean just a literal slash.
- More generally speaking: Many people think they can live completely without backslashes in regular expressions. Others insert them everywhere just in case. (We linguists call that "hypercorrectness," but among the great unwashed masses, the word often seems to be interpreted as "super correct." It really means "wrong for a particular reason," more or less.) Ordinarily, a backslash always works to quote the following character in Procmail regular expressions, so to match a dot you'd use \. -- that's a backslash and a dot. A lone dot, of course, will match any one character. (It would be prudent to not count on \{ matching a literal opening brace in all future versions of Procmail. An unescaped opening brace can

rather safely be assumed to work as it always has, matching literally.)

The exceptions which do not necessarily match only themselves when backslash-escaped, or even don't match anything at all, are the `\` special token and the `\<` and `\>` word-boundary operators. (The characters `</>` just match themselves. Repeat: To match a literal foreslash, just type a literal foreslash. Ditto-ditto for less-than and more-than. The only use for double backslash is to match a literal backslash character, except at the beginning of a regex, but you don't want to learn about that yet. Read the backslash part of the gotchas section when you've grown a little bit bigger.)

The `procmailrc` manual page has a (supposedly) complete listing of everything that Procmail does differently from Egrep. And remember, nothing beats testing your assumptions.

- Redundant regular expressions will slow you down.

A trailing `.*` or *(something)?* is always completely redundant.

A leading `^.*` doesn't serve any useful purpose, either.

A `.*` just after `^TO` or `^TO_` is also superfluous and will actually dilute the effect of those macros, because they already contain a wildcard expression at the end which is better bounded and thus both more efficient and more user-friendly (unless you specifically *want* matches to start anywhere, including, for example, in the middle of a word) than just "any character any number of times."

- To reiterate one of the points from the previous paragraphs, many neophytes don't realize that regular expressions don't "look at" their context at all. Yes, the regular expression `"dogs?"` will indeed match anything which contains "dog" or "dogs", but also "dogma" and "endogenous" and any other word containing the three letters d-o-g. And so, the optional `s` at the end is not really doing anything useful there. "dog" will match "dogs" just fine, and the `s` is not preventing matching on words which don't contain an `s`. A better expression might be `\<dogs?\>` -- this will require the match to be a "word" (token) of its own.

- Redundant lock files, while harmless, will also slow you down. If you receive a hundred messages a week and you have a hundred buddies on the same host who have similar `.procmailrcs`, the numbers will add up fast.

A lot of model `.procmailrc` files you find on the Net will contain a `LOCKFILE=some/file` clause which in most circumstances can be seen as redundant. (It basically ensures that you can only receive no more than one message at a time, which is nice if you absolutely need to keep your log file clean, but potentially a large waste of resources as it can create numerous apparently "hanging" Procmail processes waiting for their turn if you receive many mail messages in rapid succession.)

- On a related note, don't change Procmail's built-in ideas of the values for the variables `DEFAULT` or `SENDMAIL` unless you are fairly confident you know what you are doing. That goes doubly so for `ORGMAIL` -- if you're changing it then you're probably going about the problem the wrong way.
- If you declare a variable and want it expanded in a condition line, don't forget the `$` modifier on the condition itself. See below.
- Attempting to lock things you shouldn't lock (such as `/dev/null`, or using your actual inbox as the lock file for itself :-)) can leave Procmail hanging and consuming *lots* of resources. (Attempting to lock `/dev/null` is special-cased in newer versions of Procmail so that shouldn't be dangerous. As for the "elegance" of attempting that ...)
- You frequently see stacked recipes with braces where a single recipe would suffice. If there's only one action and one set of conditions, you shouldn't need braces for any of your simple newbie-level stuff.

```
:0
* condition
{
    :0c
```

```
    ! staff@example.org  
}
```

is just a redundant paraphrase of

```
:0c  
* condition  
! staff@example.org
```

Don't let that `c` flag confuse you, it won't do anything until all conditions have been met. (This could perhaps be more explicitly documented.)

- Elaborate systems where you deliver somewhere else with a `:c` flag (perhaps multiple times) and then eventually kill off the message by delivering it to `/dev/null` can usually be cleaned up one way or another.
- A more general phrasing of the previous tip: Make sure you understand the concept of "deliveredness".

Basically, Procmail will keep going until a recipe with a "delivering" action is successfully executed. Then Procmail considers its job done. The definition of "deliveredness" is explicated in the manuals, but it more or less amounts to having a message saved to a file, forwarded to another address, or written to a pipeline.

A corollary is that if you want Procmail to continue processing a message after this happens, you need to tell it so (usually with an `f` or `c` flag). Another corollary (specifically in reference to the previous paragraph) is that if you "deliver" a message somewhere, it's quite meaningless to then spawn off a clone to deliver another copy to `/dev/null`.

If the abstraction level of the manual pages doesn't match your taste, perhaps you want to try some of the tutorials on the Links page (you could even try the Rocket Science ones if you feel you understand the basics) or the Quick Reference.

Q: I'm having a hard time getting this `\/` and `$MATCH` thing to work for me. I can't seem to get `$MATCH` to contain what I expect it to. Sometimes the whole regular expression fails to match when I include the `\/` token. What gives?

A: This is a bit complicated and basically belongs in the "advanced" section. But here's a pretty good explanation. (Thanks to Philip Guenther.)

Help, I get this error message ...

Troubleshooting tips

The light bulb contains the seeds of its own revolution.

Before you panic, try adding `VERBOSE=yes` to your `.procmailrc`, send yourself a test message, and examine the information about Procmail's actions that are written to the log. This tells you, in excruciating detail, about every decision Procmail makes while processing your `rc` file.

If you have a message which is being processed differently from what you had expected, save it to a file so you can rerun it through Procmail from the command line, or show it to others when you decide you need to ask for help.

Some common error messages are explained briefly above. Go back and look for "couldn't

determine implicit lockfile". (Search for the error you're getting instead if this is not it.)

All of Procmail's own error messages and some plausible explanations are listed on the Procmail manual page. If you haven't before, now is the time to read the manual pages. Note that there are several; `procmailrc(5)` explains the syntax of the `rc` files while `procmailex(5)` contains some good real-world examples. There is also `procmailsc(5)` which discusses the scoring system.

Free tip: You'll avoid a lot of sarcastic replies if you make sure you try this *before* you post to the Procmail mailing list or some newsgroup about your problems.

Another free tip: Here's an alternate `rc` file you can use as a skeleton for interactive experiments. It will

- tell Procmail to use a harmless subdirectory under the `/tmp` directory for all messages you feed it, instead of dumping messages amid your real mail (so you can safely run a bunch of contrived or dangerous tests and not screw up your inbox)
- display all log messages on standard error instead of in a log file
- give you a chance to play with advanced features of Procmail you'd never dare to try in your regular `.procmailrc` without testing them first :-)

Do note that an interactive Procmail gets a different environment and possibly runs on a different machine than the Procmail in your `.forward`, though.

```
TMPDIR=/tmp/$USER
MAILDIR=$TMPDIR/procmail.out
DEFAULT=$MAILDIR/$USER
VERBOSE=yeah
SHELL=/bin/sh

:0
* ? test -d $TMPDIR || mkdir $TMPDIR
* ? test -d $MAILDIR || mkdir $MAILDIR
{
}
:OE
{
    # Bail out if either directory didn't exist and couldn't be created
    EXITCODE=127
    HOST
}

# ... your experimental recipes here
```

With this recipe, you can use Procmail directly from the command line, typing in experimental messages as you go along, or feeding it a message on standard input, like this:

```
procmail experiments.rc <test.mbox
```

(assuming you had saved the above example `rc` file in `experiments.rc` and have a test message for it in the file `test.mbox`).

