discard this page

The RAND *MH*
Message Handling System:
User's Manual

UCI Version

December 5, 1995
6.8.3a #10[UCI]

1. INTRODUCTION

Although people can travel cross-country in hours and can reach others by telephone in seconds, communications still depend heavily upon paper, most of which is distributed through the mails.

There are several major reasons for this continued dependence on written documents. First, a written document may be proofread and corrected prior to its distribution, giving the author complete control over his words. Thus, a written document is better than a telephone conversation in this respect. Second, a carefully written document is far less likely to be misinterpreted or poorly translated than a phone conversation. Third, a signature offers reasonable verification of authorship, which cannot be provided with media such as telegrams.

However, the need for fast, accurate, and reproducible document distribution is obvious. One solution in widespread use is the telefax. Another that is rapidly gaining popularity is electronic mail. Electronic mail is similar to telefax in that the data to be sent are digitized, transmitted via phone lines, and turned back into a document at the receiver. The advantage of electronic mail is in its compression factor. Whereas a telefax must scan a page in very fine lines and send all of the black and white information, electronic mail assigns characters fixed codes which can be transmitted as a few bits of information. Telefax presently has the advantage of being able to transmit an arbitrary page, including pictures, but electronic mail is beginning to deal with this problem. Electronic mail also integrates well with current directions in office automation, allowing documents prepared with sophisticated equipment at one site to be quickly transferred and printed at another site.

Currently, most electronic mail is intraorganizational, with mail transfer remaining within one computer. As computer networking becomes more common, however, it is becoming more feasible to communicate with anyone whose computer can be linked to your own via a network.

The pioneering efforts on general-purpose electronic mail were by organizations using the DoD ARPAnet[1]. The capability to send messages between computers existed before the ARPAnet was developed, but it was used only in limited ways. With the advent of the ARPAnet, tools began to be developed which made it convenient for individuals or organizations to distribute messages over broad geographic areas, using diverse computer facilities. The interest and activity in message systems has now reached such proportions that steps have been taken within the DoD to coordinate and unify the development of military message systems. The use of electronic mail is expected to increase dramatically in the next few years. The utility of such systems in the command and control and intelligence environments is clear, and applications in these areas will probably lead the way. As the costs for sending and handling electronic messages continue their rapid decrease, such uses can be expected to spread rapidly into other areas and, of course, will not be limited to the DoD.

A message system provides tools that help users (individuals or organizations) deal with messages in various ways. Messages must be composed, sent, received, stored, retrieved, forwarded, and replied to. Today's best interactive computer systems provide a variety of word-processing and information handling capabilities. The message handling facilities should be well integrated with the rest of the system, so as to be a graceful extension of overall system capability.

The message system described in this report, *MH*, provides most of the features that can be found in other message systems and also incorporates some new ones. It has been built on the UNIX time-sharing system[2], a popular operating system for the DEC PDP-11[1] and VAX-

11 classes of computers. A ‘‘secure’’ operating system similar to UNIX is currently being developed[3], and that system will also run *MH*.

This report provides a complete description of *MH* and thus may serve as a user’s manual, although parts of the report will be of interest to non-users as well. Sections 2 and 3, the Overview and Tutorial, present the key ideas of *MH* and will give those not familiar with message systems an idea of what such systems are like.

MH consists of a set of commands which use some special files and conventions. The final section is divided into three parts. The first part covers the information a user needs to know in addition to the commands. Then, each of the *MH* commands is described in detail. Finally, other obscure details are revealed. A summary of the commands is given in Appendix A, and the syntax of message sequences is given in Appendix B.

A novel approach has been taken in the design of *MH*. Instead of creating a large subsystem that appears as a single command to the user (such as MS[4]), *MH* is a collection of separate commands which are run as separate programs. The file and directory system of UNIX are used directly. Messages are stored as individual files (datasets), and collections of them are grouped into directories. In contrast, most other message systems store messages in a complicated data structure within a monolithic file. With the *MH* approach, UNIX commands can be interleaved with commands invoking the functions of the message handler. Conversely, existing UNIX commands can be used in connection with messages. For example, all the usual UNIX editing, text-formatting, and printing facilities can be applied directly to individual messages. MH, therefore, consists of a relatively small amount of new code; it makes extensive use of other UNIX software to provide the capabilities found in other message systems.

[1] PDP and VAX are trademarks of Digital Equipment Corporation.

2. OVERVIEW

There are three main aspects of *MH* : the way messages are stored (the message database), the user's profile (which directs how certain actions of the message handler take place), and the commands for dealing with messages.

Under *MH*, each message is stored as a separate file. A user can take any action with a message that he could with an ordinary file in UNIX. A UNIX directory in which messages are stored is called a folder. Each folder contains some standard entries to support the message-handling functions. The messages in a folder have numerical names. These folders (directories) are entries in a particular directory path, described in the user profile, through which *MH* can find message folders. Using the UNIX ''link'' facility, it is possible for one copy of a message to be ''filed'' in more than one folder, providing a message index facility. Also, using the UNIX tree-structured file system, it is possible to have a folder within a folder, nested arbitrarily deep, and have the full power of the *MH* commands available.

Each user of *MH* has a user profile, a file in his **$HOME** (initial login) directory called *.mh_profile*. This profile contains several pieces of information used by the *MH* commands: a path name to the directory that contains the message folders and parameters that tailor *MH* commands to the individual user's requirements. There is also another file, called the user context, which contains information concerning which folder the user last referenced (the ''current'' folder). It also contains most of the necessary state information concerning how the user is dealing with his messages, enabling *MH* to be implemented as a set of individual UNIX commands, in contrast to the usual approach of a monolithic subsystem.

In *MH*, incoming mail is appended to the end of a file in a system spooling area for the user. This area is called the mail drop directory, and the file is called the user's mail drop. Normally when the user logins in, s/he is informed of new mail (or the *MH* program *msgchk* may be run). The user adds the new messages to his/her collection of *MH* messages by invoking the command *inc*. The *inc* (incorporate) command adds the new messages to a folder called ''inbox'', assigning them names which are consecutive integers starting with the next highest integer available in inbox. *inc* also produces a *scan* summary of the messages thus incorporated. A folder can be compacted into a single file, for easy storage, by using the *packf* command. Also, messages within a folder can be sorted by date and time with the *sortm* command.

There are four commands for examining the messages in a folder: *show*, *prev*, *next*, and *scan*. The *show* command displays a message in a folder, *prev* displays the message preceding the current message, and *next* displays the message following the current message. *MH* lets the user choose the program that displays individual messages. A special program, *mhl*, can be used to display messages according to the user's preferences. The *scan* command summarizes the messages in a folder, normally producing one line per message, showing who the message is from, the date, the subject, etc.

The user may move a message from one folder to another with the command *refile*. Messages may be removed from a folder by means of the command *rmm*. In addition, a user may query what the current folder is and may specify that a new folder become the current folder, through the command *folder*. All folders may be summarized with the *folders* command. A message folder (or subfolder) may be removed by means of the command *rmf*.

A set of messages based on content may be selected by use of the command *pick*. This command searches through messages in a folder and selects those that match a given set of criteria. These messages are then bound to a ''sequence'' name for use with other *MH* commands.

The *mark* command manipulates these sequences.

There are five commands enabling the user to create new messages and send them: *comp*, *dist*, *forw*, *repl*, and *send*. The *comp* command provides the facility for the user to compose a new message; *dist* redistributes mail to additional addressees; *forw* enables the user to forward messages; and *repl* facilitates the generation of a reply to an incoming message. The last three commands may optionally annotate the original message. Messages may be arbitrarily annotated with the *anno* command. Once a draft has been constructed by one of the four above composition programs, a user–specifiable program is run to query the user as to the disposition of the draft prior to sending. *MH* provides the simple *whatnow* program to start users off. If a message is not sent directly by one of these commands, it may be sent at a later time using the command *send*. *MH* allows the use of any UNIX editor when composing a message. For rapid entry, a special editor, *prompter*, is provided. For programs, a special mail-sending program, *mhmail*, is provided.

MH supports a personal aliasing facility which gives users the capability to considerably shorten address typein and use meaningful names for addresses. The *ali* program can be used to query *MH* as to the expansion of a list of aliases. After composing a message, but prior to sending, the *whom* command can be used to determine exactly who a message would go to.

MH provides a natural interface for telling the user's shell the names of *MH* messages and folders. The *mhpath* program achieves this capability.

Finally, *MH* supports the UCI BBoards facility. *bbc* can be used to query the status of a group of BBoards, while *msh* can be used to read them. The *burst* command can be used to ‘‘shred’’ digests of messages into individual messages.

All of the elements summarized above are described in more detail in the following sections. Many of the normal facilities of UNIX provide additional capabilities for dealing with messages in various ways. For example, it is possible to print messages on the line-printer without requiring any additional code within *MH* . Using standard UNIX facilities, any terminal output can be redirected to a file for repeated or future viewing. In general, the flexibility and capabilities of the UNIX interface with the user are preserved as a result of the integration of *MH* into the UNIX structure.

3. TUTORIAL

This tutorial provides a brief introduction to the *MH* commands. It should be sufficient to allow the user to read his mail, do some simple manipulations of it, and create and send messages.

A message has two major pieces: the header and the body. The body consists of the text of the message (whatever you care to type in). It follows the header and is separated from it by an empty line. (When you compose a message, the form that appears on your terminal shows a line of dashes after the header. This is for convenience and is replaced by an empty line when the message is sent.) The header is composed of several components, including the subject of the message and the person to whom it is addressed. Each component starts with a name and a colon; components must not start with a blank. The text of the component may take more than one line, but each continuation line must start with a blank. Messages typically have ‘‘To:’’, ‘‘cc:’’, and ‘‘Subject:’’ components. When composing a message, you should include the ‘‘To:’’ and ‘‘Subject:’’ components; the ‘‘cc:’’ (for people you want to send copies to) is not necessary.

The basic *MH* commands are *inc*, *scan*, *show*, *next*, *prev*, *rmm*, *comp*, and *repl*. These are described below.

inc

When you get the message ‘‘You have mail’’, type the command *inc*. You will get a ‘‘scan listing’’ such as:

```
7+   7/13  Cas       revival of measurement work
8   10/ 9  Norm      NBS people and publications
9   11/26  To:norm   question ≪Are there any functions
```

This shows the messages you received since the last time you executed this command (*inc* adds these new messages to your inbox folder). You can see this list again, plus a list of any other messages you have, by using the *scan* command.

scan

The scan listing shows the message number, followed by the date and the sender. (If you are the sender, the addressee in the ‘‘To:’’ component is displayed. You may send yourself a message by including your name among the ‘‘To:’’ or ‘‘cc:’’ addressees.) It also shows the message’s subject; if the subject is short, the first part of the body of the message is included after the characters ≪.

show

This command shows the current message, that is, the first one of the new messages after an *inc*. If the message is not specified by name (number), it is generally the last message referred to by an *MH* command. For example,

show 5   will show message 5.

You can use the show command to copy a message or print a message.

show > x	will copy the message to file x.
show | lpr	will print the message, using the *lpr* command.
next	will show the message that follows the current message.
prev	will show the message previous to the current message.
rmm	will remove the current message.
rmm 3	will remove message 3.

comp

The *comp* command puts you in the editor to write or edit a message. Fill in or delete the ‘‘To:’’, ‘‘cc:’’, and ‘‘Subject:’’ fields, as appropriate, and type the body of the message. Then exit normally from the editor. You will be asked ‘‘What now?’’. Type a carriage return to see the options. Typing **send** will cause the message to be sent; typing **quit** will cause an exit from *comp*, with the message draft saved.

If you quit without sending the message, it will be saved in a file called <name>/Mail/draft (where <name> is your **$HOME** directory). You can resume editing the message later with ‘‘comp –use’’; or you can send the message later, using the *send* command.

comp –editor prompter

This command uses a different editor and is useful for preparing ‘‘quick and dirty’’ messages. It prompts you for each component of the header. Type the information for that component, or type a carriage return to omit the component. After that, type the body of the message. Backspacing is the only form of editing allowed with this editor. When the body is complete, type a carriage return followed by <EOT> (usually <CTRL-D>). This completes the initial preparation of the message; from then on, use the same procedures as with *comp* (above).

repl
repl n

This command makes up an initial message form with a header that is appropriate for replying to an existing message. The message being answered is the current message if no message number is mentioned, or n if a number is specified. After the header is completed, you can finish the message as in *comp* (above).

This is enough information to get you going using *MH*. There are more commands, and the commands described here have more features. Subsequent sections explain *MH* in complete detail. The system is quite powerful if you want to use its sophisticated features, but the foregoing commands suffice for sending and receiving messages.

There are numerous additional capabilities you may wish to explore. For example, the *pick* command will select a subset of messages based on specified criteria such as sender and/or subject. Groups of messages may be designated, as described in Sec. IV, under **Message Naming**. The file *.mh_profile* can be used to tailor your use of the message system to your needs and preferences, as described in Sec. IV, under **The User Profile**. In general, you may learn additional features of the system selectively, according to your requirements, by studying the relevant sections of this manual. There is no need to learn all the details of the system at once.

4. DETAILED DESCRIPTION

This section describes the *MH* system in detail, including the components of the user profile, the conventions for message naming, and some of the other *MH* conventions. Readers who are generally familiar with computer systems will be able to follow the principal ideas, although some details may be meaningful only to those familiar with UNIX.

THE USER PROFILE

The first time an *MH* command is issued by a new user, the system prompts for a ‘‘Path’’ and creates an *MH* ‘‘profile’’.

Each *MH* user has a profile which contains tailoring information for each individual program. Other profile entries control the *MH* path (where folders and special files are kept), folder and message protections, editor selection, and default arguments for each *MH* program. Each user of *MH* also has a context file which contains current state information for the *MH* package (the location of the context file is kept in the user’s *MH* directory, or may be named in the user profile). When a folder becomes the current folder, it is recorded in the user’s context. (Other state information is kept in the context file, see the manual entry for *mh–profile* (5) for more details.) In general, the term ‘‘profile entry’’ refer to entries in either the profile or context file. Users of *MH* needn’t worry about the distinction, *MH* handles these things automatically.

The *MH* profile is stored in the file *.mh_profile* in the user’s **$HOME** directory[1]. It has the format of a message without any body. That is, each profile entry is on one line, with a keyword followed by a colon (:) followed by text particular to the keyword.
⇒ *This file must not have blank lines.*
The keywords may have any combination of upper and lower case. (See the information of *mh–mail* later on in this manual for a description of message formats.)

For the average *MH* user, the only profile entry of importance is ‘‘Path’’. Path specifies a directory in which *MH* folders and certain files such as ‘‘draft’’ are found. The argument to this keyword must be a legal UNIX path that names an existing directory. If this path is not absolute (i.e., does not begin with a **/**), it will be presumed to start from the user’s **$HOME** directory. All folder and message references within *MH* will relate to this path unless full path names are used.

Message protection defaults to 644, and folder protection to 711. These may be changed by profile entries ‘‘Msg-Protect’’ and ‘‘Folder-Protect’’, respectively. The argument to these keywords is an octal number which is used as the UNIX file mode[2].

When an *MH* program starts running, it looks through the user’s profile for an entry with a keyword matching the program’s name. For example, when *comp* is run, it looks for a ‘‘comp’’ profile entry. If one is found, the text of the profile entry is used as the default switch setting until all defaults are overridden by explicit switches passed to the program as arguments. Thus the profile entry ‘‘comp: –form standard.list’’ would direct *comp* to use the file ‘‘standard.list’’ as the message skeleton. If an explicit form switch is given to the *comp* command, it will override the switch obtained from the profile.

[1] By defining the envariable **$MH**, you can specify an alternate profile to be used by *MH* commands.

[2] See *chmod* (1) in the *UNIX Programmer’s Manual* [5].

In UNIX, a program may exist under several names, either by linking or aliasing. The actual invocation name is used by an *MH* program when scanning for its profile defaults[3]. Thus, each *MH* program may have several names by which it can be invoked, and each name may have a different set of default switches. For example, if *comp* is invoked by the name *icomp*, the profile entry ‘‘icomp’’ will control the default switches for this invocation of the *comp* program. This provides a powerful definitional facility for commonly used switch settings.

The default editor for editing within *comp*, *repl*, *forw*, and *dist*, is usually *prompter*, but might be something else at your site, such as */usr/ucb/ex* or */bin/e*. A different editor may be used by specifying the profile entry ‘‘Editor: ’’. The argument to ‘‘Editor’’ is the name of an executable program or shell command file which can be found via the user’s $PATH defined search path, excluding the current directory. The ‘‘Editor:’’ profile specification may in turn be overridden by a ‘–editor <editor>’ profile switch associated with *comp*, *repl*, *forw*, or *dist*. Finally, an explicit editor switch specified with any of these four commands will have ultimate precedence.

During message composition, more than one editor may be used. For example, one editor (such as *prompter*) may be used initially, and a second editor may be invoked later to revise the message being composed (see the discussion of *comp* in Section 5 for details). A profile entry ‘‘<lasteditor>–next: <editor>’’ specifies the name of the editor to be used after a particular editor. Thus ‘‘comp: –e prompter’’ causes the initial text to be collected by *prompter*, and the profile entry ‘‘prompter–next: ed’’ names ed as the editor to be invoked for the next round of editing.

Some of the *MH* commands, such as *show*, can be used on message folders owned by others, if those folders are readable. However, you cannot write in someone else’s folder. All the *MH* command actions not requiring write permission may be used with a ‘‘read-only’’ folder.

Table 1 lists examples of some of the currently defined profile entries, typical arguments, and the programs that reference the entries.

[3] Unfortunately, the shell does not preserve aliasing information when calling a program, hence if a program is invoked by an alias different than its name, the program will examine the profile entry for it’s name, not the alias that the user invoked it as. The correct solution is to create a (soft) link in your *$HOME/bin* directory to the *MH* program of your choice. By giving this link a different name, you can use an alternate set of defaults for the command.

Table 1

PROFILE COMPONENTS

Keyword and Argument	*MH* Programs that use Component
Path: Mail	All
Current-Folder: inbox	Most
Editor: /usr/ucb/ex	*comp, dist, forw, repl*
Inbox: inbox	*inc, rmf*
Msg–Protect: 644	*inc*
Folder–Protect: 711	*inc, pick, refile*
<program>: default switches	All
prompter–next: ed	*comp, dist, forw, repl*

Path should be present. Current–Folder is maintained automatically by many *MH* commands (see the **Context** sections of the individual commands in Sec. IV). All other entries are optional, defaulting to the values described above.

MESSAGE NAMING

Messages may be referred to explicitly or implicitly when using *MH* commands. A formal syntax of message names is given in Appendix B, but the following description should be sufficient for most *MH* users. Some details of message naming that apply only to certain commands are included in the description of those commands.

Most of the *MH* commands accept arguments specifying one or more folders, and one or more messages to operate on. The use of the word ‘‘msg’’ as an argument to a command means that exactly one message name may be specified. A message name may be a number, such as 1, 33, or 234, or it may be one of the ‘‘reserved’’ message names: first, last, prev, next, and cur. (As a shorthand, a period (.) is equivalent to cur.) The meanings of these names are straightforward: ‘‘first’’ is the first message in the folder; ‘‘last’’ is the last message in the folder; ‘‘prev’’ is the message numerically previous to the current message; ‘‘next’’ is the message numerically following the current message; ‘‘cur’’ (or ‘‘.’’) is the current message in the folder. In addition, *MH* supports user–defined–sequences; see the description of the *mark* command for more information.

The default in commands that take a ‘‘msg’’ argument is always ‘‘cur’’.

The word ‘‘msgs’’ indicates that several messages may be specified. Such a specification consists of several message designations separated by spaces. A message designation is either a message name or a message range. A message range is a specification of the form name1–name2 or name1:n, where name1 and name2 are message names and n is an integer. The first form designates all the messages from name1 to name2 inclusive; this must be a non-empty range. The second form specifies up to n messages, starting with name1 if name1 is a number, or first, cur, or next, and ending with name1 if name1 is last or prev. This interpretation of n is overridden if n is preceded by a plus sign or a minus sign; +n always means up to n messages starting with name1, and –n always means up to n messages ending with name1. Repeated specifications of the same message have the same effect as a single specification of the message. Examples of specifications are:

1 5 7–11 22
first 6 8 next
first–10
last:5

The message name ‘‘all’’ is a shorthand for ‘‘first–last’’, indicating all of the messages in the folder.

In commands that accept ‘‘msgs’’ arguments, the default is either cur or all, depending on which makes more sense.

In all of the *MH* commands, a plus sign preceding an argument indicates a folder name. Thus, ‘‘+inbox’’ is the name of the user’s standard inbox. If an explicit folder argument is given to an *MH* command, it will become the current folder (that is, the ‘‘Current-Folder:’’ entry in the user’s profile will be changed to this folder). In the case of the *refile* command, which can have multiple output folders, a new source folder (other than the default current folder) is specified by ‘–src +folder’.

OTHER MH CONVENTIONS

One very powerful feature of *MH* is that the *MH* commands may be issued from any current directory, and the proper path to the appropriate folder(s) will be taken from the user’s profile. If the *MH* path is not appropriate for a specific folder or file, the automatic prepending of the *MH* path can be avoided by beginning a folder or file name with **/**, or with **./** or **../** component. Thus any specific absolute path may be specified along with any path relative to the current working directory.

Arguments to the various programs may be given in any order, with the exception of a few switches whose arguments must follow immediately, such as ‘–src +folder’ for *refile*.

Whenever an *MH* command prompts the user, the valid options will be listed in response to a <RETURN>. (The first of the listed options is the default if end-of-file is encountered, such as from a command file.) A valid response is any *unique* abbreviation of one of the listed options.

Standard UNIX documentation conventions are used in this report to describe *MH* command syntax. Arguments enclosed in brackets ([ ]) are optional; exactly one of the arguments enclosed within braces ({ }) must be specified, and all other arguments are required. The use of ellipsis dots (...) indicates zero or more repetitions of the previous item. For example, ‘‘+folder ...’’ would indicate that one or more ‘‘+folder’’ arguments is required and ‘‘[+folder ...]’’ indicates that 0 or more ‘‘+folder’’ arguments may be given.

MH departs from UNIX standards by using switches that consist of more than one character, e.g. ‘–header’. To minimize typing, only a unique abbreviation of a switch need be typed; thus, for ‘–header’, ‘–hea’ is probably sufficient, depending on the other switches the command accepts. Each *MH* program accepts the switch ‘–help’ (which **must** be spelled out fully) and produces a syntax description and a list of switches. In the list of switches, parentheses indicate required characters. For example, all ‘–help’ switches will appear as ‘‘–(help)’’, indicating that no abbreviation is accepted. Furthermore, the ‘–help’ switch tells the version of the *MH* program you invoked.

Many *MH* switches have both on and off forms, such as ‘–format’ and ‘–noformat’. In many of the descriptions which follow, only one form is defined; the other form, often used to nullify profile switch settings, is assumed to be the opposite.

MH COMMANDS

The *MH* package comprises several programs:

ali (1)	– list mail aliases
anno (1)	– annotate messages
bbc (1)	– check on BBoards
bboards (1)	– the UCI BBoards facility
burst (1)	– explode digests into messages
comp (1)	– compose a message
dist (1)	– redistribute a message to additional addresses
folder (1)	– set/list current folder/message
folders (1)	– list all folders
forw (1)	– forward messages
inc (1)	– incorporate new mail
mark (1)	– mark messages
mhl (1)	– produce formatted listings of MH messages
mhmail (1)	– send or read mail
mhook (1)	– MH receive–mail hooks
mhparam (1)	– print MH profile components
mhpath (1)	– print full pathnames of MH messages and folders
msgchk (1)	– check for messages
msh (1)	– MH shell (and BBoard reader)
next (1)	– show the next message
packf (1)	– compress a folder into a single file
pick (1)	– select messages by content
prev (1)	– show the previous message
prompter (1)	– prompting editor front end
rcvstore (1)	– incorporate new mail asynchronously
refile (1)	– file messages in other folders
repl (1)	– reply to a message
rmf (1)	– remove folder
rmm (1)	– remove messages
scan (1)	– produce a one line per message scan listing
send (1)	– send a message
show (1)	– show (list) messages
slocal (1)	– special local mail delivery
sortm (1)	– sort messages
vmh (1)	– visual front–end to MH
whatnow (1)	– prompting front–end for send
whom (1)	– report to whom a message would go

These programs are described below. The form of the descriptions conforms to the standard form for the description of UNIX commands.

NAME

ali – list mail aliases

SYNOPSIS

ali [–alias aliasfile] [–list] [–nolist] [–normalize] [–nonormalize] [–user] [–nouser] aliases ... [–help]

DESCRIPTION

Ali searches the named mail alias files for each of the given *aliases*. It creates a list of addresses for those *aliases*, and writes that list on standard output. If the '–list' option is specified, each address appears on a separate line; otherwise, the addresses are separated by commas and printed on as few lines as possible.

The '–user' option directs *ali* to perform its processing in an inverted fashion: instead of listing the addresses that each given alias expands to, *ali* will list the aliases that expand to each given address. If the '–normalize' switch is given, *ali* will try to track down the official hostname of the address.

The files specified by the profile entry ''Aliasfile:'' and any additional alias files given by the '–alias aliasfile' switch will be read. Each *alias* is processed as described in *mh–alias* (5).

Files

$HOME/.mh_profile	The user profile
/etc/passwd	List of users
/etc/group	List of groups

Profile Components

Path:	To determine the user's MH directory
Aliasfile:	For a default alias file

See Also

mh–alias(5)

Defaults

'–alias /usr/bs/mh-beta/lib/MailAliases'
'–nolist'
'–nonormalize'
'–nouser'

Context

None

Bugs

The '–user' option with '–nonormalize' is not entirely accurate, as it does not replace local nicknames for hosts with their official site names.

NAME

anno – annotate messages

SYNOPSIS

anno [+folder] [msgs] [–component field] [–inplace] [–noinplace] [–date] [–nodate] [–text body] [–help]

DESCRIPTION

Anno annotates the specified messages in the named folder using the field and body. Annotation is optionally performed by *dist*, *forw*, and *repl*, to keep track of your distribution of, forwarding of, and replies to a message. By using *anno*, you can perform arbitrary annotations of your own. Each message selected will be annotated with the lines

field: date
field: body

The '–nodate' switch inhibits the date annotation, leaving only the body annotation. The '–inplace' switch causes annotation to be done in place in order to preserve links to the annotated message.

The field specified should be a valid 822-style message field name, which means that it should consist of alphanumerics (or dashes) only. The body specified is arbitrary text.

If a '–component field' is not specified when *anno* is invoked, *anno* will prompt the user for the name of field for the annotation.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user's MH directory
Current–Folder:	To find the default current folder

See Also

dist (1), forw (1), repl (1)

Defaults

'+folder' defaults to the current folder
'msgs' defaults to cur
'–noinplace'
'–date'

Context

If a folder is given, it will become the current folder. The first message annotated will become the current message.

NAME

bbc – check on BBoards

SYNOPSIS

bbc [bboards ...] [–topics] [–check] [–read] [–quiet] [–verbose] [–archive] [–noarchive] [–protocol] [–noprotocol] [–mshproc program] [switches for *mshproc*] [–rcfile rcfile] [–norcfile] [–file BBoardsfile] [–user BBoardsuser] [–host host] [–help]

DESCRIPTION

bbc is a BBoard reading/checking program that interfaces to the BBoard channel.

The *bbc* program has three action switches which direct its operation:

The '–read' switch invokes the *msh* program on the named *BBoards*. If you also specify the '–archive' switch, then *bbc* will invoke the *msh* program on the archives of the named *BBoards*. If no *BBoards* are given on the command line, and you specified '–archive', *bbc* will not read your 'bboards' profile entry, but will read the archives of the ''system'' *BBoard* instead.

The '–check' switch types out status information for the named *BBoards*. *bbc* can print one of several messages depending on the status of both the BBoard and the user's reading habits. As with each of these messages, the number given is the item number of the last item placed in the BBoard. This number (which is marked in the messages as the ''BBoard–Id'') is ever increasing. Hence, when *bbc* says ''n items'', it really means that the highest BBoard–Id is ''n''. There may, or may not actually be ''n'' items in the BBoard. Some common messages are:

BBoard — n items unseen
This message tells how many items the user has not yet seen. When invoked with the '–quiet' switch, this is the only informative line that *bbc* will possibly print out.

BBoard — empty
The BBoard is empty.

BBoard — n items (none seen)
The BBoard has items in it, but the user hasn't seen any.

BBoard — n items (all seen)
The BBoard is non–empty, and the user has seen everything in it.

BBoard — n items seen out of m
The BBoard has at most m–n items that the user has not seen.

The '–topics' switch directs *bbc* to print three items about the named *BBoards*: it's official name, the number of items present, and the date and time of the last update. If no *BBoards* are named, then all BBoards are listed. If the '–verbose' switch is given, more information is output.

The '–quiet' switch specifies that *bbc* should be silent if no *BBoards* are found with new information. The '–verbose' switch specifies that *bbc* is to consider you to be interested in *BBoards* that you've already seen everything in.

To override the default *mshproc* and the profile entry, use the '–mshproc program' switch. Any arguments not understood by *bbc* are passed to this program. The '–protocol' switch tells *bbc* that your *mshproc* knows about the special *bbc* protocol for reporting back information. *msh* (1), the default *mshproc*, knows all about this.

The '–file BBoardsfile' switch tells *bbc* to use a non–standard *BBoards* file when performing its calculations. Similarly, the '–user BBoardsuser' switch tells *bbc* to use a non–standard username. Both of these switches are useful for debugging a new *BBoards* or *POP* file.

If the local host is configured as an NNTP BBoards client, or if the '–host host' switch is given, then *bbc* will query the NNTP service host as to the status of the BBoards. For NNTP BBoards clients, the '–user user' and the '–rpop' switches are ignored.

The *.bbrc* file in the user's **$HOME** directory is used to keep track of what messages have been read. The '–rcfile rcfile' switch overrides the use of *.bbrc* for this purpose. If the value given to the switch is not absolute, (i.e., does not begin with a **/**), it will be presumed to start from the current working directory. If this switch is not given (or the '–norcfile' switch is given), then *bbc* consults the envariable **$MHBBRC**, and honors it similarly.

Files

$HOME/.mh_profile	The user profile
$HOME/.bbrc	BBoard ''current'' message information

Profile Components

Path:	To determine the user's MH directory
bboards:	To specify interesting BBoards
mshproc:	Program to read a given BBoard

See Also

bbl(1), bboards(1), msh(1)

Defaults

'–read'
'–noarchive'
'–protocol'
'bboards' defaults to ''system''
'–file /usr/bboards/BBoards'
'–user bboards'

Context

None

Bugs

The '–user' switch takes effect only if followed by the '–file' switch.

NAME

bboards – the UCI BBoards facility

SYNOPSIS

bbc [–check] [–read] bboards ... [–help]

DESCRIPTION

The home directory of *bboards* is where the BBoard system is kept. This documentation describes some of the nuances of the BBoard system.

Note that if your system is configured to use the Network News Transfer Protocol (**NNTP**) to read BBoards, (your system does seem to be configured this way), then there is no local bboards setup; instead, *bbc* opens an **NNTP** connection to the local server.

BBoards, BBoard–IDs

A BBoard is just a file containing a group of messages relating to the same topic. These files live in the ˜bboards home directory. Each message in a BBoard file has in its header the line ‘‘BBoard-Id: n’’, where ‘‘n’’ is an ascending decimal number. This id-number is unique for each message in a BBoards file. It should NOT be confused with the message number of a message, which can change as messages are removed from the BBoard.

BBoard Handling

To read BBoards, use the *bbc* and *msh* programs. The *msh* command is a monolithic program which contains all the functionality of *MH* in a single program. The ‘–check’ switch to *bbc* lets you check on the status of BBoards, and the ‘–read’ switch tells *bbc* to invoke *msh* to read those BBoards.

Creating a BBoard

Both public, and private BBoards are supported. Contact the mail address *PostMaster* if you’d like to have a BBoard created.

BBoard addresses

Each BBoard has associated with it 4 addresses, these are (for the ficticious BBoard called ‘‘hacks’’):

hacks : The Internet wide distribution list.
dist-hacks : The local BBoard.
hacks-request : The people responsible for the BBoard at the Internet level.
local-hacks-request : The people responsible for the BBoard locally.

Files

$HOME/.mh_profile	The user profile
$HOME/.bbrc	BBoard information

Profile Components

Path:	To determine the user’s MH directory
bboards:	To specify interesting BBoards
mshproc:	Program to read a given BBoard

See Also

bbc(1), bbl(1), bbleader(1), msh(1)

Defaults

The default bboard is ‘‘system’’

Context

None

NAME

burst – explode digests into messages

SYNOPSIS

burst [+folder] [msgs] [–inplace] [–noinplace] [–quiet] [–noquiet] [–verbose] [–noverbose] [–help]

DESCRIPTION

Burst considers the specified messages in the named folder to be Internet digests, and explodes them in that folder.

If '–inplace' is given, each digest is replaced by the ''table of contents'' for the digest (the original digest is removed). *Burst* then renumbers all of the messages following the digest in the folder to make room for each of the messages contained within the digest. These messages are placed immediately after the digest.

If '–noinplace' is given, each digest is preserved, no table of contents is produced, and the messages contained within the digest are placed at the end of the folder. Other messages are not tampered with in any way.

