Index

#, 7
822-local, 5, 6

A .alert, 11

B /bin/mail, 18
body, 10
Boolean operators, 5

C checking .mailfilter, 8
checkmail, 18
chkmf, 2
ckmail, 2
ckmf, 9
comment, 7
compound statement, 5
conditionals, 5

D delivered, 4, 6
delivery states, 6

E else, 5
else if, 5
exit, 7
extra, 10

F file, 4
flagmail, 2, 9, 10, 11, 12
flock, 14
From, 10

H header fields, 6

I if, 5
ignore, 4

L LaTeX, 20

M mail, 2
mailfilter, 2
.mailfilter, 3, 10
malias, 2, 17
mlist, 2, 16, 17

P PATH, 6
pipe, 4
print, 7
printf, 9

R rcvstore, 7
rcvtrip, 12
regexp, 7
regular expressions, 7
resend, 2, 13

S size, 10
strip_addr, 2
strip_addr, 14
system variables, 6

T ttyalert, 2, 9, 10, 11, 12
ttybiff, 11, 12

U unixfile, 4

V variables, 6
system, 6

X X windows, 9
xalert, 2, 9, 11, 12

-h mtaname This flag prompts *checkmail* to connect to the named mta. By default, *checkmail* attempts to connect to an mta running on the local machine.

Here are two example displays from the *checkmail* tool.

```
john% xcheckmail -h sheriff
Please wait while attempt to connect to sheriff....connected
The messages from pac on sheriff are as follows:
msg.aa03130 is pending delivery to p.pat@cs.nott.ac.uk

john% xcheckmail -h sheriff -a
Please wait while attempt to connect to sheriff....connected
The messages to or from pac on sheriff are as follows:
msg.aa03130 is pending delivery to p.pat@cs.nott.ac.uk
msg.aa03140 is from  P.Pat@cs.nott.ac.uk
```

Chapter 4

Miscellaneous Utilities

There are a number of small utility programs provided with PP that may be of interest to end users. These are described here.

4.1 A /bin/mail replacement

PP comes with a very simple user interface for submitting messages. It has approximately the functionality normally assumed by */bin/mail* except that reading mail is not supported - only submission. That is, it takes a list of addresses on the command line, and reads the body of the message on the standard input. By default all the addresses are marked as `To:` recipients. The command has a few options, each option stays in force until overriden.

`-t:` The following arguments, up to the next flag, are `To:` addresses.

`-c:` The following arguments are `Cc:` addresses.

`-s:` The following arguments are text for the `Subject:` line.

This program is most useful in scripts and programs as a simple way of sending a message.

4.2 Message status checkup

PP provides a tool, *checkmail*, that connects to an mta and finds out about messages related to the user.

There are two flags to alter *checkmail*'s behaviour.

-a This flag prompts *checkmail* to find out about messages to and from the user. By default, *checkmail* only finds out about messages from the user.

enters a loop. Within this loop, you choose the lists you wish to modify. When you have finished modifying the required lists, you can choose to `quit` the loop and exit *mlist*.

3.2.1 *malias*.

The *malias* tool is a passive version of *mlist*. Using the *malias* tool, you can only view the members of specified lists. You are unable to modify, in any way, the contents of a list.

add A list of possible mail addresses is expected with this option (either following the add command or on the next line). *mlist* checks that each address is a valid mail address and that the normalised address does not already exist in the list. Address that pass both these tests are added to the list.

remove A regular expression is expected with this option. Any address that matches this expression is a candidate of removing from the list. *mlist* asks the user whether each candidate should be removed.

find As with the *remove* option, a regular expression is expected. Any address that matches the expressions is printed out.

verify This option goes through the list checking that each member is a valid mail address.

As well as these maintainance options, there are a few inspection options

print This option prints out all the information concerning the list. This information consists of all the members of the list, the list moderator's mail address, and the user ids of the moderators of the list.

list This option displays all the lists know to the local PP system. Each list also has a short description of its purpose.

help This option displays all the options that are available to you.

If the list you require does not exist, there is one further option, `create`. This option prompts you for all the details of the list you require and posts them to the postmaster. If the postmaster decides to grant you this list, he or she will then do the appropriate administration to create the list. Once the postmaster has done this, you can use *mlist* to manipulate the list.

