NAME

procmailex − procmail rcfile examples

SYNOPSIS

$HOME/.procmailrc examples

DESCRIPTION

For a description of the rcfile format see **procmailrc**(5).

The weighted scoring technique is described in detail in the **procmailsc**(5) man page.

This man page shows several example recipes. For examples of complete rcfiles you can check the NOTES section in **procmail**(1), or look at the example rcfiles part of the procmail source distribution (procmail*/examples/?procmailrc).

EXAMPLES

Sort out all mail coming from the scuba-dive mailing list into the mailfolder scubafile (uses the locallockfile scubafile.lock).

```
:0:
* ^TOscuba
scubafile
```

Forward all mail from peter about compilers to william (and keep a copy of it here in petcompil).

```
:0
* ^From.*peter
* ^Subject:.*compilers
{
  :0 c
  ! william@somewhere.edu

  :0
  petcompil
}
```

An equivalent solution that accomplishes the same:

```
:0 c
* ^From.*peter
* ^Subject:.*compilers
! william@somewhere.edu

  :0 A
  petcompil
```

An equivalent, but slightly slower solution that accomplishes the same:

```
:0 c
* ^From.*peter
* ^Subject:.*compilers
! william@somewhere.edu

:0
* ^From.*peter
* ^Subject:.*compilers
petcompil
```

If you are fairly new to procmail and plan to experiment a little bit it often helps to have a *safety net* of some sort. Inserting the following two recipes above all other recipes will make sure that of all arriving mail always the last 32 messages will be preserved. In order for it to work as intended, you have to create a directory named ‘backup’ in $MAILDIR prior to inserting these two recipes.

```
:0 c
backup

:0 ic
| cd backup && rm −f dummy ‘ls −t msg.* | sed −e 1,32d‘
```

If your system doesn’t generate or generates incorrect leading ‘From ’ lines on every mail, you can fix this by calling up procmail with the −f- option. To fix the same problem by different means, you could have inserted the following two recipes above all other recipes in your rcfile. They will filter the header of any mail through formail which will strip any leading ‘From ’, and automatically regenerates it subsequently.

```
:0 fhw
| formail −I "From " −a "From "
```

Add the headers of all messages that didn’t come from the postmaster to your private header collection (for statistics or mail debugging); and use the lockfile ‘headc.lock’. In order to make sure the lockfile is not removed until the pipe has finished, you have to specify option ‘w’; otherwise the lockfile would be removed as soon as the pipe has accepted the mail.

```
:0 hwc:
* !^FROM_MAILER
| uncompress headc.Z; cat >>headc; compress headc
```

Or, if you would use the more efficient gzip instead of compress:

```
:0 hwc:
* !^FROM_MAILER
| gzip >>headc.gz
```

Forward all mails shorter than 1000 bytes to my home address (no lockfile needed on this recipe).

```
:0
* < 1000
! myname@home
```

Split up incoming digests from the surfing mailing list into their individual messages, and store them into surfing, using surfing.lock as the locallockfile.

```
:0:
* ^Subject:.*surfing.*Digest
| formail +1 −ds >>surfing
```

Store everything coming from the postmaster or mailer-daemon (like bounced mail) into the file postm, using postm.lock as the locallockfile.

```
:0:
* ^FROM_MAILER
postm
```

A simple autoreply recipe. It makes sure that neither mail from any daemon (like bouncing mail or mail from mailing-lists), nor autoreplies coming from yourself will be autoreplied to. If this precaution would not be taken, disaster could result (‘ringing’ mail). In order for this recipe to autoreply to all the incoming mail, you should of course insert it before all other recipes in your rcfile. However, it is advisable to put it *after* any recipes that process the mails from subscribed mailinglists; it generally is not a good idea to generate autoreplies to mailinglists (yes, the !^FROM_DAEMON regexp should already catch those, but if the mailinglist doesn’t follow accepted conventions, this might *not* be *enough*).

```
:0 h c
* !^FROM_DAEMON
* !^X-Loop: your@own.mail.address
| (formail −r −I"Precedence: junk" \
    −A"X-Loop: your@own.mail.address" ; \
  echo "Mail received.") | $SENDMAIL −t
```

