The Elm Filter System Guide

What the filter program is, what it does,
and how to use it

The Elm Mail System
(Version 2.4)

Syd Weinstein, Elm Coordinator
Datacomp Systems, Inc.
3837 Byron Road
Huntingdon Valley, PA 19006-2320

email: elm@DSI.COM or dsinc!elm

The Elm Filter System Guide

(The Elm Mail System, Version 2.4)

October 1, 1992

Syd Weinstein
Datacomp Systems, Inc.
3837 Byron Road
Huntingdon Valley, PA 19006-2320

email: elm@DSI.COM or dsinc!elm

Derived from
''The Elm Mail System, Version 2.0''
by
Dave Taylor
Intuitive Systems
Mountain View, California
email: taylor@intuitive.com or limbo!taylor

One of the greatest problems with the burgeoning electronic mail explosion is that I tend to get lots of mail that I don't care about. Amusingly, perhaps, I have the equivalent of electronic junk mail. Not amusing, however, is the fact that this can rapidly accumulate and end up taking over my mailbox.

At the same time I often get mail that, while it is interesting and important, can easily be filed to be read later, without ever actually having to clutter up my incoming mailbox.

This, then, is what *filter* does! The *filter* program allows you to define a set of rules by which all incoming mail is screened, and a subsequent set of actions to perform based on whether the conditions are met or not. *Filter* also has the ability to mail a summary of what actions it performed on the incoming mail as often as you'd like.

Writing the Rules

The language for writing *filter* rules is pretty simple, actually. The fundamental structure is:

> if (*condition*) then *action*

Where *condition* is constructed by an arbitrary number of individual conditions of the form ''*field relation value*'' (an optional further type of rule is of the form ''always *action*'' but should only be used as the last rule in the ruleset, for obvious reasons). The *field* value can be:

subject
sender
from
to
lines

For the field **lines**, the *relation* can be any of the standard relationships (‘>’, ‘<’, ‘>=’, ‘<=’, ‘!=’ and ‘=’). **contains** is equivalent to the relation ‘=’ or you can skip the relationship entirely (e.g. ‘‘subject = "joe"’’, ‘‘subject contains "joe"’’, and ‘‘subject "joe"’’ are all equivalent). The *value* is any quoted string that is to be matched against or a number if **lines** is the *field* being considered.

The relation **matches** (or ‘~’) takes an *egrep* like regular expression as the *value*. The regular expression must be enclosed in pairs of ‘/’. If you need a ‘/’ in your expression write it as ‘‘\ /’’.

Individual conditions are joined together by using the word **and**, and the logic of a condition can be flipped by using **not** as the first word (e.g. ‘‘subject "fun" and not subject "games"’’). We’ll see more examples of this later.

Note that the ‘‘or’’ logical conjunction isn’t a valid part of the *filter* conditional statement.

Finally, *action* can be any of:

Action	**Description**
delete	Deletes the message
save *foldername*	Saves a copy of the message in the specified foldername.
savecopy *foldername*	Does the same as save, but also puts a copy in your mailbox.
forward *address*	Sends the message to the specified address.
forwardc *address*	Sends the message to the specified address and leaves a copy in your mailbox.
execute *command*	Feeds the message to the specified command (or complex sequence of commands) as standard input.
executec *command*	Like **execute** but also leaves a copy in your mailbox.
leave	Leaves the message in your mailbox.

Note that the **execute** command assumes that you are doing something useful with incoming mail. The command does *not* automatically append the message to your mail spool. Use **executec** for that. A command such as:

```
if (from = "dave") then execute "echo mail from dave > /dev/tty"
```

will result in *losing* your mail. A more suitable use for execute would be:

```
if (from = "boss") then execute "vacation pat"
```

Foldernames can contain any of a number of macros, too, as we’ll see in the example ruleset below. The macros available for the string fields are:

Macro	**Meaning**
%d	day of the month
%D	day of the week (0-6)
%h	hour of the day (0-23)

%m	month of the year (1-12)
%r	return address of message
%s	subject of original message
%S	‘‘Re: *subject of original message*’’
%t	current hour and minute in HH:MM format
%y	year (last two digits)
%&	the string that matched the last regular expression
%1-%9	the corresponding subexpression in the last regular expression that matched

If a message has a subject of ‘‘This is a test’’, and the rule is

```
if (subject matches /([a-z]+) a ([a-z]+)/) then "echo %& %1 %2"
```

then **%&** expands to ‘‘is a test’’, **%1** expands to ‘‘is’’ and **%2** expands to ‘‘test’’.

Foldernames may contain a leading ‘‘~/’’ which expands to your home directory.

The rules file can also contain comments (any line starting with a ‘#’) and blank lines. Metacharacters may be escaped by a preceding ‘\’ to remove any special meaning. If you need a ‘%’ character as part of an argument (such as an address), use ‘‘\%’’. If you need a double quote in a double quoted string, use ‘‘\"’’. If you need a single quote in a single quoted string, use ‘‘\´’’. If you need a backslash in any kind of string, use ‘‘\\’’.

