PP

Volume 3: Users Guide

Pete Cowen
X-Tel Services Ltd

Julian Onions
X-Tel Services Ltd

December 19, 1991
(Version 6.0)

Contents

1 Introduction **1**

2 Local delivery in PP **3**
2.1 Local RFC 822-based Channel 3
2.1.1 Format of the .mailfilter File 3
2.1.2 Examples 7
2.2 Mailfilter Checkup Tool 8
2.3 New Message Notification 9
2.3.1 Use of flagmail with xalert and ttybiff 12
2.3.2 Examples 13
2.4 Automatic Vacation Notices 13
2.5 Forwarding Mail 14
2.6 Delivery Tools 14
2.6.1 strip_addr 15
2.6.2 flock 15

3 Distribution List Maintenance **17**
3.1 File based Distribution Lists 17
3.1.1 What you can and cannot do with *mlist* 17
3.1.2 How *mlist* can run 18
3.1.3 *malias*. 19
3.2 X.500 Directory based distribution lists 19
3.2.1 The Basics 19
3.2.2 The DIT Entries for the list 20
3.2.3 ORNames 21
3.2.4 The DL tool 22
3.2.5 Creating a List 23
3.2.6 Managing a List 23
3.2.7 Maintaining a List 24
3.2.8 Printing a List 25

4 Miscellaneous Utilities **26**
4.1 A /bin/mail Replacement 26
4.2 Message Status Checkup 26

List of Figures

2.1 Example .mailfilter File 8
2.2 Grammar for the .mailfilter File 16

Chapter 1

Introduction

This manual is concerned with the user-visible tools associated with PP. Other tools which are management oriented or system related are described in other volumes.

Most of the work of a message transfer system is not visible to the user, who typically interacts with some intermediary (user agent or UA). As such this manual is a collection of the tools and interfaces that are directly visible to the user. Conceptually, they may be considered as a part of the UA, which for convenience and performance are implemented as a part of the MTA.

The main components of this volume consider the following:

`local delivery:` Local delivery is one of the more obviously noticeable parts of PP to the user. This is where the message leaves the transfer system and enters the user's domain. As such, this interface is tailorable to a considerable degree. PP does not restrict the user to a given delivery program, but can be configured on both a per recipient and a per message-type basis. This is conceptually a user agent function, but it is often more efficient to do this operation at the user agent/MTA boundary.

`distribution lists:` Maintenance and use of distribution lists is another area where the end user is directly involved. This verges on administration, but is administration that can be delegated to given users for each list. The areas discussed here are the administration of distribution lists and the information associated with such lists.

`user oriented management:` There are certain administrative functions that it is useful to make available to end users. These include checking the progress of individual messages and checking the validity of addresses and usernames.

Much of the information presented here is often available in the online UNIX[1] manuals but usually in a much less verbose format. These are listed here:

[1] UNIX is a trademark of AT&T Bell Laboratories.

chkmf **(1)** Check the *.mailfilter* syntax.

ckmail **(1)** Check status of your mail.

flagmail **(1)** Notify a server program of new mail.

mail **(1)** A very dumb mail submission program.

malias **(1)** Show what mailing lists expand to.

mlist **(1)** Distribution list maintenance program.

ttyalert **(1)** Alert of new mail on the terminal.

xalert **(1)** Alert of new mail in an X window.

resend **(1)** Resubmit a message to more recipients.

strip_addr **(1)** Remove comments and extraneous parts from an RFC 822 address.

*mailfilter***(5)** Brief description of the *.mailfilter* syntax.

Chapter 2

Local delivery in PP

The local channel is the mechanism by which PP delivers mail to users. There may be several of these channels tailored to specific mailbox formats. A user may have more than one local channel associated with him/her where this is useful, such as having two formats of mail (e.g., X.400 and RFC 822) delivered in different ways.

2.1 Local RFC 822-based Channel

The RFC 822 local channel has considerable control over the delivery of the message. In particular, the channel has a particular search path which is followed when the message is attempted to be delivered. If the message is considered delivered, or rejected by any one of these methods, then the further methods of delivery are attempted. The later methods of delivery are only attempted if the earlier ones are inconclusive. (This does not apply to the *.mailfilter* processing which normally processes the entire file.)

1. If the user has a *.mailfilter* file in his/her home directory and the file is owned by that user and not writable by anyone other than that user, the commands within that file are obeyed. The syntax is defined in Section 2.1.1. If the user does not have a *.mailfilter* or the actions in it do not deliver, reject, or temporarily fail the message, the next method is attempted.

2. If there is a system defined *.mailfilter* then the commands in that are obeyed. Again if this does not exists, or the actions are not conclusive, the next method is tried.

3. If the above two methods appear inconclusive, the message is written to a file in a well-defined place (system configured) surrounded by delimiters.

2.1.1 Format of the .mailfilter File

The *.mailfilter* file allows quite complex processing of incoming messages to be performed. The format takes the form of a small interpreted language in

which expressions can be tested and evaluated prior to performing actions. In particular conditional actions are the most useful.