The '–quiet' switch directs *burst* to be silent about reporting messages that are not in digest format.

The '–verbose' switch directs *burst* to tell the user the general actions that it is taking to explode the digest.

It turns out that *burst* works equally well on forwarded messages and blind–carbon–copies as on Internet digests, provided that the former two were generated by *forw* or *send*.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user's MH directory
Current–Folder:	To find the default current folder
Msg–Protect:	To set mode when creating a new message

See Also

Proposed Standard for Message Encapsulation (aka RFC–934),
inc(1), msh(1), pack(1)

Defaults

'+folder' defaults to the current folder
'msgs' defaults to cur
'–noinplace'
'–noquiet'
'–noverbose'

Context

If a folder is given, it will become the current folder. If '–inplace' is given, then the first message burst becomes the current message. This leaves the context ready for a *show* of the table of contents of the digest, and a *next* to see the first message of the digest. If '–noinplace' is given, then the first message extracted from the first digest burst becomes the current message. This leaves the context in a similar, but not identical, state to the context achieved when using '–inplace'.

Bugs

The *burst* program enforces a limit on the number of messages which may be *burst* from a single message. This number is on the order of 1000 messages. There is usually no limit on the number of messages which may reside in the folder after the *burst*ing.

Although *burst* uses a sophisticated algorithm to determine where one encapsulated message ends and another begins, not all digestifying programs use an encapsulation algorithm. In degenerate cases, this usually results in *burst* finding an encapsulation boundary prematurely and splitting a single encapsulated message into two or more messages. These erroneous digestifying programs should be fixed.

Furthermore, any text which appears after the last encapsulated message is not placed in a seperate message by *burst*. In the case of digestified messages, this text is usally an ‘‘End of digest’’ string. As a result of this possibly un–friendly behavior on the part of *burst*, note that when the ‘–inplace’ option is used, this trailing information is lost. In practice, this is not a problem since correspondents usually place remarks in text prior to the first encapsulated message, and this information is not lost.

NAME

comp – compose a message

SYNOPSIS

comp [+folder] [msg] [–draftfolder +folder] [–draftmessage msg] [–nodraftfolder] [–editor editor] [–noedit] [–file file] [–form formfile] [–use] [–nouse] [–whatnowproc program] [–nowhatnowproc] [–help]

DESCRIPTION

Comp is used to create a new message to be mailed. It copies a message form to the draft being composed and then invokes an editor on the draft (unless '–noedit' is given, in which case the initial edit is suppressed).

The default message form contains the following elements:

```
To:
cc:
Subject:
--------
```

If the file named ''components'' exists in the user's MH directory, it will be used instead of this form. The file specified by '–form formfile' will be used if given. You may also start *comp* using the contents of an existing message as the form. If you supply either a '+folder' or 'msg' argument, that message will be used as the form. You may not supply both a '–form formfile' and a '+folder' or 'msg' argument. The line of dashes or a blank line must be left between the header and the body of the message for the message to be identified properly when it is sent (see *send* (1)). The switch '–use' directs *comp* to continue editing an already started message. That is, if a *comp* (or *dist*, *repl*, or *forw*) is terminated without sending the draft, the draft can be edited again via ''comp –use''.

If the draft already exists, *comp* will ask you as to the disposition of the draft. A reply of **quit** will abort *comp*, leaving the draft intact; **replace** will replace the existing draft with the appropriate form; **list** will display the draft; **use** will use the draft for further composition; and **refile +folder** will file the draft in the given folder, and give you a new draft with the appropriate form. (The '+folder' argument to **refile** is required.)

The '–draftfolder +folder' and '–draftmessage msg' switches invoke the *MH* draft folder facility. This is an advanced (and highly useful) feature. Consult the **Advanced Features** section of the *MH* manual for more information.

The '–file file' switch says to use the named file as the message draft.

The '–editor editor' switch indicates the editor to use for the initial edit. Upon exiting from the editor, *comp* will invoke the *whatnow* program. See *whatnow* (1) for a discussion of available options. The invocation of this program can be inhibited by using the '–nowhatnowproc' switch. (In truth of fact, it is the *whatnow* program which starts the initial edit. Hence, '–nowhatnowproc' will prevent any edit from occurring.)

Files

/usr/bs/mh-beta/lib/components	The message skeleton
or <mh–dir>/components	Rather than the standard skeleton
$HOME/.mh_profile	The user profile
<mh–dir>/draft	The draft file

Profile Components

Path:	To determine the user's MH directory
Draft–Folder:	To find the default draft–folder
Editor:	To override the default editor
Msg–Protect:	To set mode when creating a new message (draft)
fileproc:	Program to refile the message
whatnowproc:	Program to ask the ''What now?'' questions

See Also

dist(1), forw(1), repl(1), send(1), whatnow(1), mh-profile(5)

Defaults

'+folder' defaults to the current folder
'msg' defaults to the current message
'–nodraftfolder'
'–nouse'

Context

None

Bugs

If *whatnowproc* is *whatnow*, then *comp* uses a built–in *whatnow*, it does not actually run the *whatnow* program. Hence, if you define your own *whatnowproc*, don't call it *whatnow* since *comp* won't run it.

NAME

dist – redistribute a message to additional addresses

SYNOPSIS

dist [+folder] [msg] [–annotate] [–noannotate] [–draftfolder +folder] [–draftmessage msg] [–nodraftfolder] [–editor editor] [–noedit] [–form formfile] [–inplace] [–noinplace] [–whatnowproc program] [–nowhatnowproc] [–help]

DESCRIPTION

Dist is similar to *forw*. It prepares the specified message for redistribution to addresses that (presumably) are not on the original address list.

The default message form contains the following elements:

```
Resent-To:
Resent-cc:
```

If the file named ‘‘distcomps’’ exists in the user’s MH directory, it will be used instead of this form. In either case, the file specified by ‘–form formfile’ will be used if given. The form used will be prepended to the message being resent.

If the draft already exists, *dist* will ask you as to the disposition of the draft. A reply of **quit** will abort *dist*, leaving the draft intact; **replace** will replace the existing draft with a blank skeleton; and **list** will display the draft.

Only those addresses in ‘‘Resent–To:’’, ‘‘Resent–cc:’’, and ‘‘Resent–Bcc:’’ will be sent. Also, a ‘‘Resent–Fcc: folder’’ will be honored (see *send* (1)). Note that with *dist*, the draft should contain only ‘‘Resent–xxx:’’ fields and no body. The headers and the body of the original message are copied to the draft when the message is sent. Use care in constructing the headers for the redistribution.

If the ‘–annotate’ switch is given, the message being distributed will be annotated with the lines:

```
Resent: date
Resent: addrs
```

where each address list contains as many lines as required. This annotation will be done only if the message is sent directly from *dist*. If the message is not sent immediately from *dist*, ‘‘comp –use’’ may be used to re–edit and send the constructed message, but the annotations won’t take place. The ’–inplace’ switch causes annotation to be done in place in order to preserve links to the annotated message.

See *comp* (1) for a description of the ‘–editor’ and ‘–noedit’ switches. Note that while in the editor, the message being resent is available through a link named ‘‘@’’ (assuming the default *whatnowproc*). In addition, the actual pathname of the message is stored in the envariable **$editalt**, and the pathname of the folder containing the message is stored in the envariable **$mhfolder**.

The ‘–draftfolder +folder’ and ‘–draftmessage msg’ switches invoke the *MH* draft folder facility. This is an advanced (and highly useful) feature. Consult the **Advanced Features** section of the *MH* manual for more information.

Upon exiting from the editor, *dist* will invoke the *whatnow* program. See *whatnow* (1) for a discussion of available options. The invocation of this program can be inhibited by using the ‘–nowhatnowproc’ switch.

(In truth of fact, it is the *whatnow* program which starts the initial edit. Hence, '–nowhatnowproc' will prevent any edit from occurring.)

Files

/usr/bs/mh-beta/lib/distcomps	The message skeleton
or <mh–dir>/distcomps	Rather than the standard skeleton
$HOME/.mh_profile	The user profile
<mh–dir>/draft	The draft file

Profile Components

Path:	To determine the user's MH directory
Current–Folder:	To find the default current folder
Draft–Folder:	To find the default draft–folder
Editor:	To override the default editor
fileproc:	Program to refile the message
whatnowproc:	Program to ask the ''What now?'' questions

See Also

comp(1), forw(1), repl(1), send(1), whatnow(1)

Defaults

'+folder' defaults to the current folder
'msg' defaults to cur
'–noannotate'
'–nodraftfolder'
'–noinplace'

Context

If a folder is given, it will become the current folder. The message distributed will become the current message.

History

Dist originally used headers of the form ''Distribute–xxx:'' instead of ''Resent–xxx:''. In order to conform with the ARPA Internet standard, RFC–822, the ''Resent–xxx:'' form is now used. *Dist* will recognize ''Distribute–xxx:'' type headers and automatically convert them to ''Resent–xxx:''.

Bugs

Dist does not *rigorously* check the message being distributed for adherence to the transport standard, but *post* called by *send* does. The *post* program will balk (and rightly so) at poorly formatted messages, and *dist* won't correct things for you.

If *whatnowproc* is *whatnow*, then *dist* uses a built–in *whatnow*, it does not actually run the *whatnow* program. Hence, if you define your own *whatnowproc*, don't call it *whatnow* since *dist* won't run it.

If your current working directory is not writable, the link named ''@'' is not available.

NAME

folder, folders – set/list current folder/message

SYNOPSIS

folder [+folder] [msg] [–all] [–create] [–nocreate] [–print] [–fast] [–nofast] [–header] [–noheader] [–recurse] [–norecurse] [–total] [–nototal] [–list] [–nolist] [–push] [–pop] [–pack] [–nopack] [–verbose] [–noverbose] [–help]

folders

DESCRIPTION

Since the *MH* environment is the shell, it is easy to lose track of the current folder from day to day. When *folder* is given the '–print' switch (the default), *folder* will list the current folder, the number of messages in it, the range of the messages (low–high), and the current message within the folder, and will flag extra files if they exist. An example of this summary is:

```
inbox+             has   16 messages (    3–  22); cur=    5.
```

If a '+folder' and/or 'msg' are specified, they will become the current folder and/or message. By comparison, when a '+folder' argument is given, this corresponds to a ''cd'' operation in the *shell*; when no '+folder' argument is given, this corresponds roughly to a ''pwd'' operation in the *shell*.

If the specified (or default) folder doesn't exist, the default action is to query the user as to whether the folder should be created; when standard input is not a tty, the answer to the query is assumed to be ''yes''.

Specifying '–create' will cause *folder* to create new folders without any query. (This is the easy way to create an empty folder for use later.) Specifying '–nocreate' will cause *folder* to exit without creating a non-existant folder.

Multiple Folders

Specifying '–all' will produce a summary line for each top-level folder in the user's MH directory, sorted alphabetically. (If *folder* is invoked by a name ending with ''s'' (e.g., *folders*), '–all' is assumed). Specifying '–recurse' with '–all' will also produce a line for all sub-folders. These folders are all preceded by the read–only folders, which occur as ''atr–cur–'' entries in the user's *MH* context. For example,

```
Folder                 # of messages (  range  )   cur msg (other files)
/fsd/rs/m/tacc         has   35 messages (    1–  35); cur=  23.
/rnd/phyl/Mail/EP has   82 messages (    1–108); cur=  82.
ff                     has   no messages.
inbox+                 has   16 messages (    3–  22); cur=    5.
mh                     has   76 messages (    1–  76); cur=  70.
notes                  has     2 messages (    1–    2); cur=    1.
ucom                   has 124 messages (    1–124); cur=    6; (others).
                TOTAL= 339 messages in 7 folders
```

The ''+'' after inbox indicates that it is the current folder. The ''(others)'' indicates that the folder 'ucom' has files which aren't messages. These files may either be sub–folders, or files that don't belong under the MH file naming scheme.

The header is output if either a '–all' or a '–header' switch is specified; it is suppressed by '–noheader'. A

'–total' switch will produce only the summary line.

If '–fast' is given, only the folder name (or names in the case of '–all') will be listed. (This is faster because the folders need not be read.)

If a '+folder' is given along with the '–all' switch, *folder* will, in addition to setting the current folder, list the top–level folders for the current folder (with '–norecurse') or list all sub-folders under the current folder recursively (with '–recurse'). In this case, if a 'msg' is also supplied, it will become the current message of '+folder'.

The '–recurse' switch lists each folder recursively, so use of this option effectively defeats the speed enhancement of the '–fast' option, since each folder must be searched for subfolders. Nevertheless, the combination of these options is useful.

Compacting a Folder

The '–pack' switch will compress the message names in the designated folders, removing holes in message numbering. The '–verbose' switch directs *folder* to tell the user the general actions that it is taking to compress the folder.

The Folder Stack

The '–push' switch directs *folder* to push the current folder onto the *folder–stack*, and make the '+folder' argument the current folder. If '+folder' is not given, the current folder and the top of the *folder–stack* are exchanged. This corresponds to the ''pushd'' operation in the *CShell*.

The '–pop' switch directs *folder* to discard the top of the *folder–stack*, after setting the current folder to that value. No '+folder' argument is allowed. This corresponds to the ''popd'' operation in the *CShell*. The '–push' switch and the '–pop' switch are mutually exclusive: the last occurrence of either one overrides any previous occurrence of the other. Both of these switches also set '–list' by default.

The '–list' switch directs *folder* to list the contents of the *folder–stack*. No '+folder' argument is allowed. After a successful '–push' or '–pop', the '–list' action is taken, unless a '–nolist' switch follows them on the command line. This corresponds to the ''dirs'' operation in the *CShell*. The '–push', '–pop', and '–list' switches turn off '–print'.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user's MH directory
Current–Folder:	To find the default current folder
Folder–Protect:	To set mode when creating a new folder
Folder–Stack:	To determine the folder stack

See Also

refile(1), mhpath(1)

Defaults

‘+folder’ defaults to the current folder
‘msg’ defaults to none
‘–nofast’
‘–noheader’
‘–nototal’
‘–nopack’
‘–norecurse’
‘–noverbose’
‘–print’ is the default if no ‘–list’, ‘–push’, or ‘–pop’ is specified
‘–list’ is the default if ‘–push’, or ‘–pop’ is specified

Context

If ‘+folder’ and/or ‘msg’ are given, they will become the current folder and/or message.

History

In previous versions of *MH*, the ‘–fast’ switch prevented context changes from occurring for the current folder. This is no longer the case: if ‘+folder’ is given, then *folder* will always change the current folder to that.

Bugs

‘–all’ forces ‘–header’ and ‘–total’.

There is no way to restore the default behavior (to ask the user whether to create a non-existant folder) after ‘–create’ or ‘–nocreate’ is given.

NAME

forw – forward messages

SYNOPSIS

forw [+folder] [msgs] [–annotate] [–noannotate] [–draftfolder +folder] [–draftmessage msg]
[–nodraftfolder] [–editor editor] [–noedit] [–filter filterfile] [–form formfile] [–format] [–noformat]
[–inplace] [–noinplace] [–mime] [–nomime] [–whatnowproc program] [–nowhatnowproc] [–help]

forw [+folder] [msgs] [–digest list] [–issue number] [–volume number] [other switches for *forw*] [–help]

DESCRIPTION

Forw may be used to prepare a message containing other messages. It constructs the new message from the components file or '–form formfile' (see *comp*), with a body composed of the message(s) to be forwarded. An editor is invoked as in *comp*, and after editing is complete, the user is prompted before the message is sent.

The default message form contains the following elements:

```
To:
cc:
Subject:
--------
```

If the file named ''forwcomps'' exists in the user's MH directory, it will be used instead of this form. In either case, the file specified by '–form formfile' will be used if given.

If the draft already exists, *forw* will ask you as to the disposition of the draft. A reply of **quit** will abort *forw*, leaving the draft intact; **replace** will replace the existing draft with a blank skeleton; and **list** will display the draft.

If the '–annotate' switch is given, each message being forwarded will be annotated with the lines

```
Forwarded: date
Forwarded: addrs
```

where each address list contains as many lines as required. This annotation will be done only if the message is sent directly from *forw*. If the message is not sent immediately from *forw*, ''comp –use'' may be used to re–edit and send the constructed message, but the annotations won't take place. The '–inplace' switch causes annotation to be done in place in order to preserve links to the annotated message.

See *comp* (1) for a description of the '–editor' and '–noedit' switches.

Although *forw* uses the '–form formfile' switch to direct it how to construct the beginning of the draft, the '–filter filterfile', '–format', and '–noformat' switches direct *forw* as to how each forwarded message should be formatted in the body of the draft. If '–noformat' is specified, then each forwarded message is output exactly as it appears. If '–format' or '–filter filterfile' is specified, then each forwarded message is filtered (re–formatted) prior to being output to the body of the draft. The filter file for *forw* should be a standard form file for *mhl*, as *forw* will invoke *mhl* to format the forwarded messages. The default message filter (what you get with '–format') is:

```
width=80,overflowtext=,overflowoffset=10
leftadjust,compress,compwidth=9
Date:formatfield="%<(nodate{text})%{text}%|%(tws{text})%>"
From:
To:
cc:
Subject:
:
body:nocomponent,overflowoffset=0,noleftadjust,nocompress
```

If the file named ‘‘mhl.forward’’ exists in the user’s MH directory, it will be used instead of this form. In either case, the file specified by ‘–filter filterfile’ will be used if given. To summarize: ‘–noformat’ will reproduce each forwarded message exactly, ‘–format’ will use *mhl* and a default filterfile, ‘‘mhl.forward’’, to format each forwarded message, and ‘–filter filterfile’ will use the named filterfile to format each forwarded message with *mhl*.

Each forwarded message is separated with an encapsulation delimiter and dashes in the first column of the forwarded messages will be prepended with ‘– ’ so that when received, the message is suitable for bursting by *burst* (1). This follows the Internet RFC–934 guidelines.

For users of *prompter* (1), by specifying prompter’s ‘-prepend’ switch in the .mh_profile file, any commentary text is entered before the forwarded messages. (A major win!)

To use the MIME rules for encapsulation, specify the ‘–mime’ switch. This directs *forw* to generate an *mhn* composition file. Note that MH will not invoke *mhn* automatically, unless you add this line to your .mh_profile file:

```
automhnproc: mhn
```

Otherwise, you must specifically give the command

```
What now? edit mhn
```

prior to sending the draft.

To automate this somewhat, create a link to *prompter* called *rapid* and put these lines in your .mh_profile file:

```
forw: -editor rapid -mime
rapid: -rapid
rapid-next: mhn
```

Then, you can simply do:

forw msgs
To: *mailbox*
cc:
Subject: *whatever*

--------Enter initial text

blah, blah, blah.

```
<CTRL-D>
--------

What now? edit
What now? send
```

The *edit* command invokes *mhn* automatically.

The ‘–draftfolder +folder’ and ‘–draftmessage msg’ switches invoke the *MH* draft folder facility. This is an advanced (and highly useful) feature. Consult the **Advanced Features** section of the *MH* manual for more information.

Upon exiting from the editor, *forw* will invoke the *whatnow* program. See *whatnow* (1) for a discussion of available options. The invocation of this program can be inhibited by using the ‘–nowhatnowproc’ switch. (In truth of fact, it is the *whatnow* program which starts the initial edit. Hence, ‘–nowhatnowproc’ will prevent any edit from occurring.)

The ‘–digest list’, ‘–issue number’, and ‘–volume number’ switches implement a digest facility for *MH*. Specifying these switches enables and/or overloads the following escapes:

Type	*Escape*	*Returns*	*Description*
component	*digest*	string	Argument to ‘–digest’
function	*cur*	integer	Argument to ‘–volume’
function	*msg*	integer	Argument to ‘–issue’

Consult the **Advanced Features** section of the *MH* User’s Manual for more information on making digests.

Files

/usr/bs/mh-beta/lib/forwcomps	The message skeleton
or <mh–dir>/forwcomps	Rather than the standard skeleton
/usr/bs/mh-beta/lib/digestcomps	The message skeleton if ‘–digest’ is given
or <mh–dir>/digestcomps	Rather than the standard skeleton
/usr/bs/mh-beta/lib/mhl.forward	The message filter
or <mh–dir>/mhl.forward	Rather than the standard filter
$HOME/.mh_profile	The user profile
<mh–dir>/draft	The draft file

Profile Components

Path:	To determine the user’s MH directory
Current–Folder:	To find the default current folder
Draft–Folder:	To find the default draft–folder
Editor:	To override the default editor
Msg–Protect:	To set mode when creating a new message (draft)
fileproc:	Program to refile the message
mhlproc:	Program to filter messages being forwarded
whatnowproc:	Program to ask the ‘‘What now?’’ questions

See Also

Proposed Standard for Message Encapsulation (aka RFC–934),
comp(1), dist(1), repl(1), send(1), whatnow(1), mh–format(5)

Defaults

‘+folder’ defaults to the current folder
‘msgs’ defaults to cur
‘–noannotate’
‘–nodraftfolder’
‘–noformat’
‘–noinplace’
‘–nomime’

Context

If a folder is given, it will become the current folder. The first message forwarded will become the current message.

Bugs

If *whatnowproc* is *whatnow*, then *forw* uses a built–in *whatnow*, it does not actually run the *whatnow* program. Hence, if you define your own *whatnowproc*, don’t call it *whatnow* since *forw* won’t run it.

When *forw* is told to annotate the messages it forwards, it doesn’t actually annotate them until the draft is successfully sent. If from the *whatnowproc*, you *push* instead of *send*, it’s possible to confuse *forw* by re–ordering the file (e.g., by using ‘folder –pack’) before the message is successfully sent. *Dist* and *repl* don’t have this problem.

To avoid prepending the leading dash characters in forwarded messages, there is a ‘–nodashmunging’ option. See the ‘‘Hidden Features’’ section of the *MH Administrator’s Guide* for more details.

NAME

inc – incorporate new mail

SYNOPSIS

inc [+folder] [–audit audit–file] [–noaudit] [–changecur] [–nochangecur] [–form formatfile] [–format string] [–file name] [–silent] [–nosilent] [–truncate] [–notruncate] [–width columns] [–host host] [–user user] [–apop] [–noapop] [–rpop] [–norpop] [–pack file] [–nopack] [–help]

DESCRIPTION

Inc incorporates mail from the user's incoming mail drop into an *MH* folder. If '+folder' isn't specified, a folder in the user's *MH* directory will be used, either that specified by the ''Inbox:'' entry in the user's profile, or the folder named ''inbox''. The new messages being incorporated are assigned numbers starting with the next highest number in the folder. If the specified (or default) folder doesn't exist, the user will be queried prior to its creation. As the messages are processed, a *scan* listing of the new mail is produced.

If the user's profile contains a ''Msg–Protect: nnn'' entry, it will be used as the protection on the newly created messages, otherwise the *MH* default of 0644 will be used. During all operations on messages, this initially assigned protection will be preserved for each message, so *chmod*(1) may be used to set a protection on an individual message, and its protection will be preserved thereafter.

If the switch '–audit audit–file' is specified (usually as a default switch in the profile), then *inc* will append a header line and a line per message to the end of the specified audit–file with the format:

≪inc≫ date
<scan line for first message>
<scan line for second message>
<etc.>

This is useful for keeping track of volume and source of incoming mail. Eventually, *repl*, *forw*, *comp*, and *dist* may also produce audits to this (or another) file, perhaps with ''Message–Id:'' information to keep an exact correspondence history. ''Audit–file'' will be in the user's MH directory unless a full path is specified.

Inc will incorporate even improperly formatted messages into the user's MH folder, inserting a blank line prior to the offending component and printing a comment identifying the bad message.

In all cases, the user's mail drop will be zeroed, unless the '–notruncate' switch is given.

If the profile entry ''Unseen–Sequence'' is present and non–empty, then *inc* will add each of the newly incorporated messages to each sequence named by the profile entry. This is similar to the ''Previous–Sequence'' profile entry supported by all *MH* commands which take 'msgs' or 'msg' arguments. Note that *inc* will not zero each sequence prior to adding messages.

The interpretation of the '–form formatfile', '–format string', and '–width columns' switches is the same as in *scan* (1).

By using the '–file name' switch, one can direct *inc* to incorporate messages from a file other than the user's maildrop. Note that the name file will NOT be zeroed, unless the '–truncate' switch is given.

If the envariable **$MAILDROP** is set, then *inc* uses it as the location of the user's maildrop instead of the default (the '-file name' switch still overrides this, however). If this envariable is not set, then *inc* will consult the profile entry ‘‘MailDrop’’ for this information. If the value found is not absolute, then it is interpreted relative to the user’s *MH* directory. If the value is not found, then *inc* will look in the standard system location for the user’s maildrop.

The ‘–silent’ switch directs *inc* to be quiet and not ask any questions at all. This is useful for putting *inc* in the background and going on to other things.

If the local host is configured as a POP client, or if the ‘–host host’ switch is given, then *inc* will query the POP service host as to the status of mail waiting. If the ‘–user user’ switch is not given, then the current username is used. Normally, *inc* will prompt for a password to use. However, if the ‘–apop’ switch is given, *inc* will generate authentication credentials to provide for origin authentication and replay protection, but which do not involve sending a password in the clear over the network. Otherwise, if the ‘–rpop’ switch is given, then *inc* will try to use a ‘‘trusted’’ connection (ala the BSD r-commands).

If *inc* uses POP, then the ‘–pack file’ switch is considered. If given, then *inc* simply uses the POP to *packf* (1) the user’s maildrop from the POP service host to the named file. This switch is provided for those users who prefer to use *msh* to read their maildrops.

Files

$HOME/.mh_profile	The user profile
/usr/bs/mh-beta/lib/mtstailor	tailor file
/usr/spool/mail/$USER	Location of mail drop

Profile Components

Path:	To determine the user’s MH directory
Alternate–Mailboxes:	To determine the user’s mailboxes
Inbox:	To determine the inbox, default ‘‘inbox’’
Folder–Protect:	To set mode when creating a new folder
Msg–Protect:	To set mode when creating a new message and audit–file
Unseen–Sequence:	To name sequences denoting unseen messages

See Also

Post Office Protocol - version 3 (aka RFC–1081),
mhmail(1), scan(1), mh–mail(5), post(8)

Defaults

‘+folder’ defaulted by ‘‘Inbox’’ above
‘–noaudit’
‘–changecur’
‘–format’ defaulted as described above
‘–nosilent’
‘–truncate’ if ‘–file name’ not given, ‘–notruncate’ otherwise
‘–width’ defaulted to the width of the terminal
‘–nopack’
‘–rpop’

Context

The folder into which messages are being incorporated will become the current folder. The first message incorporated will become the current message, unless the ‘–nochangecur’ option is specified. This leaves the context ready for a *show* of the first new message.

Bugs

The argument to the '–format' switch must be interpreted as a single token by the shell that invokes *inc*. Therefore, one must usually place the argument to this switch inside double–quotes.

NAME

mark – mark messages

SYNOPSIS

mark [+folder] [msgs] [–sequence name ...] [–add] [–delete] [–list] [–public] [–nopublic] [–zero] [–nozero] [–help]

DESCRIPTION

The *mark* command manipulates message sequences by adding or deleting message numbers from folder–specific message sequences, or by listing those sequences and messages. A message sequence is a keyword, just like one of the ‘‘reserved’’ message names, such as ‘‘first’’ or ‘‘next’’. Unlike the ‘‘reserved’’ message names, which have a fixed semantics on a per–folder basis, the semantics of a message sequence may be defined, modified, and removed by the user. Message sequences are folder–specific, e.g., the sequence name ‘‘seen’’ in the context of folder ‘‘+inbox’’ need not have any relation whatsoever to the sequence of the same name in a folder of a different name.

Three action switches direct the operation of *mark*. These switches are mutually exclusive: the last occurrence of any of them overrides any previous occurrence of the other two.

The ‘–add’ switch tells *mark* to add messages to sequences or to create a new sequence. For each sequence named via the ‘–sequence name’ argument (which must occur at least once) the messages named via ‘msgs’ (which defaults to ‘‘cur’’ if no ‘msgs’ are given), are added to the sequence. The messages to be added need not be absent from the sequence. If the ‘–zero’ switch is specified, the sequence will be emptied prior to adding the messages. Hence, ‘–add –zero’ means that each sequence should be initialized to the indicated messages, while ‘–add –nozero’ means that each sequence should be appended to by the indicated messages.

The ‘–delete’ switch tells *mark* to delete messages from sequences, and is the dual of ‘–add’. For each of the named sequences, the named messages are removed from the sequence. These messages need not be already present in the sequence. If the ‘–zero’ switch is specified, then all messages in the folder are appended to the sequence prior to removing the messages. Hence, ‘–delete –zero’ means that each sequence should contain all messages except those indicated, while ‘–delete –nozero’ means that only the indicated messages should be removed from each sequence. As expected, the command ‘mark –sequence seen –delete all’ deletes the sequence ‘‘seen’’ from the current folder.

When creating (or modifying) a sequence, the ‘–public’ switch indicates that the sequence should be made readable for other *MH* users. In contrast, the ‘–nopublic’ switch indicates that the sequence should be private to the user’s *MH* environment.

The ‘–list’ switch tells *mark* to list both the sequences defined for the folder and the messages associated with those sequences. *Mark* will list the name of each sequence given by ‘–sequence name’ and the messages associated with that sequence. If ‘–sequence’ isn’t used, all sequences will be listed, with private sequences being so indicated. The ‘–zero’ switch does not affect the operation of ‘–list’.

The current restrictions on sequences are:

> The name used to denote a message sequence must consist of an alphabetic character followed by zero or more alphanumeric characters, and cannot be one of the (reserved) message names ‘‘new’’, ‘‘first’’, ‘‘last’’, ‘‘all’’, ‘‘next’’, or ‘‘prev’’.
>
> Only a certain number of sequences may be defined for a given folder. This number is usually limited

to 26 (10 on small systems).

Message ranges with user–defined sequence names are restricted to the form ‘‘name:n’’ or ‘‘name:-n’’, and refer to the first or last ‘n’ messages of the sequence ‘name’, respectively. Constructs of the form ‘‘name1–name2’’ are forbidden.

Files

$HOME/.mh_profile The user profile

Profile Components

Path:	To determine the user’s MH directory
Current–Folder:	To find the default current folder

See Also

pick (1), mh-sequence (5)

Defaults

‘+folder’ defaults to the current folder
‘–add’ if ‘–sequence’ is specified, ‘–list’ otherwise
‘msgs’ defaults to cur (or all if ‘–list’ is specified)
‘–nopublic’ if the folder is read–only, ‘–public’ otherwise
‘–nozero’

Context

If a folder is given, it will become the current folder.

Helpful Hints

Use ‘‘pick sequence –list’’ to enumerate the messages in a sequence (such as for use by a shell script).

NAME

mhl – produce formatted listings of MH messages

SYNOPSIS

/usr/bs/mh-beta/lib/mhl [–bell] [–nobell] [–clear] [–noclear] [–folder +folder] [–form formfile]
[–length lines] [–width columns] [–moreproc program] [–nomoreproc] [files ...] [–help]

DESCRIPTION

Mhl is a formatted message listing program. It can be used as a replacement for *more* (1) (the default *showproc*). As with *more*, each of the messages specified as arguments (or the standard input) will be output. If more than one message file is specified, the user will be prompted prior to each one, and a <RETURN> or <EOT> will begin the output, with <RETURN> clearing the screen (if appropriate), and <EOT> (usually CTRL–D) suppressing the screen clear. An <INTERRUPT> (usually CTRL–C) will abort the current message output, prompting for the next message (if there is one), and a <QUIT> (usually CTRL-\) will terminate the program (without core dump).

The '–bell' option tells *mhl* to ring the terminal's bell at the end of each page, while the '–clear' option tells *mhl* to clear the scree at the end of each page (or output a formfeed after each message). Both of these switches (and their inverse counterparts) take effect only if the profile entry *moreproc* is defined but empty, and *mhl* is outputting to a terminal. If the *moreproc* entry is defined and non-empty, and *mhl* is outputting to a terminal, then *mhl* will cause the *moreproc* to be placed between the terminal and *mhl* and the switches are ignored. Furthermore, if the '–clear' switch is used and *mhl's* output is directed to a terminal, then *mhl* will consult the **$TERM** and **$TERMCAP** envariables to determine the user's terminal type in order to find out how to clear the screen. If the '–clear' switch is used and *mhl's* output is not directed to a terminal (e.g., a pipe or a file), then *mhl* will send a formfeed after each message.

To override the default *moreproc* and the profile entry, use the '–moreproc program' switch. Note that *mhl* will never start a *moreproc* if invoked on a hardcopy terminal.

The '–length length' and '–width width' switches set the screen length and width, respectively. These default to the values indicated by **$TERMCAP**, if appropriate, otherwise they default to 40 and 80, respectively.

The default format file used by *mhl* is called *mhl.format* (which is first searched for in the user's *MH* directory, and then sought in the */usr/bs/mh-beta/lib* directory), this can be changed by using the '–form formatfile' switch.

Finally, the '–folder +folder' switch sets the *MH* folder name, which is used for the ''messagename:'' field described below. The envariable **$mhfolder** is consulted for the default value, which *show*, *next,* and *prev* initialize appropriately.

Mhl operates in two phases: 1) read and parse the format file, and 2) process each message (file). During phase 1, an internal description of the format is produced as a structured list. In phase 2, this list is walked for each message, outputting message information under the format constraints from the format file.