Unless you specify an alternate file via the **–f** option, the rules file needs to reside in your *.elm* directory off your home directory and be called *.elm/filter–rules*. Here’s an example:

```
# $HOME/.elm/filter-rules
#
# Filter rules for the Elm Filter program. Don´t change without some
# serious thought. (remember - order counts)
#
# (for Dave Taylor)
# rule 1
if (from contains "!uucp") then delete
# rule 2
to "postmaster" ? save "/tmp/postmaster-mail.%d"
# rule 3
if (to "culture" and lines > 20) ? save "~/Mail/culture"
# rule 4
subject = "filter test" ? forward "hpldat!test"
# rule 5
if [ subject = "elm" ] savecopy "~/Mail/elm-incoming"
# rule 6
subject = "display-to-console" ? execute "cat - > /dev/console"
```

(Notice the loose syntax — there are lots of valid ways to specify a rule in the *filter* program!!)

To translate these into English:

1. All messages from ‘‘uucp’’ should be summarily deleted.

2. All mail to ‘‘postmaster’’ should be saved in a folder (file) called */tmp/postmaster–mail.n*, where *n* is the

numeric-day-of-the-week.

3. All mail addressed to ''culture'' with at least 20 lines should be automatically appended to the folder *$HOME/Mail/culture*.

4. All messages that contain the subject ''filter test'' should be forwarded to me, but via the address ''hpldat!test'' (to force a non-user forward).

5. All messages with a subject that contains the word ''elm'' should be saved in the folder *$HOME/Mail/elm−incoming* and also dropped into my mailbox.

6. Any message with the subject ''display-to-console'' will be immediately written to the console.

Notice that the *order* of the rules is very important. If we, for example, were to get a message from ''uucp'' that had the subject ''filter test'', the *filter* program would match rule 1 and delete the message. It would never be forwarded to ''hpldat!test''. It is for this reason that great care should be taken with the ordering of the rules.

Checking the rules out

The *filter* program has a convenient way of check out the rules you have written. Simply invoke it with the **−r** (**rules**) option:

```
% filter −r
Rule 1: if (from = "!uucp") then
        Delete
Rule 2: if (to = "postmaster") then
        Save  /tmp/postmaster−mail.3
Rule 3: if (to = "culture" and lines > 20) then
        Save  ˜/Mail/culture
Rule 4: if (subject = "filter test") then
        Forward  hpldat!test
Rule 5: if (subject="elm") then
        Copy  and  Save  ˜/Mail/elm−incoming
Rule 6: if (subject="display−to−console") then
        Execute "cat − > /dev/console"
```

There are a few things to notice — first off, these are the parsed and rebuilt rules, so we see them all in a consistent format. Also, notice on the filename for rule 2 that *filter* has correctly expanded the **%d** macro to be the day of the week (in this case, Wednesday, denoted by the digit '3').

It is **highly** recommended that you always check your ruleset before actually using it to filter your mail!

Actually Using the Program

Now the bad news. If you aren't running *sendmail* or some other Mail Transport Agent that supports delivery to a program you cannot use this program as currently written. Why? Because the *filter* program expects to be put in your *.forward* file (or equivalent concept for MTAs other than *sendmail*) that causes the messages to be delivered to the *filter* program. That is something that only *sendmail* and other smart MTAs look at!

The format for the entry in the *.forward* file (located in your home directory) is simply:

"|/usr/local/bin/filter"

Alright, it isn't quite *that* simple! Since *filter* is invoked by processes that don't know where you are logged in, you need to have some way to trap the error messages. For ease of use, it was decided to have all the messages written to the file specified by **−o** (or *stderr*) which means that you have two main choices for the actual entry. Either:

"|/usr/local/bin/filter −o /dev/console"

which will log all errors on the system console (each error is prefixed with ‘‘filter (*username*)’’ to distinguish it), or:

"|/usr/local/bin/filter −o /tmp/joe.filter_errors"

if you want to have a copy saved to a file. Note that the quotes are a required part of the line. A possible strategy would be to have the errors written to a file and to then have a few lines in your *.login* (or equivalent) script like:

```
if ( −f /tmp/joe.filter_errors) then
   echo  " "
   echo "Filter program errors;"
   cat /tmp/joe.filter_errors
   echo " "
endif
```

You can also use the **−v** option in combination with the above to have a more verbose log file (including action taken with date/time stamp) saved by having in your *.forward* file:

```
"|/usr/local/bin/filter −vo /tmp/joe.filter_errors"
```

Suffice it to say, you can get pretty tricky with all this!!

Summarizing the Actions Taken

The *filter* program can keep a log of all actions performed, including what rules it matched against, in your *.elm* directory in a file called *.elm/filterlog*. You can either directly operate on this file, or, much more recommended, you can use one of the two summarization options to *filter* and let *it* do the work for you!