[Click here to download a copy of `experiments.rc`]

(Incidentally, testing Procmail from the command line like this is much more convenient than sending yourself a test message and waiting for Sendmail to deliver it to you. It also frees you from having to put diagnostic logging in your regular `.procmailrc`, or having to put verbose logging in your regular log file. FWIW, here's an old message from the mailing list with an example of how to

debug a "mystery recipe.") [Nyfb abgr gur zbeavp sbyybj-hc :-]

Known bugs, common gotchas, and funny quirks

This listing is by no means exhaustive. Philip Guenther's todo list and the companion warts list contains some items which are not here, although I do try to keep in synch with Philip generally.

Known bugs

This is mostly for version 3.11pre4 and newer. If you have an older version, you should probably upgrade. The HISTORY file in the distribution package contains a list of fixed bugs in earlier versions (although it's not very detailed).

Not all delivering recipes are logged after all (3.11pre7)

With LOGABSTRACT=all Procmail is supposed to log all deliveries. This is not working correctly in 3.11pre7; expect a fix in the next version.

Procmail's memory handling is poison to some *BSD systems.

This is fixed in 3.12, although stress testing the system with the newest Procmail before putting it into production use probably wouldn't hurt.

This bug would cause trouble on at least FreeBSD systems, and possibly other related *BSD:s as well (AIX, apparently; BSDI?, NetBSD?, OpenBSD??. NeXTStep??. others?). More information about affected platforms would be welcome.

If you're stuck with an older version, try simply changing malloc's options.

(Michael S. Fischer writes:

```
"Changing malloc's options seemed to do the trick for now (ln -s '<'
/etc/malloc.conf)."
```

See the *malloc(3)* man page if you're on FreeBSD and wonder what this means.)

Closure before \ / doesn't work correctly.

See Stan Ryckman's bug report on the Procmail mailing list for details. (See the followups in the same thread, too.)

Philip Guenther writes:

```
Actually, it has nothing particular to do with the '+' operator -- you get the same
effect with the '*' operator. The real problem is that if you have something like "
+.*", the '+' is superfluous from the point of view of what the regex should match,
but it adds to the combinations that procmail tries as it goes. Along the way,
procmail loses track of where the match really started. Yes, it's a mess. The
workaround is to remove the superfluous operator ('+' in this case). That'll also
make it match faster.
```

A fix is expected in the near future, albeit probably not yet in 3.12.

Newline immediately after \ / is left out

This will be fixed in 3.12.

Exceeding LINEBUF can give you core dumps.

I'm not sure on what systems and under what circumstances exactly this happens. You should definitely avoid grabbing huge amounts of data into MATCH and then evaluating it, unless you know that your LINEBUF can accommodate all the data.

A fix is being advertised for version 3.12 (although it will only fix dynamic overflows; an overlong line in your .procmailrc will still be unsafe).

Inode bug in non-MH mail directories

The algorithm for creating the name for the next mail file involves the assumption that existing mail files are named according to their inode numbers. This is generally correct if Procmail has been allowed to name those files, but probably incorrect otherwise (e.g. if the files have been restored from a backup, or moved to a different file system).

TIMEOUT problems

Procmail attempts to kill off spawned shells which exceed the TIMEOUT setting but won't always succeed.

Variable capture clobbers variable's old value

I.e. the following doesn't work as expected:

```
:0
variable=| echo "$variable" | tr A-Z a-z
```

The value of `variable` will be empty by the time the `echo` executes.

formail -r breaks RFC822

Always use `formail -rt` if you don't know what this means. Perhaps you should always use it anyway. Never mind that the documentation says something funny about how the `-t` option implies "trusting" the sender. Without `-t`, you stand a good chance of replying to the envelope sender, which is a violation of the standard. (See next item.)

The precedence of different headers when doing a formail -r or formail -rt is not documented

Here's some documentation.

Upgrading to 3.12/3.13 breaks Procmail on systems with group-writable home directories (e.g. Red Hat Linux)

This is known to the Red Hat folks, and a fixed RPM is apparently in the works. See <http://developer.redhat.com/bugzilla/show%5Fbug.cgi?id=2242>

(Apparently this can also bite when upgrading to Red Hat 6 from an earlier version.)

The general explanation is that Procmail was tweaked to be more paranoid about security issues, and will not accept an `rc` file if it is group-writable. This is a non-issue on systems where the only member of the group will be the user herself; for those systems, there is a `GROUP_PER_USER` switch in `config.h` which needs to be enabled before compiling Procmail. (This was not done for Red Hat's `procmail-3.13-1` RPM but should be fixed in the next. An independently fixed RPM seems to be available at <http://cs.wilpaterson.edu/~scottw/shebang/> but this appears not to be "official".)

The manual says the regex engine is egrep compatible

'Tain't so. This statement is probably a holdover from older times when basically it was, or tried to be.

(Some of the documentation somehow makes some people think stuff is getting passed to the normal external `egrep(1)`, whichever version happens to be installed. This is certainly not the case; Procmail has its own built-in regex engine, which is roughly modeled after `egrep` but still its very own implementation with its own features and drawbacks. Read the rest of this section, and the manual pages.)

The scope of flags is not explicitly documented

This is supposed to mean, some flags affect how conditions are matched (i.e. they basically take effect immediately when Procmail sees them) while others affect how the action is taken (if it's taken; in other words, they don't have any effect at all if the conditions fail).

It would probably be sufficient if the manual pointed out this fact somewhere; the rest should come from that (but if the manual is augmented with a more systematic overview of the flags, this feature should probably be explicitly mentioned for each flag). The help message (which contains a very brief listing of the flags, in case you hadn't noticed) should perhaps be tweaked a bit as well.

Common gotchas

The following things are frequently regarded as unintuitive or even "broken" by many people.

There is no ^FROM macro

Many people incorrectly infer that since there is a special ^TO macro which catches all variants of To:, Cc:, Resent-To:, etcetera, there also ought to be a ^FROM which would catch, say, From:, Sender:, Reply-To:, and so forth. But this is in fact not the case, and Stephen van den Berg has repeatedly stated that he will not add such a special macro. Of course, rolling your own isn't too hard. Just define a variable which contains the stuff you think you want in there, e.g.

```
FROM="^(From[ ]|(Old-|X-)?(Resent-)?(From|Reply-To|Sender):)(.*\<)\>

# Use thusly:
:0
* $ ${FROM}billg@microsoft\.com
| $HOME/bin/throw-virtual-cake
```

This is not completely orthogonal to what ^TO does; you use it at your own risk, of course. (This is provided as an example. Compare to the documentation for ^TO_ and pick the things you like from both.)

(Oh, and by the way, that "mysterious extra dollar sign" is required to actually expand the FROM variable. See below.)

Procmail strips off all whitespace after backslash plus newline

This frequently bites newbies. Most other programs will treat backslash plus newline as a token delimiter. Not Procmail.

```
:0
| formail -AMoo: -IFoo: -r\
  -aBoo:
```

will actually be glued together as

```
:0
| formail -AMoo: -IFoo: -r-aBoo:
```

which will not work as expected. Solution: If you want a space, put it before the line-continuation backslash.

Backslashes at the beginning of a line are parsed differently from other backslashes

Get used to it, or use some workaround. Here's one idea:

```
* Condition which wants to continue \
( ) with leading whitespace
```

You can use something else besides the empty parens. The basic idea is just to avoid a backslash in the first column, which is what you'd ordinarily expect to work to tell Procmail that the following whitespace is to be taken literally. The parens work well as a substitute for a backslash in this particular case.