If you have done this and the list you requested now appears in the output of the `list` option but *mlist* still thinks it does not know about that list, then it is possible that the postmaster overlooked creating an empty file to contain the list when he or she was adding the list to PP's tables.

3.2 How *mlist* can run.

mlist takes each list specified on the command line in turn, and enables you to modify them. If no lists are specified on the command line, *mlist*

Chapter 3

Distribution List Maintance.

PP provides an interactive tool for maintaining distribution lists, *mlist*. This tool allows you, the user, to view, modify and generally manage the members of a given list. What you are allowed to do to a list, depends on the access rights set on that list.

The access rights for a list are set when the one of the managers of that list first uses *mlist* to manage the list. There are four modes of access rights:

free anyone can add or remove members of the list;

public anyone can add themselves to or remove themselves from the list;

private only managers of the list can add or remove members;

secret only managers of the list can view or alter the members of a list.

If you are a manager of a list, the access rights are bypassed and you are given *free* access to the list.

3.1 What you can and can't do with *mlist*.

For all modes of access except *free*, the maintainance of the list is done via a series of questions. For example,

```
You (pac) are not in this list
Do you wish to be added ? [y/n]
```

With *free* access, the maintainance is done via a menu of options

```
-- Mail filter grammer

<statementlist> ::= <statement>
                  | <statatament> <statementlist>

<statement> ::= <expression> ";"
                  | ";"
                  | "print" <prexp> ";"
                  | "if" <condition> <statement>
                  | "if" <condition> <statement> "else" <statement>
                  | "{" <statementlist> "}"

<condition> ::= "(" <expression> ")"

<expression> ::=  "(" <expression> ")"
                  | <expression> "==" <regex>
                  | <expression> "!=" <regex>
                  | <variable> "=" <expression>
                  | <expression> "==" <expression>
                  | <expression> "!=" <expression>
                  | <expression> "||" <expression>
                  | <expression> "&&" <expression>
                  | "!" <expression>
                  | [ "ignore" ] "file" <string>
                  | [ "ignore" ] "pipe" <string>
                  | "exit" [ <expression> ]
                  | <field>
                  | <variable>
                  | <string>

<prexp> ::= <expression>
                  | <prexp> "," <expression>

<string> ::= quoted string of characters (e.g. "abced")

<variable> ::= list of characters of the form [a-zA-Z_][a-zA-Z0-9_]*

<field> ::= "$(" list of characters and - ")"

<regex> ::= "/" regular expression (see regexp(3)) "/"
```

Figure 2.2: Grammer for the mailfilter file

2.6 Delivery tools

There are one or two small tools that are provided to help with delivery processing.

2.6.1 strip_addr

The *strip_addr* utility takes a list of address either on the command line or on the standard input and issues those same addresses in a basic form. In particular all address comments are removed as are name and list constructs. This utility may be useful in processing mail where only the basic address is wanted.

strip_addr takes two optional flags. These are:

`-p:` Interpret the `%` character as a routing construct. If this flag is not present, the `%` character will be taken as just another character.

`-r:` Strip out routes as well.

An example of use might be:

```
strip_addr 'Julian Onions <jpo(thats me)@there.com>'
jpo@there.com
```

2.6.2 flock

The *flock* utility takes a single file followed by a command. The file is first locked in the current PP locking style, and then the command is executed. When the command exits, the file is unlocked and *flock* exists with the returned status of the command.

`-e value:` Set the exit code depending on this value. A value of 0 means the program always exits with failure (the default). If the value is 1, the exit code is always good. A value of 2 will give an exit status according to the following table:

Condition	exit code
A message was sent	1
A message was not sent as this this sender has already received a message	2
A message was not sent as this message was not sent directly to this user	3
A parse error was found	4
Anything else	99

`-f:` Set the text of the from field (defaults to `MESSAGE AGENT`).

`-l file:` Set the database to this filename (default *triplog*).

`-n:` Use this file as the text of the message to send back after the standard preamble (default *ttripnote*).

`-s:` Use this file to determine the users signature (default *.signature*).

`-t text:` Modify the preamble to use this line (several of these may be given).

`-u user:` Pretend to be this user when answering.