The ''mhl.format'' form file contains information controlling screen clearing, screen size, wrap–around control, transparent text, component ordering, and component formatting. Also, a list of components to ignore may be specified, and a couple of ''special'' components are defined to provide added functionality. Message output will be in the order specified by the order in the format file.

Each line of mhl.format has one of the formats:

```
;comment
:cleartext
variable[,variable...]
component:[variable,...]
```

A line beginning with a ‘;’ is a comment, and is ignored. A line beginning with a ‘:’ is clear text, and is output exactly as is. A line containing only a ‘:’ produces a blank line in the output. A line beginning with ‘‘component:’’ defines the format for the specified component, and finally, remaining lines define the global environment.

For example, the line:

```
width=80,length=40,clearscreen,overflowtext="***",overflowoffset=5
```

defines the screen size to be 80 columns by 40 rows, specifies that the screen should be cleared prior to each page, that the overflow indentation is 5, and that overflow text should be flagged with ‘‘***’’.

Following are all of the current variables and their arguments. If they follow a component, they apply only to that component, otherwise, their affect is global. Since the whole format is parsed before any output processing, the last global switch setting for a variable applies to the whole message if that variable is used in a global context (i.e., bell, clearscreen, width, length).

variable	*type*	*semantics*
width	integer	screen width or component width
length	integer	screen length or component length
offset	integer	positions to indent ‘‘component: ’’
overflowtext	string	text to use at the beginning of an overflow line
overflowoffset	integer	positions to indent overflow lines
compwidth	integer	positions to indent component text after the first line is output
uppercase	flag	output text of this component in all upper case
nouppercase	flag	don’t uppercase
clearscreen	flag/G	clear the screen prior to each page
noclearscreen	flag/G	don’t clearscreen
bell	flag/G	ring the bell at the end of each page
nobell	flag/G	don’t bell
component	string/L	name to use instead of ‘‘component’’ for this component
nocomponent	flag	don’t output ‘‘component: ’’ for this component
center	flag	center component on line (works for one–line components only)
nocenter	flag	don’t center
leftadjust	flag	strip off leading whitespace on each line of text
noleftadjust	flag	don’t leftadjust
compress	flag	change newlines in text to spaces
nocompress	flag	don’t compress
split	flag	don’t combine multiple fields into a single field
nosplit	flag	combine multiple fields into a single field

newline	flag	print newline at end of components (default)
nonewline	flag	don't print newline at end of components
formatfield	string	format string for this component (see below)
addrfield	flag	field contains addresses
datefield	flag	field contains dates

To specify the value of integer–valued and string–valued variables, follow their name with an equals–sign and the value. Integer–valued variables are given decimal values, while string–valued variables are given arbitrary text bracketed by double–quotes. If a value is suffixed by ''/G'' or ''/L'', then its value is useful in a global–only or local–only context (respectively).

A line of the form:

```
ignores=component,...
```

specifies a list of components which are never output.

The component ''MessageName'' (case–insensitive) will output the actual message name (file name) preceded by the folder name if one is specified or found in the environment. The format is identical to that produced by the '–header' option to *show*.

The component ''Extras'' will output all of the components of the message which were not matched by explicit components, or included in the ignore list. If this component is not specified, an ignore list is not needed since all non–specified components will be ignored.

If ''nocomponent'' is NOT specified, then the component name will be output as it appears in the format file.

The default format is:

```
: -- using template mhl.format --
overflowtext="***",overflowoffset=5
leftadjust,compwidth=9
ignores=msgid,message-id,received
Date:formatfield="%<(nodate{text})%{text}%|%(pretty{text})%>"
To:
cc:
:
From:
Subject:
:
extras:nocomponent
:
body:nocomponent,overflowtext=,overflowoffset=0,noleftadjust
```

The variable ''formatfield'' specifies a format string (see *mh–format* (5)). The flag variables ''addrfield'' and ''datefield'' (which are mutually exclusive), tell *mhl* to interpret the escapes in the format string as either addresses or dates, respectively.

By default, *mhl* does not apply any formatting string to fields containing address or dates (see *mh–mail* (5) for a list of these fields). Note that this results in faster operation since *mhl* must parse both addresses and dates in order to apply a format string to them. If desired, *mhl* can be given a default format string for either address or date fields (but not both). To do this, on a global line specify: either the flag addrfield or datefield, along with the apropriate formatfield variable string.

Files

/usr/bs/mh-beta/lib/mhl.format	The message template
or <mh–dir>/mhl.format	Rather than the standard template
$HOME/.mh_profile	The user profile

Profile Components

moreproc:	Program to use as interactive front–end

See Also

show(1), ap(8), dp(8)

Defaults

'–bell'
'–noclear'
'–length 40'
'–width 80'

Context

None

Bugs

There should be some way to pass 'bell' and 'clear' information to the front–end.

On hosts where *MH* was configured with the BERK option, address parsing is not enabled.

The ''nonewline'' option interacts badly with ''compress'' and ''split''.

NAME

mhmail – send or read mail

SYNOPSIS

mhmail [addrs ... [–body text] [–cc addrs ...] [–from addr] [–subject subject]] [–help]

DESCRIPTION

MHmail is intended as a replacement for the standard Bell mail program (*bellmail* (1)), compatible with *MH*. When invoked without arguments, it simply invokes *inc* (1) to incorporate new messages from the user's maildrop. When one or more users is specified, a message is read from the standard input and spooled to a temporary file. *MHmail* then invokes *post* (8) with the name of the temporary file as its argument to deliver the message to the specified user.

The '–subject subject' switch can be used to specify the ''Subject:'' field of the message. The '–body text' switch can be used to specify the text of the message; if it is specified, then the standard input is not read. Normally, addresses appearing as arguments are put in the ''To:'' field. If the '–cc' switch is used, all addresses following it are placed in the ''cc:'' field.

By using '–from addr', you can specify the ''From:'' header of the draft. Naturally, *post* will fill–in the ''Sender:'' header correctly.

This program is intended for the use of programs such as *at* (1), which expect to send mail automatically to various users. Normally, real people (as opposed to the ''unreal'' ones) will prefer to use *comp* (1) and *send* (1) to send messages.

Files

/usr/bs/mh-beta/bin/inc	Program to incorporate a maildrop into a folder
/usr/bs/mh-beta/lib/post	Program to deliver a message
/tmp/mhmail*	Temporary copy of message

Profile Components

None

See Also

inc(1), post(8)

Defaults

None

Context

If *inc* is invoked, then *inc*'s context changes occur.

NAME

mhn – multi-media MH

SYNOPSIS

mhn [[+folder] [msgs] | [–file file]]
[–part number]... [–type content]...
[–list [–headers] [–noheaders]
[–realsize] [–norealsize]] [-nolist]
[–show [–serialonly] [–noserialonly]
[–form formfile] [–pause] [–nopause]] [–noshow]
[–store [–auto] [–noauto]] [–nostore]
[–cache] [–nocache] [–rcache policy] [–wcache policy]
[–check] [–nocheck]
[–ebcdicsafe] [–noebcdicsafe]
[–rfc934mode] [–norfc934mode]
[–verbose] [–noverbose]
[–help]

DESCRIPTION

The *mhn* command manipulates multi-media messages as specified in RFC 1521.

Four action switches direct the operation of *mhn*, namely '–list', '–show', '–store', and '-cache'. Any of these switches may be used concurrently. Normally these action switches will operate on the content of each of the named messages. However, by using the '–part' and '–type' switches, the scope of the operation can be focused on particular subparts (of a multipart content) and/or particular content types.

A part specification consists of a series of numbers separated by dots. For example, in a multipart content containing three parts, these would be named as 1, 2, and 3, respectively. If part 2 was also a multipart content containing two parts, these would be named as 2.1 and 2.2, respectively. Note that the '–part' switch is effective for only messages containing a multipart content. If a message has some other kind of content, or if the part is itself another multipart content, the '–part' switch will not prevent the content from being acted upon.

A content specification consists of a content type and a subtype. The initial list of ''standard'' content types and subtypes can be found in RFC 1521. A list of commonly used contents is briefly reproduced here:

Type	Subtypes
----	--------
text	plain
multipart	mixed, alternative, digest, parallel
message	rfc822, partial, external-body
application	octet-stream, postscript
image	jpeg, gif, x-pbm, x-pgm, x-ppm, x-xwd
audio	basic
video	mpeg

Subtypes are mandatory. To specify a content, regardless of its subtype, just use the name of the content, e.g., ‘‘audio’’. To specify a specific subtype, separate the two with a slash, e.g., ‘‘audio/basic’’. Note that regardless of the values given to the ‘–type’ switch, a multipart content (of any subtype listed above) is always acted upon. Further note that if the ‘–type’ switch is used, and it is desirable to act on a message/external-body content, then the ‘–type’ switch must be used twice: once for message/external-body and once for the content externally referenced.

Each content may optionally have an integrity check associated with it. If present and the ‘-check’ switch is given, then *mhn* will attempt to verify the integrity of the content.

The option ‘–file file’ directs *mhn* to use the specified file as the source message, rather than a message from a folder. Note that the file should be a validly formatted message, just like any other *MH* message. It should **NOT** be in mail drop format (to convert a file in mail drop format to a folder of *MH* messages, see *inc* (1)).

Listing the Contents

The ‘–list’ switch tells *mhn* to list the table of contents associated with the named messages. The ‘–headers’ switch indicates that a one-line banner should be displayed above the listing. The ‘–realsize’ switch tells *mhn* to evaluate the ‘‘native’’ (decoded) format of each content prior to listing. This provides an accurate count at the expense of a small delay.

Showing the Contents

The ‘–show’ switch tells *mhn* to display the contents of the named messages. The headers of the message are displayed with the *mhlproc*, using format file *mhl.headers*. (The choice of format file can be overridden by the ‘–form formfile’ switch.)

mhn will look for information in the user’s profile to determine how the different contents should be displayed. This is accomplished by consulting a display string, and executing it under **/bin/sh**, with the standard input set to the content. The display string may contain these escapes:

%a additional arguments
%e exclusive execution
%f filename containing content
%F %e, %f, and stdin is terminal not content
%l display listing prior to displaying content
%p %l, and ask for confirmation
%s subtype
%d content description

For those display strings containing the e- or F-escape, *mhn* will execute at most one of these at any given time. Although the F-escape expands to be the filename containing the content, the e-escape has no expansion as far as the shell is concerned.

When the p-escape prompts for confirmation, typing INTR (usually control-C) will tell *mhn* not to display that content. (The p-escape can be disabled by specifying '–nopause'.) Further, when *mhn* is display a content, typing QUIT (usually control-\) will tell *mhn* to wrap things up immediately.

Note that if the content being displayed is multipart, but not one of the subtypes listed above, then the f- and F-escapes expand to multiple filenames, one for each subordinate content. Further, stdin is not redirected from the terminal to the content.

First, *mhn* will look for an entry of the form:

mhn-show-<type>/<subtype>

to determine the command to use to display the content. If this isn't found, *mhn* will look for an entry of the form:

mhn-show-<type>

to determine the display command. If this isn't found, *mhn* has two default values:

mhn-show-text/plain: %pmoreproc '%F'
mhn-show-message/rfc822: %pshow -file '%F'

If neither apply, *mhn* will check to see if the message has a application/octet-stream content with parameter ‘‘type=tar’’. If so, *mhn* will use an appropriate command. If not, *mhn* will complain.

Example entries might be:

mhn-show-audio/basic: raw2audio 2>/dev/null | play
mhn-show-image: xv '%f'
mhn-show-application/PostScript: lpr -Pps

Note that when using the f- or F-escape, it's a good idea to use single-quotes around the escape. This prevents misinterpretation by the shell of any funny characters that might be present in the filename.

Because the text content might be in a non-ASCII character set, when *mhn* encounters a ‘‘charset’’ parameter for this content, it checks to see whether the environment variable $MM_CHARSET is set and whether the value of this environment variable is equal to the value of the charset parameter. If not, then *mhn* will look for an entry of the form:

mhn-charset-<charset>

which should contain a command creating an environment to render the character set. This command string should containing a single ‘‘%s’’, which will be filled-in with the command to display the content.

An example entry might be:

mhn-charset-iso-8859-1: xterm -fn '-*-*-medium-r-normal-*-*-120-*-*-c-*-iso8859-*' -e %s

Note that many pagination programs strip off the high-order bit. However, newer releases of the *less* program have modest support for single-octet character sets. The source to *less* version 177, which has such support, is found in the MH source tree under **miscellany/less-177**. In order to view messages sent in the ISO 8859/1 character set using *less*, put these lines in your .login file:

setenv LESSCHARSET latin1
setenv LESS "-f"

The first line tells *less* to use 8859/1 definition for determing whether a character is ‘‘normal’’, ‘‘control‘‘, or ‘‘binary’’. The second line tells *less* not to warn you if it encounters a file that has non-ASCII characters. Then, simply set the **moreproc** profile entry to *less*, and it will get called automatically. (To handle other single-octet character sets, look at the *less* (1) manual entry for information about the **LESSCHARDEF** environment variable.)

Finally, *mhn* will process each message serially -- it won't start showing the next message until all the commands executed to display the current message have terminated. In the case of a multipart content (of any subtype listed above), the content contains advice indicating if the parts should be displayed serially or in parallel. Because this may cause confusion, particularly on uni-window displays, the ‘–serialonly’ switch can be given to tell *mhn* to never display parts in parallel.

Storing the Contents

The ‘–store’ switch tells *mhn* to store the contents of the named messages in ‘‘native’’ (decoded) format. Two things must be determined: the directory to store the content, and the filenames. Files are written in the directory given by the **mhn-storage** profile entry, e.g.,

mhn-storage: /tmp

If this entry isn't present, the current working directory is used.

mhn will look for information in the user's profile to determine how the different contents should be stored.

This is achieved through the use of a formatting string, which may contain these escapes:

%m message number
%P .part
%p part
%s subtype

If the content isn't part of a multipart (of any subtype listed above) content, the p-escapes are ignored. Note that if the formatting string starts with a ''+'' character, then these escapes are ignored, and the content is stored in the named folder. (A formatting string consisting solely of a ''+'' character indicates the current folder.) Further, a formatting string consisting solely of a ''-'' character indicates the standard-output.

First, *mhn* will look for an entry of the form:

mhn-store-<type>/<subtype>

to determine the formatting string. If this isn't found, *mhn* will look for an entry of the form:

mhn-store-<type>

to determine the formatting string. If this isn't found, *mhn* will check to see if the content is application/octet-stream with parameter ''type=tar''. If so, *mhn* will choose an appropriate filename. If the content is not application/octet-stream, then *mhn* will check to see if the content is a message. If so, *mhn* will use the value ''+''. If not, *mhn* will use the value ''%m%P.%s''.

Note that if the formatting string starts with a '/', then content will be stored in the full path given (rather than using the value of **mhn-storage** or the current working directory.) Similarly, if the formatting string starts with a '|', then *mhn* will execute a command which should ultimately store the content. Note that before executing the command, *mhn* will change to the appropriate directory. Also note that if the formatting string starts with a '|', then *mhn* will also honor the a-escape when processing the formatting string.

Example entries might be:

mhn-store-text: %m%P.txt
mhn-store-audio/basic: | raw2audio -e ulaw -s 8000 -c 1 > %m%P.au
mhn-store-application/PostScript: %m%P.ps

Further, note that when asked to store a content containing a partial message, *mhn* will try to locate all of the portions and combine them accordingly. Thus, if someone's sent you a message in several parts, you might put them all in their own folder and do:

mhn all -store

This will store exactly one message, containing the sum of the parts. Note that if *mhn* can not locate each part, it will not store anything.

Finally, if the '–auto' switch is given and the content contains information indicating the filename the content should be stored as (and if the filename doesn't begin with a '/'), then the filename from the content will be used instead.

External Access

For contents of type message/external-body, *mhn* supports these access-types:

afs
anon-ftp
ftp
local-file
mail-server

For the ‘‘anon-ftp’’ and ‘‘ftp’’ access types, if your system supports a SOCKETs interface to TCP/IP, then *mhn* will use a built-in FTP client. Otherwise, *mhn* will look for the **mhn-access-ftp** profile entry, e.g.,

mhn-access-ftp: myftp.sh

to determine the pathname of a program to perform the FTP retrieval. This program is invoked with these arguments:

domain name of FTP-site
username
password
remote directory
remote filename
local filename
‘‘ascii’’ or ‘‘binary’’

The program should terminate with a zero-valued exit-status if the retrieval is successful.

The Content Cache

When *mhn* encounters an external content containing a ‘‘Content-ID:’’ field, and if the content allows caching, then depending on the caching behavior of *mhn*, the content might be read from or written to a cache.

The caching behavior of *mhn* is controlled with the ‘–rcache’ and ‘–wcache’ switches, which define the policy for reading from, and writing to, the cache, respectively. One of four policies may be specified: ‘‘public’’, indicating that *mhn* should make use of a publically-accessible content cache; ‘‘private’’, indicating that *mhn* should make use of the user’s private content cache; ‘‘never’’, indicating that *mhn* should never make use of caching; and, ‘‘ask’’, indicating that *mhn* should ask the user.

There are two directories where contents may be cached: the profile entry **mhn-cache** names a directory containing world-readable contents, and, the profile entry **mhn-private-cache** names a directory containing private contents. The former should be an absolute (rooted) directory name. For example,

mhn-cache: /tmp

might be used if you didn’t care that the cache got wiped after each reboot of the system. The latter is interpreted relative to the user’s MH directory, if not rooted, e.g.,

mhn-private-cache: .cache

(which is the default value).

Caching the Contents

When you encounter a content of type message/external-body with access type ‘‘mail-server’’, *mhn* will ask you if may send a message to a mail-server requesting the content, e.g.,

```
% show 1
Retrieve content by asking mail-server@...

SEND file

? yes
mhn: request sent
```

Regardless of your decision, *mhn* can’t perform any other processing on the content.

However, if *mhn* is allowed to request the content, then when it arrives, there should be a top-level ‘‘Content-ID:’’ field which corresponds to the value in the original message/external-body content. You should now use the ‘-cache’ switch to tell *mhn* to enter the arriving content into the content cache, e.g.,

```
% mhn -cache 2
caching message 2 as file ...
```

You can then re-process the original message/external-body content, and ‘‘the right thing should happen’’, e.g.,

```
% show 1
  ...
```

Composing the Contents

The *mhn* program can also be used as a simple editor to aid in composing multi-media messages. When invoked by a *whatnow* program, *mhn* will expect the body of the draft to be formatted as an ‘‘*mhn* composition file.’’

The syntax of this is straight-forward:

```
body       ::=   1*(content | EOL)

content    ::=   directive | plaintext

directive  ::=   "#" type "/" subtype
                     0*(";" attribute "=" value)
                     [ "(" comment ")" ]
                     [ "<" id ">" ]
                     [ "[" description "]" ]
                     [ filename ]
                     EOL

                 | "#@" type "/" subtype
                     0*(";" attribute "=" value)
                     [ "(" comment ")" ]
                     [ "<" id ">" ]
                     [ "[" description "]" ]
                     external-parameters
                     EOL

                 | "#forw"
                     [ "<" id ">" ]
                     [ "[" description "]" ]
                     [ "+"folder ] [ 0*msg ]
                     EOL

                 | "#begin"
                       [ "<" id ">" ]
                       [ "[" description "]" ]
                       [   "alternative"
                         | "parallel"
                         | something-else    ]
                       EOL
                     1*body
                   "#end" EOL

plaintext  ::=   [ "Content-Description:"
                       description EOL EOL ]
                     1*line
                   [ "#" EOL ]

                 | "#<" type "/" subtype
                     0*(";" attribute "=" value)
                     [ "(" comment ")" ]
                     [ "[" description "]" ]
                     EOL
                     1*line
                   [ "#" EOL ]

line       ::=   "##" text EOL
```

```
  -- interpreted as "#"text EOL
| text EOL
```

Basically, the body contains one or more contents. A content consists of either a directive, indicated with a ‘‘#’’ as the first character of a line; or, plaintext (one or more lines of text). The continuation character, ‘‘\‘‘, may be used to enter a single directive on more than one line, e.g.,

```
#@application/octet-stream; \
    type=tar; \
    x-conversions=compress
```

There are four kinds of directives: ‘‘type’’ directives, which name the type and subtype of the content; ‘‘external-type’’ directives, which also name the type and subtype of the content; the ‘‘forw’’ directive, which is used to forward a digest of messages; and, the ‘‘begin’’ directive, which is used to create a multipart content.

For the type directives, the user may optionally specify the name of a file containing the contents in ‘‘native’’ (decoded) format. (If the filename starts with the ‘‘|’’ character, then this gives a command whose output is captured accordingly.) If a filename is not given, *mhn* will look for information in the user’s profile to determine how the different contents should be composed. This is accomplished by consulting a composition string, and executing it under **/bin/sh**, with the standard output set to the content. The composition string may contain these escapes:

%a additional arguments
%f filename containing content
%F %f, and stdout is not re-directed
%s subtype

First, *mhn* will look for an entry of the form:

mhn-compose-<type>/<subtype>

to determine the command to use to compose the content. If this isn’t found, *mhn* will look for an entry of the form:

mhn-compose-<type>

to determine the composition command. If this isn’t found, *mhn* will complain.

An example entry might be:

mhn-compose-audio/basic: record | raw2audio -F

Because commands like these will vary, depending on the display environment used for login, composition strings for different contents should probably be put in the file specified by the **$MHN** environment variable, instead of directly in your user profile.

The external-type directives are used to provide a reference to a content, rather than enclosing the contents itself. Hence, instead of providing a filename as with the type directives, external-parameters are supplied.

These look like regular parameters, so they must be separated accordingly, e.g.,

```
#@application/octet-stream; \
    type=tar; \
    x-conversions=compress [] \
    access-type=anon-ftp; \
    name="mh-mime.tar.Z"; \
    directory="mrose/mh-mime"; \
    site="ftp.ics.uci.edu"
```

By specifying ‘‘[]’’, an empty description string is given, and the start of the external-parameters is identified. These parameters are of the form:

access-type=	usually *anon-ftp* or *mail-server*
name=	filename
permission=	read-only or read-write
site=	hostname
directory=	directoryname (optional)
mode=	usually *ascii* or *image* (optional)
size=	number of octets
server=	mailbox
subject=	subject to send
body=	command to send for retrieval

For the forw directive, the user may optionally specify the name of the folder and which messages are to be forwarded. if a folder is not given, it defaults to the current folder. Similarly, if a message is not given, it defaults to the current message. Hence, the forw directive is similar to the *forw* (1) command, except that the former uses the MIME rules for encapsulation rather than those specified in RFC 934. Usage of the ‘–rfc934mode’ switch indicates whether *mhn* should attempt to utilize the encapsulation rules in such a way as to appear that RFC 934 is being used. If given, then RFC 934-compliant user-agents should be able to burst the message on reception -- providing that the messages being encapsulated do not contain encapsulated messages themselves. The drawback of this approach is that the encapsulations are generated by placing an extra newline at the end of the body of each message.

For the begin directive, the user must specify at least one content between the begin and end pairs.

For all of these directives, the user may include a brief description of the content between the ‘‘[’’ character and the ‘‘]’’ character. By default, *mhn* will generate a unique ‘‘Content-ID:’’ for each directive; however, the user may override this by defining the ID using the ‘‘<’’ and ‘‘>’’ characters. Putting this all together, here is a brief example of what a user’s components file might look like:

```
To:
cc:
Subject:
--------
#audio/basic [Flint phone]  \
   |raw2audio -F < /home/mrose/lib/multi-media/flint.au
#image/gif   [MTR’s photo] \
                  /home/mrose/lib/multi-media/mrose.gif
```

For a later example, we’ll call this components file *mhncomps*.

As noted earlier, in addition to directives, plaintext can be present. Plaintext is gathered, until a directive is found or the draft is exhausted, and this is made to form a text content. If the plaintext must contain a ‘‘#’’ at the beginning of a line, simply double it, e.g.,

```
##when sent, this line will start with only one #
```

If you want to end the plaintext prior to a directive, e.g., to have two plaintext contents adjacent, simply insert a line containing a single ‘‘#’’ character, e.g.,

```
this is the first content
#
and this is the second
```

Finally, if the plaintext starts with a line of the form:

```
Content-Description: text
```

then this will be used to describe the plaintext content. **NOTE WELL:** you must follow this line with a blank line before starting your text.

By default, plaintext is captured as a text/plain content. You can override this by starting the plaintext with ‘‘#<’’ followed by a content-type specification, e.g.,

```
#<text/richtext
this content will be tagged as text/richtext
#
and this content will be tagged as text/plain
```

Note that if you use the ‘‘#<’’ plaintext-form, then the content-description must be on the same line which identifies the content type of the plaintext.

If *mhn* is successful, it renames the original draft to start with the ‘‘,’’ character and end with the string ‘‘.orig’’, e.g., if you are editing the file ‘‘draft’’, it will be renamed to ‘‘,draft.orig’’. This allows you to easily recover the *mhn* composition file.

If the ‘-check’ switch is given, *mhn* will associate an integrity check with each content.

Automatic Composition

Note that MH will not invoke *mhn* automatically, unless you add this line to your .mh_profile file:

```
automhnproc: mhn
```

Otherwise, you must specifically give the command

```
What now? edit mhn
```

prior to sending the draft.

You can easily tailor MH to help you remember to do this. Suppose you have these lines in your profile:

```
mcomp:          -editor mprompter -form mhncomps
mprompter:      -noprepend -norapid
mprompter-next: mhn
```

where *mcomp* is a link to *comp* (1), and *mprompter* is a link to *prompter* (1). Then to send a message using the *mhncomps* components file above, the sequence is:

% **mcomp**
To: **user@host**
cc:
Subject: **multi-media message**

#audio/basic [Flint phone] \
    |raw2audio -F < /home/mrose/lib/multi-media/flint.au
#image/gif [MTR's photo] \
            /home/mrose/lib/multi-media/mrose.gif

--------Enter additional text

This message contains three contents.
<CTRL-D>

What now? **edit** (this invokes *mhn*)

What now? **send**

You have to remember to type the additional edit command, but it should be fairly obvious from the interaction.

Finally, you should consider adding this line to your profile:

```
lproc: show
```

This way, if you decide to **list** after invoking *mhn* as your editor, the command

```
What now? list
```

will work as you expect.

Sending Files via Mail

When you want to send a bunch of files to someone, you can run the *viamail* shell script, which is similar the tarmail command:

```
/usr/bs/mh-beta/lib/viamail mailpath ‘‘subject’’ files ...
```

viamail will archive the directories/files you name with *tar* (1), and then mail the compressed archive to the ‘mailpath’ with the given ‘subject’. The archive will be automatically split up into as many messages as necessary in order to get past most mailers.

Sometimes you want *viamail* to pause after posting a partial message. This is usually the case when you are running *sendmail* and expect to generate a lot of partial messages. If the first argument given to *viamail* starts with a dash, then it is interpreted as the number of seconds to pause in between postings, e.g.,

```
/usr/bs/mh-beta/lib/viamail -300 mailpath ‘‘subject’’ files ...
```

will pause 5 minutes in between each posting.

When these messages are received, invoke *mhn* once, with the list of messages, and the ‘–store’ command. The *mhn* program will then store exactly one message containing the archive. You can then use ‘–show’ to find out what’s inside; possibly followed by ‘–store’ to write the archive to a file where you can subsequently uncompress and untar it, e.g.,

```
% mhn -list all
 msg part  type/subtype             size description
   1       message/partial           47K part 1 of 4
   2       message/partial           47K part 2 of 4
   3       message/partial           47K part 3 of 4
   4       message/partial           18K part 4 of 4
% mhn -store all
% mhn -list -verbose last
 msg part  type/subtype             size description
   5       application/octet-stream 118K
             (extract with uncompress | tar xvpf -)
             type=tar
             x-conversions=compress
% mhn -show last
 msg part  type/subtype             size description
   5       application/octet-stream 118K
-- headers of message, followed by tar listing appears here
% mhn -store last
% uncompress < 5.tar.Z | tar xvpf -
```

Alternately, by using the '–auto' switch, *mhn* will automatically do the extraction for you, e.g.,

```
% mhn -list all
 msg part  type/subtype              size description
   1       message/partial           47K part 1 of 4
   2       message/partial           47K part 2 of 4
   3       message/partial           47K part 3 of 4
   4       message/partial           18K part 4 of 4
% mhn -store all
% mhn -list -verbose last
 msg part  type/subtype              size description
   5       application/octet-stream 118K
             (extract with uncompress | tar xvpf -)
             type=tar
             x-conversions=compress
% mhn -show last
 msg part  type/subtype              size description
   5       application/octet-stream 118K
-- headers of message, followed by tar listing appears here
% mhn -store -auto last
-- tar listing appears here as files are extracted
```

As the second *tar* listing is generated, the files are extracted. A prudent user will never put '–auto' in the .mh_profile file. The correct procedure is to first use '–show', to find out what will be extracted. Then *mhn* can be invoked with '–store' and '–auto' to perform the extraction.

User Environment

Because the display environment in which *mhn* operates may vary for a user, *mhn* will look for the environment variable **$MHN**. If present, this specifies the name of an additional user profile which should be read. Hence, when a user logs in on a particular display device, this environment variable should be set to refer to a file containing definitions useful for the display device. Normally, only entries of the form

```
mhn-show-<type>/<subtype>
mhn-show-<type>
```

need be present. Finally, *mhn* will attempt to consult one other additional user profile, e.g.,

```
/usr/bs/mh-beta/lib/mhn_defaults
```

which is created automatically during MH installation.

Files

$HOME/.mh_profile	The user profile
$MHN	Additional profile entries
/usr/bs/mh-beta/lib/mhn_defaults	System-default profile entries
/usr/bs/mh-beta/lib/mhl.headers	The headers template

Profile Components

Path:	To determine the user's MH directory
Current–Folder:	To find the default current folder
mhlproc:	Default program to display message headers
mhn-access-ftp:	Program to retrieve contents via FTP
mhn-cache	Public directory to store cached external contents
mhn-charset-<charset>	Template for environment to render character sets
mhn-compose-<type>*	Template for composing contents
mhn-private-cache	Personal directory to store cached external contents
mhn-show-<type>*	Template for displaying contents
mhn-storage	Directory to store contents
mhn-store-<type>*	Template for storing contents
moreproc:	Default program to display text/plain content

See Also

mhl(1)
MIME: Mechanisms for Specifying and Describing the Format of Internet Message Bodies (RFC 1521),
Proposed Standard for Message Encapsulation (RFC 934).

Defaults

'+folder' defaults to the current folder
'msgs' defaults to cur
'–noauto'
'–nocache'
'–nocheck'
'–noebcdicsafe'
'–form mhl.headers'
'–headers'
'–pause'
'–rcache ask'
'–realsize'
'–rfc934mode'
'–noserialonly'
'–show'
'–noverbose'
'–wcache ask'

Context

If a folder is given, it will become the current folder. The last message selected will become the current message.

Bugs

Partial messages contained within a multipart content are not reassembled with the '–store' switch.

NAME

mhook, rcvdist, rcvpack, rcvtty – MH receive-mail hooks

SYNOPSIS

/usr/bs/mh-beta/lib/rcvdist [–form formfile] [switches for *postproc*] address ... [–help]

/usr/bs/mh-beta/lib/rcvpack file [–help]

/usr/bs/mh-beta/lib/rcvtty [command] [–form formatfile] [–format string] [–bell] [–nobell] [–newline] [–nonewline] [–biff] [–help]

DESCRIPTION

A receive–mail hook is a program that is run whenever you receive a mail message. You do **NOT** invoke the hook yourself, rather the hook is invoked on your behalf by your system's Message Transport Agent. See *slocal* (1) for details on how to activate receive–mail hooks on your system.

Four programs are currently available as part of *MH*, *rcvdist* (redistribute incoming messages to additional recipients), *rcvpack* (save incoming messages in a *packf*'d file), and *rcvtty* (notify user of incoming messages). The fourth program, *rcvstore* (1) is described separately. They all reside in the */usr/bs/mh-beta/lib/* directory.

The *rcvdist* program will resend a copy of the message to all of the addresses listed on its command line. It uses the format string facility described in *mh–format* (5).

The *rcvpack* program will append a copy of the message to the file listed on its command line. Its use is obsoleted by the ''file'' action of *slocal*.

The *rcvtty* program executes the named file with the message as its standard input, and writes the resulting output on your terminal.

If no file is specified, or is bogus, etc., then *rcvtty* will instead write a one–line scan listing. Either the '–form formatfile' or '–format string' option may be used to override the default output format (see *mh–format* (5)). A newline is output before the message output, and the terminal bell is rung after the output. The '–nonewline' and '–nobell' options will inhibit these functions.

In addition to the standard *mh–format* (5) escapes, *rcvtty* also recognizes the following additional *component* escapes:

Escape	*Returns*	*Description*
body	string	the (compressed) first part of the body
dtimenow	date	the current date
folder	string	the name of the current folder

Normally, *rcvtty* obeys write permission as granted by *mesg* (1). With the '–biff' option, *rcvtty* will obey the notification status set by *biff* (1) instead. If the terminal access daemon (TTYD) is available on your system, then *rcvtty* will give its output to the daemon for output instead of writing on the user's terminal.