A more complicated autoreply recipe that implements the functional equivalent of the well known **vacation**(1) program. This recipe is based on the same principles as the last one (prevent ‘ringing’ mail). In addition to that however, it maintains a vacation database by extracting the name of the sender and inserting it in the vacation.cache file if the name was new (the vacation.cache file is maintained by formail which will make sure that it always contains the most recent names, the size of the file is limited to a maximum of approximately 8192 bytes). If the name was new, an autoreply will be sent.

As you can see, the following recipe has comments **between** the conditions. This is allowed. Do **not** put comments on the same line as a condition though.

```
SHELL=/bin/sh    # for other shells, this might need adjustment

:0 Whc: vacation.lock
 # Perform a quick check to see if the mail was addressed to us
* $^To:.*\<$\LOGNAME\>
 # Don't reply to daemons and mailinglists
* !^FROM_DAEMON
 # Mail loops are evil
* !^X-Loop: your@own.mail.address
| formail −rD 8192 vacation.cache

  :0 ehc        # if the name was not in the cache
  | (formail −rI"Precedence: junk" \
       −A"X-Loop: your@own.mail.address" ; \
     echo "I received your mail,"; \
     echo "but I won't be back until Monday."; \
     echo "-- "; cat $HOME/.signature \
    ) | $SENDMAIL −oi −t
```

Store all messages concerning TeX in separate, unique filenames, in a directory named texmail (this directory has to exist); there is no need to use lockfiles in this case, so we won't.

```
:0
* (^TO|^Subject:.*)TeX[^t]
texmail
```

The same as above, except now we store the mails in numbered files (MH mail folder).

```
:0
* (^TO|^Subject:.*)TeX[^t]
texmail/.
```

Or you could file the mail in several directory folders at the same time. The following recipe will deliver the mail to two MH-folders and one directory folder. It is actually only one file with two extra hardlinks.

```
:0
* (^TO|^Subject:.*)TeX[^t]
texmail/. wordprocessing dtp/.
```

Store all the messages about meetings in a folder that is in a directory that changes every month. E.g. if it were January 1994, the folder would have the name ‘94-01/meeting’ and the locallockfile would be ‘94-01/meeting.lock’.

```
:0:
* meeting
‘date +%y-%m‘/meeting
```

The same as above, but, if the ‘94-01’ directory wouldn’t have existed, it is created automatically:

```
MONTHFOLDER=‘date +%y-%m‘

:0 Wic
* ? test ! −d $MONTHFOLDER
| mkdir $MONTHFOLDER

:0:
* meeting
${MONTHFOLDER}/meeting
```

The same as above, but now by slightly different means:

```
MONTHFOLDER=‘date +%y-%m‘
DUMMY=‘test −d $MONTHFOLDER || mkdir $MONTHFOLDER‘

:0:
* meeting
${MONTHFOLDER}/meeting
```

If you are subscribed to several mailinglists and people cross-post to some of them, you usually receive several duplicate mails (one from every list). The following simple recipe eliminates duplicate mails. It tells formail to keep an 8KB cache file in which it will store the Message-IDs of the most recent mails you received. Since Message-IDs are guaranteed to be unique for every new mail, they are ideally suited to weed out duplicate mails. Simply put the following recipe at the top of your rcfile, and no duplicate mail will get past it.