Backslash as the first character of a condition has the same problem; similar workarounds should do:

```
* VAR ?? ()\string which starts with a literal dollar sign
* VAR ?? [$]string which also starts with a literal dollar sign
```

```
* VAR ?? (\$)third alternative solution
```

The empty parentheses are almost an idiom among veteran Procmail users, but you see all variants in regular use.

It's easy to forget the \$ modifier when interpolating a variable into a condition line

Compare these:

```
bar='quux'

:0
* foo$bar
baz

:0
* $ foo$bar
mumble
```

The first recipe looks for "foo, newline, bar". The second looks for "foo" adjacent to "quux". The basic problem is that the dollar sign is overloaded to mean, on one hand, newline in regular expressions, and on the other, interpolation of environment variables. So in a situation where you need both mechanisms, there has to be a way to tell them apart. (To identify a newline in a condition which has a \$ modifier, (\$) should always be unambiguous.)

Getting a newline into the action line is tricky

Sometimes, you'd want to write a recipe with an action line which contains a newline. Let's say you have a sed script to trim off a pompous .signature and leave a notice about what happened:

```
sed '/^-- $/, $c\
-- \
sed: .signature removed'
```

and you'd like to run this over mail from a particular user. But Procmail won't easily permit you to split the action line over several physical lines. How do you fix that?

One approach is to define the newline as a variable, and then interpolate that variable when you need it:

```
NL="
"
:0bfiw
* ^From:(.*\<)dquayle@aol\.com\>
| sed '/^-- $/, $c\'"$NL"'-- \'"$NL"'sed: .signature removed'
```

(Don't let that quoting scare you. If you look at it closely, you'll notice that those are just several quoted strings glued together in order to pass it all as a single argument to sed. The newlines are in double quotes while all the other stuff is in single quotes. Do pay attention to the first lines which define the NL variable, though -- that's NL equals double quote, newline, double quote.)

In this particular example, and many times in practice, it's probably neater to put the whole sed script in a variable and then simply interpolate that, newlines and all:

```
SEDKLUDE='/^-- $/, $c\
-- \
sed: .signature removed'

:0bfiw
* ^From:(.*\<)dquayle@aol\.com\>
| sed -e "$SEDKLUDE"
```

You can, in principle, use backslashes to escape a newline on the action line, but the number of backslashes you need for that is not easily deduced (see previous notes about Procmail's quirky treatment of backslashes in some situations). It's often easier to just try to think of a workaround.

Procmail doesn't cope well with carriage returns in the .procmailrc file

Some people like to edit their .procmailrc file on a local PC and then upload the file to their shell account with ftp. If you do this in "binary" mode, the line endings will likely consist of DOS carriage return/line feed pairs, rather than just line feeds (the convention on Unix).

Procmail will not ignore these "extra" characters, and the errors that will be logged will look quite odd (like this maybe:

```
"rocmail: skipped "
```

This depends on what you use to view the log). Solution: upload the file again using the correct transfer mode, or modify it in place with a Unix editor, or the following shell script:

```
#!/bin/sh
if ! mv .procmailrc .procmailrc.old ; then
    echo Could not move .procmailrc to .procmailrc.old >&2
    exit 1
fi
tr -d '\015' .procmailrc.old >.procmailrc
```

You should remove .procmailrc.old after making sure the new file works as intended. To discover the problem in the first place, programs like *viz(1)*, *cat(1) -A*, or *od(1)* can be helpful.

There is no way to globally modify how Procmail behaves when there's an error.

When there is an error executing an action, Procmail merrily skips to the next recipe and tries that instead. This frequently means that when the recipe that should have matched didn't, everything gets written to your \$DEFAULT mailbox (or spam-filtered by some particularly ominous spam filter that would otherwise have been bypassed, if you have that sort of stuff near the end of your .procmailrc).

For a single error-prone recipe, you can always put one with an :e flag immediately after it, but this is hardly something you want to do for each and every recipe.

Procmail always requires enough memory to read in the entire message it's processing.

This is rather self-evident, given the way Procmail works, but it has some unfortunate consequences (especially considering the FreeBSD memory issue above). It would be nice if there was a back door of some sort, so you could say e.g. "if a message is bigger than 32 megabytes, just deliver it, okay?" (or maybe look at only the first 32 megabytes of it when reading the recipes).

A fix for this involving mmap() is scheduled for some future release (not 3.12 though).

If your login shell is a C shell (csh or tcsh), prepare for havoc.

Procmail works *very* badly with the C family of shells (you could almost say it doesn't, plain and simple, but the symptoms are murky and hard to put your finger on), at least for some people.

As a precaution, always set SHELL=/bin/sh at the beginning of your .procmailrc (If you are forced to work with tcsh and would like to look further into this, the FAQ author has some ideas for things to check.)

\$@ is only available to external programs

This is actually documented (all right, in the BUGS section of the manual, so perhaps it's not a feature :-)) -- if you need to use \$@ internally, you'll have to go via an external program such as echo:

```
:0ir
ARGS=| echo "$@"
```

Using "\$ARGS" subsequently is still not exactly the same as "\$@" is to the shell.

\$SENDMAILFLAGS gets passed in double-quoted by the ! action

You won't be able to change SENDMAILFLAGS to a string containing whitespace because the whitespace gets passed to Sendmail as part of the argument. You can bypass this by changing all occurrences of ! actions with

```
| $SENDMAIL $SENDMAILFLAGS ...
```

but probably (er, make that hopefully), forthcoming (hopefully forthcoming) versions of Procmail should be fixed and not quote the flags.

Diagnostics in the log file don't tell you which line in the .rc file the error is happening on

Run the problem message through Procmail again with `VERBOSE=yes` and it should become apparent where the problem is.

formail -A or -I don't imply -x

If you use formail's `-A` or `-I` flags in a command which also uses `-x`, you have to explicitly add an `-x` for the fields you just added.

In other words,

```
formail -rt -XTo: -A"X-Loop: me@my.isp"
```

will actually kill the `X-Loop` header immediately after inserting it. You have to extract it explicitly, even though you just added it:

```
formail -rt -XTo: -A"X-Loop: me@my.isp" -XX-Loop:
```

formail has a fixed order for parsing its arguments

This means you sometimes have to feed the output of formail to another instance of formail to achieve the end result you want. (Given this, it's surprising how seldom you need to in practice :-)

Older versions of formail and lockfile won't tell you what their version number is.

This was fixed in 3.12, but if you're stuck with an older version, well, you're stuck. (But you really should upgrade.)

You can look inside the binary with `strings(1)` -- however, you will probably only find a date but no version number. (Dates in April 1997 seem to correspond to the versions that come with version 3.11pre7 of Procmail. [pre5 and pre6 were also issued around April.]

Funny quirks

- The `\|` operator is a bit unwieldy. The way that it changes the longest-match behavior of the regex engine will bite you one day if it hasn't already. See the related question in the previous section.
- There is no general way to limit what gets grabbed into `MATCH` -- either you get everything after the `\|` operator, or nothing at all. It would be nice to be able to specify a trailing context for what you want to match, which would still not get copied into `MATCH`.
- The `$(VAR)` interpolation operator will add a set of empty parentheses at the beginning of the interpolated value.
- No mechanism exists for running a program without feeding it a message on standard input. (You can use the `i` flag to suppress warnings about data not being read, but it's a kind of a kludge.)
- Procmail doesn't have a convenient way to tell what version of Procmail you're running from

inside an `.rc` file. There is a patch for this. It will probably be adopted in slightly modified form for 3.12.

- It's not exactly clear what kind of comsat notice should be generated when delivering to a program. Currently, it's as if the program had been a file. Your comsat will probably be confused all over.
- Some people think the manual needs improvement.
- It is not obvious what the difference between a "quirk" and a "gotcha" is, exactly, nor for that matter always what the difference between a "gotcha" and a "bug" is. If you're a speculative type, speculate.