Files

/usr/bs/mh-beta/lib/mtstailor	tailor file
$HOME/.maildelivery	The file controlling local delivery
/usr/bs/mh-beta/lib/maildelivery	Rather than the standard file

See Also

rcvstore (1), mh–format(5), slocal(1)

Bugs

Only two return codes are meaningful, others should be.

NAME

mhparam – print MH profile components

SYNOPSIS

mhparam [components] [-all] [-component] [-nocomponent] [–help]

DESCRIPTION

Mhparam writes the value of the specified profile component to the standard output separated by newlines. If the profile component is not present, the default value (or nothing if there is no default) is printed.

If more than one component is specified in the ‘components’ list, the component value is preceded by the component name. If ‘–component’ is specified, the component name is displayed even when only one component is specified. If ‘–nocomponent’ is specified, the component name is not displayed even when more than one component is specified.

If ‘–all’ is specified, all components if the MH profile are displayed and other arguments are ignored.

Examples:

```
% mhparam path
Mail

% mhparam mhlproc
/usr/bs/mh-beta/lib/mhl

% mhparam –component path
Path:       Mail

% mhparam AliasFile rmmproc
AliasFile:  aliases
rmmproc:    rmmproc

% mhparam –nocomponent AliasFile rmmproc
aliases
rmmproc
```

Mhparam is also useful in back–quoted operations:

```
% fgrep cornell.edu ‘mhpath +‘/‘mhparam aliasfile‘
```

Files

$HOME/.mh_profile The user profile

See Also

mh-profile (5)

Defaults

‘–nocomponent’ if only one component is specified
‘–component’ if more than one component is specified
‘components’ defaults to none

Context

None

NAME

mhpath – print full pathnames of MH messages and folders

SYNOPSIS

mhpath [+folder] [msgs] [–help]

DESCRIPTION

Mhpath expands and sorts the message list ‘msgs’ and writes the full pathnames of the messages to the standard output separated by newlines. If no ‘msgs’ are specified, *mhpath* outputs the folder pathname instead. If the only argument is ‘+’, your MH *Path* is output; this can be useful is shell scripts.

Contrasted with other MH commands, a message argument to *mhpath* may often be intended for *writing*. Because of this:

1) the name ‘‘new’’ has been added to *mhpath*’s list of reserved message names (the others are ‘‘first’’, ‘‘last’’, ‘‘prev’’, ‘‘next’’, ‘‘cur’’, and ‘‘all’’). The new message is equivalent to the message after the last message in a folder (and equivalent to 1 in a folder without messages). The ‘‘new’’ message may not be used as part of a message range.

2) Within a message list, the following designations may refer to messages that do not exist: a single numeric message name, the single message name ‘‘cur’’, and (obviously) the single message name ‘‘new’’. All other message designations must refer to at least one existing message.

3) An empty folder is not in itself an error.

Message numbers greater than the highest existing message in a folder as part of a range designation are replaced with the next free message number.

Examples: The current folder foo contains messages 3 5 6. Cur is 4.

```
% mhpath
/r/phyl/Mail/foo

% mhpath all
/r/phyl/Mail/foo/3
/r/phyl/Mail/foo/5
/r/phyl/Mail/foo/6

% mhpath 2001
/r/phyl/Mail/foo/7

% mhpath 1–2001
/r/phyl/Mail/foo/3
/r/phyl/Mail/foo/5
/r/phyl/Mail/foo/6

% mhpath new
/r/phyl/Mail/foo/7

% mhpath last new
/r/phyl/Mail/foo/6
```

```
/r/phyl/Mail/foo/7

% mhpath last–new
bad message list ‘‘last–new’’.

% mhpath cur
/r/phyl/Mail/foo/4

% mhpath 1–2
no messages in range ‘‘1–2’’.

% mhpath first:2
/r/phyl/Mail/foo/3
/r/phyl/Mail/foo/5

% mhpath 1 2
/r/phyl/Mail/foo/1
/r/phyl/Mail/foo/2
```

MHpath is also useful in back–quoted operations:

```
% cd ‘mhpath +inbox‘

% echo ‘mhpath +‘
/r/phyl/Mail
```

Files

$HOME/.mh_profile The user profile

Profile Components

Path:	To determine the user’s MH directory
Current–Folder:	To find the default current folder

See Also

folder(1)

Defaults

‘+folder’ defaults to the current folder
‘msgs’ defaults to none

Context

None

Bugs

Like all MH commands, *mhpath* expands and sorts [msgs]. So don't expect

```
mv `mhpath 501 500`
```

to move 501 to 500. Quite the reverse. But

```
mv `mhpath 501` `mhpath 500`
```

will do the trick.

Out of range message 0 is treated far more severely than large out of range message numbers.

NAME

msgchk – check for messages

SYNOPSIS

msgchk [–date] [–nodate] [–notify all/mail/nomail] [–nonotify all/mail/nomail] [–host host] [–user user]
[–apop] [–noapop] [–rpop] [–norpop] [users ...] [–help]

DESCRIPTION

The *msgchk* program checks all known mail drops for mail waiting for you. For those drops which have mail for you, *msgchk* will indicate if it believes that you have seen the mail in question before.

The '–notify type' switch indicates under what circumstances *msgchk* should produce a message. The default is '–notify all' which says that *msgchk* should always report the status of the users maildrop. Other values for 'type' include 'mail' which says that *msgchk* should report the status of waiting mail; and, 'nomail' which says that *msgchk* should report the status of empty maildrops. The '–nonotify type' switch has the inverted sense, so '–nonotify all' directs *msgchk* to never report the status of maildrops. This is useful if the user wishes to check *msgchk*'s exit status. A non–zero exit status indicates that mail was **not** waiting for at least one of the indicated users.

If *msgchk* produces output, then the '–date' switch directs *msgchk* to print out the last date mail was read, if this can be determined.

If the local host is configured as a POP client, or if the '–host host' switch is given, *msgchk* will query the POP service host as to the status of mail waiting. If the '–user user' switch is not given, then the current username is used. Normally, *msgchk* will prompt for a password to use. However, if the '–apop' switch is given, *msgchk* will generate authentication credentials to provide for origin authentication and replay protection, but which do not involve sending a password in the clear over the network. Otherwise, if the '–rpop' switch is given, then *msgchk* will try to use a ''trusted'' connection (ala the BSD r-commands).

Files

$HOME/.mh_profile	The user profile
/usr/bs/mh-beta/lib/mtstailor	tailor file
/usr/spool/mail/$USER	Location of mail drop

Profile Components

None

See Also

Post Office Protocol - version 3 (aka RFC–1081),
inc(1)

Defaults

'user' defaults to the current user
'–date'
'–notify all'
'–rpop'

Context

None

NAME

msh – MH shell (and BBoard reader)

SYNOPSIS

msh [–prompt string] [–scan] [–noscan] [–topcur] [–notopcur] [file] [–help]

DESCRIPTION

msh is an interactive program that implements a subset of the normal *MH* commands operating on a single file in *packf*'d format. That is, *msh* is used to read a file that contains a number of messages, as opposed to the standard *MH* style of reading a number of files, each file being a separate message in a folder. *msh*'s chief advantage is that the normal *MH* style does not allow a file to have more than one message in it. Hence, *msh* is ideal for reading *BBoards*, as these files are delivered by the transport system in this format. In addition, *msh* can be used on other files, such as message archives which have been *pack*ed (see *packf* (1)). Finally, *msh* is an excellent *MH* tutor. As the only commands available to the user are *MH* commands, this allows *MH* beginners to concentrate on how commands to *MH* are formed and (more or less) what they mean.

When invoked, *msh* reads the named file, and enters a command loop. The user may type most of the normal *MH* commands. The syntax and semantics of these commands typed to *msh* are identical to their *MH* counterparts. In cases where the nature of *msh* would be inconsistent (e.g., specifying a '+folder' with some commands), *msh* will duly inform the user. The commands that *msh* currently supports (in some slightly modified or restricted forms) are:

ali
burst
comp
dist
folder
forw
inc
mark
mhmail
mhn
msgchk
next
packf
pick
prev
refile
repl
rmm
scan
send
show
sortm
whatnow
whom

In addition, *msh* has a ''help'' command which gives a brief overview. To terminate *msh*, type CTRL–D, or use the ''quit'' command. If *msh* is being invoked from *bbc*, then typing CTRL–D will also tell *bbc* to exit as well, while using the ''quit'' command will return control to *bbc*, and *bbc* will continue examining the list of BBoards that it is scanning.

If the file is writable and has been modified, then using ‘‘quit’’ will query the user if the file should be updated.

The ‘–prompt string’ switch sets the prompting string for *msh*.

You may wish to use an alternate *MH* profile for the commands that *msh* executes; see *mh-profile* (5) for details about the **$MH** envariable.

When invoked from *bbc*, two special features are enabled: First, the ‘–scan’ switch directs *msh* to do a ‘scan unseen’ on start–up if new items are present in the BBoard. This feature is best used from *bbc*, which correctly sets the stage. Second, the *mark* command in *msh* acts specially when you are reading a BBoard, since *msh* will consult the sequence ‘‘unseen’’ in determining what messages you have actually read. When *msh* exits, it reports this information to *bbc*. In addition, if you give the *mark* command with no arguments, *msh* will interpret it as ‘mark –sequence unseen –delete –nozero all’ Hence, to discard all of the messages in the current BBoard you’re reading, just use the *mark* command with no arguments.

Normally, the ‘‘exit’’ command is identical to the ‘‘quit’’ command in *msh*. When run under *bbc* however, ‘‘exit’’ directs *msh* to mark all messages as seen and then ‘‘quit’’. For speedy type–in, this command is often abbreviated as just ‘‘e’’.

When invoked from *vmh*, another special feature is enabled: The ‘topcur’ switch directs *msh* to have the current message ‘‘track’’ the top line of the *vmh* scan window. Normally, *msh* has the current message ‘‘track’’ the center of the window (under ‘–notopcur’, which is the default).

msh supports an output redirection facility. Commands may be followed by one of

> *file*	write output to *file*
>> *file*	append output to *file*
| *command*	pipe output to UNIX *command*

If *file* starts with a ‘~’ (tilde), then a *csh*-like expansion takes place. Note that *command* is interpreted by *sh* (1). Also note that *msh* does NOT support history substitutions, variable substitutions, or alias substitutions.

When parsing commands to the left of any redirection symbol, *msh* will honor ‘\’ (back–slash) as the quote next–character symbol, and ‘"’ (double–quote) as quote–word delimiters. All other input tokens are separated by whitespace (spaces and tabs).

Files

$HOME/.mh_profile	The user profile
/usr/bs/mh-beta/lib/mtstailor	tailor file

Profile Components

Path:	To determine the user’s MH directory
Msg–Protect:	To set mode when creating a new ‘file’
fileproc:	Program to file messages
showproc:	Program to show messages

See Also

bbc(1)

Defaults

‘file’ defaults to ‘‘./msgbox’’
‘–prompt (msh) ’
‘–noscan’
‘–notopcur’

Context

None

Bugs

The argument to the ‘–prompt’ switch must be interpreted as a single token by the shell that invokes *msh*. Therefore, one must usually place the argument to this switch inside double–quotes.

There is a strict limit of messages per file in *packf*’d format which *msh* can handle. Usually, this limit is 1000 messages.

Please remember that *msh* is not the *CShell*, and that a lot of the nice facilities provided by the latter are not present in the former.

In particular, *msh* does not understand back–quoting, so the only effective way to use *pick* inside *msh* is to always use the ‘–seq select’ switch. Clever users of *MH* will put the line

```
pick:  –seq  select  –list
```

in their .mh_profile file so that *pick* works equally well from both the shell and *msh*.

sortm always uses ‘‘–noverbose’’ and if ‘‘–textfield field‘‘ is used, ‘‘–limit 0’’.

The *msh* program inherits most (if not all) of the bugs from the *MH* commands it implements.

NAME

next – show the next message

SYNOPSIS

next [+folder] [–header] [–noheader] [–showproc program] [–noshowproc] [switches for *showproc*] [–help]

DESCRIPTION

Next performs a *show* on the next message in the specified (or current) folder. Like *show*, it passes any switches on to the program *showproc*, which is called to list the message. This command is almost exactly equivalent to ‘‘show next’’. Consult the manual entry for *show* (1) for all the details.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user’s MH directory
Current–Folder:	To find the default current folder
showproc:	Program to show the message

See Also

show(1), prev(1)

Defaults

‘+folder’ defaults to the current folder
‘–header’

Context

If a folder is specified, it will become the current folder. The message that is shown (i.e., the next message in sequence) will become the current message.

Bugs

next is really a link to the *show* program. As a result, if you make a link to *next* and that link is not called *next*, your link will act like *show* instead. To circumvent this, add a profile–entry for the link to your *MH* profile and add the argument *next* to the entry.

NAME

packf – compress an MH folder into a single file

SYNOPSIS

packf [+folder] [msgs] [–file name] [–help]

DESCRIPTION

Packf takes messages from a folder and copies them to a single file. Each message in the file is separated by four CTRL–A's and a newline (identical to the way messages are stored in your receiving mail drop). Messages packed can be unpacked using *inc*.

If the *name* given to the '–file name' switch exists, then the messages specified will be appended to the end of the file, otherwise the file will be created and the messages appended.

Files

$HOME/.mh_profile	The user profile
.msgbox.map	A binary index of the file

Profile Components

Path:	To determine the user's MH directory
Current–Folder:	To find the default current folder
Msg–Protect:	To set mode when creating a new 'file'

See Also

inc(1)

Defaults

'+folder' defaults to the current folder
'msgs' defaults to all
'–file ./msgbox'

Context

If a folder is given, it will become the current folder. The first message packed will become the current message.

Bugs

Packf doesn't handle the old UUCP-style ''mbox'' format (used by *SendMail*). To pack messages into this format, use the script */usr/bs/mh-beta/lib/packmbox*. Note that *packmbox* does not take the '–file' option of *packf*, and instead writes its output on *stdout*.

NAME

pick – select messages by content

SYNOPSIS

```
pick  ⎧–cc            ⎫ [+folder] [msgs] [–help]
      ⎪–date          ⎪ [–before date] [–after date] [–datefield field]
      ⎪–from          ⎪
      ⎨–search        ⎬ pattern [–and ...] [–or ...] [–not ...] [–lbrace ... –rbrace]
      ⎪–subject       ⎪
      ⎪–to            ⎪ [–sequence name ...] [–public] [–nopublic] [–zero] [–nozero]
      ⎩– –component   ⎭ [–list] [–nolist]
```

typically:

```
scan ‘pick –from jones‘
pick –to holloway –sequence select
show ‘pick –before friday‘
```

DESCRIPTION

Pick searches messages within a folder for the specified contents, and then identifies those messages. Two types of search primitives are available: pattern matching and date constraint operations.

A modified *grep*(1) is used to perform the matching, so the full regular expression (see *ed*(1)) facility is available within ‘pattern’. With ‘–search’, ‘pattern’ is used directly, and with the others, the grep pattern constructed is:

‘‘component[\t]*:.*pattern’’

This means that the pattern specified for a ‘–search’ will be found everywhere in the message, including the header and the body, while the other pattern matching requests are limited to the single specified component. The expression

‘– –component pattern’

is a shorthand for specifying

‘–search ‘‘component[\t]*:.*pattern’’ ’

It is used to pick a component which is not one of ‘‘To:’’, ‘‘cc:’’, ‘‘Date:’’, ‘‘From:’’, or ‘‘Subject:’’. An example is ‘pick – –reply–to pooh’.

Pattern matching is performed on a per–line basis. Within the header of the message, each component is treated as one long line, but in the body, each line is separate. Lower–case letters in the search pattern will match either lower or upper case in the message, while upper case will match only upper case.

Note that since the ‘–date’ switch is a pattern matching operation (as described above), to find messages sent on a certain date the pattern string must match the text of the ‘‘Date:’’ field of the message.

Independent of any pattern matching operations requested, the switches ‘–after date’ or ‘–before date’ may also be used to introduce date/time contraints on all of the messages. By default, the ‘‘Date:’’ field is consulted, but if another date yielding field (such as ‘‘BB–Posted:’’ or ‘‘Delivery–Date:’’) should be used, the ‘–datefield field’ switch may be used.

With '–before' and '–after', *pick* will actually parse the date fields in each of the messages specified in 'msgs' and compare them to the date/time specified. If '–after' is given, then only those messages whose ''Date:'' field value is chronologically after the date specified will be considered. The '–before' switch specifies the complimentary action.

Both the '–after' and '–before' switches take legal 822–style date specifications as arguments. *Pick* will default certain missing fields so that the entire date need not be specified. These fields are (in order of defaulting): timezone, time and timezone, date, date and timezone. All defaults are taken from the current date, time, and timezone.

In addition to 822–style dates, *pick* will also recognize any of the days of the week (''sunday'', ''monday'', and so on), and the special dates ''today'', ''yesterday'' (24 hours ago), and ''tomorrow'' (24 hours from now). All days of the week are judged to refer to a day in the past (e.g., telling *pick* ''saturday'' on a ''tuesday'' means ''last saturday'' not ''this saturday'').

Finally, in addition to these special specifications, *pick* will also honor a specification of the form ''–dd'', which means ''dd days ago''.

Pick supports complex boolean operations on the searching primitives with the '–and', '–or', '–not', and '–lbrace ... –rbrace' switches. For example,

```
pick –after yesterday –and –lbrace –from freida –or –from fear –rbrace
```

identifies messages recently sent by ''frieda'' or ''fear''.

The matching primitives take precedence over the '–not' switch, which in turn takes precedence over '–and' which in turn takes precedence over '–or'. To override the default precedence, the '–lbrace' and '–rbrace' switches are provided, which act just like opening and closing parentheses in logical expressions.

If no search criteria are given, all the messages specified on the command line are selected (this defaults to ''all'').

Once the search has been performed, if the '–list' switch is given, the message numbers of the selected messages are written to the standard output separated by newlines. This is *extremely* useful for quickly generating arguments for other *MH* programs by using the ''backquoting'' syntax of the shell. For example, the command

```
scan ‘pick +todo –after ''31 Mar 83 0123 PST''‘
```

says to *scan* those messages in the indicated folder which meet the appropriate criterion. Note that since *pick* 's context changes are written out prior to *scan* 's invocation, you need not give the folder argument to *scan* as well.

Regardless of the operation of the '–list' switch, the '–sequence name' switch may be given once for each sequence the user wishes to define. For each sequence named, that sequence will be defined to mean exactly those messages selected by *pick*. For example,

```
pick –from frated –seq fred
```

defines a new message sequence for the current folder called ''fred'' which contains exactly those messages that were selected.

Note that whenever *pick* processes a '–sequence name' switch, it sets '–nolist'.

By default, *pick* will zero the sequence before adding it. This action can be disabled with the '–nozero' switch, which means that the messages selected by *pick* will be added to the sequence, if it already exists, and any messages already a part of that sequence will remain so.

The '–public' and '–nopublic' switches are used by *pick* in the same way *mark* uses them.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user's MH directory
Current–Folder:	To find the default current folder

See Also

mark(1)

Defaults

'+folder' defaults to the current folder
'msgs' defaults to all
'–datefield date'
'–nopublic' if the folder is read–only, '–public' otherwise
'–zero'
'–list' is the default if no '–sequence', '–nolist' otherwise

Context

If a folder is given, it will become the current folder.

History

In previous versions of *MH*, the *pick* command would *show*, *scan*, or *refile* the selected messages. This was rather ''inverted logic'' from the UNIX point of view, so *pick* was changed to define sequences and output those sequences. Hence, *pick* can be used to generate the arguments for all other *MH* commands, instead of giving *pick* endless switches for invoking those commands itself.

Also, previous versions of *pick* balked if you didn't specify a search string or a date/time constraint. The current version does not, and merely matches the messages you specify. This lets you type something like:

```
show `pick last:20 –seq fear`
```

instead of typing

```
mark –add –nozero –seq fear last:20
show fear
```

Finally, timezones used to be ignored when comparing dates: they aren't any more.

Helpful Hints

Use ''pick sequence –list'' to enumerate the messages in a sequence (such as for use by a shell script).

Bugs

The argument to the '–after' and '–before' switches must be interpreted as a single token by the shell that invokes *pick*. Therefore, one must usually place the argument to this switch inside double–quotes. Furthermore, any occurance of '–datefield' must occur prior to the '–after' or '–before' switch it applies to.

If *pick* is used in a back–quoted operation, such as

```
scan `pick –from jones`
```

and *pick* selects no messages (e.g., no messages are from ''jones''), then the shell will still run the outer command (e.g., ''scan''). Since no messages were matched, *pick* produced no output, and the argument given to the outer command as a result of backquoting *pick* is empty. In the case of *MH* programs, the outer command now acts as if the default 'msg' or 'msgs' should be used (e.g., ''all'' in the case of *scan*). To prevent this unexpected behavior, if '–list' was given, and if its standard output is not a tty, then *pick* outputs the illegal message number ''0'' when it fails. This lets the outer command fail gracefully as well.

The pattern syntax ''[l-r]'' is not supported; each letter to be matched must be included within the square brackets.

NAME

prev – show the previous message

SYNOPSIS

prev [+folder] [–header] [–noheader] [–showproc program] [–noshowproc] [–switches for *showproc*] [–help]

DESCRIPTION

Prev performs a *show* on the previous message in the specified (or current) folder. Like *show*, it passes any switches on to the program named by *showproc*, which is called to list the message. This command is almost exactly equivalent to ‘‘show prev’’. Consult the manual entry for *show* (1) for all the details.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user’s MH directory
Current–Folder:	To find the default current folder
showproc:	Program to show the message

See Also

show(1), next(1)

Defaults

‘+folder’ defaults to the current folder
‘–header’

Context

If a folder is specified, it will become the current folder. The message that is shown (i.e., the previous message in sequence) will become the current message.

Bugs

prev is really a link to the *show* program. As a result, if you make a link to *prev* and that link is not called *prev*, your link will act like *show* instead. To circumvent this, add a profile–entry for the link to your *MH* profile and add the argument *prev* to the entry.

NAME

prompter – prompting editor front-end for MH

SYNOPSIS

prompter [–erase chr] [–kill chr] [–prepend] [–noprepend] [–rapid] [–norapid] [–doteof] [–nodoteof] file [–help]

DESCRIPTION

This program is normally not invoked directly by users but takes the place of an editor and acts as an editor front–end. It operates on an 822–style message draft skeleton specified by file, normally provided by *comp*, *dist*, *forw*, or *repl*.

Prompter is an editor which allows rapid composition of messages. It is particularly useful to network and low–speed (less than 2400 baud) users of *MH*. It is an *MH* program in that it can have its own profile entry with switches, but it is not invoked directly by the user. The commands *comp*, *dist*, *forw*, and *repl* invoke *prompter* as an editor, either when invoked with '–editor prompter', or by the profile entry ''Editor: prompter'', or when given the command 'edit prompter' at ''What now?'' level.

For each empty component *prompter* finds in the draft, the user is prompted for a response; A <RETURN> will cause the whole component to be left out. Otherwise, a '\' preceding a <RETURN> will continue the response on the next line, allowing for multiline components. Continuation lines **must** begin with a space or tab.

Each non–empty component is copied to the draft and displayed on the terminal.

The start of the message body is denoted by a blank line or a line of dashes. If the body is non–empty, the prompt, which isn't written to the file, is

''--------Enter additional text'',

or (if '–prepend' was given)

''--------Enter initial text''.

Message–body typing is terminated with an end–of–file (usually CTRL–D). With the '–doteof' switch, a period on a line all by itself also signifies end–of–file. At this point control is returned to the calling program, where the user is asked ''What now?''. See *whatnow* for the valid options to this query.

By using the '–prepend' switch, the user can add type–in to the beginning of the message body and have the rest of the body follow. This is useful for the *forw* command.

By using the '–rapid' switch, if the draft already contains text in the message–body, it is not displayed on the user's terminal. This is useful for low–speed terminals.

The line editing characters for kill and erase may be specified by the user via the arguments '–kill chr' and '–erase chr', where chr may be a character; or '\nnn', where ''nnn'' is the octal value for the character.

An interrupt (usually CTRL–C) during component typing will abort *prompter* and the *MH* command that invoked it. An interrupt during message–body typing is equivalent to CTRL–D, for historical reasons. This means that *prompter* should finish up and exit.

The first non–flag argument to *prompter* is taken as the name of the draft file, and subsequent non–flag arguments are ignored.

Files

$HOME/.mh_profile	The user profile
/tmp/prompter*	Temporary copy of message

Profile Components

prompter–next:	To name the editor to be used on exit from *prompter*
Msg–Protect:	To set mode when creating a new draft

See Also

comp(1), dist(1), forw(1), repl(1), whatnow(1)

Defaults

'–prepend'
'–norapid'
'–nodoteof'

Context

None

Helpful Hints

The '–rapid' option is particularly useful with *forw*, and '–noprepend' is useful with *comp –use*.

The user may wish to link *prompter* under several names (e.g., ''rapid'') and give appropriate switches in the profile entries under these names (e.g., ''rapid: -rapid''). This facilitates invoking prompter differently for different *MH* commands (e.g., ''forw: -editor rapid'').

Bugs

Prompter uses *stdio* (3), so it will lose if you edit files with nulls in them.

NAME

rcvstore – incorporate new mail asynchronously

SYNOPSIS

/usr/bs/mh-beta/lib/rcvstore [+folder] [–create] [–nocreate] [–sequence name ...] [–public] [–nopublic]
[–zero] [–nozero] [–help]

DESCRIPTION

Rcvstore incorporates a message from the standard input into an *MH* folder. If ‘+folder’ isn’t specified, a folder in the user’s *MH* directory will be used, either that specified by the ‘‘Inbox:’’ entry in the user’s profile, or the folder named ‘‘inbox’’. The new message being incorporated is assigned the next highest number in the folder. If the specified (or default) folder doesn’t exist, then it will be created if the ‘–create’ option is specified, otherwise *rcvstore* will exit.

If the user’s profile contains a ‘‘Msg–Protect: nnn’’ entry, it will be used as the protection on the newly created messages, otherwise the *MH* default of 0644 will be used. During all operations on messages, this initially assigned protection will be preserved for each message, so *chmod*(1) may be used to set a protection on an individual message, and its protection will be preserved thereafter.

Rcvstore will incorporate anything except zero length messages into the user’s MH folder.

If the profile entry ‘‘Unseen–Sequence’’ is present and non–empty, then *rcvstore* will add the newly incorporated message to each sequence named by the profile entry. This is similar to the ‘‘Previous–Sequence’’ profile entry supported by all *MH* commands which take ‘msgs’ or ‘msg’ arguments. Note that *rcvstore* will not zero each sequence prior to adding messages.

Furthermore, the incoming messages may be added to user-defined sequences as they arrive by appropriate use of the ‘–sequence’ option. As with *pick*, use of the ‘–zero’ and ‘–nozero’ switches can also be used to zero old sequences or not. Similarly, use of the ‘–public’ and ‘–nopublic switches may be used to force additions to public and private sequences.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user’s MH directory
Folder–Protect:	To set mode when creating a new folder
Inbox:	To find the default inbox
Msg–Protect:	To set mode when creating a new message
Unseen–Sequence:	To name sequences denoting unseen messages

See Also

inc(1), pick(1), mh–mail(5)

Defaults

‘+folder’ defaults to ‘‘inbox’’
‘–create’
‘–nopublic’ if the folder is read–only, ‘–public’ otherwise
‘–nozero’

Context

No context changes will be attempted, with the exception of sequence manipulation.

Bugs

If you use the ‘‘Unseen–Sequence’’ profile entry, *rcvstore* could try to update the context while another *MH* process is also trying to do so. This can cause the context to become corrupted. To avoid this, do not use *rcvstore* if you use the ‘‘Unseen–Sequence’’ profile entry.

NAME

refile – file message in other folders

SYNOPSIS

refile [msgs] [–draft] [–link] [–nolink] [–preserve] [–nopreserve] [–src +folder] [–file file] [–rmmproc program] [–normmproc] +folder ... [–help]

DESCRIPTION

Refile moves (*mv* (1)) or links (*ln* (1)) messages from a source folder into one or more destination folders. If you think of a message as a sheet of paper, this operation is not unlike filing the sheet of paper (or copies) in file cabinet folders. When a message is filed, it is linked into the destination folder(s) if possible, and is copied otherwise. As long as the destination folders are all on the same file system, multiple filing causes little storage overhead. This facility provides a good way to cross–file or multiply–index messages. For example, if a message is received from Jones about the ARPA Map Project, the command

```
refile  cur  +jones  +Map
```

would allow the message to be found in either of the two folders 'jones' or 'Map'.

The option '–file file' directs *refile* to use the specified file as the source message to be filed, rather than a message from a folder. Note that the file should be a validly formatted message, just like any other *MH* message. It should **NOT** be in mail drop format (to convert a file in mail drop format to a folder of *MH* messages, see *inc* (1)).

If a destination folder doesn't exist, *refile* will ask if you want to create it. A negative response will abort the file operation. If the standard input for *refile* is *not* a tty, then *refile* will not ask any questions and will proceed as if the user answered ''yes'' to all questions.

The option '–link' preserves the source folder copy of the message (i.e., it does a *ln*(1) rather than a *mv*(1)), whereas, '–nolink' deletes the filed messages from the source folder. Normally, when a message is filed, it is assigned the next highest number available in each of the destination folders. Use of the '–preserve' switch will override this message renaming, but name conflicts may occur, so use this switch cautiously.

If '–link' is not specified (or '–nolink' is specified), the filed messages will be removed from the source folder, by renaming them with a site-dependent prefix (usually a comma).

If the user has a profile component such as

```
rmmproc:        /bin/rm
```

then *refile* will instead call the named program to delete the message files. The user may specify '–rmmproc program' on the command line to override this profile specification. The `-normmproc' option forces the message files to be deleted by renaming them as described above.

The '–draft' switch tells *refile* to file the <mh–dir>/draft.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user's MH directory
Current–Folder:	To find the default current folder
Folder–Protect:	To set mode when creating a new folder
rmmproc:	Program to delete the message

See Also

folder(1)

Defaults

'–src +folder' defaults to the current folder
'msgs' defaults to cur
'–nolink'
'–nopreserve'

Context

If '–src +folder' is given, it will become the current folder. If neither '–link' nor 'all' is specified, the current message in the source folder will be set to the last message specified; otherwise, the current message won't be changed.

If the Previous–Sequence profile entry is set, in addition to defining the named sequences from the source folder, *refile* will also define those sequences for the destination folders. See *mh–sequence* (5) for information concerning the previous sequence.

Bugs

Since *refile* uses your *rmmproc* to delete the message, the *rmmproc* must **NOT** call *refile* without specifying '–normmproc', or you will create an infinte loop.

NAME

repl – reply to a message

SYNOPSIS

repl [+folder] [msg] [–annotate] [–noannotate] [–cc all/to/cc/me] [–nocc all/to/cc/me] [–draftfolder +folder] [–draftmessage msg] [–nodraftfolder] [–editor editor] [–noedit] [–fcc +folder] [–filter filterfile] [–form formfile] [–inplace] [–noinplace] [–query] [–noquery] [–width columns] [–whatnowproc program] [–nowhatnowproc] [–help]

DESCRIPTION

Repl aids a user in producing a reply to an existing message. *Repl* uses a reply template to guide its actions when constructing the message draft of the reply. In its simplest form (with no arguments), it will set up a message–form skeleton in reply to the current message in the current folder, and invoke the whatnow shell. The default reply template will direct *repl* to construct the composed message as follows:

```
To: <Reply–To> or <From>
cc: <cc>, <To>, and yourself
Subject: Re: <Subject>
In–reply–to: Your message of <Date>.
             <Message–Id>
```

where field names enclosed in angle brackets (< >) indicate the contents of the named field from the message to which the reply is being made. A reply template is simply a format file. See *mh–format* (5) for the details.

The '–cc type' switch takes an argument which specifies who gets placed on the ''cc:'' list of the reply. The '–query' switch modifies the action of '–cc type' switch by interactively asking you if each address that normally would be placed in the ''To:'' and ''cc:'' list should actually be sent a copy. (This is useful for special–purpose replies.) Note that the position of the '–cc' and '–nocc' switches, like all other switches which take a positive and negative form, is important.

Lines beginning with the fields ''To:'', ''cc:'', and ''Bcc:'' will be standardized and have duplicate addresses removed. In addition, the '–width columns' switch will guide *repl*'s formatting of these fields.

If the file named ''replcomps'' exists in the user's MH directory, it will be used instead of the default form. In either case, the file specified by '–form formfile' will be used if given.

If the draft already exists, *repl* will ask you as to the disposition of the draft. A reply of **quit** will abort *repl*, leaving the draft intact; **replace** will replace the existing draft with a blank skeleton; and **list** will display the draft.

See *comp* (1) for a description of the '–editor' and '–noedit' switches. Note that while in the editor, the message being replied to is available through a link named ''@'' (assuming the default *whatnowproc*). In addition, the actual pathname of the message is stored in the envariable **$editalt**, and the pathname of the folder containing the message is stored in the envariable **$mhfolder**.