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

Beware if you have delivery problems in recipes below this one and procmail tries to requeue the mail, then on the next queue run, this mail will be considered a duplicate and will be thrown away. For those not quite so confident in their own scripting capabilities, you can use the following recipe instead. It puts duplicates in a separate folder instead of throwing them away. It is up to you to periodically empty the folder of course.

```
:0 Whc: msgid.lock
| formail −D 8192 msgid.cache

:0 a:
duplicates
```

Procmail can deliver to MH folders directly, but, it does not update the unseen sequences the real MH manages. If you want procmail to update those as well, use a recipe like the following which will file everything that contains the word spam in the body of the mail into an MH folder called spamfold. Note the local lockfile, which is needed because MH programs do not lock the sequences file. Asynchronous invocations of MH programs that change the sequences file may therefore corrupt it or silently lose changes. Unfortunately, the lockfile doesn’t completely solve the problem as rcvstore could be invoked while ‘show’ or ‘mark’ or some other MH program is running. This problem is expected to be fixed in some future version of MH, but until then, you’ll have to balance the risk of lost or corrupt sequences against the benefits of the unseen sequence.

```
:0 :spamfold/$LOCKEXT
* B ?? spam
| rcvstore +spamfold
```

When delivering to emacs folders (i.e., mailfolders managed by any emacs mail package, e.g., RMAIL or VM) directly, you should use emacs-compatible lockfiles. The emacs mailers are a bit braindamaged in that respect, they get very upset if someone delivers to mailfolders which they already have in their internal buffers. The following recipe assumes that $HOME equals /home/john.

```
MAILDIR=Mail

:0:/usr/local/lib/emacs/lock/!home!john!Mail!mailbox
* ^Subject:.*whatever
mailbox
```

Alternatively, you can have procmail deliver into its own set of mailboxes, which you then periodically empty and copy over to your emacs files using **movemail**. Movemail uses mailbox.lock local lockfiles per mailbox. This actually is the preferred mode of operation in conjunction with procmail.

To extract certain headers from a mail and put them into environment variables you can use any of the following constructs:

```
SUBJECT=‘formail −xSubject:‘    # regular field
FROM=‘formail −rt −xTo:‘        # special case

:0 h                        # alternate method
KEYWORDS=| formail −xKeywords:
```

If you are using temporary files in a procmailrc file, and want to make sure that they are removed just before procmail exits, you could use something along the lines of:

```
TEMPORARY=$HOME/tmp/pmail.$$
TRAP="/bin/rm −f $TEMPORARY"
```

The TRAP keyword can also be used to change the exitcode of procmail. I.e. if you want procmail to return an exitcode of ‘1’ instead of its regular exitcodes, you could use:

```
EXITCODE=""
TRAP="exit 1;"   # The trailing semi-colon is important
            # since exit is not a standalone program
```

Or, if the exitcode does not need to depend on the programs run from the TRAP, you can use a mere:

```
EXITCODE=1
```

The following recipe prints every incoming mail that looks like a postscript file.

```
:0 Bb
* ^^%!
| lpr
```

The following recipe does the same, but is a bit more selective. It only prints the postscript file if it comes from the print-server. The first condition matches only if it is found in the header. The second condition only matches at the start of the body.

```
:0 b
* ^From[ :].*print-server
* B ?? ^^%!
| lpr
```

The same as above, but now by slightly different means:

```
:0
* ^From[ :].*print-server
{
  :0 B b
  * ^^%!
  | lpr
}
```

Likewise:

```
:0 HB b
* ^^(.+$)*From[ :].*print-server
* ^^(.+$)*^%!
| lpr
```

Suppose you have two accounts, you use both accounts regularly, but they are in very distinct places (i.e., you can only read mail that arrived at either one of the accounts). You would like to forward mail arriving at account one to account two, and the other way around. The first thing that comes to mind is using .forward files at both sites; this won't work of course, since you will be creating a mail loop. This mail loop can be avoided by inserting the following recipe in front of all other recipes in the $HOME/.procmailrc files on both sites. If you make sure that you add the same X-Loop: field at both sites, mail can now safely be forwarded to the other account from either of them.

```
:0 c
* !^X-Loop: yourname@your.main.mail.address
| formail -A "X-Loop: yourname@your.main.mail.address" | \
    $SENDMAIL -oi yourname@the.other.account
```