The difference between the two is best demonstrated by example:

% **filter −s**

```
                Summary of Filter Activity
                --------------------------
A total of 418 messages were filtered:
The default rule of putting mail into your mailbox
         applied 364 times (87%)
Rule #1: (delete message)
         applied 1 time (0%)
Rule #2: (save in "˜/Filtered−Mail/netnews.12")
         applied 8 times (2%)
Rule #3: (save in "˜/Filtered−Mail/postmaster.12")
         applied 14 times (3%)
Rule #5: (save in "˜/Filtered−Mail/risks.12")
         applied 3 times (1%)
Rule #6: (save in "˜/Filtered−Mail/rays.12")
         applied 28 times (7%)
```

versus:

% **filter –S**
[the output as listed above, followed by:]
```
Explicit log of each action;
Mail from taylor about Filter Summary
         PUT in mailbox: the default action
Mail from news@hplabsz.hpl.hp.com about Newsgroup comp.editors created
         PUT in mailbox: the default action
Mail from root about Log file: cleanuplog
         PUT in mailbox: the default action
```
[etc.]

To actually use either of the summarizing options, there are two ways that are recommended. The preferred way is to have a line in either your *crontab* file (ask your administrator for help with this) that invokes the *filter* program as often as you desire with the **–s** option. For example, I have a summary mailed to me every morning at 8 o'clock:

```
0 8 * * * "/usr/local/bin/filter –s | elm –s ´Filter Summary´ taylor"
```

An alternative is to have your *.login* (or equivalent) execute the command each time you log in.

Note that if you want to have your log files cleared out each time the summary is generated you'll need to use the **–c** option too. Also, if you want to keep a long list of actions performed you can do this by saving it as you display it. A way to do this would be, if you were to have the invocation in your *.login* (or equivalent) script, to use:

```
echo "Filter Log;"
filter –c –s | tee –a PERM.filter.log
```

which would append a copy of all the output to the file *PERM.filter.log* and would avoid you having to read larger and larger summaries of what the program had done.

If you prefer not to have *filter* generate log files, use the **–q** option. Messages that are written to *stderr* or the **–o** file are still generated; only message to the log files are quashed.

Further Testing of the Ruleset

Using the *readmsg* command, it is quite easy to test the rules you've written to see if they do what you desire.

For example, we can use the **–n** option to *filter*, which means ''don't actually do this, just tell me what rule you matched, if any, and what action you would have performed'' (you can see why a single letter option is easier to type in!!), and feed it each message in our mailbox by using a command like:

% **set message=1**
% **set total_messages=`messages`**
% **while (1)**
> **if ($message > $total_messages) exit**
> **readmsg –h $message | filter –n**
> **@ message++**
> **end**

which will then hand each of the messages in your mailbox to the *filter* program and display what action would have been taken with that message and why.

For example, if we do this for a few interesting messages in my mailbox, we'd end up with output like:

```
Mail from taylor about filter test
     FORWARDED to hpldat!taylor by rule;
       subject="filter test"  ? forward "hpldat!test"
Mail from bradley%hplkab@hplabsc about Re: AI-ED mailing address for HP
     PUT in mailbox: the default action
Mail from taylor about display-to-console
     EXECUTED "cat - > /dev/console"
```

(Sharp users will notice that this is exactly the same format as the longer summary listing.)

What Forwarded Messages Look Like

When a message is forwarded to another user by the *action* being specified as ‘‘**forward** *address*’’, then *filter* can generate one of two styles of message. If the message is to you, then it simply adds it to your mailbox in such a way as to ensure that the return address is that of the person who sent the message.

If not, then the message is enclosed in a message of the form:

```
From taylor Thu Oct  2 15:07:04 1986
Date: Thu, 2 Oct 86 15:06:58 pdt
Subject: "filter test"
From: The filter of taylor@hpldat <taylor>
To: hpldat!taylor
X-Filtered-By: filter, version 1.4
-- Begin filtered message --

   From taylor Thu Oct  2 15:06:41 1986
   Date: Thu, 2 Oct 86 15:06:33 pdt
   From: Dave Taylor <taylor>
   Subject: filter test
   Just a simple test.
-- End of filtered message --
```

The subject of the actual message is the same as the subject of the message being forwarded, but in quotes. The **From:** header indicates how the message was sent, and the **X-Filtered-By:** header identifies what version of *filter* was used.

Areas to Improve

While the *filter* program as presented herein is obviously a nice addition to the set of tools available for dealing with electronic mail, there are some key features that are missing and will be added in the future based on demand.

As I see it, the main things missing are:

1. Perhaps more *actions* available (but what?)

2. Certainly the ability to filter based on any field or combination of fields.

Warnings and Things to Look Out For

Since this is a pretty simple program, there are a few pitfalls, some of which have already been mentioned:

Order counts in the rules. Beware!

Matching is pretty simple — make sure your patterns are sufficiently exclusive before having any destructive rules.

The log files can grow indefinitely unless you use the **−c** or **−q** options or you trim the files by other means.

Finally, as with the rest of the Elm mail system, I welcome feedback and suggestion on how to improve this program!!