Related quick questions

Q: What are some typical installation problems?

A: Here are some of the most typical problems with getting Procmail to run in the first place:

- If you seem never to be able to execute Procmail from your `.forward` file, it could be that your site is using `smrsh`, the Sendmail Restricted Shell, for mail delivery. In this case, you should get a bounce message saying something like "Address <foo@bar> is unsafe for mailing to programs" or "Cannot use <metacharacter> in command". Your admin has to explicitly tell `smrsh` that Procmail is a program you are allowed to call up from your `.forward` (if you are -- you might not be, since there are security risks involved; users can basically run any program they wish from within Procmail, thereby invalidating the whole idea with `smrsh`. Look for a compile-time fix in an upcoming version of Procmail, though. The administrator will then be able to restrict which programs the user's Procmail can execute ... and then you might even be able to persuade your admin to install Procmail as your local MDA for this feature alone).
(It could also be that you're running Red Hat or some other system where your home directory is group writable, without the group-writable option compiled into Procmail; see the Red Hat problem in the Bugs and Gotchas section for a discussion.)
- Another "mysterious" condition to watch out for is when mail is actually usually processed on a different host than the one where you installed Procmail. See Appendix B. One of the typical symptoms is that you can send yourself test messages which get processed by Procmail, but mail from the outside world bypasses Procmail completely or doesn't even get delivered. (Mail to the local host from yourself usually gets processed locally, whereas outside mail always comes via the mail server. You can look at `Received:` lines from old messages to find out what routes mail to you usually takes -- the lines closest to the top are the most important. If the mail server is a different architecture, or can't mount your home directory, or whatever, you will have problems. Ask your site admin for help.)
- A good safety measure is to put `SHELL=/bin/sh` right at the top of your `.procmailrc`. There are various things that will not work, but instead either fail rather mysteriously or produce very unexpected results if you try to run Procmail scripts under `tcsh`. Some things to look out for are pipelines which just don't do what you expect them to, or error codes which never get passed back to Procmail.

See also various points in Stephen's original Procmail FAQ.

Q: What are all those mysterious things I'm supposed to stick in my `.forward` file?

A: First off, for the record, you don't need a `.forward` if Procmail is your local MDA. Ask your admin.

If Procmail is not your local MDA, you need something more or less like what the manual suggests. Here's one popular variation:

```
"|IFS=' '&&p=/usr/local/bin/procmail&&test -f $p&&exec $p -f-||exit 75#whatever"
```

(That should all be on a single line. Unwrap it if it appears wrapped. Unwarp it if it appears warped. The surrounding double quotes are significant, and should be included in the file.)

This is just a simple (but slightly convoluted) shell script.

Whitespaces are often left out as much as possible, partly out of paranoia, partly because people want it all to fit onto a single line in their editor.

Please note that this is *not* guaranteed to work out of the box. Your first attempt at a `.forward` should be what your local manual page suggests (and please, people, change `#whatever` to your own login name with a hash mark in front).

Here's what it does:

- The double quotes are required in order to tell Sendmail that this is all just a single directive.
- The pipe character just after the leading quote [-- you can try to put it just before the leading quote if you have problems with getting this variant to work, just make sure you don't let any whitespace slip in --] tells Sendmail to try to execute this and feed the mail message to it on standard input.
- The `&&` "and" connective means, do the next action only if the previous action(s) succeeded.
- The `||` "or" connective means, do the next action if the preceding action(s) failed.
- `IFS=' '` sets the shell's "internal field separator" to a single space. This is mostly just in order to prevent some old Sendmail hacks which were based on setting the IFS to something else.
- The variable `p` is set to where-ever you expect to find your Procmail binary. This allows us to use the convenient shorthand `$p` instead of this longish pathname, and also means that if you ever move it, there is only one place where you need to remember to change it.
- The `test` will fail if the file named in `$p` doesn't exist.
- The `exec` replaces this script we're currently executing with Procmail if the binary is successfully loaded. The parameter `-f-` says to Procmail to regenerate missing `From_` lines and update the timestamp on existing ones.
- The `exit` only happens if either the `test` or the `exec` failed. If this is the case, the exit code number 75 is passed back to Sendmail, which interprets it as a temporary mailer failure and requeues the message (we hope). (See also the *EXITCODE* topic elsewhere in this FAQ.) Note that if the `exec` succeeded, `sendmail` will never reach here. Whatever the `exec` executes replaces whatever was reading the `.forward` file and whatever exit code that returns will be returned instead.
- The `#comment` doesn't do anything, but serves to make sure your `.forward` file is unique, in case your site is running a braindead over-optimizing Sendmail. (See the explanation for this in Stephen's original FAQ.)

The Filtering Mail FAQ has a section with more variants you can try if the one in the manual page and this one don't seem to work.

Q: I have this recipe which I basically copied from someone's working `.procmailrc` but it gives me funny errors such as "Badly placed ()'s" or "formail: not found" or "formail: formail: cannot open". What's happening here?

A: The first error message is a Csh error message (also "Too many ('s)" -- add the following to

your `.procmailrc` and try again:

```
SHELL=/bin/sh
```

The second and third errors comes from a Bourne shell which can't find the command named `formail`. It probably means that your `PATH` is not set up properly, or that a program which the recipe needs doesn't exist on your system. (You might also see `"/bin/sh: formail: cannot open"` which makes a lot more sense than the example above. Don't ask me why some Bourne shells will print the command name twice.)

It could also be because the recipe author used the (somewhat dubious) convention of defining commands as Procmail variables and using these variables in the actual recipes:

```
:0
* ? $FORMAIL -XFrom: -XSender: | $FGREP cyberpromo.com
/dev/null
```

The above will only work if the variables `FORMAIL` and `FGREP` are set to suitable values such as `FORMAIL=/usr/local/bin/formail` and `FGREP=/usr/bin/fgrep` earlier in your `.procmailrc`. (And you can't just snatch these values from here either -- ask your local gurus [or the recipe author] to help you out if you don't understand what this does.)

Q: Yikes! Where did my mail go??

A: Usually, to a file in your `MAILDIR`. Look in the log for clues (you *were* using the logging facility, yes? Yes?) and in your `MAILDIR` for files called something funny such as `"*"` or `":0:"` or `"# I wonder if Procmail allows comments here"` or `"forwardee@address.net"` (the quotes are not part of the file names and these file names are not necessarily representative).

Q: Okay, I looked in the log, and it said it wrote it all to the file `"*"`. Problem is, I don't know why -- I'm sure this recipe was correctly transcribed from the manual page snippet I saw. Here's the relevant `VERBOSE` log excerpt:

```
procmail: Skipped "^Subject:.*test"
procmail: Locking "*.lock"
procmail: Opening "*"
procmail: Unlocking "*.lock"
```

A: Make sure you are using a reasonably new version of Procmail. According to archaeologists, versions prior to 3.something don't understand the `:0` syntax

Q: Why is Procmail writing to the console? Can I stop that?

A: It's not, unless you specifically messed with this setting when you compiled it. Unmess and recompile. Another popular mistake is to somehow enable the MMDF message separator string (`^A^A^A^A`, four ASCII 01:s) before compiling. (This setting is the first thing in `config.h` and is kind of easy to uncomment so it's not all that hard to change it by mistake if you're using, uh, ahem, something else than the One True Text Editor.)

Q: What are these fields that get written to the log?

A: In order of appearance

- the `From_` line of the original (which conveniently includes a fresh date stamp, which you can use for various things such as determining the approximate time of day without calling `date(1)` for each incoming message);
- The `Subject`;;
- the delivery point -- either a file, in your `MAILDIR` if it doesn't have an explicit path (see previous question, by the way), or a shell script pipeline, which will be your local Sendmail or imitation in the case of a forwarding recipe;
- and the size of the message in bytes, on the same line.