2.5 Forwarding mail

There is a simple program which will forward mail to other recipients. This is called *resend*. This program takes a message as standard input with extra recipients given on the command line. The program will then submit the message as a `resent` message. The new recipients will appear in the `resent-to` and `resent-cc` lines as appropriate.

There are two flags that control the operation of this program:

`-t:` The next recipients should be specified as `Resent-To:` recipients.

`-c:` The next recipients should be specified as `Resent-Cc:` recipients.

2.3.2 Examples

Here are a few examples of use.

First, the simple notification that should be put in the *.mailfilter* file.

```
ignore pipe "flagmail -s $(size)";
```

This does the most basic notification. If more control is required over the displayed message more complex strings can be built up.

```
if ($(subject) == /request/) {
    file "request-mbox";
    ignore pipe "flagmail -s $(size) -e REQUEST";
}
```

Finally, if you are adventurous, you might like to play with the format of the message given. For instance

```
ignore pipe
  "flagmail -s $(size) -f '%5size %15from (%20subject) ** %body'";
```

This would give an output in the following format for a typical message

```
1023  Julian Onions   (Re: Delivery Report ) ** Also
```

2.4 Automatic Vacation Notices

PP provides a simple program to notify originators that the recipients will not read their message for some time. This is typically installed when the recipient is expecting to be unable to read mail for some time, such as going on vacation.

The program, *rcvtrip*, is installed in the *.mailfilter* file in a fashion similar to the following.

```
ignore pipe "rcvtrip"
```

The program reads the message and check to see if the recipient is mentioned directly in either the `to` or `cc` line. If so, and the originator has not been replied to in a similar style before, then a reply is composed. This consists of some standard text, followed by the contents of the file *tripnote* in the users home directory.

There are several flags to this program.

`-F:` Always reply to the user, even if the message is not sent directly to this user.

`-port number:` Set the UDP port numbr to the given number. The default is to choose a random unused UDp port.

ttybiff supports the following flags similar to *xalert*:

`-f hostfile:` use the given file to write the host and port number in instead of the default.

`-u username:` as for *xalert*'s `-username` switch.

`-p port:` as for *xalert*'s `-port` switch.

`-s:` only display on the standard output, not on other terminals where the user is logged on.

2.3.1 Use of flagmail with xalert and ttybiff

`Xalert` works with *flagmail* or *ttybiff* to provide the notification. `Flagmail` is responsible for building up the format string and sending it to either *xalert* or *ttybiff*. This is achieved using a UDP connection. `flagmail` contacts the server program by looking for a file with a hostname and udp port in (by default *.alert*). If this exists then it attempts to contact a server on that host to display the message. If either of these steps fail, then it reverts to the behaviour of *ttyalert* except that there is a delay while it waits for UDP timeouts to occur.

The *.alert* has the format

```
<hostname> <udp-port-no>
```

flagmail will use this information to contact the remote process, which can be any process which understands the *flagmail* protocol.

The *xalert* program is just a simple display process that displays the messages sent by *flagmail*. It displays these message in a text widget which is scrollable. There are three buttons.

`Quit:` which does the obvious thing,

`Clear:` which clears the message window, and

`Reset:` which rewrites the *.alert* file - used if two xalerts are started or the *.alert* file is removed.

The *ttybiff* attempts to locate the user in the same manner as *ttyalert* or if that fails to display the message on the standard output.

```
if (delivered)
    ignore pipe "flagmail -s $(size)"
```

See the example in Figure 2.1 on page 8 for a more complex example of usage.

Flags

Both *flagmail* and *ttyalert* respond to a number of command line options to change the behavior. These are:

`-d:` Runs the program in a debugging mode. This stops the program running in the background. Only useful if you are trying to debug the program.

`-e string:` Set the `extra` variable. This is used to override the default extra information which is normally `NEW:`.

`-f format:` Set the entire format string to the given string. This is how you override the internal format string with something entirely different.

`-s size:` Set the message size parameter.

`-w width:` Set the default output line width. the line produced will be truncated to this width (default 80 characters).

In addition the *flagmail* program has two other options to tailor the interaction with *xalert* or other servers.

`-h hostfile:` Use the given file to read the host and port to communicate with. This overrides the default of *.alert* in the home directory.