Although *repl* uses the '–form formfile' switch to direct it how to construct the beginning of the draft, the '–filter filterfile' switch directs *repl* as to how the message being replied–to should be formatted in the body of the draft. If '–filter' is not specified, then the message being replied–to is not included in the body of the draft. If '–filter filterfile' is specified, then the message being replied–to is filtered (re–formatted) prior to being output to the body of the draft. The filter file for *repl* should be a standard form file for *mhl*, as *repl* will invoke *mhl* to format the message being replied–to. There is no default message filter ('–filter' must be followed by a file name). A filter file that is commonly used is:

```
:
body:nocomponent,compwidth=9,offset=9
```

which says to output a blank line and then the body of the message being replied–to, indented by one tab–stop. Another format popular on USENET is:

```
message-id:nocomponent, nonewline, formatfield=‘‘In message %{text}, ’’
from:nocomponent, formatfield=‘‘%(friendly{text}) writes:’’
body:component=‘‘>’’, overflowtext=‘‘>’’, overflowoffset=0
```

Which cites the Message-ID and author of the message being replied–to, and then outputs each line of the body prefaced with the ''>'' character.

If the '–annotate' switch is given, the message being replied–to will be annotated with the lines

```
Replied: date
Replied: addrs
```

where the address list contains one line for each addressee. The annotation will be done only if the message is sent directly from *repl*. If the message is not sent immediately from *repl*, ''comp –use'' may be used to re–edit and send the constructed message, but the annotations won't take place. The '–inplace' switch causes annotation to be done in place in order to preserve links to the annotated message.

The '–fcc +folder' switch can be used to automatically specify a folder to receive Fcc:s. More than one folder, each preceeded by '–fcc' can be named.

In addition to the standard *mh–format* (5) escapes, *repl* also recognizes the following additional *component* escape:

Escape	*Returns*	*Description*
fcc	string	Any folders specified with '–fcc folder'

To avoid reiteration, *repl* strips any leading 'Re: ' strings from the *subject* component.

The '–draftfolder +folder' and '–draftmessage msg' switches invoke the *MH* draft folder facility. This is an advanced (and highly useful) feature. Consult the **Advanced Features** section of the *MH* manual for more information.

Upon exiting from the editor, *repl* will invoke the *whatnow* program. See *whatnow* (1) for a discussion of available options. The invocation of this program can be inhibited by using the '–nowhatnowproc' switch. (In truth of fact, it is the *whatnow* program which starts the initial edit. Hence, '–nowhatnowproc' will prevent any edit from occurring.)

Files

/usr/bs/mh-beta/lib/replcomps	The reply template
or <mh–dir>/replcomps	Rather than the standard template
$HOME/.mh_profile	The user profile
<mh–dir>/draft	The draft file

Profile Components

Path:	To determine the user's MH directory
Alternate–Mailboxes:	To determine the user's mailboxes
Current–Folder:	To find the default current folder
Draft–Folder:	To find the default draft–folder
Editor:	To override the default editor
Msg–Protect:	To set mode when creating a new message (draft)
fileproc:	Program to refile the message
mhlproc:	Program to filter message being replied–to
whatnowproc:	Program to ask the ''What now?'' questions

See Also

comp(1), dist(1), forw(1), send(1), whatnow(1), mh–format(5)

Defaults

'+folder' defaults to the current folder
'msg' defaults to cur
'–nocc all' at ATHENA sites, '–cc all' otherwise
'–noannotate'
'–nodraftfolder'
'–noinplace'
'–noquery'
'–width 72'

Context

If a folder is given, it will become the current folder. The message replied–to will become the current message.

History

Prior to using the format string mechanism, '–noformat' used to cause address headers to be output as–is. Now all address fields are formatted using Internet standard guidelines.

Bugs

If any addresses occur in the reply template, addresses in the template that do not contain hosts are defaulted incorrectly. Instead of using the localhost for the default, *repl* uses the sender's host. Moral of the story: if you're going to include addresses in a reply template, include the host portion of the address.

The '–width columns' switch is only used to do address-folding; other headers are not line–wrapped.

If *whatnowproc* is *whatnow*, then *repl* uses a built–in *whatnow*, it does not actually run the *whatnow* program. Hence, if you define your own *whatnowproc*, don't call it *whatnow* since *repl* won't run it.

If your current working directory is not writable, the link named ''@'' is not available.

NAME

rmf – remove an MH folder

SYNOPSIS

rmf [+folder] [–interactive] [–nointeractive] [–help]

DESCRIPTION

Rmf removes all of the messages (files) within the specified (or default) folder, and then removes the folder (directory) itself. If there are any files within the folder which are not a part of *MH*, they will *not* be removed, and an error will be produced. If the folder is given explicitly or the '–nointeractive' option is given, then the folder will be removed without confirmation. Otherwise, the user will be asked for confirmation. If *rmf* can't find the current folder, for some reason, the folder to be removed defaults to '+inbox' (unless overridden by user's profile entry ''Inbox'') with confirmation.

Rmf irreversibly deletes messages that don't have other links, so use it with caution.

If the folder being removed is a subfolder, the parent folder will become the new current folder, and *rmf* will produce a message telling the user this has happened. This provides an easy mechanism for selecting a set of messages, operating on the list, then removing the list and returning to the current folder from which the list was extracted.

Rmf of a read–only folder will delete the private sequence and cur information (i.e., ''atr–*seq*–*folder*'' entries) from the profile without affecting the folder itself.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user's MH directory
Current–Folder:	To find the default current folder
Inbox:	To find the default inbox

See Also

rmm(1)

Defaults

'+folder' defaults to the current folder, usually with confirmation
'–interactive' if +folder' not given, '–nointeractive' otherwise

Context

Rmf will set the current folder to the parent folder if a subfolder is removed; or if the current folder is removed, it will make ''inbox'' current. Otherwise, it doesn't change the current folder or message.

Bugs

Although intuitively one would suspect that *rmf* works recursively, it does not. Hence if you have a sub–folder within a folder, in order to *rmf* the parent, you must first *rmf* each of the children.

NAME

rmm – remove messages

SYNOPSIS

rmm [+folder] [msgs] [–help]

DESCRIPTION

Rmm removes the specified messages by renaming the message files with preceding commas. Many sites consider files that start with a comma to be a temporary backup, and arrange for *cron* (8) to remove such files once a day.

If the user has a profile component such as

    rmmproc: /bin/rm

then instead of simply renaming the message file, *rmm* will call the named program to delete the file. Note that at most installations, *cron* (8) is told to remove files that begin with a comma once a night.

Some users of csh prefer the following:

    alias rmm 'refile +d'

where folder +d is a folder for deleted messages, and

    alias mexp 'rm `mhpath +d all`'

is used to ``expunge'' deleted messages.

The current message is not changed by *rmm*, so a *next* will advance to the next message in the folder as expected.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user's MH directory
Current–Folder:	To find the default current folder
rmmproc:	Program to delete the message

See Also

rmf(1)

Defaults

'+folder' defaults to the current folder
'msgs' defaults to cur

Context

If a folder is given, it will become the current folder.

Bugs

Since *refile* uses your *rmmproc* to delete the message, the *rmmproc* must **NOT** call *refile* without specifying '–normmproc', or you will create an infinte loop.

NAME

scan – produce a one line per message scan listing

SYNOPSIS

scan [+folder] [msgs] [–clear] [–noclear] [–form formatfile] [–format string] [–header] [–noheader]
[–width columns] [–reverse] [–noreverse] [–file filename] [–help]

DESCRIPTION

Scan produces a one–line–per–message listing of the specified messages. Each *scan* line contains the message number (name), the date, the ''From:'' field, the ''Subject'' field, and, if room allows, some of the body of the message. For example:

```
15+  7/ 5 Dcrocker  nned    ≪Last week I asked some of
16 - 7/ 5 dcrocker  message id format    ≪I recommend
18   7/ 6 Obrien    Re: Exit status from mkdir
19   7/ 7 Obrien    ''scan'' listing format in MH
```

The '+' on message 15 indicates that it is the current message. The '–' on message 16 indicates that it has been replied to, as indicated by a ''Replied:'' component produced by an '–annotate' switch to the *repl* command.

If there is sufficient room left on the *scan* line after the subject, the line will be filled with text from the body, preceded by <<, and terminated by >> if the body is sufficiently short. *Scan* actually reads each of the specified messages and parses them to extract the desired fields. During parsing, appropriate error messages will be produced if there are format errors in any of the messages.

The '–header' switch produces a header line prior to the *scan* listing. Currently, the name of the folder and the current date and time are output (see the **HISTORY** section for more information).

If the '–clear' switch is used and *scan's* output is directed to a terminal, then *scan* will consult the **$TERM** and **$TERMCAP** envariables to determine your terminal type in order to find out how to clear the screen prior to exiting. If the '–clear' switch is used and *scan's* output is not directed to a terminal (e.g., a pipe or a file), then *scan* will send a formfeed prior to exiting.

For example, the command:

(scan –clear –header; show all –show pr –f) | lpr

produces a scan listing of the current folder, followed by a formfeed, followed by a formatted listing of all messages in the folder, one per page. Omitting '–show pr –f' will cause the messages to be concatenated, separated by a one–line header and two blank lines.

If *scan* encounters a message without a ''Date:'' field, rather than leaving that portion of the scan listing blank, the date is filled–in with the last write date of the message, and post–fixed with a '*'. This is particularly handy for scanning a *draft folder*, as message drafts usually aren't allowed to have dates in them.

To override the output format used by *scan*, the '–format string' or '–form file' switches are used. This permits individual fields of the scan listing to be extracted with ease. The string is simply a format string and the file is simply a format file. See *mh–format* (5) for the details.

In addition to the standard *mh–format* (5) escapes, *scan* also recognizes the following additional *component* escapes:

Escape	*Returns*	*Description*
body	string	the (compressed) first part of the body
dtimenow	date	the current date
folder	string	the name of the current folder

Also, if no date header was present in the message, the *function* escapes which operate on {*date* } will return values for the date of last modification of the message file itself.

scan will update the *MH* context prior to starting the listing, so interrupting a long *scan* listing preserves the new context. *MH* purists hate this idea.

Files

$HOME/.mh_profile The user profile

Profile Components

Path:	To determine the user's MH directory
Alternate–Mailboxes:	To determine the user's mailboxes
Current–Folder:	To find the default current folder

See Also

inc(1), pick(1), show(1), mh–format(5)

Defaults

'+folder' defaults to the folder current
'msgs' defaults to all
'–format' defaulted as described above
'–noheader'
'–width' defaulted to the width of the terminal

Context

If a folder is given, it will become the current folder.

History

Prior to using the format string mechanism, '–header' used to generate a heading saying what each column in the listing was. Format strings prevent this from happening.

Bugs

The argument to the '–format' switch must be interpreted as a single token by the shell that invokes *scan*. Therefore, one must usually place the argument to this switch inside double–quotes.
The value of each *component* escape is set by *scan* to the contents of the first message header *scan* encounters with the corresponding component name; any following headers with the same component name are ignored.

The switch '–reverse', makes *scan* list the messages in reverse order; this should be considered a bug.

The '–file filename' switch allows the user to obtain a *scan* listing of a maildrop file as produced by *packf*. This listing includes every message in the file. The user should use *msh* for more selective processing of the file. '–reverse' is ignored with this option.

NAME

send – send a message

SYNOPSIS

send [–alias aliasfile] [–draft] [–draftfolder +folder] [–draftmessage msg] [–nodraftfolder] [–filter filterfile] [–nofilter] [–format] [–noformat] [–forward] [–noforward] [–mime] [–nomime] [–msgid] [–nomsgid] [–push] [–nopush] [–split seconds] [–verbose] [–noverbose] [–watch] [–nowatch] [–width columns] [file ...] [–help]

DESCRIPTION

Send will cause each of the specified files to be delivered (via *post* (8)) to each of the destinations in the ‘‘To:’’, ‘‘cc:’’, ‘‘Bcc:’’, and ‘‘Fcc:’’ fields of the message. If *send* is re–distributing a message, as invoked from *dist*, then the corresponding ‘‘Resent–xxx’’ fields are examined instead.

If ‘–push’ is specified, *send* will detach itself from the user’s terminal and perform its actions in the background. If *push* ’d and the draft can’t be sent, then the ‘–forward’ switch says that draft should be forwarded with the failure notice sent to the user. This differs from putting *send* in the background because the output is trapped and analyzed by *MH*.

If ‘–verbose’ is specified, *send* will indicate the interactions occurring with the transport system, prior to actual delivery. If ‘–watch’ is specified *send* will monitor the delivery of local and network mail. Hence, by specifying both switches, a large detail of information can be gathered about each step of the message’s entry into the transport system.

The ‘–draftfolder +folder’ and ‘–draftmessage msg’ switches invoke the *MH* draft folder facility. This is an advanced (and highly useful) feature. Consult the **Advanced Features** section of the *MH* manual for more information.

If ‘–split’ is specified, *send* will split the draft into one or more partial messages prior to sending. This makes use of the multi-media content feature in MH. Note however that if *send* is invoked under *dist* (1), then this switch is ignored -- it makes no sense to redistribute a message in this fashion. Sometimes you want *send* to pause after posting a partial message. This is usually the case when you are running *sendmail* and expect to generate a lot of partial messages. The argument to ‘–split’ tells it how long to pause between postings.

Send with no *file* argument will query whether the draft is the intended file, whereas ‘–draft’ will suppress this question. Once the transport system has successfully accepted custody of the message, the file will be renamed with a leading comma, which allows it to be retrieved until the next draft message is sent. If there are errors in the formatting of the message, *send* will abort with a (hopefully) helpful error message.

If a ‘‘Bcc:’’ field is encountered, its addresses will be used for delivery, and the ‘‘Bcc:’’ field will be removed from the message sent to sighted recipients. The blind recipients will receive an entirely new message with a minimal set of headers. Included in the body of the message will be a copy of the message sent to the sighted recipients. If ‘–filter filterfile’ is specified, then this copy is filtered (re–formatted) prior to being sent to the blind recipients. Otherwise, to use the MIME rules for encapsulation, specify the ‘-mime’ switch.

Prior to sending the message, the fields ‘‘From: user@local’’, and ‘‘Date: now’’ will be appended to the headers in the message. If the envariable **$SIGNATURE** is set, then its value is used as your personal name when constructing the ‘‘From:’’ line of the message. If this envariable is not set, then *send* will consult the profile entry ‘‘Signature’’ for this information. On hosts where *MH* was configured with the UCI option, if **$SIGNATURE** is not set and the ‘‘Signature’’ profile entry is not present, then the file **$HOME**/.signature is consulted. If ‘–msgid’ is specified, then a ‘‘Message–ID:’’ field will also be added to the message.

If *send* is re–distributing a message (when invoked by *dist*), then ‘‘Resent–’’ will be prepended to each of these fields: ‘‘From:’’, ‘‘Date:’’, and ‘‘Message–ID:’’. If the message already contains a ‘‘From:’’ field, then a ‘‘Sender: user@local’’ field will be added as well. (An already existing ‘‘Sender:’’ field is an error!)

By using the ‘–format’ switch, each of the entries in the ‘‘To:’’ and ‘‘cc:’’ fields will be replaced with ‘‘standard’’ format entries. This standard format is designed to be usable by all of the message handlers on the various systems around the Internet. If ‘–noformat’ is given, then headers are output exactly as they appear in the message draft.

If an ‘‘Fcc: folder’’ is encountered, the message will be copied to the specified folder for the sender in the format in which it will appear to any non–Bcc receivers of the message. That is, it will have the appended fields and field reformatting. The ‘‘Fcc:’’ fields will be removed from all outgoing copies of the message.

By using the ‘–width columns’ switch, the user can direct *send* as to how long it should make header lines containing addresses.

The files specified by the profile entry ‘‘Aliasfile:’’ and any additional alias files given by the ‘–alias aliasfile’ switch will be read (more than one file, each preceeded by ‘–alias’, can be named). See *mh–alias* (5) for more information.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user’s MH directory
Draft–Folder:	To find the default draft–folder
Aliasfile:	For a default alias file
Signature:	To determine the user’s mail signature
mailproc:	Program to post failure notices
postproc:	Program to post the message

See Also

comp(1), dist(1), forw(1), repl(1), mh–alias(5), post(8)

Defaults

‘file’ defaults to <mh–dir>/draft
‘–alias /usr/bs/mh-beta/lib/MailAliases’
‘–nodraftfolder’
‘–nofilter’
‘–format’
‘–forward’
‘–nomime’
‘–nomsgid’
‘–nopush’
‘–noverbose’
‘–nowatch’
‘–width 72’

Context

None

Bugs

Under some configurations, it is not possible to mointor the mail delivery transaction; '–watch' is a no-op on those systems.

Using '–split 0' doesn't work correctly.

NAME

show – show (list) messages

SYNOPSIS

show [+folder] [msgs] [–draft] [–header] [–noheader] [–showproc program] [–noshowproc]
[switches for *showproc*] [–help]

DESCRIPTION

Show lists each of the specified messages to the standard output (typically, the terminal). Typically, the messages are listed exactly as they are, with no reformatting. A program named by the *showproc* profile component is invoked to do the listing, and any switches not recognized by *show* are passed along to that program. The default program is known as *more* (1). To override the default and the *showproc* profile component, use the '–showproc program' switch. For example, '–show pr' will cause the *pr* (1) program to list the messages. The *MH* command *mhl* can be used as a *showproc* to show messages in a more uniform format. Normally, this program is specified as the *showproc* is the user's .mh_profile. See *mhl* (1) for the details. If the '–noshowproc' option is specified, '/bin/cat' is used instead of *showproc*.

If you have messages with multi-media contents, the profile entry *mhnproc* defines the name of a program to manipulate multi-media messages. (The *mhn* (1) program, which is suitable for this purpose, is the default.) If the '–noshowproc' option is NOT specified, and if one or more named messages has a multimedia content, then the program indicated by *mhnproc* will be run instead of *showproc*. The use of the *mhnproc* can also be disabled if the environment variable **$NOMHNPROC** is set. Note that the *mhnproc* may be invoked even for textual contents, depending on the character set involved. The environment variable $MM_CHARSET should be set to the terminal's character set to avoid gratuitous invocations of the *mhnproc*.

The '–header' switch tells *show* to display a one–line description of the message being shown. This description includes the folder and the message number.

If no 'msgs' are specified, the current message is used. If more than one message is specified, *more* will prompt for a <RETURN> prior to listing each message. *more* will list each message, a page at a time. When the end of page is reached, *more* will ring the bell and wait for a <SPACE> or <RETURN>. If a <RETURN> is entered, *more* will print the next line, whereas <SPACE> will print the next screenful. To exit *more*, type ''q''.

If the standard output is not a terminal, no queries are made, and each file is listed with a one–line header and two lines of separation.

''show –draft'' will list the file <mh–dir>/draft if it exists.

If the profile entry ''Unseen–Sequence'' is present and non–empty, then *show* will remove each of the messages shown from each sequence named by the profile entry. This is similar to the ''Previous–Sequence'' profile entry supported by all *MH* commands which take 'msgs' or 'msg' arguments.

Files

$HOME/.mh_profile The user profile

Profile Components

Path:	To determine the user’s MH directory
Current–Folder:	To find the default current folder
Unseen–Sequence:	To name sequences denoting unseen messages
showproc:	Program to show messages
mhnproc:	Program to show messages with multi-media content

See Also

mhl(1), more(1), next(1), pick(1), prev(1), scan(1)

Defaults

‘+folder’ defaults to the current folder
‘msgs’ defaults to cur
‘–header’

Context

If a folder is given, it will become the current folder. The last message shown will become the current message.

Bugs

The ‘–header’ switch doesn’t work when ‘msgs’ expands to more than one message. If the *showproc* is *mhl*, then is problem can be circumvented by referencing the ‘‘messagename’’ field in the *mhl* format file.

Show updates the user’s context before showing the message. Hence *show* will mark messages as seen prior to the user actually seeing them. This is generally not a problem, unless the user relies on the ‘‘unseen’’ messages mechanism, and interrupts *show* while it is showing ‘‘unseen’’ messages.

If *showproc* is *mhl*, then *show* uses a built–in *mhl*: it does not actually run the *mhl* program. Hence, if you define your own *showproc*, don’t call it *mhl* since *show* won’t run it.

If *more* (1) is your showproc (the default), then avoid running *show* in the background with only its standard output piped to another process, as in

```
show | imprint &
```

Due to a bug in *more*, show will go into a ‘‘tty input’’ state. To avoid this problem, re–direct *show*’s diagnostic output as well. For users of *csh*:

```
show |& imprint &
```

For users of *sh*:

```
show 2>&1 | imprint &
```

NAME

slocal – special local mail delivery

SYNOPSIS

/usr/bs/mh-beta/lib/slocal [address info sender]
[–addr address] [–info data] [–sender sender]
[–user username] [–mailbox mbox] [–file file]
[–maildelivery deliveryfile] [–verbose] [–noverbose] [–debug] [–help]

DESCRIPTION

Slocal is a program designed to allow you to have your inbound mail processed according to a complex set of selection criteria. You do not normally invoke *slocal* yourself, rather *slocal* is invoked on your behalf by your system's Message Transfer Agent.

The message selection criteria used by *slocal* is specified in the file *.maildelivery* in the user's home directory. The format of this file is given below.

The message delivery address and message sender are determined from the Message Transfer Agent envelope information, if possible. Under *SendMail*, the sender will obtained from the UUCP ‘‘From ’’ line, if present. The user may override these values with command line arguments, or arguments to the ‘–addr’ and ‘–sender’ switches.

The message is normally read from the standard input. The ‘–file’ switch sets the name of the file from which the message should be read, instead of reading stdin. The ‘–user’ switch tells *slocal* the name of the user for whom it is delivering mail. The ‘–mailbox’ switch tells *slocal* the name of the user's maildrop file.

The ‘–info’ switch may be used to pass an arbitrary argument to sub-processes which *slocal* may invoke on your behalf. The ‘–verbose’ switch causes *slocal* to give information on stdout about its progress. The ‘–debug’ switch produces more verbose debugging output on stderr.

Message Transfer Agents

If your MTA is *SendMail*, you should include the line

‘‘| /usr/bs/mh-beta/lib/slocal –user username’’

in your .forward file in your home directory. This will cause *SendMail* to invoke *slocal* on your behalf.

If your MTA is *MMDF-I*, you should (symbolically) link /usr/bs/mh-beta/lib/slocal to the file bin/rcvmail in your home directory. This will cause *MMDF-I* to invoke *slocal* on your behalf with the correct ‘‘*address info sender*’’ arguments.

If your MTA is *MMDF-II*, then you should not use *slocal*. An equivalent functionality is already provided by *MMDF-II*; see maildelivery(5) for details.

The Maildelivery File

The *.maildelivery* file controls how local delivery is performed. Each line of this file consists of five fields, separated by white-space or comma. Since double-quotes are honored, these characters may be included in a single argument by enclosing the entire argument in double-quotes. A double-quote can be included by preceding it with a backslash. Lines beginning with '#' are ignored. The format of each line in the *.mail-delivery* file is:

header pattern action result string

header:

The name of a header field that is to be searched for a pattern. This is any field in the headers of the message that might be present. The following special fields are also defined:

source	the out-of-band sender information
addr	the address that was used to cause delivery to the recipient
default	this matches *only* if the message hasn't been delivered yet
*	this always matches

pattern:

The sequence of characters to match in the specified header field. Matching is case-insensitive, but does not use regular expressions.

action:

The action to take to deliver the message:

destroy This action always succeeds.

file or > Append the message to the file named by **string**. The message is appended to the file in the maildrop format which is used by your message transport system. If the message can be appended to the file, then this action succeeds. When writing to the file, a ''Delivery–Date: date'' header is added which indicates the date and time that message was appended to the file.

mbox Identical to *file*, but always appends the message using the format used by *packf* (the MMDF mailbox format).

pipe or | Pipe the message as the standard input to the command named by **string**, using the Bourne shell *sh*(1) to interpret the string. Prior to giving the string to the shell, it is expanded with the following built-in variables:

$(sender)	the out-of-band sender information
$(address)	the address that was used to cause delivery to the recipient
$(size)	the size of the message in bytes
$(reply–to)	either the ''Reply–To:'' or ''From:'' field of the message
$(info)	the out-of-band information specified

qpipe or *<caret>* Similar to *pipe*, but executes the command directly, after built-in variable expansion, without assistance from the shell. This action can be used to avoid quoting special characters which your shell might interpret.

result:

Indicates how the action should be performed:

A Perform the action. If the action succeeds, then the message is considered delivered.

R Perform the action. Regardless of the outcome of the action, the message is not considered delivered.

? Perform the action only if the message has not been delivered. If the action succeeds, then the message is considered delivered.

N Perform the action only if the message has not been delivered and the previous action succeeded. If this action succeeds, then the message is considered delivered.

To summarize, here’s an example:

```
#field   pattern      action  result  string
# lines starting with a '#' are ignored, as are blank lines
#
# file mail with mmdf2 in the ``To:'' line into file mmdf2.log
To       mmdf2        file    A       mmdf2.log
# Messages from mmdf pipe to the program err-message-archive
From     mmdf         pipe    A       /bin/err-message-archive
# Anything with the ``Sender:'' address ``mh-workers''
# file in mh.log if not filed already
Sender   mh-workers   file    ?       mh.log
# ``To:'' unix – put in file unix-news
To       Unix         >       A       unix-news
# if the address is jpo=ack – send an acknowledgement copy back
addr     jpo=ack      |       R       ``/bin/resend –r $(reply-to)''
# anything from steve – destroy!
From     steve        destroy A       –
# anything not matched yet – put into mailbox
default  –            >       ?       mailbox
# always run rcvtty
*        –            |       R       /mh/lib/rcvtty
```

The file is always read completely, so that several matches can be made and several actions can be taken. The *.maildelivery* file must be owned either by the user or by root, and must be writable only by the owner. If the *.maildelivery* file cannot be found, or does not perform an action which delivers the message, then the file /usr/bs/mh-beta/lib/maildelivery is read according to the same rules. This file must be owned by the root and must be writable only by the root. If this file cannot be found or does not perform an action which delivers the message, then standard delivery to the user’s maildrop is performed.

Sub-process environment

When a process is invoked, its environment is: the user/group-ids are set to recipient’s ids; the working directory is the recipient’s home directory; the umask is 0077; the process has no /dev/tty; the standard input is set to the message; the standard output and diagnostic output are set to /dev/null; all other file-descriptors are closed; the envariables **$USER**, **$HOME**, **$SHELL** are set appropriately, and no other envariables exist.

The process is given a certain amount of time to execute. If the process does not exit within this limit, the process will be terminated with extreme prejudice. The amount of time is calculated as ((size x 60) + 300) seconds, where size is the number of bytes in the message.

The exit status of the process is consulted in determining the success of the action. An exit status of zero means that the action succeeded. Any other exit status (or abnormal termination) means that the action failed.

In order to avoid any time limitations, you might implement a process that began by *forking*. The parent would return the appropriate value immediately, and the child could continue on, doing whatever it wanted for as long as it wanted. This approach is somewhat risky if the parent is going to return an exit status of zero. If the parent is going to return a non-zero exit status, then this approach can lead to quicker delivery into your maildrop.

Files

/usr/bs/mh-beta/lib/mtstailor	MH tailor file
$HOME/.maildelivery	The file controlling local delivery
/usr/bs/mh-beta/lib/maildelivery	Rather than the standard file
/usr/spool/mail/$USER	The default maildrop

See Also

rcvstore(1), mhook(1), mh–format(5) , maildelivery(5)

Defaults

'–noverbose'
'–maildelivery .maildelivery'
'–mailbox /usr/spool/mail/$USER'
'–file' defaults to stdin
'–user' defaults to the current user

Context

None

History

Slocal is designed to be backward-compatible with the *maildelivery* facility provided by *MMDF-II*. Thus, the *.maildelivery* file syntax is limited, as is the functionality of *slocal*.

In addition to an exit status of zero, the *MMDF* values *RP_MOK* (32) and *RP_OK* (9) mean that the message has been fully delivered. Any other non-zero exit status, including abnormal termination, is interpreted as the *MMDF* value *RP_MECH* (200), which means ''use an alternate route'' (deliver the message to the maildrop).

Bugs

Only two return codes are meaningful, others should be.

Slocal is designed to be backwards-compatible with the *maildelivery* functionality provided by **MMDF-II**.

Versions of *MMDF* with the *maildelivery* mechanism aren't entirely backwards-compatible with earlier versions of *MMDF*. If you have an *MMDF-I* old-style hook, the best you can do is to have a one-line *.maildelivery* file:

```
default – pipe A ''bin/rcvmail $(address) $(info) $(sender)''
```

NAME

sortm – sort messages

SYNOPSIS

sortm [+folder] [msgs] [–datefield field] [–textfield field] [–notextfield] [–limit days] [–nolimit] [–verbose] [–noverbose] [–help]

DESCRIPTION

Sortm sorts the specified messages in the named folder according to the chronological order of the ‘‘Date:’’ field of each message.

The ‘–verbose’ switch directs *sortm* to tell the user the general actions that it is taking to place the folder in sorted order.

The ‘–datefield field’ switch tells *sortm* the name of the field to use when making the date comparison. If the user has a special field in each message, such as ‘‘BB–Posted:’’ or ‘‘Delivery–Date:’’, then the ‘–datefield’ switch can be used to direct *sortm* which field to examine.

The ‘–textfield field’ switch causes *sortm* to sort messages by the specified text field. If this field is ‘‘subject’’, any leading "re:" is stripped off. In any case, all characters except letters and numbers are stripped and the resulting strings are sorted datefield–major, textfield–minor, using a case insensitive comparison.

With ‘–textfield field’, if ‘–limit days’ is specified, messages with similar textfields that are dated within ‘days’ of each other appear together. Specifying ‘–nolimit’ makes the limit infinity. With ‘–limit 0’, the sort is instead made textfield–major, date–minor.

For example, to order a folder by date-major, subject-minor, use:

sortm -textfield subject +folder

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user’s MH directory
Current–Folder:	To find the default current folder

See Also

folder (1)

Defaults

‘+folder’ defaults to the current folder
‘msgs’ defaults to all
‘–datefield date’
‘–notextfield’
‘–noverbose’
‘–nolimit’

Context

If a folder is given, it will become the current folder. If the current message is moved, *sortm* will preserve its status as current.

History

Timezones used to be ignored when comparing dates: they aren’t any more.

Messages which were in the folder, but not specified by ‘msgs’, used to be moved to the end of the folder; now such messages are left untouched.

Sortm sometimes did not preserve the message numbering in a folder (e.g., messages 1, 3, and 5, might have been renumbered to 1, 2, 3 after sorting). This was a bug, and has been fixed. To compress the message numbering in a folder, use ‘‘*folder –pack*’’ as always.

Bugs

If *sortm* encounters a message without a date–field, or if the message has a date–field that *sortm* cannot parse, then *sortm* attempts to keep the message in the same relative position. This does not always work. For instance, if the first message encountered lacks a date which can be parsed, then it will usually be placed at the end of the messages being sorted.

When *sortm* complains about a message which it can’t temporally order, it complains about the message number *prior* to sorting. It should indicate what the message number will be *after* sorting.

NAME

vmh – visual front-end to MH

SYNOPSIS

vmh [–prompt string] [–vmhproc program] [–novmhproc] [switches for *vmhproc*] [–help]

DESCRIPTION

vmh is a program which implements the server side of the *MH* window management protocol and uses *curses* (3) routines to maintain a split–screen interface to any program which implements the client side of the protocol. This latter program, called the *vmhproc*, is specified using the '–vmhproc program' switch.

The upshot of all this is that one can run *msh* on a display terminal and get a nice visual interface. To do this, for example, just add the line

    mshproc: vmh

to your .mh_profile. (This takes advantage of the fact that *msh* is the default *vmhproc* for *vmh*.)

In order to facilitate things, if the '–novmhproc' switch is given, and *vmh* can't run on the user's terminal, the *vmhproc* is run directly without the window management protocol.

After initializing the protocol, *vmh* prompts the user for a command to be given to the client. Usually, this results in output being sent to one or more windows. If a output to a window would cause it to scroll, *vmh* prompts the user for instructions, roughly permitting the capabilities of *less* or *more* (e.g., the ability to scroll backwards and forwards):

SPACE		advance to the next windowful
RETURN	*	advance to the next line
y	*	retreat to the previous line
d	*	advance to the next ten lines
u	*	retreat to the previous ten lines
g	*	go to an arbitrary line (preceed g with the line number)
G	*	go to the end of the window (if a line number is given, this acts like 'g')
CTRL–L		refresh the entire screen
h		print a help message
q		abort the window

(A '*' indicates that a numeric prefix is meaningful for this command.)

Note that if a command resulted in more than one window's worth of information being displayed, and you allow the command which is generating information for the window to gracefully finish (i.e., you don't use the 'q' command to abort information being sent to the window), then *vmh* will give you one last change to peruse the window. This is useful for scrolling back and forth. Just type 'q' when you're done.

To abnormally terminate *vmh* (without core dump), use <QUIT> (usually CTRL–\). For instance, this does the ''right'' thing with *bbc* and *msh*.

Files

$HOME/.mh_profile   The user profile

Profile Components

Path: To determine the user's MH directory

See Also

msh(1)

Defaults

'–prompt (vmh) '
'–vmhproc msh'

Context

None

Bugs

The argument to the '–prompt' switch must be interpreted as a single token by the shell that invokes *vmh*. Therefore, one must usually place the argument to this switch inside double–quotes.

At present, there is no way to pass signals (e.g., interrupt, quit) to the client. However, generating QUIT when *vmh* is reading a command from the terminal is sufficient to tell the client to go away quickly.