This is documented under `LOGABSTRACT` in the `procmailrc` manual page.

Q: I copied the recipe from the `procmailex` manual page for checking for duplicates, but now I get a lot of log entries saying this invocation of `formail` is failing. What's up?

A: It's supposed to work that way. What the recipe does is, if `formail` detects a duplicate, it will *succeed*, which makes Procmail think the message is now delivered.

Q: Uh, I'm not sure I understand that.

A: It's a bit of a play with side effects. For reference, here's that recipe again:

```
:0Wh:msgid.lock
| formail -D 8192 msgid.cache
```

Let's paraphrase this recipe as a dialog. There are no conditions, so the action line is done for every message that gets as far as this recipe. The `h` flag says to only show the headers of the message to the action line, and the `w` flag says to wait for the program in the action line to finish (so we can look at Formail's exit code).

Procmail says, "hey `formail`, save this message for me."

Formail eats the message (actually just the headers, because of the `h` flag), extracts the Message-Id, and checks if it's already in the cache file. If not, `formail` says "sorry, no can do." Procmail catches this error, says "aw shucks, you couldn't save it? I have to continue processing this message then" and wanders off to the next recipe.

However, if the Message-Id was already in the cache, Formail doesn't generate an error, and Procmail (somewhat stupidly, it might seem) assumes Formail took care of that message (i.e. saved it somewhere, for example) and that is is thus now delivered and no longer a problem of Procmail's.

Q: I have a filter which works fine most of the time, but sometimes it will fail and I get a log message that Procmail has "rescued" lost data. Huh? Why is that?

A: It's a feature. Normally, you want Procmail to keep track of what your recipes are doing, and salvage your data if something goes wrong with one of them. If you have a recipe like this:

```
# Old Stan always writes boring messages,
# we only really need to see the first five lines of'em
:0fbw
* ^From: stanislaus@poland\.com
| head -5
```

it will work some of the time, when the message from Stan is short enough, but that's a coincidence. With a longer message, though, Unix starts paying attention to what is happening, because it will have to buffer some of the data, and then when the buffered data is never read, an error occurs. The error is passed back to Procmail, and Procmail tries to be nice and give you back

your original message as it was before this malicious program truncated it. Never mind that in this case you *wanted* to truncate the data. Anyway, the fix is easy: Just add an `:i` flag to the recipe (`:0fbwi` instead of `:0fbw`) to make Procmail ignore the error.

Q: `biff` works just fine for notifying me of mail in my system mailbox, but I don't seem to get notifications for mail delivered to mailboxes in my home directory, or it keeps telling me I have mail in the system mailbox when in fact it was delivered to a different mailbox. How can I fix that?

A: Procmail by default sends a notification message for all delivered messages. The `COMSAT` pseudovvariable controls this; see the documentation for details. So for starters, make sure you have `COMSAT` set to the right thing -- in a verbose log, you should see something like

```
procmail: Notified comsat: "you@offset:file"
```

for each delivered message.

Unfortunately, many systems come with an ancient version of `comsat(8)` which doesn't pay attention to the last part of this notification. Thus, you will see apparently random `biff` notices, or none at all for mail delivered outside your system spool.

The solution is to [get your admin to] install a `comsat` daemon which understands the newer, extended form of the `comsat` protocol.

How do I ...?

Running Procmail

This section attempts to explain a little something about the context in which Procmail runs. Specifically, you need to understand this if you wonder why you can't match on the BCC header, how to cope with many users sharing the same POP mailbox, or generally want to do something like a virtual domain.

If that's not something you worry about, you should perhaps still read the next section. But if you want to skip directly to the other questions, you can [click here](#).

Why Headers Don't Matter

Learn the following by heart:

SMTP doesn't care what's in the headers of a message. It goes where the envelope says. The headers don't matter. The headers **don't matter**. The message goes where the envelope says.

What, the astute reader asks, does this have with Procmail to do?

Well, Procmail is frequently used to handle mail. And Procmail can only look at what's in a message's headers. And since the headers might just contain more or less anything, trying to let Procmail figure out whom a message for is pretty futile.

Nevertheless, this is something a lot of people are trying to do.

If you are content with the answer "Procmail is the wrong tool for this," you can stop reading now and skip to the next question.

Let's take an example. The following message looks like it was sent from `foo@bar.net` to `nospam-list@nospam.org`. And yet, it was actually sent by some mailing list package at `nospam.org` and landed in your inbox, because you're a subscriber to `nospam-list@nospam.org`. How can this be?

```
From nospam-list-request@nospam.org Sun Nov 15 18:39:39 1998
Received: by yourhost from nospam.org and so on and so forth
From: Frank O. Olsen <foo@bar.net>
To: People Who Don't Want Unwanted Mail <nospam-list@nospam.org>
Subject: How?
```

```
How do I get off this list?
```

```
--
Frank
< ... 18 lines of .signature trimmed ... >
```

When a message is transferred over an SMTP connection, it's encapsulated. It's kind of like the message is nicely folded into an envelope, and the programs which handle it just look at whatever is scribbled onto the envelope. Then when the message reaches the MTA at the final destination, this program (typically Sendmail or Qmail) opens the envelope and delivers the actual message.

What happens when the Sendmail at `nospam.org` sends the above message is that it passes in the `-request` address as the envelope sender (which turns up in the `From_` pseudo-header) and your address as the envelope recipient. Fine and dandy, says your local MTA, and asks `nospam.org` to follow up with the actual message. Only at this stage are the headers (and the body, of course) transmitted -- obviously, what's in the message itself doesn't matter anymore, because the envelope data is what says where the message came from and where it should go.

The `Bcc:` header, as implemented in e.g. Sendmail, is really just a special case of this: Before sending the message, Sendmail copies any `Bcc:` recipients from the headers to the envelope, zaps the `Bcc:` headers, and sends it off just like any other message.

Normally this doesn't affect Procmail a lot, because in the typical scenario, Procmail already knows implicitly who a message is for -- the envelope sender data determines whose `.forward` (if applicable) and/or `.procmailrc` gets read, and all of this is nicely sorted out by the time your Procmail recipes get a chance to do anything like deliver the message somewhere.

But what if you want to use your account as a switchboard for many other addresses? Procmail is a fairly nice program for this sort of application, but if you are feeding mail for several distinct recipients into your switchboard `.procmailrc`, you have to make sure you also somehow tell Procmail where the message should ultimately go.

Here are some ways to accomplish that:

- The actual headers or the body contents determine where the message should go, and you don't really care about any of the envelope information. If you merely want to forward everything with "Precedence: junk" to somewhere else, this is trivial to do and precisely what Procmail was made for.
- Make sure the MTA passes in the relevant information on Procmail's command line when it

starts up. This is rather simple if you have control over the MTA, but virtually impossible if Procmail is invoked via your `.forward` file.

- Make sure the relevant information is in the headers. It is technically fairly easy in most MTA:s to add something like `X-Envelope-To:` or `Apparently-To:` to the headers before passing it on to Procmail for delivery.
- Be unsure of whether the information is there, and try to wing it anyhow. This is a surprisingly popular option, but it's by no means trivial to get it working acceptably, and factually impossible to make completely watertight. Some cases you should think about now rather than later include:
 - Are you sure there isn't a better tool for what you are trying to do? Check out Sendmail's `virtusertable` and related constructs, for example. For simple applications, even plain old aliases work superbly. For POP, Fetchmail is a popular solution which relies on some heuristics when it can't get the envelope information. You can try to emulate these heuristics in Procmail, but it's definitely not easy (and why not just use fetchmail then. Check out its "multidrop" facility).
 - How do you cope when more than one user at your site is Cc:ed the same message?
 - How do you cope with mailing lists? They typically use something technically similar to BCC (see above example).