`-p number:` Use the following number to override the UDP port found in the file.

xalert has only a few command line flags besides the generic X flags. These are:

`-filename hostfile:` use the given file to write the host and port number in instead of the default (*.alert*).

`-user username:` *xalert* filters out requests destined for the given user. Normally it will not receive messages destined for another user, but this will change *xalert*'s idea of the current user anyway.

Both *flagmail* and *ttyalert* use the same method of preparing and formatting the data to be displayed. This is done by using a format string to indicate how the information should be layed out. The format string can contain verbatim text, header fields and other attributes of the message.

Fields within the format string are indicated by percent (`%`) symbols. The general format bares some relationship to the *printf*(3) format.

```
%[digits]string
```

The digits are optional and indicate how large the field should be padded out to or truncated to. The string is any message field, or one of a set of special fields. The case of the string is not sensitive.

The special fields are

Field	Meaning
`body`	The first few characters of the body of the message
`size`	The size in bytes of the message
`extra`	A special field set on the command line
`from`	the from field is treated in a special way

The `body` field is the first few characters of the body with all new lines removed and multiple white space compressed to single spaces.

The `size` is set on the command line by the use of an optional flag. This is usually done in the *.mailfilter* file by use of the `$(size)` variable.

The `extra` field is used to allow simple changes to the default format string. This just allows a simple flag to adjust the output from the default format string rather than rewriting the format string. If not set, it defaults to the value `NEW:␣`.

The `From` field is treated slightly specially, in that if there is a name component in front of the opening angle bracket, then the rest of the field is ignored.

The default format string is the following

```
%extra(%size) %20from %subject << %50body
```

This gives output something like the following

```
NEW: (3402) Steve Kille  Re: state of unix-niftp? << I
```

The whole output line is truncated to the number of characters set on the command line (defaulting to 80). This allows the later parts of the string to be large in the assurance that they will get chopped off if they exceed the given size.

An example usage in a *.mailfilter* file might look like the following

It is *STRONGLY* recommended that you check your *.mailfilter* file each time it is modified. This will certainly pick out syntax errors and similar, however it will not discover errors such as the following which may be common causes of problems.

- Unable to find the program to be run (not in search path).
- The process that is executed aborts. This may be because some things in the environment are missing - if this is the case it is best to wrap the process in a shell script. Alternatively the process may expect the standard input/ouput be connected to a terminal. This is difficult to get around.

An example run of *ckmf* might look like this:

```
john% ckmf ~jpo/.mailfilter < /dev/null
pipe into process 'rcvstore +inbox'
pipe into process 'flagmail -s 0 -e 'inbox: ''
Result: delivered OK
john%
```

2.3 New message notification

PP provides a couple of simple tools for alerting the user of new messages arriving. Both methods share a lot of common features (and code for that matter).

The basic idea of the notification is to run a process from the delivery process which reads the message, extracts some information, formats it and then gives it to the user. This may be either by displaying this formatted information on the users terminal, or on a client of an *X windows* display.

There are four process involved. they are:

`ttyalert:` This process reads gathers data from a message and simply displays it on the users terminal if logged in.

`xalert:` This process runs as a client of an X windowing system server. It displays messages sent to it by *flagmail*.

`ttybiff:` This process runs in the background normally. it displays messagessent to it by *flagmail*.

`flagmail:` This process gathers the data from the message, formats it and attempts to pass it to *xalert* (or similar process obeying the same protocol). If it fails it emulates the *ttyalert* program.

```
PATH="/usr/local/lib/mh:/usr/local/bin:/usr/bin:/bin:/usr/ucb";
folder = FALSE;
if ($(to) == /jpo/ || $(to) == /onions/) {
        pipe "rcvstore +to"; folder = "to";
} else if ($(sender) == /bind-request/) {
        pipe "rcvstore +bind"; folder = "bind";
} else if ($(sender) == /mailgroup/) {
        pipe "rcvstore +janet/mailgroup"; folder = "janet/mailgroup";
} else if ($(sender) == /namedropper/) {
        pipe "rcvstore +mail/namedroppers"; folder = "namedroppers";
} else if ($(sender) == /pp.*-request/) {
        pipe "rcvstore +pp"; folder = "pp";
}
if (!delivered) {
        pipe "rcvstore +inbox";
        if (!delivered) {
                ignore pipe "flagmail -f 'DELIVERY FAILED'";
                exit (TEMPFAIL);
        }
        folder = "inbox";
}
if (folder)
        ignore pipe "flgamail -s $(size) -e '$(folder): '";
```

Figure 2.1: Example .mailfilter file

```
if ($(to) == /jpo/)
pipe "rcvstore +to";
else
pipe "rcvstore +inbox";
ignore pipe "flagmail -s $(size)";
```

An more complex example of a *.mailfilter* file is shown in Figure 2.1. The grammer is specified in Figure 2.2.