If someone sends you a mail with the word 'retrieve' in the subject, the following will automatically send back the contents of info_file to the sender. Like in all recipes where we send mail, we watch out for mail loops.

```
:0
* !^From +YOUR_USERNAME
* !^Subject:.*Re:
* !^FROM_DAEMON
* ^Subject:.*retrieve
| (formail -r ; cat info_file) | $SENDMAIL -oi -t
```

Now follows an example for a very simple fileserver accessible by mail. For more demanding applications, I suggest you take a look at **SmartList** (available from the same place as the procmail distribution). As listed, this fileserver sends back at most one file per request, it ignores the body of incoming mails, the Subject: line has to look like "Subject: send file the_file_you_want" (the blanks are significant), it does not return files that have names starting with a dot, nor does it allow files to be retrieved that are outside the fileserver directory tree (if you decide to munge this example, make sure you do not inadvertently loosen this last restriction).

```
:0
* ^Subject: send file [0-9a-z]
* !^X-Loop: yourname@your.main.mail.address
* !^Subject:.*Re:
* !^FROM_DAEMON
* !^Subject: send file .*[/.]\.
{
 MAILDIR=$HOME/fileserver # chdir to the fileserver directory

 :0 fhw                  # reverse mailheader and extract name
 * ^Subject: send file \/[^ ]*
 | formail -rA "X-Loop: yourname@your.main.mail.address"

 FILE="$MATCH"           # the requested filename

 :0 ah
 | cat - ./$FILE 2>&1 | $SENDMAIL -oi -t
}
```

The following example preconverts all plain-text mail arriving in certain encoded MIME formats into a more compact 8-bit format which can be used and displayed more easily by most programs. The **mimencode**(1) program is part of Nathaniel Borenstein's metamail package.

```
:0
* ^Content-Type: *text/plain
{
 :0 fbw
 * ^Content-Transfer-Encoding: *quoted-printable
 | mimencode -u -q

   :0 Afhw
   | formail -I "Content-Transfer-Encoding: 8bit"

 :0 fbw
 * ^Content-Transfer-Encoding: *base64
 | mimencode -u -b

   :0 Afhw
   | formail -I "Content-Transfer-Encoding: 8bit"
}
```

The following one is rather exotic, but it only serves to demonstrate a feature. Suppose you have a file in your HOME directory called ".urgent", and the (one) person named in that file is the sender of an incoming mail, you'd like that mail to be stored in $MAILDIR/urgent instead of in any of the normal mailfolders it would have been sorted in. Then this is what you could do (beware, the filelength of $HOME/.urgent should be well below $LINEBUF, increase LINEBUF if necessary):

```
URGMATCH=‘cat $HOME/.urgent‘

:0:
* $^From.*${URGMATCH}
urgent
```

An entirely different application for procmail would be to conditionally apply filters to a certain (outgoing) text or mail. A typical example would be a filter through which you pipe all outgoing mail, in order to make sure that it will be MIME encoded only if it needs to be. I.e. in this case you could start procmail in the middle of a pipe like:

```
cat newtext | procmail ./mimeconvert | mail chris@where.ever
```

The **mimeconvert** rcfile could contain something like (the =0x80= and =0xff= should be substituted with the real 8-bit characters):

```
DEFAULT=|     # pipe to stdout instead of
        # delivering mail as usual
:0 Bfbw
* [=0x80=-=0xff=]
| mimencode −q

 :0 Afhw
 | formail −I ’MIME-Version: 1.0’ \
   −I ’Content-Type: text/plain; charset=ISO-8859-1’ \
   −I ’Content-Transfer-Encoding: quoted-printable’
```

SEE ALSO

procmail(1), **procmailrc**(5), **procmailsc**(5), **sh**(1), **csh**(1), **mail**(1), **mailx**(1), **binmail**(1), **uucp**(1), **aliases**(5), **sendmail**(8), **egrep**(1), **grep**(1), **biff**(1), **comsat**(8), **mimencode**(1), **lockfile**(1), **formail**(1)

AUTHORS

Stephen R. van den Berg
<srb@cuci.nl>
Philip A. Guenther
<guenther@sendmail.com>