Some of the "related questions" below discuss some of these issues to some extent.

The default settings with Sendmail 8.9.x for Procmail as LDA don't actually pass in the actual envelope recipient, but at least with plussed addresses you get the part after the plus (as in, if I get mail to `era+pr@iki.fi` I'd have access to the information that it was sent to something +pr, but I don't know if it was sent specifically to `era` or to an alias which points to this address).

With Sendmail, using the Procmail mailer (not just Procmail as the default delivery agent) is a solution which works nicely (but you can't invoke that from plain aliases, you have to use something like the `mailertable`). (Thanks to Philip Guenther for information about how to do this with Sendmail.)

With qmail, you can have the MTA insert envelope information into the headers which subsequently get stripped by the local delivery agent. This is nice for POP sites where both the upstream and the local host run qmail. (Thanks to Bart Schaefer for this information.)

Related quick questions

Q: Why can't I match on the `BCC:` header?

A: You can't match on the `BCC:` header because it is not present in the message you receive. See above. This is what the "Blind" in "Blind Carbon Copy" stands for; *none* of the recipients are supposed to see who's been `BCC:ed`. (Okay, so the spec has a few more twists. This is how Sendmail does it. Some other mailers will show the BCC list to the people who are being `BCC:ed`.)

Q: The `Received:` header seems to often contain something like `Received: from elsewhere (...) by somewhere (...) for <recipient@host.domain>` even if the recipient was `Bcc:ed` -- can't I rely on that?

A: No. Next question?

The problem is that (a) not all MTA:s add this information, and (b) even if yours sometimes does, it might not always. E.g. Sendmail won't stamp the recipient in the `Received:` header if the message

was sent to several recipients at your site.

Q: How do I implement a virtual domain in Procmail?

Or, is Procmail even the right tool for this?

A: The short answer is no; don't. See the section Why Headers Don't Matter, above.

If you stubbornly refuse to take no for an answer, you might want to look at Chris Lindsey's "solution"

Q: How can Procmail figure out who to Cc: when there are several recipients?

A: If you're asking this, you should probably be reading the answer to the previous question first. The rest of this is just a demonstration of how futile this is.

It has been asked a number of times if it's possible to implement a good forwarding scheme which can take care of things like Cc:s to different users in the same virtual domain.

Here's what you might have started out with:

```
:0
* ^TO_jill@mydomain\.virtual
! jill@real.address

:0
* ^TO_jane@mydomain\.virtual
! jane@somewhere.else
```

The problem now is that if something comes in `To:` both `jane` and `jill`, (with infinite variations of `Cc:s`, `Resent-To:`, etc) it will only be forwarded to `jill`. (Or, if you have a `:c` flag on each recipe, the recipients will each get one copy per local recipient of the same message. Or, if your upstream already exploded the message into one copy per recipient at your site, `jill` will get both her own copy and `jane's`, or both will get two copies. Or, you mess up while trying to fix this and your mail server goes into a headspin and you wake up the next morning in Mongolia.)

The short answer is that Procmail is not the right tool for this.

If you really want this to work, you will also want `BCC:s` to work (see the `BCC` question above), which is usually not possible without tweaking your Sendmail configuration. In which case you might as well implement some or all of the actual forwarding mechanism in Sendmail, which is the more appropriate tool anyway.

If you are set on getting this to work anyway, see the "Kevorkian" answer below.

The basic problem is that when mail gets delivered into mailboxes at the upstream, the envelope information (like who the message is for) is usually discarded. So you shouldn't *deliver* into a mailbox that is not the correct mailbox, you should continue to *transport* the mail until you are ready to deliver to the final destination.

You can tweak Sendmail to copy the envelope recipient information into the message header (see below), but you don't have to do that if you don't deliver the message prematurely.

In a typical my-home-computer-is-my-domain setting, you could instead transport mail to your home computer using UUCP. `fetchmail` can also handle this for you, allowing you to have all mail for your domain delivered to a single POP account with the envelope information intact. (Look for "multidrop" in the `fetchmail` documentation. It appears that it actually relies more or less on the headers, not on the actual envelope recipient data.)

Q: Procmail is broken in this respect, isn't it?

A: No. If every problem begins to look like a nail, you need to have more tools than just the hammer in your toolbox.

Search backward for "stubborn" if you really really want to do this anyway.

Q: But I don't want to bug my upstream every time I create a new local address.

A: You bug your upstream to set this up correctly for you, once. (If they insist on having a list of your local users available before they can deliver your mail correctly, it means they *want* to be bugged. And you are a paying customer, correct? Anyway, having a list of users at the upstream makes sense for some things; if they want one, you should agree on a procedure for updating this list which is as painless as possible for both parties. `man rdist` and all that.)

Q: Yada yada yada. So how do I set up this under Sendmail so that Procmail does get hold of the envelope recipient information?

A: Miquel van Smoorenburg has written a patch for this which can be found at <ftp://ftp.cistron.nl/pub/people/miquels/sendmail/8.9.2-envto+etrnprog.dif> -- allegedly this, or something like it, is being considered for distribution with Sendmail 8.10.

See also <http://www.rosat.mpe-garching.mpg.de/mailling-lists/procmail/1996-11/msg00251.html> which is a tangentially relevant message about this from Philip Guenther. (Or look at this local copy.)

Also check out the Sendmail FAQ, which discusses this at some length. And please visit <http://www.sendmail.org/> if you haven't already.

Q: But I *want* to shoot myself in the foot. Won't you help me?

A: Download a copy of `kevorkian.pl` (never mind the Simpsons references). See the Cc: to many question above for details. (If you just want to look at the script, but your browser wants to download it and/or execute it, try the copy named `kevorkian.txt` instead.)

Unrelated quick questions

This is the kitchen sink.

Q: I have a recipe which I would like to run on many messages which are already in one big happy mailbox file. How can I tell Procmail to not treat them all as a single long message?

A: You don't, you use `formail` to split it it back into separate messages and feed each into its own little procmail process.

```
formail -ns procmail experiments.rc < test.mbox
```

(This will use the recipes in the rc file `experiments.rc`.)

Be careful with the `-n` switch if you have a large amount of messages to process.

Q: How do I do different actions depending on the time of day, day of week, etc?

A: The "brute force" answer is to run `date(1)` each time you receive a message, and compare the results against some regular expression, but a simple and efficient shortcut is to use the date stamp which is already in the `From_` line -- it will be in your (machine's) local time zone and reasonably close (usually within seconds or even split-seconds) of the message's arrival.

Here's an example which forwards messages elsewhere, but only on weekdays between 9 and 5. (Actually, until 16:59:59.)

```
:0
* ^From [^      ]+[^      ]+(Mon|Tue|Wed|Thu|Fri) ... .. \
  ([ 0]9|1[1-6]:[0-5][0-9]:[0-5][0-9])
! frankz@golden.net
```

If you want to save to a folder with today's date, you can similarly extract the time out of this time stamp and use that as part of the folder name:

```
:0
* ^From [^      ]+[^      ]+... \/. . . . . [+]? . . . .
{
  FROM_=$MATCH

  :0
  * FROM_ ?? ^^\/[a-z][a-z][a-z]
  { MONTH=$MATCH }

  # Exercise: Make sure month name is always properly capitalized,
  # and/or convert it to a month number

  :0
  * FROM_ ?? ^^\. . . \/([ 0][0-9]|1[0-2])
  { DAY=$MATCH }

  :0 # Y10K problem! :-)
  * FROM_ ?? ()\/[1-9][0-9][0-9][0-9]$$
  { YEAR=$MATCH }

  # Should probably bail out here if any of MONTH DAY YEAR unset

  :0: # save to daily-yyyy-mmm-dd
  daily-$YEAR-$MONTH-$DAY
}
```

The mailing list archives have numerous variations on this theme.