Acts strangely (loses peer or botches window management protocol with peer) on random occasions.

NAME

whatnow – prompting front-end for send

SYNOPSIS

whatnow [–draftfolder +folder] [–draftmessage msg] [–nodraftfolder] [–editor editor] [–noedit]
[–prompt string] [file] [–help]

DESCRIPTION

Whatnow is the default program that queries the user about the disposition of a composed draft. It is normally invoked by one of *comp*, *dist*, *forw*, or *repl* after the initial edit.

When started, the editor is started on the draft (unless '–noedit' is given, in which case the initial edit is suppressed). Then, *whatnow* repetitively prompts the user with ''What now?'' and awaits a response. The valid responses are:

display	to list the message being distributed/replied–to on the terminal
edit	to re–edit using the same editor that was used on the preceding round unless a profile entry ''<lasteditor>–next: <editor>'' names an alternate editor
edit <editor>	to invoke <editor> for further editing
list	to list the draft on the terminal
push	to send the message in the background
quit	to terminate the session and preserve the draft
quit –delete	to terminate, then delete the draft
refile +folder	to refile the draft into the given folder
send	to send the message
send –watch	to cause the delivery process to be monitored
whom	to list the addresses that the message will go to
whom –check	to list the addresses and verify that they are acceptable to the transport service

For the **edit** response, any valid switch to the editor is valid. Similarly, for the **send** and **whom** responses, any valid switch to *send* (1) and *whom* (1) commands, respectively, are valid. For the **push** response, any valid switch to *send* (1) is valid (as this merely invokes *send* with the '–push' option). For the *refile* response, any valid switch to the *fileproc* is valid. For the **display** and **list** responses, any valid argument to the *lproc* is valid. If any non–switch arguments are present, then the pathname of the draft will be excluded from the argument list given to the *lproc* (this is useful for listing another *MH* message).

See *mh–profile* (5) for further information about how editors are used by MH. It also discusses how complex envariables can be used to direct *whatnow*'s actions.

The '–prompt string' switch sets the prompting string for *whatnow*.

The '–draftfolder +folder' and '–draftmessage msg' switches invoke the *MH* draft folder facility. This is an advanced (and highly useful) feature. Consult the **Advanced Features** section of the *MH* manual for more information.

Files

$HOME/.mh_profile	The user profile
<mh–dir>/draft	The draft file

Profile Components

Path:	To determine the user's MH directory
Draft–Folder:	To find the default draft–folder
Editor:	To override the default editor
<lasteditor>–next:	To name an editor to be used after exit from <lasteditor>
automhnproc:	Program to automatically run prior to sending if the draft is an *mhn* composition file
fileproc:	Program to refile the message
lproc:	Program to list the contents of a message
sendproc:	Program to use to send the message
whomproc:	Program to determine who a message would go to

See Also

send(1), whom(1)

Defaults

'–prompt ''What Now? '''

Context

None

Bugs

The argument to the '–prompt' switch must be interpreted as a single token by the shell that invokes *whatnow*. Therefore, one must usually place the argument to this switch inside double–quotes.

If the initial edit fails, *whatnow* deletes your draft (by renaming it with a leading comma); failure of a later edit preverves the draft.

If *whatnowproc* is *whatnow*, then *comp*, *dist*, *forw*, and *repl* use a built–in *whatnow*, and do not actually run the *whatnow* program. Hence, if you define your own *whatnowproc*, don't call it *whatnow* since it won't be run.

If *sendproc* is *send*, then *whatnow* uses a built–in *send*, it does not actually run the *send* program. Hence, if you define your own *sendproc*, don't call it *send* since *whatnow* won't run it.

NAME

whom – report to whom a message would go

SYNOPSIS

whom [–alias aliasfile] [–check] [–nocheck] [–draft] [–draftfolder +folder] [–draftmessage msg]
[–nodraftfolder] [file] [–help]

DESCRIPTION

Whom is used to expand the headers of a message into a set of addresses and optionally verify that those addresses are deliverable at that time (if '–check' is given).

The '–draftfolder +folder' and '–draftmessage msg' switches invoke the *MH* draft folder facility. This is an advanced (and highly useful) feature. Consult the **Advanced Features** section of the *MH* manual for more information.

The files specified by the profile entry ''Aliasfile:'' and any additional alias files given by the '–alias aliasfile' switch will be read (more than one file, each preceeded by '–alias', can be named). See *mh–alias* (5) for more information.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user's MH directory
Draft–Folder:	To find the default draft–folder
Aliasfile:	For a default alias file
postproc:	Program to post the message

See Also

mh–alias(5), post(8)

Defaults

'file' defaults to <mh–dir>/draft
'–nocheck'
'–alias /usr/bs/mh-beta/lib/MailAliases'

Context

None

Bugs

With the '–check' option, *whom* makes no guarantees that the addresses listed as being ok are really deliverable, rather, an address being listed as ok means that at the time that *whom* was run the address was thought to be deliverable by the transport service. For local addresses, this is absolute; for network addresses, it means that the host is known; for uucp addresses, it (often) means that the *UUCP* network is available for use.

MORE DETAILS

This section describes some of the more intense points of the *MH* system, by expanding on topics previously discussed. The format presented conforms to the standard form for the description of UNIX documentation.

NAME

mh-alias – alias file for MH message system

SYNOPSIS

any *MH* command

DESCRIPTION

This describes both *MH* personal alias files and the (primary) alias file for mail delivery, the file

/usr/bs/mh-beta/lib/MailAliases

It does **not** describe aliases files used by the message transport system. Each line of the alias file has the format:

alias : address–group

or

alias ; address–group

or

< alias–file

or

; comment

where:

```
address–group  :=  address–list
                |  ‘‘<’’ file
                |  ‘‘=’’ UNIX–group
                |  ‘‘+’’ UNIX–group
                |  ‘‘*’’

address–list   :=  address
                |  address–list, address
```

Continuation lines in alias files end with ‘\’ followed by the newline character.

Alias–file and file are UNIX file names. UNIX–group is a group name (or number) from */etc/group*. An address is a ‘‘simple’’ Internet–style address. Througout this file, case is ignored, except for alias–file names.

If the line starts with a ‘<’, then the file named after the ‘<’ is read for more alias definitions. The reading is done recursively, so a ‘<’ may occur in the beginning of an alias file with the expected results.

If the address–group starts with a ‘<’, then the file named after the ‘<’ is read and its contents are added to the address–list for the alias.

If the address–group starts with an ‘=’, then the file */etc/group* is consulted for the UNIX–group named after the ‘=’. Each login name occurring as a member of the group is added to the address–list for the alias.

In contrast, if the address–group starts with a ‘+’, then the file */etc/group* is consulted to determine the group–id of the UNIX–group named after the ‘+’. Each login name occurring in the */etc/passwd* file whose group–id is indicated by this group is added to the address–list for the alias.

If the address–group is simply ‘*’, then the file */etc/passwd* is consulted and all login names with a userid greater than some magic number (usually 200) are added to the address–list for the alias.

In match, a trailing * on an alias will match just about anything appropriate. (See example below.)

An approximation of the way aliases are resolved at posting time is (it’s not really done this way):

1) Build a list of all addresses from the message to be delivered, eliminating duplicate addresses.

2) If this draft originated on the local host, then for those addresses in the message that have no host specified, perform alias resolution.

3) For each line in the alias file, compare ‘‘alias’’ against all of the existing addresses. If a match, remove the matched ‘‘alias’’ from the address list, and add each new address in the address–group to the address list if it is not already on the list. The alias itself is not usually output, rather the address–group that the alias maps to is output instead. If ‘‘alias’’ is terminated with a ‘;’ instead of a ‘:’, then both the ‘‘alias’’ and the address are output in the correct format. (This makes replies possible since *MH* aliases and personal aliases are unknown to the mail transport system.)

Since the alias file is read line by line, forward references work, but backward references are not recognized, thus, there is no recursion.

Example:

</usr/bs/mh-beta/lib/BBoardAliases
sgroup: fred, fear, freida
b-people: Blind List: bill, betty;
fred: frated@UCI
UNIX–committee: <unix.aliases
staff: =staff
wheels: +wheel
everyone: *
news.*: news

The first line says that more aliases should immediately be read from the file */usr/bs/mh-beta/lib/BBoardAliases*. Following this, ‘‘fred’’ is defined as an alias for ‘‘frated@UCI’’, and ‘‘sgroup’’ is defined as an alias for the three names ‘‘frated@UCI’’, ’’fear’’, and ’’freida’’.

The alias ‘‘b-people’’ is a blind list which includes the addresses ‘‘bill’’ and ‘‘betty’’; the message will be delieved to those addresses, but the message header will show only ‘‘Blind List: ;’’ (not the addresses).

Next, the definition of ‘‘UNIX–committee’’ is given by reading the file *unix.aliases* in the users *MH* directory, ‘‘staff’’ is defined as all users who are listed as members of the group ‘‘staff’’ in the */etc/group* file, and ‘‘wheels’’ is defined as all users whose group–id in */etc/passwd* is equivalent to the ‘‘wheel’’ group.

Finally, ‘‘everyone’’ is defined as all users with a user–id in */etc/passwd* greater than 200, and all aliases of the form ‘‘news.<anything>’’ are defined to be ‘‘news’’.

The key thing to understand about aliasing in *MH* is that aliases in *MH* alias files are expanded into the headers of messages posted. This aliasing occurs first, at posting time, without the knowledge of the message transport system. In contrast, once the message transport system is given a message to deliver to a list of addresses, for each address that appears to be local, a system–wide alias file is consulted. These aliases are **NOT** expanded into the headers of messages delivered.

Helpful Hints

To use aliasing in *MH* quickly, do the following:

First, in your *.mh_profile*, choose a name for your alias file, say ‘‘aliases’’, and add the line:

Aliasfile: aliases

Second, create the file ‘‘aliases’’ in your *MH* directory.

Third, start adding aliases to your ‘‘aliases’’ file as appropriate.

Files

/usr/bs/mh-beta/lib/MailAliases	Primary alias file

Profile Components

Aliasfile:	For a default alias file

See Also

ali(1), send(1), whom(1), group(5), passwd(5), conflict(8), post(8)

Defaults

None

Context

None

History

In previous releases of *MH*, only a single, system–wide mh–alias file was supported. Now that *MH* uses *MMDF* as a transport system, the system–wide aliasing facility can be more consistently controlled by the latter. This means that at most sites, the system–wide mh–alias file will be empty (or trivial at best). Hence, the semantics of mh–alias were extended to support personal alias files. Users of *MH* no longer need to bother mail–system administrators for keeping information in the system–wide alias file, as each *MH* user can create/modify/remove aliases at will from any number of personal files.

Bugs

Although the forward-referencing semantics of *mh–alias* files prevent recursion, the ‘‘< alias–file’’ command may defeat this. Since the number of file descriptors is finite (and very limited), such infinite recursion will terminate with a meaningless diagnostic when all the fds are used up.

Forward references do not work correctly inside blind lists.

NAME

mh-format – format file for MH message system

SYNOPSIS

some *MH* commands

DESCRIPTION

Several *MH* commands utilize either a *format* string or a *format* file during their execution. For example, *scan* (1) uses a format string which directs it how to generate the scan listing for each message; *repl* (1) uses a format file which directs it how to generate the reply to a message, and so on.

Format strings are designed to be efficiently parsed by *MH* which means they are not necessarily simple to write and understand. This means that novice, casual, or even advanced users of *MH* should not have to deal with them. Some canned scan listing formats are in /usr/bs/mh-beta/lib/scan.time, /usr/bs/mh-beta/lib/scan.size, and /usr/bs/mh-beta/lib/scan.timely. Look in /usr/bs/mh-beta/lib for other *scan* and *repl* format files which may have been written at your site.

It suffices to have your local *MH* expert actually write new format commands or modify existing ones. This manual section explains how to do that. Note: familiarity with the C *printf* routine is assumed.

A format string consists of ordinary text, and special multi-character *escape* sequences which begin with ‘%’. When specifying a format string, the usual C backslash characters are honored: ‘\b’, ‘\f’, ‘\n’, ‘\r’, and ‘\t’. Continuation lines in format files end with ‘\’ followed by the newline character. There are three types of *escape* sequences: header *components*, built-in *functions*, and flow *control*.

A *component* escape is specified as ‘%{*component*}’, and exists for each header found in the message being processed. For example ‘%{date}’ refers to the ‘‘Date:’’ field of the appropriate message. All component escapes have a string value. Normally, component values are compressed by converting any control characters (tab and newline included) to spaces, then eliding any leading or multiple spaces. However, commands may give different interpretations to some component escapes; be sure to refer to each command’s manual entry for complete details.

A *function* escape is specified as ‘%(*function*)’. All functions are built-in, and most have a string or numeric value.

Control-flow escapes

A *control* escape is one of: ‘%<’, ‘%?’, ‘%|’, or ‘%>’. These are combined into the conditional execution construct:

```
%<condition
        format text 1
%?condition2
        format text 2
%?condition3
        format text 3
...
%|
        format text N
%>
```

Extra white space is shown here only for clarity. These constructs may be nested without ambiguity. They form a general **if–elseif–else–endif** block where only one of the *format text* segments is interpreted.

The ‘%<’ and ‘%?’ control escapes causes a condition to be evaluated. This condition may be either a *component* or a *function*. The four constructs have the following syntax:

```
%<{component}
%<(function)
%?{component}
%?(function)
```

These control escapes test whether the function or component value is non-zero (for integer-valued escapes), or non-empty (for string-valued escapes).

If this test evaulates true, then the format text up to the next corresponding control escape (one of ‘%|’, ‘%?’, or ‘%>’) is interpreted normally. Next, all format text (if any) up to the corresponding ‘%>’ control escape is skipped. The ‘%>’ control escape is not interpreted; normal interpretation resumes after the ‘%>’ escape.

If the test evaluates false, however, then the format text up to the next corresponding control escape (again, one of ‘%|’, ‘%?’, or ‘%>’) is skipped, instead of being interpreted. If the control escape encountered was ‘%?’, then the condition associated with that control escape is evaluated, and interpretation proceeds after that test as described in the previous paragraph. If the control escape encountered was ‘%|’, then the format text up to the corresponding ‘%>’ escape is interpreted normally. As above, the ‘%>’ escape is not interpreted and normal interpretation resumes after the ‘%>’ escape.

The ‘%?’ control escape and its following format text is optional, and may be included zero or more times. The ‘%|’ control escape and its following format text is also optional, and may be included zero or one times.

Function escapes

Most functions expect an argument of a particular type:

Argument	*Description*	*Example Syntax*
literal	A literal number,	%(*func* 1234)
	or string	%(*func* text string)
comp	Any header component	%(*func*{*in-reply-to*})
date	A date component	%(*func*{*date*})
addr	An address component	%(*func*{*from*})
expr	An optional component,	%(*func*(*func2*))
	function or control,	%(*func* %<{*reply-to*}%|%{*from*}%>)
	perhaps nested	%(*func*(*func2*{*comp*}))

The types *date* and *addr* have the same syntax as *comp*, but require that the header component be a date string, or address string, respectively.

All arguments except those of type *expr* are required. For the *expr* argument type, the leading ‘%’ must be omitted for component and function escape arguments, and must be present (with a leading space) for control escape arguments.

The evaluation of format strings is based on a simple machine with an integer register *num*, and a text string register *str*. When a function escape is processed, if it accepts an optional *expr* argument which is not present, it reads the current value of either *num* or *str* as appropriate.

Return values

Component escapes write the value of their message header in *str*. Function escapes write their return value in *num* for functions returning *integer* or *boolean* values, and in *str* for functions returning string values. (The *boolean* type is a subset of integers with usual values 0=false and 1=true.) Control escapes return a *boolean* value, and set *num*.

All component escapes, and those function escapes which return an *integer* or *string* value, pass this value back to their caller in addition to setting *str* or *num*. These escapes will print out this value unless called as part of an argument to another escape sequence. Escapes which return a *boolean* value do pass this value back to their caller in *num*, but will never print out the value.

Function	*Argument*	*Return*	*Description*
msg		integer	message number
cur		integer	message is current
size		integer	size of message
strlen		integer	length of *str*
width		integer	output buffer size in bytes
charleft		integer	bytes left in output buffer
timenow		integer	seconds since the UNIX epoch
me		string	the user's mailbox
eq	literal	boolean	*num* == *arg*
ne	literal	boolean	*num* != *arg*
gt	literal	boolean	*num* > *arg*
match	literal	boolean	*str* contains *arg*
amatch	literal	boolean	*str* starts with *arg*
plus	literal	integer	*arg* plus *num*
minus	literal	integer	*arg* minus *num*
divide	literal	integer	*num* divided by *arg*
modulo	literal	integer	*num* modulo *arg*
num	literal	integer	Set *num* to *arg*
lit	literal	string	Set *str* to *arg*
getenv	literal	string	Set *str* to environment value of *arg*
profile	literal	string	Set *str* to profile component *arg* value
nonzero	expr	boolean	*num* is non-zero
zero	expr	boolean	*num* is zero
null	expr	boolean	*str* is empty
nonnull	expr	boolean	*str* is non-empty
void	expr		Set *str* or *num*
comp	comp	string	Set *str* to component text
compval	comp	integer	*num* set to ‘‘**atoi**(*comp*)’’
trim	expr		trim trailing white-space from *str*
putstr	expr		print *str*
putstrf	expr		print *str* in a fixed width
putnum	expr		print *num*
putnumf	expr		print *num* in a fixed width

These functions require a date component as an argument:

Function	*Argument*	*Return*	*Description*
sec	date	integer	seconds of the minute
min	date	integer	minutes of the hour
hour	date	integer	hours of the day (0-23)
wday	date	integer	day of the week (Sun=0)
day	date	string	day of the week (abbrev.)
weekday	date	string	day of the week
sday	date	integer	day of the week known? (0=implicit,–1=unknown)
mday	date	integer	day of the month
yday	date	integer	day of the year
mon	date	integer	month of the year
month	date	string	month of the year (abbrev.)
lmonth	date	string	month of the year
year	date	integer	year (may be > 100)
zone	date	integer	timezone in hours
tzone	date	string	timezone string
szone	date	integer	timezone explicit? (0=implicit,–1=unknown)
date2local	date		coerce date to local timezone
date2gmt	date		coerce date to GMT
dst	date	integer	daylight savings in effect?
clock	date	integer	seconds since the UNIX epoch
rclock	date	integer	seconds prior to current time
tws	date	string	official 822 rendering
pretty	date	string	user-friendly rendering
nodate	date	integer	*str* not a date string

These functions require an address component as an argument. The return value of functions noted with '*' pertain only to the first address present in the header component.

Function	*Argument*	*Return*	*Description*
proper	addr	string	official 822 rendering
friendly	addr	string	user-friendly rendering
addr	addr	string	mbox@host or host!mbox rendering*
pers	addr	string	the personal name*
note	addr	string	commentary text*
mbox	addr	string	the local mailbox*
mymbox	addr	integer	the user's addresses? (0=no,1=yes)
host	addr	string	the host domain*
nohost	addr	integer	no host was present*
type	addr	integer	host type* (0=local,1=network, –1=uucp,2=unknown)
path	addr	string	any leading host route*
ingrp	addr	integer	address was inside a group*
gname	addr	string	name of group*
formataddr	expr		append *arg* to *str* as a (comma separated) address list
putaddr	literal		print *str* address list with *arg* as optional label;

get line width from *num*

When escapes are nested, evaluation is done from inner-most to outer-most. The outer-most escape must begin with ‘%’; the inner escapes must not. For example,

%<(mymbox{from}) To: %{to}%>

writes the value of the header component ‘‘From:’’ to *str*; then (*mymbox*) reads *str* and writes its result to *num*; then the control escape evaluates *num*. If *num* is non-zero, the string ‘‘To: ’’ is printed followed by the value of the header component ‘‘To:’’.

A minor explanation of (*mymbox*{*comp*}) is in order. In general, it checks each of the addresses in the header component ‘‘*comp*’’ against the user’s mailbox name and any *Alternate-Mailboxes*. It returns true if any address matches, however, it also returns true if the ‘‘*comp*’’ header is not present in the message. If needed, the (*null*) function can be used to explicitly test for this condition.

When a function or component escape is interpreted and the result will be immediately printed, an optional field width can be specified to print the field in exactly a given number of characters. For example, a numeric escape like %4(*size*) will print at most 4 digits of the message size; overflow will be indicated by a ‘?’ in the first position (like ‘?234’). A string escape like %4(*me*) will print the first 4 characters and truncate at the end. Short fields are padded at the right with the fill character (normally, a blank). If the field width argument begins with a leading zero, then the fill character is set to a zero.

As above, the functions (*putnumf*) and (*putstrf*) print their result in exactly the number of characters specified by their leading field width argument. For example, %06(*putnumf*(*size*)) will print the message size in a field six characters wide filled with leading zeros; %14(*putstrf*{*from*}) will print the ‘‘From:’’ header component in fourteen characters with trailing spaces added as needed. For *putstrf*, using a negative value for the field width causes right-justification of the string within the field, with padding on the left up to the field width. The functions (*putnum*) and (*putstr*) print their result in the minimum number of characters required, and ignore any leading field width argument.

The available output width is kept in an internal register; any output past this width will be truncated.

Comments may be inserted in most places where a function argument is not expected. A comment begins with ‘%;’ and ends with a (non-escaped) newline.

With all this in mind, here’s the default format string for *scan*. It’s been divided into several pieces for readability. The first part is:

%4(msg)%<(cur)+%| %>%<{replied}–%?{encrypted}E%| %>

which says that the message number should be printed in four digits, if the message is the current message then a ‘+’ else a space should be printed, and if a ‘‘Replied:’’ field is present then a ‘–’ else if an ‘‘Encrypted:’’ field is present then an ‘E’ otherwise a space should be printed. Next:

%02(mon{date})/%02(mday{date})

the month and date are printed in two digits (zero filled) separated by a slash. Next,

%<{date} %|*>

If a ‘‘Date:’’ field was present, then a space is printed, otherwise a ‘*’. Next,

```
%<(mymbox{from})%<{to}To:%14(friendly{to})%>%>
```

if the message is from me, and there is a ‘‘To:’’ header, print ‘To:’ followed by a ‘‘user-friendly’’ rendering of the first address in the ‘‘To:’’ field. Continuing,

```
%<(zero)%17(friendly{from})%>
```

if either of the above two tests failed, then the ‘‘From:’’ address is printed in a ‘‘user-friendly’’ format. And finally,

```
%{subject}%<{body}<<%{body}%>
```

the subject and initial body (if any) are printed.

For a more complicated example, next consider the default *replcomps* format file.

```
%(lit)%(formataddr %<{reply-to}
```

This clears *str* and formats the ‘‘Reply-To:’’ header if present. If not present, the else-if clause is executed.

```
%?{from}%?{sender}%?{return-path}%>)\
```

This formats the ‘‘From:’’, ‘‘Sender:’’ and ‘‘Return-Path:’’ headers, stopping as soon as one of them is present. Next:

```
%<(nonnull)%(void(width))%(putaddr To: )\n%>\
```

If the *formataddr* result is non-null, it is printed as an address (with line folding if needed) in a field *width* wide with a leading label of ‘‘To: ’’.

```
%(lit)%(formataddr{to})%(formataddr{cc})%(formataddr(me))\
```

str is cleared, and the ‘‘To:’’ and ‘‘Cc:’’ headers, along with the user’s address (depending on what was specified with the ‘‘–cc’’ switch to *repl*) are formatted.

```
%<(nonnull)%(void(width))%(putaddr cc: )\n%>\
```

If the result is non-null, it is printed as above with a leading label of ‘‘cc: ’’.

```
%<{fcc}Fcc: %{fcc}\n%>\
```

If a ‘‘–fcc folder’’ switch was given to *repl* (see *repl* (1) for more details about %{*fcc*}), an ‘‘Fcc:’’ header is output.

```
%<{subject}Subject: Re: %{subject}\n%>\
```

If a subject component was present, a suitable reply subject is output.

```
%<{date}In-reply-to: Your message of "\
%<(nodate{date})%{date}%|%(pretty{date})%>."%<{message-id}
        %{message-id}%>\n%>\
```

If a date component was present, an ‘‘In-Reply-To:’’ header is output with the preface ‘‘Your message of ’’. If the date was parseable, it is output in a user-friendly format, otherwise it is output as-is. The message-id is included if present. As with all plain-text, the row of dashes are output as-is.

This last part is a good example for a little more elaboration. Here’s that part again in pseudo-code:

```
if (comp_exists(date))  then
        print (‘‘In-reply-to: Your message of \‘‘’’)
        if (not_date_string(date.value) then
                print (date.value)
        else
                print (pretty(date.value))
        endif
        print (‘‘\’’’’)
        if (comp_exists(message-id)) then
                print (‘‘\n\t’’)
                print (message-id.value)
        endif
        print (‘‘\n’’)
endif
```

Although this seems complicated, in point of fact, this method is flexible enough to extract individual fields and print them in any format the user desires.

Files

None

Profile Components

None

See Also

scan(1), repl(1), ap(8), dp(8)

Defaults

None

Context

None

History

This software was contributed for MH 6.3. Prior to this, output format specifications were much easier to write, but considerably less flexible.

Bugs

On hosts where *MH* was configured with the BERK option, address parsing is not enabled.

NAME

mh-mail – message format for MH message system

SYNOPSIS

any *MH* command

DESCRIPTION

MH processes messages in a particular format. It should be noted that although neither Bell nor Berkeley mailers produce message files in the format that *MH* prefers, *MH* can read message files in that antiquated format.

Each user possesses a mail drop box which initially receives all messages processed by *post* (8). *Inc* (1) will read from that drop box and incorporate the new messages found there into the user's own mail folders (typically '+inbox'). The mail drop box consists of one or more messages. To facilitate the separation of messages, each message begins and ends with a line consisting of nothing but four CTRL–A (octal 001) characters.

Messages are expected to consist of lines of text. Graphics and binary data are not handled. No data compression is accepted. All text is clear ASCII 7-bit data.

The general ''memo'' framework of RFC–822 is used. A message consists of a block of information in a rigid format, followed by general text with no specified format. The rigidly formatted first part of a message is called the header, and the free-format portion is called the body. The header must always exist, but the body is optional. These parts are separated by an empty line, i.e., two consecutive newline characters. Within *MH*, the header and body may be separated by a line consisting of dashes:

```
To:
cc:
Subject:
--------
```

The header is composed of one or more header items. Each header item can be viewed as a single logical line of ASCII characters. If the text of a header item extends across several real lines, the continuation lines are indicated by leading spaces or tabs.

Each header item is called a component and is composed of a keyword or name, along with associated text. The keyword begins at the left margin, may NOT contain spaces or tabs, may not exceed 63 characters (as specified by RFC–822), and is terminated by a colon (':'). Certain components (as identified by their keywords) must follow rigidly defined formats in their text portions.

The text for most formatted components (e.g., ''Date:'' and ''Message–Id:'') is produced automatically. The only ones entered by the user are address fields such as ''To:'', ''cc:'', etc. Internet addresses are assigned mailbox names and host computer specifications. The rough format is ''local@domain'', such as ''MH@UCI'', or ''MH@UCI–ICSA.ARPA''. Multiple addresses are separated by commas. A missing host/domain is assumed to be the local host/domain.

As mentioned above, a blank line (or a line of dashes) signals that all following text up to the end of the file is the body. No formatting is expected or enforced within the body.

Following is a list of header components that are considered meaningful to various MH programs.
Date:

Added by *post* (8), contains date and time of the message's entry into the transport system.

From:
Added by *post* (8), contains the address of the author or authors (may be more than one if a ''Sender:'' field is present). Replies are typically directed to addresses in the ''Reply–To:'' or ''From:'' field (the former has precedence if present).

Sender:
Added by *post* (8) in the event that the message already has a ''From:'' line. This line contains the address of the actual sender. Replies are never sent to addresses in the ''Sender:'' field.

To:
Contains addresses of primary recipients.

cc:
Contains addresses of secondary recipients.

Bcc:
Still more recipients. However, the ''Bcc:'' line is not copied onto the message as delivered, so these recipients are not listed. *MH* uses an encapsulation method for blind copies, see *send* (1).

Fcc:
Causes *post* (8) to copy the message into the specified folder for the sender, if the message was successfully given to the transport system.

Message–ID:
A unique message identifier added by *post* (8) if the '–msgid' flag is set.

Subject:
Sender's commentary. It is displayed by *scan* (1).

In–Reply–To:
A commentary line added by *repl* (1) when replying to a message.

Resent–Date:
Added when redistributing a message by *post* (8).

Resent–From:
Added when redistributing a message by *post* (8).

Resent–To:
New recipients for a message resent by *dist* (1).

Resent–cc:
Still more recipients. See ''cc:'' and ''Resent–To:''.

Resent–Bcc:
Even more recipients. See ''Bcc:'' and ''Resent–To:''.

Resent–Fcc:
Copy resent message into a folder. See ''Fcc:'' and ''Resent–To:''.

Resent–Message–Id:
A unique identifier glued on by *post* (8) if the '–msgid' flag is set. See ''Message–Id:'' and ''Resent–To:''.

Resent:
Annotation for *dist* (1) under the '–annotate' option.

Forwarded:
Annotation for *forw* (1) under the '–annotate' option.

Replied:
Annotation for *repl* (1) under the '–annotate' option.

Files

/usr/spool/mail/$USER Location of mail drop

Profile Components

None

See Also

Standard for the Format of ARPA Internet Text Messages (aka RFC–822)

Defaults

None

Context

None

NAME

mh-profile – user profile customization for MH message handler

SYNOPSIS

.mh_profile

DESCRIPTION

Each user of *MH* is expected to have a file named *.mh_profile* in his or her home directory. This file contains a set of user parameters used by some or all of the *MH* family of programs. Each line of the file is of the format

profile–component: *value*

The possible profile components are exemplified below. Only 'Path:' is mandatory. The others are optional; some have default values if they are not present. In the notation used below, (profile, default) indicates whether the information is kept in the user's *MH* profile or *MH* context, and indicates what the default value is.

Path: Mail
: Locates *MH* transactions in directory ''Mail''. (profile, no default)

context: context
: Declares the location of the *MH* context file, see the **HISTORY** section below. (profile, default: <mh–dir>/context)

Current–Folder: inbox
: Keeps track of the current open folder. (context, default: folder specified by ''Inbox'')

Inbox: inbox
: Defines the name of your inbox. (profile, default: inbox)

Previous–Sequence: pseq
: Names the sequences which should be defined as the 'msgs' or 'msg' argument given to the program. If not present, or empty, no sequences are defined. Otherwise, for each name given, the sequence is first zero'd and then each message is added to the sequence. (profile, no default)

Sequence–Negation: not
: Defines the string which, when prefixed to a sequence name, negates that sequence. Hence, ''notseen'' means all those messages that are not a member of the sequence ''seen''. (profile, no default)

Unseen–Sequence: unseen
: Names the sequences which should be defined as those messages recently incorporated by *inc*. *Show* knows to remove messages from this sequence once it thinks they have been seen. If not present, or empty, no sequences are defined. Otherwise, each message is added to each sequence name given. (profile, no default)

mh–sequences: .mh_sequences
: The name of the file in each folder which defines public sequences. To disable the use of public sequences, leave the value portion of this entry blank. (profile, default:

.mh_sequences)

atr–*seq*–*folder*: 172 178–181 212
Keeps track of the private sequence called *seq* in the specified folder. (context, no default)

Editor: /usr/ucb/ex
Defines editor to be used by *comp* (1), *dist* (1), *forw* (1), and *repl* (1). (profile, default: prompter)

Msg–Protect: 644
Defines octal protection bits for message files. See *chmod* (1) for an explanation of the octal number. (profile, default: 0644)

Folder–Protect: 711
Defines protection bits for folder directories. (profile, default: 0711)

program: default switches
Sets default switches to be used whenever the mh program *program* is invoked. For example, one could override the *Editor*: profile component when replying to messages by adding a component such as:

    repl: –editor /bin/ed

(profile, no defaults)

lasteditor–next: nexteditor
Names ‘‘nexteditor’’ to be the default editor after using ‘‘lasteditor’’. This takes effect at ‘‘What now?’’ level in *comp*, *dist*, *forw*, and *repl*. After editing the draft with ‘‘lasteditor’’, the default editor is set to be ‘‘nexteditor’’. If the user types ‘‘edit’’ without any arguments to ‘‘What now?’’, then ‘‘nexteditor’’ is used. (profile, no default)

bboards: system
Tells *bbc* which BBoards you are interested in. (profile, default: system)

Folder–Stack: *folders*
The contents of the folder-stack for the *folder* command. (context, no default)

mhe:
If present, tells *inc* to compose an *MHE* auditfile in addition to its other tasks. *MHE* is Brian Reid’s *Emacs* front-end for *MH*. An early version is supplied with the *mh.6* distribution. (profile, no default)

Alternate–Mailboxes: mh@uci–750a, bug-mh*
Tells *repl* and *scan* which addresses are really yours. In this way, *repl* knows which addresses should be included in the reply, and *scan* knows if the message really originated from you. Addresses must be separated by a comma, and the hostnames listed should be the ‘‘official’’ hostnames for the mailboxes you indicate, as local nicknames for hosts are not replaced with their official site names. For each address, if a host is not given, then that address on any host is considered to be you. In addition, an asterisk (‘*’) may appear at either or both ends of the mailbox and host to indicate wild-card matching. (profile, default: your user-id)

Aliasfile: aliases other-alias

Indicates aliases files for *ali*, *whom*, and *send*. This may be used instead of the '–alias file' switch. (profile, no default)

Draft–Folder: drafts

Indicates a default draft folder for *comp*, *dist*, *forw*, and *repl*. (profile, no default)

digest–issue–*list*: 1

Tells *forw* the last issue of the last volume sent for the digest *list*. (context, no default)

digest–volume–*list*: 1

Tells *forw* the last volume sent for the digest *list*. (context, no default)

MailDrop: .mail

Tells *inc* your maildrop, if different from the default. This is superceded by the **MAIL-DROP** envariable. (profile, default: /usr/spool/mail/$USER)

Signature: RAND MH System (agent: Marshall Rose)

Tells *send* your mail signature. This is superceded by the **SIGNATURE** envariable. If **SIGNATURE** is not set and this profile entry is not present, the ‘‘gcos’’ field of the */etc/passwd* file will be used; otherwise, on hosts where *MH* was configured with the UCI option, the file $HOME/.signature is consulted. Your signature will be added to the address *send* puts in the ‘‘From:’’ header; do not include an address in the signature text. (profile, no default)

The following profile elements are used whenever an *MH* program invokes some other program such as *more* (1). The *.mh_profile* can be used to select alternate programs if the user wishes. The default values are given in the examples.

fileproc:	/usr/bs/mh-beta/bin/refile
incproc:	/usr/bs/mh-beta/bin/inc
installproc:	/usr/bs/mh-beta/lib/install–mh
lproc:	/usr/ucb/more
mailproc:	/usr/bs/mh-beta/bin/mhmail
mhlproc:	/usr/bs/mh-beta/lib/mhl
moreproc:	/usr/ucb/more
mshproc:	/usr/bs/mh-beta/bin/msh
packproc:	/usr/bs/mh-beta/bin/packf
postproc:	/usr/bs/mh-beta/lib/post
rmmproc:	none
rmfproc:	/usr/bs/mh-beta/bin/rmf
sendproc:	/usr/bs/mh-beta/bin/send
showproc:	/usr/ucb/more
whatnowproc:	/usr/bs/mh-beta/bin/whatnow
whomproc:	/usr/bs/mh-beta/bin/whom

If you define the envariable **MH**, you can specify a profile other than *.mh_profile* to be read by the *MH* programs that you invoke. If the value of **MH** is not absolute, (i.e., does not begin with a **/**), it will be presumed to start from the current working directory. This is one of the very few exceptions in *MH* where non-absolute pathnames are not considered relative to the user's *MH* directory.