2.2 Mailfilter checkup tool

As writing *.mailfilter* files can be error prone, a tool to check that the contents conform to the laguage is provided. This is invoked with a given filename and a message on the standard input. The file is parsed and then interpreted and the actions appropriate for the given message are indicated.

```
ckmf ~/.mailfilter < msg
```

```
if ( $(from) == /namedroppers/ || $(from) == /bind/) pipe "rcvstore
+mailadmin"; else pipe "rcvstore +inbox";
```

Finally there are some variables that may appear to be header fields but which are actually derived from other sources. These are:

`size`	The size of the message in bytes as a string
`return-path`	The original sender of the message
`mailbox`	The mailbox file associated with this user
`recipient`	The address of this recipient
`userid`	Either the username or userid of this recipient
`groupid`	The group id of this recipient
`channelname`	The name of this delivery channel
`hostname`	The name of the host

Other features

There are a few other features that can be used in the *.mailfilter* file. These are summarised below.

`exit:` This statement exits the processing of the *.mailfilter* file. With no argument it exits with the current value of the `delivered` variable. If an argument is given, then that is taken as the success or failure of the delivery attempt. The argument should be one of `OK`, `TEMPFAIL`, `PERMFAIL` or a variable that evaluates to one of these.

`print:` Normally, this statement does nothing and is ignored. However when running the checking program, it will echo its arguments on the standard output. This may be useful in debugging.

`#:` This symbol starts a comment, which finishes at the end of the line.

2.1.2 Examples

The following are some simple examples of *.mailfilter* files.

This first example shows just a simple delivery making use of the MH *rcvstore* (1) utility.

```
pipe "rcvstore +inbox";
```

The next example shows slightly more complex behviour, whereby some mailfiltering occurs, and an alert is posted.

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 to 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 users shell default search path is used.

Note: This variable is not used if the 822-local channel is configure 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 exact matches including white space with the exception that letter case is not significant. These variables are distinguished by always starting with a dollar symbol ('`$`') followed by the field name in parenthases. (e.g. `$(to)` or `$(from)`)

Much more useful is comparison against a regular expression. The regular expressions used are those in *regexp*(3). These match against substrings of the fields by default and more complex patterns can be built up. Regular expression must always appear on the right hand side of the expression and are delimited by slash marks ('`/`'). An example of such usage shows the real power of the *.mailfilter* file:

```
        <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.

Secondly, 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

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.

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

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 large than 127 is taken as permanent failure.

Sometimes though, you may wish to run a command and ignore the results – that is to not have the system 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 - "\".

In the normal case, 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 on certain attributes of the message for instance.

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> )
```

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 grammer 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

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 anyone 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 to methods appear inconclusive, the message is written to a file in a well defined place (system configured) surrounded by

clude 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:

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 etraneous parts from an RFC-822 address.

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

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

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 users 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 in-

List of Figures

2.1 Example .mailfilter file 8
2.2 Grammer for the mailfilter file 15

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 . . . 8
- 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 . . . 12
- 2.4 Automatic Vacation Notices . . . 13
- 2.5 Forwarding mail . . . 14
- 2.6 Delivery tools . . . 14
  - 2.6.1 strip_addr . . . 14
  - 2.6.2 flock . . . 14

3 Distribution List Maintance. **16**
- 3.1 What you can and can't do with *mlist*. . . . 16
- 3.2 How *mlist* can run. . . . 17
  - 3.2.1 *malias*. . . . 17

4 Miscellaneous Utilities **18**
- 4.1 A /bin/mail replacement . . . 18
- 4.2 Message status checkup . . . 18

PP

Volume 3: Users Guide

Pete Cowen Julian Onions

March 18, 1991