Q: I can extract the sender's email address with `formail -zxFrom:` but this still often results in something like `Donaldus Anas <don@see.va>` or `don@see.va (Donaldus Anas)`. How can I trim this down to just `don@see.va`?

A: The canonical way to get the sender's trimmed-down address is

```
formail -rtzxTo:
```

This looks a bit counter-intuitive, but it works fine, provided that you don't mind the fact that `formail -rt` will prefer e.g. the value of `Reply-To: Over From:`.

This will in fact do two things, which is not apparent if you're not too familiar with `formail`. One: It will generate a reply header (`formail -rt`). Two: Out of this new header, it will extract the `To:` field (`formail -zxTo:`). By happy coincidence, the generated reply address will only contain the actual email terminus, without the full name or other comments. Wallah.

None of this is important if you merely want to find an address to send off a reply to (all you need for that is `formail -rt` and pass it to `Sendmail`). An address with a full name will do just fine for replying; any decent mail program should accept that (although you probably need to put it in double quotes to pass it as a single argument). For keeping a database of addresses, though, the stripped-down version is better (but still not perfect, and keep in mind that the `host.domain` part is not case sensitive).

Typical newbie mistake: `formail -rtzxFrom:`; this will actually return nothing at all, but you could in fact expect it to return *your own* address.

If this approach doesn't suit you for some reason, you could always post-process the address through some simple script which successively strips off layers of comments and whitespace. A fully RFC822-compliant script isn't trivial to write, but for many purposes, a simple `sed` script will probably do. (If you're truly masochistic, try to figure out how to get `sendmail` to strip the address and hand it back to you :-)

Perhaps you also want to look at `formail -rD` which keeps record of a limited number of addresses and expires the oldest ones as it gets fed new addresses.

Q: I've seen all these recipes which use `EXITCODE=` but how can I tell what exit codes to use?

A: The file `/usr/include/sys/exits.h` should contain a good listing. If not, ask your system administrator where to get this information. These exit codes are fairly standardized but you should still not be surprised if something turns out to be different at your site. Some things are also probably more or less `Sendmail`-centric.

If you use `Qmail` instead of `Sendmail`, the codes will be completely different. See the `Qmail` documentation.

(The mechanism when you use this from `Procmail` is that `Procmail` quits with the specified error code, and the calling program is expected to catch that and act accordingly, i.e. generate a bounce message in most situations. If you are calling `Procmail` from a program which doesn't do this, there will obviously be no bounce. So if you test `Procmail` from your shell prompt and you see `**Bounced**` or `**Requeued**` in the log, it only really means `Procmail` quit with a non-zero exit code.)

If your `$SENDMAIL` is in fact not a real `Sendmail`, it probably doesn't care about `Procmail`'s exit code. At least `Smail` is rumored to behave this way. Bottom line, as always: test your recipes.

Not all programs return sensible error codes. If you see "mailer died with error 12 and a half" in a bounce message, it probably means the recipient ran his own home-grown filter on your message and it just happened to die with a more or less random and nonstandard exit code. `Procmail`, of

course, is an excellent wrapper to put around such misbehaving programs.

Q: How can I prevent my `.forward` from being included in `sendmail`'s bounce messages?

I am rejecting some unwanted messages by setting `EXITCODE` to a suitable value (see separate question about that) but the generated bounce will include my `.forward` file, which will reveal to a human reader that my account in fact does exist, and it won't really allow me to explain the reason for the bounce, either. Can I make the bounce include a better explanation to the receiver of the bounce, or otherwise customize it?

A: If your site is using Procmail as your local MDA, you shouldn't have this problem. Perhaps you can persuade your admin to install Procmail site-wide, in which case you will have more control over generated bounce messages in general.

If that's not an option, your best bet is probably to simply generate an autoreply which looks like a bounce message, instead of letting `sendmail` generate the bounce message for you. Required reading: RFC1894, or get one of the existing bounce generators (there's at least one; see the Links page for more information).

Q: How can I extract (or kill) MIME parts from composite messages?

A: While there is nothing inherently very complicated about MIME multipart messages, the problem is not particularly a Procmail problem. You cannot write a Procmail recipe or a sed script which can parse them on its own because the MIME format is -- after all -- complicated enough to require a bit more muscle than simple regular expressions.

Jari Aalto's Procmail library has a component which attempts to do it anyway. See the Links page for details.

You can write your own Perl script to pick apart the messages (recommended: get the Perl MIME module instead of inventing your own wheel) or use a dedicated program such as `munpack` to handle them. (See the MIME FAQ for pointers.)

An unsound thing to do, which people ask about a lot anyway, is to kill all MIME messages. This should be easy (all compliant MIME messages contain a `Mime-Version:` header) but not smart, because MIME is merely an extension technology, not a sign that the message contains anything in particular that people usually want to filter with this measure (such as a huge file attachment, or HTML-encrypted message text, or text in the Latin-1 character set or a number of other character sets, including the default 7-bit US-ASCII, or anything else that MIME can be a vehicle for). Again, reading up on MIME would probably be a good idea if you are thinking about this.

Q: How do I implement a "vacation" program? A spam auto-responder?

A: Look at the `procmailex` man page. For an even simpler solution (in that it's less reading for you and presumably a better and more stable solution than you could build on your own in a couple of hours), get Alan K. Stebbens's Procmail library (see below) and tweak to suit your needs.

If unsolicited bulk e-mail, aka spam, is a problem for you, your best defense is to get your ISP to filter your mail at the SMTP and/or router level. The next best option is to install one of the many existing Procmail antispam packages. The Links page has pointers to a host of'em. If you want to discuss the minutia of detecting spam, a dedicated spam list is probably a better forum than the

Procmal-L mailing list.

Bouncing unwanted messages is not necessarily a smart thing to do.

Where can I learn more?

A small links collection

- The companion links page to this FAQ page has a lot more links. The below links are all there, along with a host of others. Check it out.
- There is now an official Procmal site at <http://www.procmal.org/>
- A Procmal Quick Reference by yours truly (still considered a beta version) is at <http://www.iki.fi/era/procmal/quickref.html>
- Another links and information collection is Infinite Ink's site. Start at the Mail Filtering and Robots page -- you'll find lots of Procmal-related links, as well as mounds of other useful information.
- Also check out the Best of Procmal archive.
- There is an archive for the whole mailing list, too, not just the "best-of" selection. Valuable snippets continue to be posted to the mailing list. Thanks to Achim Bohnet, there is even a searchable hypertext version of the archive of the Procmal-L mailing list.
- Jari Aalto has created a Procmal Tips Page (there's also an HTML version but the rendering of some of the recipes is funny, IMHO) which you can also retrieve by e-mailing jari.aalto@poboxes.com with the Subject: send pm-tips.txt
You can also say "send help" and/or browse around in Jari's ftp directory, where you'll find various Procmal libraries among sundry other files.
Warning/Guarantee: The tips page is large -- much larger than this FAQ --, sometimes somewhat whimsical, and in desperate need of proofreading. Still, a very useful resource.

Kudos

This Procmal FAQ draws heavily on Nancy McGough's excellent collection of mail filtering resources. If you only bookmark one web page this year, make sure it's an Infinite Ink one! This FAQ would also be impossible -- and perhaps unnecessary :-) -- were it not for the Procmal-L mailing list. If you're serious about Procmal, you need to be on this list. It also happens to be one of the most helpful bunches of people on the 'Net. No names mentioned, in order not to offend those I'd no doubt forget to remember -- you know who you are! Finally, none of this would be even remotely possible without the skill, dedication, and genius of Procmal's author, **Stephen R. van den Berg**. The 'Net community owes you one, Stephen.