Similarly, if you define the envariable **MHCONTEXT**, you can specify a context other than the normal context file (as specified in the *MH* profile). As always, unless the value of **MHCONTEXT** is absolute, it will be presumed to start from your *MH* directory.

MH programs also support other envariables:

MAILDROP : tells *inc* the default maildrop
This supercedes the ''MailDrop:'' profile entry.

SIGNATURE : tells *send* and *post* your mail signature
This supercedes the ''Signature:'' profile entry.

HOME : tells all *MH* programs your home directory

SHELL : tells *bbl* the default shell to run

TERM : tells *MH* your terminal type
The **TERMCAP** envariable is also consulted. In particular, these tell *scan* and *mhl* how to clear your terminal, and how many columns wide your terminal is. They also tell *mhl* how many lines long your terminal screen is.

editalt : the alternate message
This is set by *dist* and *repl* during edit sessions so you can peruse the message being distributed or replied to. The message is also available through a link called ''@'' in the current directory if your current working directory and the folder the message lives in are on the same UNIX filesystem.

mhdraft : the path to the working draft
This is set by *comp*, *dist*, *forw*, and *repl* to tell the *whatnowproc* which file to ask ''What now?'' questions about. In addition, *dist*, *forw*, and *repl* set **mhfolder** if appropriate. Further, *dist* and *repl* set **mhaltmsg** to tell the *whatnowproc* about an alternate message associated with the draft (the message being distributed or replied to), and *dist* sets **mhdist** to tell the *whatnowproc* that message re-distribution is occurring. Also, **mheditor** is set to tell the *whatnowproc* the user's choice of editor (unless overridden by '–noedit'). Similarly, **mhuse** may be set by *comp*. Finally, **mhmessages** is set by *dist*, *forw*, and *repl* if annotations are to occur (along with **mhannotate**, and **mhinplace**). It's amazing all the information that has to get passed via envariables to make the ''What now?'' interface look squeaky clean to the *MH* user, isn't it? The reason for all this is that the *MH* user can select *any* program as the *whatnowproc*, including one of the standard shells. As a result, it's not possible to pass information via an argument list.
If the WHATNOW option was set during *MH* configuration (type '–help' to an *MH* command to find out), and if this envariable is set, if the commands *refile*, *send*, *show*, or *whom* are not given any 'msgs' arguments, then they will default to using the file indicated by **mhdraft**. This is useful for getting the default behavior supplied by the default *whatnowproc*.

mhfolder : the folder containing the alternate message
This is set by *dist* and *repl* during edit sessions so you can peruse other messages in the current folder besides the one being distributed or replied to. The **mhfolder** envariable is also set by *show*, *prev*, and *next* for use by *mhl*.

MHBBRC :
If you define the envariable **MHBBRC**, you can specify a BBoards information file other than *.bbrc* to be read by *bbc*. If the value of **MHBBRC** is not absolute, (i.e., does not begin with a **/**), it will be presumed to start from the current working directory.

MHFD :
If the OVERHEAD option was set during *MH* configuration (type '–help' to an *MH* command to find out), then if this envariable is set, *MH* considers it to be the number of a file descriptor which is opened, read-only to the *MH* profile. Similarly, if the envariable **MHCONTEXTFD** is set, this is the number of a file descriptor which is opened read-only to the *MH* context. This feature of *MH* is experimental, and is used to examine possible speed improvements for *MH* startup. Note that these envariables must be set and non-empty to enable this feature. However, if OVERHEAD is enabled during *MH* configuration, then when *MH* programs call other *MH* programs, this scheme is used. These file descriptors are not closed throughout the execution of the *MH* program, so children may take advantage of this. This approach is thought to be completely safe and does result in some performance enhancements.

Files

$HOME/.mh_profile	The user profile
or $MH	Rather than the standard profile
<mh–dir>/context	The user context
or $CONTEXT	Rather than the standard context
<folder>/.mh_sequences	Public sequences for <folder>

Profile Components

All

See Also

mh(1), environ(5), mh-sequence(5)

Defaults

None

Context

All

History

In previous versions of *MH*, the current-message value of a writable folder was kept in a file called ''cur'' in the folder itself. In *mh.3*, the *.mh_profile* contained the current-message values for all folders, regardless of their writability.

In all versions of *MH* since *mh.4*, the *.mh_profile* contains only static information, which *MH* programs will **NOT** update. Changes in context are made to the *context* file kept in the users MH *directory*. This includes, but is not limited to: the ''Current–Folder'' entry and all private sequence information. Public sequence information is kept in a file called *.mh_sequences* in each folder.

To convert from the format used in releases of *MH* prior to the format used in the *mh.4* release, *install–mh* should be invoked with the '–compat' switch. This generally happens automatically on *MH* systems generated with the ''COMPAT'' option during *MH* configuration.

The *.mh_profile* may override the path of the *context* file, by specifying a ''context'' entry (this must be in lower-case). If the entry is not absolute (does not start with a **/**), then it is interpreted relative to the user's *MH* directory. As a result, you can actually have more than one set of private sequences by using different context files.

Bugs

The shell quoting conventions are not available in the .mh_profile. Each token is separated by whitespace.

There is some question as to what kind of arguments should be placed in the profile as options. In order to provide a clear answer, recall command line semantics of all *MH* programs: conflicting switches (e.g., '–header and '–noheader') may occur more than one time on the command line, with the last switch taking effect. Other arguments, such as message sequences, filenames and folders, are always remembered on the invocation line and are not superseded by following arguments of the same type. Hence, it is safe to place only switches (and their arguments) in the profile.

If one finds that an *MH* program is being invoked again and again with the same arguments, and those arguments aren't switches, then there are a few possible solutions to this problem. The first is to create a (soft) link in your *$HOME/bin* directory to the *MH* program of your choice. By giving this link a different name, you can create a new entry in your profile and use an alternate set of defaults for the *MH* command. Similarly, you could create a small shell script which called the *MH* program of your choice with an alternate set of invocation line switches (using links and an alternate profile entry is preferable to this solution).

Finally, the *csh* user could create an alias for the command of the form:

```
alias cmd 'cmd arg1 arg2 ...'
```

In this way, the user can avoid lengthy type-in to the shell, and still give *MH* commands safely. (Recall that some *MH* commands invoke others, and that in all cases, the profile is read, meaning that aliases are disregarded beyond an initial command invocation)

NAME

mh-sequence – sequence specification for MH message system

SYNOPSIS

most *MH* commands

DESCRIPTION

Most *MH* commands accept a ‘msg’ or ‘msgs’ specification, where ‘msg’ indicates one message and ‘msgs’ indicates one or more messages. To designate a message, you may use either its number (e.g., 1, 10, 234) or one of these ‘‘reserved’’ message names:

Name	*Description*
first	the first message in the folder
last	the last message in the folder
cur	the most recently accessed message
prev	the message numerically preceding ‘‘cur’’
next	the message numerically following ‘‘cur’’

In commands that take a ‘msg’ argument, the default is ‘‘cur’’. As a shorthand, ‘‘.’’ is equivalent to ‘‘cur’’.

For example: In a folder containing five messages numbered 5, 10, 94, 177 and 325, ‘‘first’’ is 5 and ‘‘last’’ is 325. If ‘‘cur’’ is 94, then ‘‘prev’’ is 10 and ‘‘next’’ is 177.

The word ‘msgs’ indicates that one or more messages may be specified. Such a specification consists of one message designation or of several message designations separated by spaces. A message designation consists either of a message name as defined above, or a message range.

A message range is specified as ‘‘name1–name2’’ or ‘‘name:n’’, where ‘name’, ‘name1’ and ‘name2’ are message names, and ‘n’ is an integer.

The specification ‘‘name1–name2’’ designates all currently-existing messages from ‘name1’ to ‘name2’ inclusive. The message name ‘‘all’’ is a shorthand for the message range ‘‘first–last’’.

The specification ‘‘name:n’’ designates up to ‘n’ messages. These messages start with ‘name’ if ‘name’ is a message number or one of the reserved names ‘‘first’’ ‘‘cur’’, or ‘‘next’’, The messages end with ‘name’ if ‘name’ is ‘‘prev’’ or ‘‘last’’. The interpretation of ‘n’ may be overridden by preceding ‘n’ with a plus or minus sign; ‘+n’ always means up to ‘n’ messages starting with ‘name’, and ‘–n’ always means up to ‘n’ messages ending with ‘name’.

In commands which accept a ‘msgs’ argument, the default is either ‘‘cur’’ or ‘‘all’’, depending on which makes more sense for each command (see the individual man pages for details). Repeated specifications of the same message have the same effect as a single specification of the message.

User–Defined Message Sequences

In addition to the ‘‘reserved’’ (pre-defined) message names given above, *MH* supports user-defined sequence names. User-defined sequences allow the *MH* user a tremendous amount of power in dealing with groups of messages in the same folder by allowing the user to bind a group of messages to a meaningful symbolic name.

The name used to denote a message sequence must consist of an alphabetic character followed by zero or more alphanumeric characters, and can not be one of the ‘‘reserved’’ message names above. After defining a sequence, it can be used wherever an *MH* command expects a ‘msg’ or ‘msgs’ argument.

Some forms of message ranges are allowed with user-defined sequences. The specification ‘‘name:n’’ may be used, and it designates up to the first ‘n’ messages (or last ‘n’ messages for ‘–n’) which are elements of the user-defined sequence ‘name’.

The specifications ‘‘name:next’’ and ‘‘name:prev’’ may also be used, and they designate the next or previous message (relative to the current message) which is an element of the user-defined sequence ‘name’. The specificaitions ‘‘name:first’’ and ‘‘name:last’’ are equivalent to ‘‘name:1’’ and ‘‘name:–1’’, respectively. The specification ‘‘name:cur’’ is not allowed (use just ‘‘cur’’ instead). The syntax of these message range specifcations is subject to change in the future.

User-defined sequence names are specific to each folder. They are defined using the *pick* and *mark* commands.

Public and Private User-Defined Sequences

There are two varieties of sequences: *public* sequences and *private* sequences. *Public* sequences of a folder are accessible to any *MH* user that can read that folder and are kept in the .mh_sequences file in the folder. *Private* sequences are accessible only to the *MH* user that defined those sequences and are kept in the user’s *MH* context file. By default, *pick* and *mark* create *public* sequences if the folder for which the sequences are being defined is writable by the *MH* user. Otherwise, *private* sequences are created. This can be overridden with the ‘–public’ and ‘–private’ switches to *mark*.

Sequence Negation

MH provides the ability to select all messages not elements of a user-defined sequence. To do this, the user should define the entry ‘‘Sequence–Negation’’ in the *MH* profile file; its value may be any string. This string is then used to preface an existing user-defined sequence name. This specification then refers to those messages not elements of the specified sequence name. For example, if the profile entry is:

    Sequence–Negation: not

then anytime an *MH* command is given ‘‘notfoo’’ as a ‘msg’ or ‘msgs’ argument, it would substitute all messages that are not elements of the sequence ‘‘foo’’.

Obviously, the user should beware of defining sequences with names that begin with the value of the ‘‘Sequence–Negation’’ profile entry.

The Previous Sequence

MH provides the ability to remember the ‘msgs’ or ‘msg’ argument last given to an *MH* command. The entry ‘‘Previous–Sequence’’ should be defined in the *MH* profile; its value should be a sequence name or multiple sequence names separated by spaces. If this entry is defined, when when an *MH* command finishes, it will define the sequence(s) named in the value of this entry to be those messages that were specified to the command. Hence, a profile entry of

Previous–Sequence: pseq

directs any *MH* command that accepts a ‘msg’ or ‘msgs’ argument to define the sequence ‘‘pseq’’ as those messages when it finishes.

Note: there can be a performance penalty in using the ‘‘Previous–Sequence’’ facility. If it is used, **all** *MH* programs have to write the sequence information to the .mh_sequences file for the folder each time they run. If the ‘‘Previous–Sequence’’ profile entry is not included, only *pick* and *mark* will write to the .mh_sequences file.

The Unseen Sequence

Finally, some users like to indicate messages which have not been previously seen by them. Both *inc* and *show* honor the profile entry ‘‘Unseen–Sequence’’ to support this activity. This entry in the .mh_profile should be defined as one or more sequence names separated by spaces. If there is a value for ‘‘Unseen–Sequence’’ in the profile, then whenever *inc* places new messages in a folder, the new messages will also be added to the sequence(s) named in the value of this entry. Hence, a profile entry of

Unseen–Sequence: unseen

directs *inc* to add new messages to the sequence ‘‘unseen’’. Unlike the behavior of the ‘‘Previous–Sequence’’ entry in the profile, however, the sequence(s) will **not** be zeroed by *inc*.

Similarly, whenever *show* (or *next* or *prev*) displays a message, that message will be removed from any sequences named by the ‘‘Unseen–Sequence’’ entry in the profile.

Files

$HOME/.mh_profile	The user profile
<mh–dir>/context	The user context
<folder>/.mh_sequences	Public sequences for <folder>

Profile Components

Sequence–Negation:	To designate messages not in a sequence
Previous–Sequence:	The last message specification given
Unseen–Sequence:	Those messages not yet seen by the user

See Also

mh(1), mark(1), pick(1), mh-profile(5)

Defaults

None

Context

All

Bugs

User-defined sequences are stored in the .mh_sequences file as a series of message specifications separated by spaces. If a user-defined sequence contains too many individual message specifications, that line in the file may become too long for *MH* to handle. This will generate the error message ‘‘.mh_sequences is poorly formatted’’. You’ll have to edit the file by hand to remove the offending line.

This can happen to users who define the ‘‘Previous–Sequence’’ entry in the *MH* profile and have a folder containing many messages with gaps in the numbering. A workaround for large folders is to minimize numbering gaps by using ‘‘folder –pack’’ often.

NAME

ap – parse addresses 822-style

SYNOPSIS

/usr/bs/mh-beta/lib/ap [–form formatfile] [–format string] [–normalize] [–nonormalize] [–width columns] addrs ... [–help]

DESCRIPTION

Ap is a program that parses addresses according to the ARPA Internet standard. It also understands many non–standard formats. It is useful for seeing how *MH* will interpret an address.

The *ap* program treats each argument as one or more addresses, and prints those addresses out in the official 822–format. Hence, it is usually best to enclose each argument in double–quotes for the shell.

To override the output format used by *ap*, the '–format string' or '–format file' switches are used. This permits individual fields of the address to be extracted with ease. The string is simply a format stringand thefile is simply a format file. See *mh–format* (5) for the details.

In addition to the standard escapes, *ap* also recognizes the following additional escape:

Escape	*Returns*	*Description*
error	string	A diagnostic if the parse failed

If the '–normalize' switch is given, *ap* will try to track down the official hostname of the address.

Here is the default format string used by *ap*:

%<{error}%{error}: %{text}%|%(putstr(proper{text}))%>

which says that if an error was detected, print the error, a ':', and the address in error. Otherwise, output the 822–proper format of the address.

Files

$HOME/.mh_profile	The user profile
/usr/bs/mh-beta/lib/mtstailor	tailor file

Profile Components

None

See Also

dp(8),
Standard for the Format of ARPA Internet Text Messages (aka RFC–822)

Defaults

'–format' defaults as described above
'–normalize'
'–width' defaults to the width of the terminal

Context

None

Bugs

The argument to the '–format' switch must be interpreted as a single token by the shell that invokes *ap*. Therefore, one must usually place the argument to this switch inside double–quotes.

On hosts where *MH* was configured with the BERK option, address parsing is not enabled.

NAME

conflict – search for alias/password conflicts

SYNOPSIS

/usr/bs/mh-beta/lib/conflict [–mail name] [–search directory] [aliasfiles...] [–help]

DESCRIPTION

Conflict is a program that checks to see if the interface between *MH* and transport system is in good shape

Conflict also checks for maildrops in /usr/spool/mail which do not belong to a valid user. It assumes that no user name will start with '.', and thus ignores files in /usr/spool/mail which begin with '.'. It also checks for entries in the *group* (5) file which do not belong to a valid user, and for users who do not have a valid group number. In addition duplicate users and groups are noted.

If the '–mail name' switch is used, then the results will be sent to the specified *name*. Otherwise, the results are sent to the standard output.

The '–search directory' switch can be used to search directories other than /usr/spool/mail and to report anomalies in those directories. The '–search directory' switch can appear more than one time in an invocation to *conflict*.

Conflict should be run under *cron* (8), or whenever system accounting takes place.

Files

/usr/bs/mh-beta/lib/mtstailor	tailor file
/etc/passwd	List of users
/etc/group	List of groups
/usr/bs/mh-beta/bin/mhmail	Program to send mail
/usr/spool/mail/	Directory of mail drop

Profile Components

None

See Also

mh–alias(5)

Defaults

'aliasfiles' defaults to /usr/bs/mh-beta/lib/MailAliases

Context

None

NAME

dp – parse dates 822-style

SYNOPSIS

/usr/bs/mh-beta/lib/dp [–form formatfile] [–format string] [–width columns] dates ... [–help]

DESCRIPTION

Dp is a program that parses dates according to the ARPA Internet standard. It also understands many non–standard formats, such as those produced by TOPS–20 sites and some UNIX sites using *ctime* (3). It is useful for seeing how *MH* will interpret a date.

The *dp* program treats each argument as a single date, and prints the date out in the official 822–format. Hence, it is usually best to enclose each argument in double–quotes for the shell.

To override the output format used by *dp*, the '–format string' or '–format file' switches are used. This permits individual fields of the address to be extracted with ease. The string is simply a format stringand thefile is simply a format file. See *mh–format* (5) for the details.

Here is the default format string used by *dp*:

```
%<(nodate{text})error: %{text}%|%(putstr(pretty{text}))%>
```

which says that if an error was detected, print the error, a ':', and the date in error. Otherwise, output the 822–proper format of the date.

Files

$HOME/.mh_profile	The user profile

Profile Components

None

See Also

ap(8)
Standard for the Format of ARPA Internet Text Messages (aka RFC–822)

Defaults

'–format' default as described above
'–width' default to the width of the terminal

Context

None

Bugs

The argument to the '–format' switch must be interpreted as a single token by the shell that invokes *dp*. Therefore, one must usually place the argument to this switch inside double–quotes.

NAME

fmtdump – decode MH format files

SYNOPSIS

/usr/bs/mh-beta/lib/fmtdump [–form formatfile] [–format string] [–help]

DESCRIPTION

Fmtdump is a program that parses an *MH* format file and produces a pseudo-language listing of the how *MH* interprets the file.

The '–format string' and '–form formatfile' switches may be used to specify a format string or format file to read. The string is simply a format string and the file is simply a format file. See *mh-format* (5) for the details.

Files

$HOME/.mh_profile	The user profile
/usr/bs/mh-beta/lib/scan.default	The default format file

Profile Components

Path:	To determine the user's MH directory

See Also

mh-format(5), mh-sequences(8)

Context

None

Bugs

The output may not be useful unless you are familiar with the internals of the mh-format subroutines.

NAME

install-mh – initialize the MH environment

SYNOPSIS

/usr/bs/mh-beta/lib/install–mh [–auto] [–compat]

DESCRIPTION

When a user runs any *MH* program for the first time, the program will invoke *install–mh* (with the '–auto' switch) to query the user for the initial *MH* environment. The user does **NOT** invoke this program directly. The user is asked for the name of the directory that will be designated as the user's *MH* directory. If this directory does not exist, the user is asked if it should be created. Normally, this directory should be under the user's home directory, and has the default name of Mail/. After *install–mh* has written the initial .mh_profile for the user, control returns to the original *MH* program.

As with all *MH* commands, *install–mh* first consults the **$HOME** envariable to determine the user's home directory. If **$HOME** is not set, then the */etc/passwd* file is consulted.

When converting from *mh.3* to *mh.4*, *install–mh* is automatically invoked with the '–compat' switch.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To set the user's MH directory

Context

With '–auto', the current folder is changed to ''inbox''.

NAME

post – deliver a message

SYNOPSIS

/usr/bs/mh-beta/lib/post [–alias aliasfile] [–filter filterfile] [–nofilter] [–format] [–noformat] [–mime] [–nomime] [–msgid] [–nomsgid] [–verbose] [–noverbose] [–watch] [–nowatch] [–width columns] file [–help]

DESCRIPTION

Post is the program called by *send* (1) to deliver the message in *file* to local and remote users. In fact, all of the functions attributed to *send* on its manual page are performed by *post*, with *send* acting as a relatively simple preprocessor. Thus, it is *post* which parses the various header fields, appends From: and Date: lines, and interacts with the *MMDF* transport system. *Post* will not normally be called directly by the user.

Post searches the ‘‘To:’’, ‘‘cc:’’, ‘‘Bcc:’’, ‘‘Fcc:’’, and ‘‘Resent–xxx:’’ header lines of the specified message for destination addresses, checks these addresses for validity, and formats them so as to conform to ARPAnet Internet Message Format protocol, unless the ‘–noformat’ flag is set. This will normally cause ‘‘*@local–site*’’ to be appended to each local destination address, as well as any local return addresses. The ‘–width columns’ switch can be used to indicate the preferred length of the header components that contain addresses.

If a ‘‘Bcc:’’ field is encountered, its addresses will be used for delivery, and the ‘‘Bcc:’’ field will be removed from the message sent to sighted recipients. The blind recipients will receive an entirely new message with a minimal set of headers. Included in the body of the message will be a copy of the message sent to the sighted recipients. If ‘–filter filterfile’ is specified, then this copy is filtered (re–formatted) prior to being sent to the blind recipients. Otherwise, to use the MIME rules for encapsulation, specify the ‘–mime’ switch.

The ‘–alias aliasfile’ switch can be used to specify a file that post should take aliases from. More than one file can be specified, each being preceded with ‘–alias’. In any event, the primary alias file is read first.

The ‘–msgid’ switch indicates that a ‘‘Message–ID:’’ or ‘‘Resent–Message–ID:’’ field should be added to the header.

The ‘–verbose’ switch indicates that the user should be informed of each step of the posting/filing process.

The ‘–watch’ switch indicates that the user would like to watch the transport system’s handling of the message (e.g., local and ‘‘fast’’ delivery).

Post consults the envariable **$SIGNATURE** to determine the sender’s personal name in constructing the ‘‘From:’’ line of the message.

Files

/usr/bs/mh-beta/lib/mtstailor	tailor file
/usr/bs/mh-beta/bin/refile	Program to process Fcc:s
/usr/bs/mh-beta/lib/mhl	Program to process Bcc:s
/usr/bs/mh-beta/lib/MailAliases	Primary alias file

Profile Components

post does **NOT** consult the user’s .mh_profile

See Also

Standard for the Format of ARPA Internet Text Messages (aka RFC–822),
mhmail(1), send(1), mh–mail(5), mh–alias(5)

Defaults

'–alias /usr/bs/mh-beta/lib/MailAliases'
'–format'
'–nomime'
'–nomsgid'
'–noverbose'
'–nowatch'
'–width 72'
'–nofilter'

Context

None

Bugs

''Reply–To:'' fields are allowed to have groups in them according to the 822 specification, but *post* won't let you use them.

5. REPORTING PROBLEMS

If problems are encountered with an *MH* program, the problems should be reported to the local maintainers of *MH*. When doing this, the name of the program should be reported, along with the version information for the program. To find out what version of an *MH* program is being run, invoke the program with the '–help' switch. In addition to listing the syntax of the command, the program will list information pertaining to its version. This information includes the version of *MH*, the host it was generated on, and the date the program was loaded. A second line of information, found on versions of *MH* after #5.380 include *MH* configuration options. For example,

```
version: MH 6.1 #1[UCI] (nrtc-gremlin) of Wed Nov  6 01:13:53 PST 1985
options: [BSD42] [MHE] [NETWORK] [SENDMTS] [MMDFII] [SMTP] [POP]
```

The '6.1 #1[UCI]' indicates that the program is from the UCI *mh.6* version of *MH*. The program was generated on the host 'nrtc-gremlin' on 'Wed Nov 6 01:13:53 PST 1985'. It's usually a good idea to send the output of the '–help' switch along with your report.

If there is no local *MH* maintainer, try the address **Bug-MH**. If that fails, use the Internet mailbox **Bug-MH@ICS.UCI.EDU**.

6. ADVANCED FEATURES

This section describes some features of *MH* that were included strictly for advanced *MH* users. These capabilities permit *MH* to exhibit more powerful behavior for the seasoned *MH* users.

USER–DEFINED SEQUENCES

User–defined sequences allow the *MH* user a tremendous amount of power in dealing with groups of messages in the same folder by allowing the user to bind a group of messages to a meaningful symbolic name. The user may choose any name for a message sequence, as long as it consists of alphanumeric characters and does not conflict with the standard *MH* reserved message names (e.g., ‘‘first’’, etc). After defining a sequence, it can be used wherever an *MH* command expects a ‘msg’ or ‘msgs’ argument.

A restricted form of message ranges are allowed with user–defined sequences. The form ‘‘name:n’’, specifies up to the first ‘n’ messages which are part of the user–defined sequence ‘name’. A leading plus sign is allowed on ‘n’, but is ignored. The interpretation of n is overridden if n is preceded by a minus sign; ‘–n’ always means up to the last ‘n’ messages which are part of the sequence ‘name’.

Although all *MH* commands expand user–defined sequences as appropriate, there are two commands that allow the user to define and manipulate them: *pick* and *mark*.

Pick and User–Defined Sequences

Most users of *MH* will use user–defined sequences only with the *pick* command. By giving the ‘–sequence name’ switch to *pick* (which can occur more than once on the command line), each sequence named is defined as those messages which *pick* matched according the the selection criteria it was given. Hence,

```
pick  –from  frated  –seq  fred
```

finds all those messages in the current folder which were from ‘‘frated’’, creates a sequence called ‘‘fred’’, and then adds them to the sequence. The user could then invoke

```
scan  fred
```

to get a *scan* listing of those messages. Note that by default, *pick* creates the named sequences before it adds the selected messages to the sequence. Hence, if the named sequence already existed, the sequence is destroyed prior to being re-defined (nothing happens to the messages that were a part of this sequence, they simply cease to be members of that sequence). By using the ‘–nozero’ switch, this behavior can be inhibited, as in

```
pick  –from  frated  –seq  sgroup
pick  –from  fear  –seq  sgroup  –nozero
pick  –from  freida  –seq  sgroup  –nozero
```

finds all those messages in the current folder which were from ‘‘frated’’, ‘‘fear’’, or ‘‘freida’’, and defines the sequence called ‘‘sgroup’’ as exactly those messages. These operations amounted to an ‘‘inclusive–or’’ of three selection criteria, using *pick*, one can also generate the ‘‘and’’ of some selection criteria as well:

```
pick –from frated –seq fred
pick –before friday –seq fred fred
```

This example defines the sequence called ‘‘fred’’ as exactly those messages from ‘‘frated’’ that were dated prior to ‘‘friday’’.[1]

Pick is normally used as a back–quoted command, for example,

```
scan ‘pick –from postmaster‘
```

Now suppose that the user decides that another command should be issued, using exactly those messages. Since, *pick* wasn’t given a ‘–sequence name’ argument in this example, the user would end–up typing the entire back–quoted command again. A simpler way is to add a default sequence name to the .mh_profile. For example,

```
pick: –seq select –list
```

will tell *pick* to always define the sequence ‘‘select’’ whenever it’s run. The ‘-list’ is necessary since the ‘–sequence name’ switch sets ‘–nolist’ whenever the former is encountered. Hence, this profile entry makes *pick* define the ‘‘select’’ sequence and otherwise behave exactly as if there was no profile entry at all.

Mark and User–Defined Sequences

The *mark* command lets the user perform low–level manipulation of sequences, and also provides a well–needed debug facility to the implementors/developers/maintainers of *MH* (the *MH*–hacks). In the future, a user–friendly ‘‘front–end’’ for *mark* will probably be developed to give the *MH* user a way to take better advantage of the underlying facilities.

Public and Private User–Defined Sequences

There are two kinds of sequences: *public* sequences, and *private* sequences. *Public* sequences of a folder are accessible to any *MH* user that can read that folder and are kept in the .mh_sequences file in the folder. *Private* sequences are accessible only to the *MH* user that defined those sequences and are kept in the user’s *MH* context file. By default, *pick* (and *mark*) create *public* sequences if the folder for which the sequences are being defined is writable by the *MH* user. Otherwise, *private* sequences are created. This can be overridden with the ‘–public’ and ‘–nopublic’ switches.

Sequence Negation

In addition to telling an *MH* command to use the messages in the sequence ‘‘seen’’, as in

```
refile seen +old
```

[1] Of course, it is much easier to simply use the built–in boolean operation of *pick* to get the desired results:

```
pick –from frated –or –from fear –or –from freida –seq sgroup
```

and

```
pick –from frated –and –before friday –seq fred
```

do exactly the same thing as the five commands listed above. Hence, the ‘–nozero’ option to *pick* is only useful to manipulate existing sequences.

it would be useful to be easily able to tell an *MH* command to use all messages *except* those in the sequence. One way of doing this would be to use *mark* and define the sequence explicitly, as in

```
mark –delete –zero seen –seq notseen
```

which, owing to *mark* 's cryptic interpretation of '–delete' and '–zero', defines the sequence ''notseen'' to be all messages not in the sequence ''seen''. Naturally, anytime the sequence ''seen'' is changed, ''notseen'' will have to be updated. Another way to achieve this is to define the entry ''Sequence–Negation:'' in the .mh_profile. If the entry was

```
Sequence–Negation: not
```

then anytime an *MH* command was given ''notseen'' as a 'msg' or 'msgs' argument, it would substitute all messages that are not a member of the sequence ''seen''. That is,

```
refile notseen +new
```

does just that. The value of the ''Sequence–Negation:'' entry in the profile can be any string. Hence, experienced users of *MH* do not use a word, but rather a special character which their shell does not interpret (users of the *CShell* use a single caret or circumflex (usually shift–6), while users of the Bourne shell use an exclamation–mark). This is because there is nothing to prevent a user of *MH* from defining a sequence with this string as its prefix, if the string is nothing by letters and digits. Obviously, this could lead to confusing behavior if the ''Sequence–Negation:'' entry leads *MH* to believe that two sequences are opposites by virtue of their names differing by the prefix string.

The Previous Sequence

Many times users find themselves issuing a series of commands on the same sequences of messages. If the user first defined these messages as a sequence, then considerable typing may be saved. If the user doesn't have this foresight, *MH* provides a handy way of having *MH* remember the 'msgs' or 'msg' argument last given to an *MH* command. If the entry ''Previous–Sequence:'' is defined in the .mh_profile, then when the command finishes, it will define the sequence(s) named in the value of this entry as being exactly those messages that were specified. Hence, a profile entry of

```
Previous–Sequence: pseq
```

directs any *MH* command that accepts a 'msg' or 'msgs' argument to define the sequence ''pseq'' as those messages when it finishes. More than one sequence name may be placed in this entry, separated with spaces. The one disadvantage of this approach is that the *MH* progams have to update the sequence information for the folder each time they run (although most programs read this information, usually only *pick* and *mark* have to write this information out).