The language is defined more formally in the grammar given in Figure 2.2. However, a more gentle introduction is given here with some examples of usage.

Commands

There are two basic commands that can be performed on a message. These commands can be performed any number of times so the message can be delivered in several places and/or to several processes.

The first command is `file`. This command takes an argument which is a string and the result is to attempt to copy the message surrounded by delimiters into the given file. This might be as in one of the following examples:

```
file "mailbox";
file "/archive/mailbox";
```

A variant of this is the `unixfile`. This writes into the given file, but in sendmail or traditional UNIX-style format; i.e., with a header line stating `From`.

```
unixfile ".mail";
```

Note that all statements in the *.mailfilter* file must be terminated with a semi-colon (";").

The second basic command is `pipe`. This command starts up a process and writes the message into it as standard input. This is shown in the next example.

```
pipe "rcvstore +inbox";
pipe "lpr -p -Ppostscript";
```

The system can tell if these commands have succeeded by the following methods. If the file writing works then a flag `delivered` is set to indicate success. Similarly, if the exit status from a `pipe` is zero, the delivery is considered a success and the flag updated. Additionally, as programs may report temporary failure, this is supported. A program may return a temporary failure by exiting with a value between 1 and 127, in which case the delivery attempt is aborted and retried at a later date. An exit status greater than 127 is taken as permanent failure.

Sometimes, though, you may wish to run a command and ignore the results; that is, to have the system not update the status of the delivery. This can be achieved by setting a variable as described later, but for simplicity, if you precede the command with the keyword `ignore`, then no notice will be taken of the return status.

```
ignore pipe "echo hello";
ignore file "/dev/null";
ignore pipe "rcvalert 'you have new mail'";
```

One last feature of these commands, header variables (described below) may be substituted in the strings associated with these commands. These are not too useful for `file` commands, but can be useful as arguments for processes. Care must be taken that appropriate quoting of these is done in case they contain special characters. As any string containing a "`$`" is substituted, a $ must be inserted by doubling it, e.g., "`$$`". If you require a quote "`"`" character in the string, it must be preceeded by a backslash character: "\".

Normally, the commands are interpreted by `/bin/sh -c <command>` but if the *822-local* channel is running in restricted mode, the command is directly executed.

Conditional Expressions

The real power of the *.mailfilter* comes with conditional expresssions. These allow the user to select particular rules depending, for instance, on certain attributes of the message.

The most basic form of condition is an `if` statement with a simple comparison. This allows you to compare two strings for equality. This takes the form of one of the following:

```
if ( <string1> == <string2> )
    <expression> ;

if ( <string1> == <string2> ) {
    <expression1> ;
    <expression2> ;
}

if ( <string1> == <string2> )
    <expression> ;
else if ( <string1> == <string3> )
    <expression> ;
else
    <expression> ;
```

These examples show a number of things. First, where you can have a single expression, you may have several expressions by enclosing them in braces[1]; this is an example of a compound expression.

Second, by use of the else statement, default actions can be taken. Finally, the `else if` allows a multi way branching construct.

Conditional expressions may also contain Boolean operators. The list is described below.

Operator	Meaning
`==`	Is equal to
`!=`	Is not equal to
`&&`	Logical and
`||`	Logical or
`!`	Logical not

[1] Any resemblance to the *C* programming language is entirely intentional!

The conditional construct begs the question "what conditions can be tested?" These are discussed in the next section.

Variables

There are two basic types of variables that can be accessed:

1. System variables. These are variables maintained by the *822-local* channel and may be updated after actions are taken.

2. Header fields. When delivering a message, all the current header fields are available for examination.

The system variables are the simplest and there are very few of these. The most used system variable is `delivered`. This variable normally contains the status of the delivery attempt. This is what the system thinks the state of your message is. This may be one of the following values.

Value	Meaning
`TRUE`	The message has been delivered
`OK`	The same as above
`FALSE`	The message has not been delivered
`PERMFAIL`	The message has permanently failed to be delivered. A delivery report should be sent back if required.
`TEMPFAIL`	The message has failed to be delivered for some temporary reason. Another attempt may succeed.

This may be used in conditional expressions to test for delivery, or in expressions to reset the delivery state. For example:

```
if ( delivered == FALSE) file "foo";

if ( delivered != TRUE ) file "bar";

# If not delivered - reject the message
if ( delivered == FALSE) delivered = PERMFAIL;
```

Another system type variable is `PATH`. If this variable is assigned a value, then it is placed in the environment as the default path. This is used to search for commands when executing `pipe` commands. It is initially set to nothing and the user's shell default search path is used.

Note: This variable is not used if the 822-local channel is configured in restricted mode.

The second class of variables, which are more useful for automatic filing and delivery, are the variables derived from the messages header. These variables evaluate to the contents of the particular field, or to the null string if there is no such variable. These variables can be compared against quoted strings, but such usage is not generally useful. Such comparisons require