Contact and copyright information

The official location of this page is <http://www.iki.fi/era/procmal/mini-faq.html> -- please use this redirector URL instead of whatever it happens to point to when referring to this page.



* To Links page * Feedback and comments are most welcome! *

Copyright: I wrote it with the help of others. Way to go.

The cover page has a "to do" section with some ideas. If you think you can help out with any of them, it would be very much appreciated.

A text-only version of this Procmail FAQ is also available by e-mail (via Procmail of course); send a message to era+pr@iki.fi with a Subject: header containing only the words `send procmail-faq ...` to get a copy of the `experiments.rc` file, use `send experiments.rc`

\$Id: mini-faq.prep,v 1.306 1999/06/23 15:02:41 era Exp \$

Appendix A -- Folder Formats

This appendix lists some common (and some less common) mail folder formats.

A "folder," in this sense, is merely another name for a file or directory which contains mail messages.

Berkeley mbox format

This is a flat file, with no explicit message delimiter in the classical sense. Each line starting with the five characters "From " and preceded by either an empty line or beginning of file is the beginning of a new message.

Some exciting more or less incompatible variants of this format exist, however the one used by Sendmail and BSD `mailx` is by far the most common these days.

MMDF format

Used by the MMDF mailer; flat file, each message is separated by a string of four ctrl-A's. Procmail can be configured relatively simply at compile-time to use this instead of Berkeley format for file folders.

MH format

This is a one-message-per-file format, where a directory forms the folder and the files in it are messages (the only exceptions are control and cache files used by certain MUAs). The file names are numbers (not necessarily in sequence), or numbers preceded by commas. The files with names starting with commas correspond to deleted messages.

Procmail can deliver to MH directories but doesn't know how to update the associated control and cache files. For that reason, some people prefer to pipe messages to the `MH rcvstore` program instead of delivering straight to an MH folder.

Used by the Rand MH system and derivatives.

"Mail directory" or "MSGPREFIX" format

Similar to MH format, except the file naming conventions are different.

More details welcome.

Maildir format

Not directly understood by Procmail, but you can call up an external program for each delivery, or get a patch to Procmail. Used by Qmail.

A Maildir folder is a directory, with three subdirectories named "tmp", "new", and "cur".

Messages are written into "tmp", then moved to "new" to commit the delivery. As messages are read, they're moved from "new" to "cur", and renamed to append flags for the message status. These files are "rigid," the contents are not changed when a message's status changes.

In Maildir folders the files have long, complex names intended to ensure that all filenames are unique, even across different machines.

Maildir is the only file-oriented mail folder format that requires no locking.

MBX format

Not directly understood by Procmail, but you can get an external program and use that to deliver to MBX. Used by some IMAP systems (?)

More details welcome.

BABYL format

This is the format used by RMAIL, the mail mode that GNU Emacs supports out of the box. It consists of one file per folder with an interesting separator between the messages and a copy of each message's original headers before the message itself. In practice, you deliver new mail into an mbox-format spool file (or something else, if you figure out how to teach the Emacs `movemail` utility how to handle new formats) and let Emacs import mail from there using the `rmail-get-new-mail` command (normally on the `g` key). The command `set-rmail-inbox-list` allows you to tell RMAIL where to get new mail from (you have to issue this separately for each RMAIL folder).

This format is not understood by any sane programs except `formail` with the `-B` option.

nml

Not understood by Procmail, but included here for completeness' sake. A format specific to the newsreader Gnus (which also has a lot of mail handling functions built in).

Appendix B -- Figuring Out the Mail Flow

For a lot of reasons, it would be good if you could figure out as much as possible about how mail is being processed on the host(s) where you intend to run Procmail.

Here's a quick rundown of what you can do to learn more about how mail flows on your site. (If you're the postmaster of a site and don't know these things already, you're probably in deep trouble. This is primarily meant for the end users. For the benefit of your users, it would be nice if you could document these things e.g. on your web pages.)

If all you have is a single host which serves all the needs of all your users, you can probably stop reading here (unless you're curious).

If your ISP or company has several relatively autonomous hosts, each might be capable of receiving and sending mail. Modern installations often try to delegate mail processing to one central "smart host" and refuse at least external mail on the other machines (users' workstations, database servers, web servers etc). An unfortunate number of sites with old software are running a number of essentially uncontrolled and unmanaged mail servers, which is confusing both to users and administrators, and probably means there is ample opportunity for spammers and other abusers to find security holes.

The typical scenario on modern sites is to have one big dedicated server which is responsible for delivering mail to local users. Other hosts are only relaying stuff to the smart host, and preferably don't accept outside mail at all. Making changes to the mail configuration means you change the configuration of the smart host, and don't have to worry about the "stupid clients," which should perhaps even be behind a firewall.

DNS, the Domain Name System, allows each domain (and even each host) to have an *MX (Mail Exchanger) record* which tells others where to route mail for this particular host. Unfortunately,

many clients erroneously fail to look for MX records, and will blindly bypass this redirection and try to send to the named host. For this reason, hosts whose names are going to be, or have been, visible to users outside your domain should probably be equipped to at least relay mail to the main MX host.

Internal hosts, inside the firewall or in the same domain, often don't care about MX hosts and are able to send mail directly to local users (one way or another). In effect, the same configuration as for "stupid hosts" can be used for this, too -- just let the "stupid hosts" forward to the main MX machine. (This is what secondary MX hosts are often set up to do as well.)

You can find out which hosts are designated as MX hosts with any standard DNS lookup tool. *nslookup*(1) is the classical tool for this, but you might want to see if you have *host*(1) and/or *dig*(1) installed, too.

Here's an example with *nslookup*:

```
$ nslookup -q=mx rwth-aachen.de
Server: localhost
Address: 127.0.0.1

Non-authoritative answer:
rwth-aachen.de preference = 20, mail exchanger = mail.rwth-aachen.de
rwth-aachen.de preference = 30, mail exchanger = Campino.Informatik.rwth-aachen

Authoritative answers can be found from:
rwth-aachen.de nameserver = netsl.rz.rwth-aachen.de
rwth-aachen.de nameserver = deneb.dfn.de
rwth-aachen.de nameserver = ns.nic.de
rwth-aachen.de nameserver = Urmel.Informatik.rwth-aachen.de
mail.rwth-aachen.de internet address = 137.226.144.9
Campino.Informatik.rwth-aachen.de internet address = 137.226.116.240
netsl.rz.rwth-aachen.de internet address = 137.226.144.3
deneb.dfn.de internet address = 192.76.176.9
ns.nic.de internet address = 193.196.32.1
Urmel.Informatik.rwth-aachen.de internet address = 137.226.112.21
```

The output from *nslookup* is slightly messy. Among all the unrelated information about name servers etc, this simply says that the hosts *mail* and *Campino.Informatik* do MX for Aachen. By having a lower preference number (zero means most preferred), *mail* is the primary MX host.

In general terms, hosts with lower preference numbers should be preferred over the ones with higher numbers, which would often be off-site hosts which are merely set up to queue the mail if and when the main MX is down, and then forward there when it comes back up again. Some spammers have found that secondary MX hosts are often less well-protected than the main MX host and try to relay spam via a secondary.

If you don't have access to command-line DNS tools, there are numerous WWW services which allow anybody to use these same tools via a simple WWW form interface.

If you have collected mail for some time, a quick look at the *Received:* headers of those messages should be enough to give you a good impression of how things are working in practice. Still, don't forget to check DNS for secondary MXes and other interesting phenomena if you write recipes which need to know which hosts are "local" or "allowed" in some sense.