The Unseen Sequence

Finally, some users like to distinguish between messages which have been previously seen by them. Both *inc* and *show* honorthe profile entry ''Unseen–Sequence'' to support this activity. Whenever *inc* places new messages in a folder, if the entry ''Unseen–Sequence'' is defined in the .mh_profile, then when the command finishes, *inc* will add the new messages to the sequence(s) named in the value of this entry. Hence, a profile entry of

```
Unseen–Sequence: unseen
```

directs *inc* to add new messages to the sequence ‘‘unseen’’. Unlike the behavior of the ‘‘Previous–Sequence’’ entry in the profile however, the sequence(s) will **not** be zero’d.

Similarly, whenever *show* (or *next* or *prev*) displays a message, they remove those messages from any sequences named by the ‘‘Unseen–Sequence’’ entry in the profile.

COMPOSITION OF MAIL

There are a number of interesting advanced facilities for the composition of outgoing mail.

The Draft Folder

The *comp*, *dist*, *forw*, and *repl* commands have two switches, ‘–draftfolder +folder’ and ‘–draftmessage msg’. If ‘–draftfolder +folder’ is used, these commands are directed to construct a draft message in the indicated folder. (The ‘‘Draft–Folder:’’ profile entry may be used to declare a default draft folder for use with *comp*, *dist*, *forw*, and *repl*) If ‘–draftmessage msg’ is not used, it defaults to ‘new’ (unless the user invokes *comp* with ‘–use’, in which case the default is ‘cur’). Hence, the user may have several message compositions in progress simultaneously. Now, all of the *MH* tools are available on each of the user’s message drafts (e.g., *show*, *scan*, *pick*, and so on). If the folder does not exist, the user is asked if it should be created (just like with *refile*). Also, the last draft message the user was composing is known as ‘cur’ in the draft folder.

Furthermore, the *send* command has these switches as well. Hence, from the shell, the user can send off whatever drafts desired using the standard *MH* ‘msgs’ convention with ‘–draftmessage msgs’. If no ‘msgs’ are given, it defaults to ‘cur’.

In addition, all five programs have a ‘–nodraftfolder’ switch, which undoes the last occurrence of ‘–draftfolder folder’ (useful if the latter occurs in the user’s *MH* profile).

If the user does not give the ‘–draftfolder +folder’ switch, then all these commands act ‘‘normally’’. Note that the ‘–draft’ switch to *send* and *show* still refers to the file called ‘draft’ in the user’s *MH* directory. In the interests of economy of expression, when using *comp* or *send*, the user needn’t prefix the draft ‘msg’ or ‘msgs’ with ‘–draftmessage’. Both of these commands accept a ‘file’ or ‘files’ argument, and they will, if given ‘–draftfolder +folder’ treat these arguments as ‘msg’ or ‘msgs’.[2] Hence,

```
send -draftf +drafts first
```

is the same as

```
send -draftf +drafts -draftm first
```

To make all this a bit more clear, here are some examples. Let’s assume that the following entries are in the *MH* profile:

```
Draft–Folder: +drafts
sendf: -draftfolder +drafts
```

Furthermore, let’s assume that the program *sendf* is a (symbolic) link in the user’s

[2] This may appear to be inconsistent, at first, but it saves a lot of typing.

$HOME/bin/ directory to *send*. Then, any of the commands

```
comp
dist
forw
repl
```

constructs the message draft in the 'draft' folder using the 'new' message number. Furthermore, they each define 'cur' in this folder to be that message draft. If the user were to use the *quit* option at 'What now?' level, then later on, if no other draft composition was done, the draft could be sent with simply

```
sendf
```

Or, if more editing was required, the draft could be edited with

```
comp -use
```

Instead, if other drafts had been composed in the meantime, so that this message draft was no longer known as 'cur' in the 'draft' folder, then the user could *scan* the folder to see which message draft in the folder should be used for editing or sending. Clever users could even employ a back-quoted *pick* to do the work:

```
comp -use `pick +drafts -to bug-mh`
```

or

```
sendf `pick +drafts -to bug-mh`
```

Note that in the *comp* example, the output from *pick* must resolve to a single message draft (it makes no sense to talk about composing two or more drafts with one invocation of *comp*). In contrast, in the *send* example, as many message drafts as desired can appear, since *send* doesn't mind sending more than one draft at a time.

Note that the argument '–draftfolder +folder' is not included in the profile entry for *send*, since when *comp*, et. al., invoke *send* directly, they supply *send* with the UNIX pathname of the message draft, and **not** a 'draftmessage msg' argument. As far as *send* is concerned, a *draft folder* is not being used.

It is important to realize that *MH* treats the draft folder like a standard *MH* folder in nearly all respects. There are two exceptions: <u>first</u>, under no circumstancs will the '–draftfolder folder' switch cause the named folder to become the current folder.[3] <u>Second</u>, although conceptually *send* deletes the 'msgs' named in the draft folder, it does not call 'delete-prog' to perform the deletion.

What Happens if the Draft Exists

[3] Obviously, if the folder appeared in the context of a standard '+folder' argument to an *MH* program, as in

```
scan +drafts
```

it might become the current folder, depending on the context changes of the *MH* program in question.

When the *comp*, *dist*, *forw*, and *repl* commands are invoked and the draft you indicated already exists, these programs will prompt the user for a reponse directing the program's action. The prompt is

> Draft ''/usr/src/uci/mh/mhbox/draft'' exists (xx bytes).
> Disposition?

The appropriate responses and their meanings are: replace: deletes the draft and starts afresh; list: lists the draft; refile: files the draft into a folder and starts afresh; and, quit: leaves the draft intact and exits. In addition, if you specified '–draftfolder folder' to the command, then one other response will be accepted: new: finds a new draft, just as if '–draftmessage new' had been given. Finally, the *comp* command will accept one more response: use: re-uses the draft, just as if '–use' had been given.

The Push Option at What now? Level

The *push* option to the ''What now?'' query in the *comp*, *dist*, *forw*, and *repl* commands, directs the command to *send* the draft in a special detached fashion, with all normal output discarded. If *push* is used and the draft can not be sent, then *MH* will send the user a message, indicating the name of the draft file, and an explanation of the failure.

The user can also invoke *send* from the shell with the '–push' switch, which makes *send* act like it had been *push* 'd by one of the composition commands.

By using *push*, the user can free the shell to do other things, because it appears to the shell that the *MH* command has finished. As a result the shell will immediately prompt for another command, despite the fact that the command is really still running. Note that if the user indicates that annotations are to be performed (with '–annotate' to *dist*, *forw*, or *repl*), the annotations will be performed after the message has been successfully sent. This action will appear to occur asynchronously. Obviously, if one of the messages that is to be annotated is removed before the draft has been successfully sent, then when *MH* tries to make the annotations, it won't be able to do so. In previous versions of *MH*, this resulted in an error message mysteriously appearing on the user's terminal. In *mh.5* and later versions, in this special circumstance, no error will be generated.

If send is *push* 'd, then the '–forward' switch is examined if a failure notice is generated. If given, then the draft is forwarded with the failure notice sent to the user. This allows rapid *burst* 'ing of the failure notice to retrieve the unsent draft.

Options at What now? Level

By default, the message composition programs call a program called *whatnow* before the initial draft composition. The *MH* user can specify any program for this. Following is some information about the default ''What now?'' level. More detailed information can be found in the *whatnow* (1) manual entry.

When using the *comp*, *dist*, *forw*, and *repl* commands at ''What now?'' level, the *edit*, *list*, *headers*, *refile*, and (for the *dist* and *repl* commands) the *display* options will pass on any additional arguments given them to whatever program they invoke.

In *mh.1* (the original RAND *MH*) and *mh.2* (the first UCI version of *MH*), *MH* used a complicated heuristic to determine if the draft should be deleted or preserved after an unsuccessful edit. In *mh.3*, *MH* was changed to preserve the draft always, since *comp*, et. al., could usually look at a draft, apply another set of heuristics, and decide if it was important or not. With the notion of a *draft folder*, in which one by default gets a 'new' message draft, the edit deletion/preservation algorithm was re-implemented, to keep the draft folder from being cluttered with aborted edits.

Also, note that by default, if the draft cannot be successfully sent, these commands return to ‘‘What now?’’ level. But, when *push* is used, this does not happen (obviously). Hence, if these commands were expected to annotate any messages, this will have to be done by hand, later on, with the *anno* command.

Finally, if the ‘–delete’ switch is not given to the *quit* option, then these commands will inform the user of the name of the unsent draft file.

Digests

The *forw* command has the beginnings of a digestifying facility, with the ‘–digest list’, ‘–issue number’, and ‘–volume number’ switches.

If *forw* is given ‘‘list’’ to the ‘–digest’ switch as the name of the discussion group, and the ‘–issue number’ switch is not given, then *forw* looks for an entry in the user’s *MH* context called ‘‘*digest*–issue–list’’ and increments its value to use as the issue number. Similarly, if the ‘–volume number’ switch is not given, then *forw* looks for ‘‘*digest*–volume–list’’ (but does not increment its value) to use as the volume number.

Having calculated the name of the digest and the volume and issue numbers, *forw* will now process the components file using the same format string mechanism used by *repl*. The current ‘%’–escapes are:

escape	*type*	*substitution*
digest	string	digest name
msg	integer	issue number
cur	integer	volume number

In addition, to capture the current date, any of the escapes valid for *dp* (8) are also valid for *forw*.

The default components file used by *forw* when in digest mode is:

```
From:     %{digest}-Request
To:       %{digest} Distribution: dist-%{digest};
Subject:  %{digest} Digest V%(cur) #%(msg)
Reply-To: %{digest}
--------
%{digest} Digest %(weekday{date}), %2(mday{date}) %(month{date}) 19%02(year{date})
                 Volume %(cur) : Issue %(msg)

Today's Topics:
```

Hence, when the ‘–digest’ switch is present, the first step taken by *forw* is to expand the format strings in the component file. The next step is to compose the draft using the standard digest encapsulation algorithm (even putting an ‘‘End of list Digest’’ trailer in the draft). Once the draft is composed by *forw*, *forw* writes out the volume and issue profile entries for the digest, and then invokes the editor.

Naturally, when composing the draft, *forw* will honor the ‘–filter filterfile’ switch, which is given to *mhl* to filter each message being forwarded prior to encapsulation in the draft. A good filter file to use, which is called *mhl.digest*, is:

```
width=80,overflowoffset=10
leftadjust,compress,compwidth=9
Date:formatfield="%<(nodate{text})%{text}%|%(tws{text})%>"
From:
Subject:
:
body:nocomponent,overflowoffset=0,noleftadjust,nocompress
```

FOLDER HANDLING

There are two interesting facilities for manipulating folders: relative folder addressing, which allows a user to shorten the typing of long folder names; and the folder–stack, which permits a user to keep a stack of current folders.

Relative Folder Addressing

By default, when ‘+folder’ is given, and the folder name is not absolute (does not start with **/**, **./**, or **../**), then the UNIX pathname of the folder is interpreted relative to the user’s *MH* directory. Although this mechanism works fine for top–level folders and their immediate sub–folders, once the depth of the sub–folder tree grows, it becomes rather unwieldly:

```
scan  +mh/mh.4/draft/flames
```

is a lot of typing. *MH* can’t do anything if the current folder was ‘‘+inbox’’, but if the current folder was, say, ‘‘+mh/mh.4/draft’’, *MH* has a short–hand notation to reference a sub–folder of the current folder. Using the ‘@folder’ notation, the *MH* user can direct any *MH* program which expects a ‘+folder’ argument to look for the folder relative to the current folder instead of the user’s *MH* directory. Hence, if the current folder *was* ‘‘+mh/mh.4/draft’’, then

```
scan  @flames
```

would do the trick handily. In addition, if the current folder *was* ‘‘+mh/mh.4/draft’’,

```
scan  @../pick
```

would scan the folder ‘‘+mh/mh.4/pick’’, since, in the UNIX fashion, it references the folder ‘‘pick’’ which is a sub–folder of the folder that is the parent of the current folder. Since most advanced *MH* users seem to exhibit a large degree of locality in referencing folders when they process mail, this convention should receive a wide range of uses.

The Folder–Stack

The *folder–stack* mechanism in *MH* gives the *MH* user a facility similar to the *CShell* ’s directory–stack. Simply put,

```
folder  –push  +foo
```

makes ‘‘foo’’ the current folder, saving the folder that was previously the current folder on the *folder–stack*. As expected,

```
folder  –pop
```

takes the top of the *folder–stack* and makes it the current folder. Each of these switches lists the *folder–stack* when they execute. It is simple to write a *pushf* command as a shell script. It's one line:

```
exec folder –push $@
```

Probably a better way is to link *folder* to the $HOME/bin/ directory under the name of *pushf* and then add the entry

```
pushf: –push
```

to the .mh_profile.

The manual page for *folder* discusses the analogy between the *CShell* directory stack commands and the switches in *folder* which manipulate the *folder–stack*. The *folder* command uses the context entry 'Folder–Stack:' to keep track of the folders in the user's stack of folders.

Appendix A

COMMAND SUMMARY

ali [–alias aliasfile] [–list] [–nolist] [–normalize] [–nonormalize] [–user] [–nouser]
aliases ... [–help]

anno [+folder] [msgs] [–component field] [–inplace] [–noinplace] [–date] [–nodate]
[–text body] [–help]

bbc [bboards ...] [–topics] [–check] [–read] [–quiet] [–verbose] [–archive] [–noarchive]
[–protocol] [–noprotocol] [–mshproc program] [switches for *mshproc*]
[–rcfile rcfile] [–norcfile] [–file BBoardsfile] [–user BBoardsuser] [–host host]
[–help]

burst [+folder] [msgs] [–inplace] [–noinplace] [–quiet] [–noquiet] [–verbose]
[–noverbose] [–help]

comp [+folder] [msg] [–draftfolder +folder] [–draftmessage msg] [–nodraftfolder]
[–editor editor] [–noedit] [–file file] [–form formfile] [–use] [–nouse]
[–whatnowproc program] [–nowhatnowproc] [–help]

dist [+folder] [msg] [–annotate] [–noannotate] [–draftfolder +folder]
[–draftmessage msg] [–nodraftfolder] [–editor editor] [–noedit]
[–form formfile] [–inplace] [–noinplace] [–whatnowproc program]
[–nowhatnowproc] [–help]

/usr/bs/mh-beta/lib/fmtdump [–form formatfile] [–format string] [–help]

folder [+folder] [msg] [–all] [–fast] [–nofast] [–header] [–noheader] [–pack] [–nopack]
[–recurse] [–norecurse] [–total] [–nototal] [–print] [–noprint] [–list] [–nolist]
[–push] [–pop] [–help]

folders

forw [+folder] [msgs] [–annotate] [–noannotate] [–draftfolder +folder]
[–draftmessage msg] [–nodraftfolder] [–editor editor] [–noedit]
[–filter filterfile] [–form formfile] [–format] [–noformat] [–inplace]
[–noinplace] [–mime] [–nomime] [–whatnowproc program]
[–nowhatnowproc] [–help]

forw [+folder] [msgs] [–digest list] [–issue number] [–volume number]
[other switches for *forw*] [–help]

inc [+folder] [–audit audit–file] [–noaudit] [–changecur] [–nochangecur] [–file name]
[–form formatfile] [–format string] [–silent] [–nosilent] [–truncate]
[–notruncate] [–width columns] [–host host] [–user user] [–apop] [–noapop]
[–rpop] [–norpop] [–pack file] [–nopack] [–help]

mark [+folder] [msgs] [–sequence name ...] [–add] [–delete] [–list] [–public]
[–nopublic] [–zero] [–nozero] [–help]

/usr/bs/mh-beta/lib/mhl [–bell] [–nobell] [–clear] [–noclear] [–folder +folder]
[–form formfile] [–length lines] [–width columns] [–moreproc program]
[–nomoreproc] [files ...] [–help]

mhmail [addrs ... [–body text] [–cc addrs ...] [–from addr] [–subject subject]] [–help]

mhn [[+folder] [msgs] | [–file file]]
[–part number]... [–type content]...
[–list [–headers] [–noheaders]
[–realsize] [–norealsize]] [-nolist]
[–show [–serialonly] [–noserialonly]
[–form formfile] [–pause] [–nopause]] [–noshow]
[–store [–auto] [–noauto]] [–nostore]
[–cache] [–nocache] [–rcache policy] [–wcache policy]
[–check] [–nocheck]
[–ebcdicsafe] [–noebcdicsafe]
[–rfc934mode] [–norfc934mode]
[–verbose] [–noverbose]
[–help]

mhparam [profile-components] [–components] [–nocomponents] [–all] [–help]

mhpath [+folder] [msgs] [–help]

msgchk [–date] [–nodate] [–notify all/mail/nomail] [–nonotify all/mail/nomail]
[–host host] [–user user] [–apop] [–noapop] [–rpop] [–norpop] [users ...]
[–help]

msh [–prompt string] [–scan] [–noscan] [–topcur] [–notopcur] [file] [–help]

next [+folder] [–header] [–noheader] [–showproc program] [–noshowproc]
[switches for *showproc*] [–help]

packf [+folder] [msgs] [-file name] [–help]

pick	–cc	[+folder] [msgs] [–help]
	–date	[–before date] [–after date] [–datefield field]
	–from	
	–search	pattern [–and ...] [–or ...] [–not ...] [–lbrace ... –rbrace]
	–subject	
	–to	[–sequence name ...] [–public] [–nopublic] [–zero] [–nozero]
	––component	[–list] [–nolist]

prev [+folder] [–header] [–noheader] [–showproc program] [–noshowproc]
    [switches for *showproc*] [–help]

prompter [–erase chr] [–kill chr] [–prepend] [–noprepend] [–rapid] [–norapid]
    [–doteof] [–nodoteof] file [–help]

/usr/bs/mh-beta/lib/rcvstore [+folder] [–create] [–nocreate] [–sequence name ...]
    [–public] [–nopublic] [–zero] [–nozero] [–help]

refile [msgs] [–draft] [–link] [–nolink] [–preserve] [–nopreserve] [–src +folder]
    [–file file] +folder ... [–help]

repl [+folder] [msg] [–annotate] [–noannotate] [–cc all/to/cc/me] [–nocc all/to/cc/me]
    [–draftfolder +folder] [–draftmessage msg] [–nodraftfolder] [–editor editor]
    [–noedit] [–fcc +folder] [–filter filterfile] [–form formfile] [–inplace]
    [–noinplace] [–query] [–noquery] [–whatnowproc program]
    [–nowhatnowproc] [–width columns] [–help]

rmf [+folder] [–interactive] [–nointeractive] [–help]

rmm [+folder] [msgs] [–help]

scan [+folder] [msgs] [–clear] [–noclear] [–form formatfile] [–format string] [–header]
    [–noheader] [–width columns] [–reverse] [–noreverse] [–file filename]
    [–help]

send [–alias aliasfile] [–draft] [–draftfolder +folder] [–draftmessage msg]
    [–nodraftfolder] [–filter filterfile] [–nofilter] [–format] [–noformat] [–forward]
    [–noforward] [–mime] [–nomime] [–msgid] [–nomsgid] [–push] [–nopush]
    [–split seconds] [–verbose] [–noverbose] [–watch] [–nowatch]
    [–width columns] [file ...] [–help]

show [+folder] [msgs] [–draft] [–header] [–noheader] [–showproc program]
    [–noshowproc] [switches for *showproc*] [–help]

sortm [+folder] [msgs] [–datefield field] [–textfield field] [–notextfield] [–limit days]
    [–nolimit] [–verbose] [–noverbose] [–help]

vmh [–prompt string] [–vmhproc program] [–novmhproc] [switches for *vmhproc*]
    [–help]

whatnow [–draftfolder +folder] [–draftmessage msg] [–nodraftfolder] [–editor editor]
    [–noedit] [–prompt string] [file] [–help]

whom [–alias aliasfile] [–check] [–nocheck] [–draft] [–draftfolder +folder]
    [–draftmessage msg] [–nodraftfolder] [file] [–help]

/usr/bs/mh-beta/lib/ap [–form formatfile] [–format string] [–normalize] [–nonormalize]
[–width columns] addrs ... [–help]

/usr/bs/mh-beta/lib/conflict [–mail name] [–search directory] [aliasfiles ...] [–help]

/usr/bs/mh-beta/lib/dp [–form formatfile] [–format string] [–width columns] dates ...
[–help]

/usr/bs/mh-beta/lib/install–mh [–auto] [–compat]

/usr/bs/mh-beta/lib/post [–alias aliasfile] [–filter filterfile] [–nofilter] [–format]
[–noformat] [–mime] [–nomime] [–msgid] [–nomsgid] [–verbose]
[–noverbose] [–watch] [–nowatch] [–width columns] file [–help]

/usr/bs/mh-beta/lib/slocal [address info sender] [–addr address] [–info data]
[–sender sender] [–user username] [–mailbox mbox] [–file file]
[–maildelivery deliveryfile] [–verbose] [–noverbose] [–debug] [–help]

Appendix B

MESSAGE NAME BNF

```
msgs                   := msgspec                  |
                          msgs msgspec

msgspec                := msg                      |
                          msg-range                |
                          msg-sequence             |
                          user-defined-sequence

msg                    := msg-name                 |
                          <number>

msg-name               := ‘‘first’’                |
                          ‘‘last’’                 |
                          ‘‘cur’’                  |
                          ‘‘.’’                    |
                          ‘‘next’’                 |
                          ‘‘prev’’

msg-range              := msg‘‘–’’msg              |
                          ‘‘all’’

msg-sequence           := msg‘‘:’’signed-number

signed-number          := ‘‘+’’<number>            |
                          ‘‘–’’<number>            |
                          <number>

user-defined-sequence  := <alpha>                  |
                          <alpha><alphanumeric>*
```

Where <number> is a decimal number greater than zero.

Msg-range specifies all of the messages in the given range and must not be empty.

Msg-sequence specifies up to <number> of messages, beginning with ‘‘msg’’ (in the case of first, cur, next, or <number>), or ending with ‘‘msg’’ (in the case of prev or last). +<number> forces ‘‘starting with msg’’, and –<number> forces ‘‘ending with number’’. In all cases, ‘‘msg’’ must exist.

User–defined sequences are defined and manipulated with the *pick* and *mark* commands.

REFERENCES

1. Crocker, D. H., J. J. Vittal, K. T. Pogran, and D. A. Henderson, Jr., ‘‘Standard for the Format of ARPA Network Text Messages,’’ *RFC733*, November 1977.

2. Thompson, K., and D. M. Ritchie, ‘‘The UNIX Time-sharing System,’’ *Communications of the ACM*, Vol. 17, July 1974, pp. 365-375.

3. McCauley, E. J., and P. J. Drongowski, ‘‘KSOS–The Design of a Secure Operating System,’’ *AFIPS Conference Proceedings*, National Computer Conference, Vol. 48, 1979, pp. 345-353.

4. Crocker, David H., *Framework and Functions of the ‘‘MS’’ Personal Message System*, The RAND Corporation, R-2134-ARPA, December 1977.

5. Thompson, K., and D. M. Ritchie, *UNIX Programmer’s Manual*, 6th ed., Western Electric Company, May 1975 (available only to UNIX licensees).

6. Crocker, D. H., ‘‘Standard for the Format of ARPA Internet Text Messages,’’ *RFC822*, August 1982.

READ THIS

Although the *MH* system was originally developed by the RAND Corporation, and is now in the public domain, the RAND Corporation assumes no responsibility for *MH* or this particular version of *MH*.

In addition, the Regents of the University of California issue the following **disclaimer** in regard to the UCI version of *MH*:

> ‘‘Although each program has been tested by its contributor, no warranty, express or implied, is made by the contributor or the University of California, as to the accuracy and functioning of the program and related program material, nor shall the fact of distribution constitute any such warranty, and no responsibility is assumed by the contributor or the University of California in connection herewith.’’

This version of *MH* is in the public domain, and as such, there are no real restrictions on its use. The *MH* source code and documentation have no licensing restrictions whatsoever. As a courtesy, the authors ask only that you provide appropriate credit to the RAND Corporation and the University of California for having developed the software.

MH is a software package that is supported neither by the RAND Corporation nor the University of California. However, since we do use the software ourselves and plan to continue using (and improving) *MH*, bug reports and their associated fixes should be reported back to us so that we may include them in future releases. The current computer mailbox for *MH* is **Bug–MH@ICS.UCI.EDU** (in the ARPA Internet), and **...!ucbvax!ucivax!bug–mh** (UUCP). Presently, there are two Internet discussion groups, **MH–Users@ICS.UCI.EDU** and **MH–Workers@ICS.UCI.EDU**. **MH–Workers** is for people discussing code changes to *MH*. **MH-Users** is for general discussion about how to use *MH*. **MH–Users** is bi-directionally gatewayed into USENET as **comp.mail.mh**.

OBTAINING MH

Since you probably already have *MH*, you may not need to read this unless you suspect you have an old version. On the Internet, you may retrieve the lastest release from one of:

> ftp://ftp.ics.uci.edu/pub/mh/mh-6.8.tar.Z
> ftp://ftp.uu.uunet/networking/mail/mh/mh-6.8.tar.Z
> http://wuarchive.wustl.edu/packages/mail/mh/mh-6.8.tar.Z

This is a tar image after being run through the compress program (approximately 2Mb). There is also an *MH* World Wide Web Home Page at ‘‘http://www.ics.uci.edu/˜mh’’ which tells what the current release of *MH* is, and how to get updates.

You may also find MH on various other hosts; to make sure you get the latest version and don’t waste your time re-fixing bugs, it’s best to get it from either ftp.ics.uci.edu or a site which mirrors ftp.ics.uci.edu.

Alternatively, you can send $75 US to the address below. This covers the cost of a 6250 BPI 9-track magtape, handling, and shipping. In addition, you’ll get a laser-printed hard-copy of the entire MH documentation set. Be sure to include your USPS address with your check. Checks must be drawn on U.S. funds and should be made payable to:

> Regents of the University of California

The distribution address is:

> Attn: MH distribution
> Office of Academic Computing
> University of California, Irvine
> Irvine, CA 92717-2225
>
> +1 714 824 5153

If you just want the hard-copies of the documentation, you still have to pay the $75. The tar image has the documentation source (the manual is in roff format, but the rest are in TeX format). Postscript formatted versions of the TeX papers are available, as are crude tty-conversions of those papers.

FOREWORD

This document describes the RAND *MH* Message Handling System. Its primary purpose is to serve as a user's manual. It has been heavily based on a previous version of the manual, prepared by Bruce Borden, Stockton Gaines, and Norman Shapiro.

MH is a particularly novel system, and thus it is often more prone to change than other pieces of production software. As such, some specific points in this manual may not be correct in the future. In all cases, the on–line sections of this manual, available through the UNIX[1] *man* command, should present the most current information.

When reading this document as a user's manual, certain sections are more interesting than others. The Preface and Summary are not particularly interesting to those interested in learning *MH*. The Introduction is slightly more interesting, as it touches upon the organization of the remainder of this document. The most useful sections are the Overview, Tutorial, and Detailed Description. The Overview should be read by all *MH* users, regardless of their expertise (beginning, novice, advanced, or hacker). The Tutorial should be read by all beginning and novice *MH* users, as it presents a nice description of the *MH* system. The Detailed Description should be read by the day–to–day user of *MH*, as it spells out all of the realities of the *MH* system. The Advanced Features section discusses some powerful *MH* capabilities for advanced users. Appendix A is particularly useful for novices, as it summarizes the invocation syntax of all the *MH* commands.

There are also several other documents which may be useful to you: *The RAND MH Message Handling System: Tutorial*, which is a tutorial for *MH*; *The RAND MH Message Handling System: The UCI BBoards Facility*, which describes the BBoards handling under *MH*; *MH.5: How to process 200 messages a day and still get some real work done*, which was presented at the 1985 Summer Usenix Conference and Exhibition in Portland, Oregon; *MH: A Multifarious User Agent*, which has been accepted for publication by Computer Networks; *MZnet: Mail Service for Personal Micro–Computer Systems*, which was presented at the First International Symposium on Computer Message Systems in Nottingham, U.K.; and, *Design of the TTI Prototype Trusted Mail Agent*, which describes a proprietary ''trusted'' mail system built on *MH*. There are also documents, mostly specific to U.C. Irvine which you may find interesting: *MH for Beginners*, and *MH for MM Users*. All of these documents exist in the *mh.6* distribution sent to your site. There's also a document, *Changes to the RAND MH Message Handling System: MH.6*, which describes user–visible changes made to *MH* since the last major release.

This manual is very large, as it describes a large, powerful system in gruesome detail. The important thing to remember is:

DON'T PANIC[2]

As explained in the tutorial, you really need to know only 5 commands to handle most of your mail.

Very advanced users may wish to consult *The RAND MH Message Handling System: Administrator's Guide*, which is also present in the *mh.6* distribution sent to your site.

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

[2] Note the large, *friendly* letters.

ACKNOWLEDGMENTS

The *MH* system described herein is based on the original RAND *MH* system. It has been extensively developed (perhaps too much so) by Marshall T. Rose and John L. Romine at the University of California, Irvine. Einar A. Stefferud, Jerry N. Sweet, and Terry P. Domae provided numerous suggestions to improve the UCI version of *MH*. Of course, a large number of people have helped *MH* along. The list of ‘‘*MH* immortals’’ is too long to list here. However, Van Jacobson deserves a special acknowledgement for his tireless work in improving the performance of *MH*. Some programs have been speeded-up by a factor of 10 or 20. All of users of *MH*, everywhere, owe a special thanks to Van. For this release, numerous *MH–Workers* sent in fixes and other changes. A handful of courageous *MH–Workers* volunteered to beta–test these changes; their help is particularly appreciated.

This manual is based on the original *MH* manual written at RAND by Bruce Borden, Stockton Gaines, and Norman Shapiro.

PREFACE

This report describes a system for dealing with messages transmitted on a computer. Such messages might originate with other users of the same computer or might come from an outside source through a network to which the user's computer is connected. Such computer-based message systems are becoming increasingly widely used, both within and outside the Department of Defense.

The message handling system *MH* was developed for two reasons. One was to investigate some research ideas concerning how a message system might take advantage of the architecture of the UNIX time-sharing operating system for Digital Equipment Corporation PDP-11 and VAX computers, and the special features of UNIX's command-level interface with the user (the ‘‘shell’’). The other reason was to provide a better and more adaptable base than that of conventional designs on which to build a command and control message system. The effort has succeeded in both regards, although this report mainly describes the message system itself and how it fits in with UNIX.

The present report should be of interest to three groups of readers. First, it is a complete reference manual for the users of *MH*. Second, it should be of interest to those who have a general knowledge of computer-based message systems, both in civilian and military applications. Finally, it should be of interest to those who build large subsystems that interface with users, since it illustrates a new approach to such interfaces.

The original *MH* system was developed by Bruce Borden, using an approach suggested by Stockton Gaines and Norman Shapiro. Valuable assistance was provided by Phyllis Kantar in the later stages of the system's implementation. Several colleagues contributed to the ideas included in this system, particularly Robert Anderson and David Crocker. In addition, valuable experience in message systems, and a valuable source of ideas, was available to us in the form of a previous message system for UNIX called MS, designed at RAND by David Crocker.

This report was originally prepared as part of the RAND project entitled ‘‘Data Automation Research’’, sponsored by Project AIR FORCE.

SUMMARY

Electronic communication of text messages is becoming commonplace. Computer-based message systems–software packages that provide tools for dealing with messages–are used in many contexts. In particular, message systems are becoming increasingly important in command and control and intelligence applications.

This report describes a message handling system called *MH*. This system provides the user with tools to compose, send, receive, store, retrieve, forward, and reply to messages. *MH* has been built on the UNIX time-sharing system, a popular operating system developed for the DEC PDP-11 and VAX classes of computers.

A complete description of *MH* is given for users of the system. For those who do not intend to use the system, this description gives a general idea of what a message system is like. The system involves some new ideas about how large subsystems can be constructed.

The interesting and unusual features of *MH* include the following: The user command interface to *MH* is the UNIX ‘‘shell’’ (the standard UNIX command interpreter). Each separable component of message handling, such as message composition or message display, is a separate command. Each program is driven from and updates a private user environment, which is stored as a file between program invocations. This private environment also contains information to ‘‘custom tailor’’ *MH* to the individual’s tastes. *MH* stores each message as a separate file under UNIX, and it utilizes the tree-structured UNIX file system to organize groups of files within separate directories or ‘‘folders’’. All of the UNIX facilities for dealing with files and directories, such as renaming, copying, deleting, cataloging, off-line printing, etc., are applicable to messages and directories of messages (folders). Thus, important capabilities needed in a message system are available in *MH* without the need (often seen in other message systems) for code that duplicates the facilities of the supporting operating system. It also allows users familiar with the shell to use *MH* with minimal effort.

CONTENTS

THE RAND MH MESSAGE HANDLING SYSTEM: USER'S MANUAL

UCI Version

Marshall T. Rose
John L. Romine

Based on the original manual by
Borden, Gaines, and Shapiro

December 5, 1995
6.8.3a #10[UCI]