The Mutt E-Mail Client

Michael Elkins

<me@cs.hmc.edu>

version 1.5.22 (2013-10-16)

Abstract

“All mail clients suck. This one just sucks less.” — me, circa 1995

Table of Contents

List of Tables

List of Examples

6.2. Managing multiple accounts

Chapter 1. Introduction

Table of Contents

Mutt is a small but very powerful text-based MIME mail client. Mutt is highly configurable, and is well suited to the mail power user with advanced features like key bindings, keyboard macros, mail threading, regular expression searches and a powerful pattern matching language for selecting groups of messages.

1. Mutt Home Page

The official homepage can be found at http://www.mutt.org/.

2. Mailing Lists

To subscribe to one of the following mailing lists, send a message with the word *subscribe* in the body to *list-name*`-request@mutt.org`.

- <`mutt-announce-request@mutt.org`> — low traffic list for announcements
- <`mutt-users-request@mutt.org`> — help, bug reports and feature requests
- <`mutt-dev-request@mutt.org`> — development mailing list

All messages posted to *mutt-announce* are automatically forwarded to *mutt-users*, so you do not need to be subscribed to both lists.

3. Getting Mutt

Mutt releases can be downloaded from ftp://ftp.mutt.org/mutt/. For a list of mirror sites, please refer to http://www.mutt.org/download.html.

For nightly tarballs and version control access, please refer to the Mutt development site.

4. Mutt Online Resources

Bug Tracking System

The official Mutt bug tracking system can be found at http://bugs.mutt.org/

Wiki

An (unofficial) wiki can be found at http://wiki.mutt.org/.

IRC

For the IRC user community, visit channel *#mutt* on irc.freenode.net.

USENET

For USENET, see the newsgroup comp.mail.mutt.

5. Contributing to Mutt

There are various ways to contribute to the Mutt project.

Especially for new users it may be helpful to meet other new and experienced users to chat about Mutt, talk about problems and share tricks.

Since translations of Mutt into other languages are highly appreciated, the Mutt developers always look for skilled translators that help improve and continue to maintain stale translations.

For contributing code patches for new features and bug fixes, please refer to the developer pages at http://dev.mutt.org/ for more details.

6. Typographical Conventions

This section lists typographical conventions followed throughout this manual. See table Table 1.1, "Typographical conventions for special terms" for typographical conventions for special terms.

Table 1.1. Typographical conventions for special terms

Item	Refers to...
`printf(3)`	UNIX manual pages, execute `man 3 printf`
`<PageUp>`	named keys
`<create-alias>`	named Mutt function
`^G`	Control+G key combination
$mail_check	Mutt configuration option
`$HOME`	environment variable

Examples are presented as:

```
mutt -v
```

Within command synopsis, curly brackets ("{}") denote a set of options of which one is mandatory, square brackets ("[]") denote optional arguments, three dots denote that the argument may be repeated arbitrary times.

7. Copyright

Chapter 2. Getting Started

Table of Contents

This section is intended as a brief overview of how to use Mutt. There are many other features which are described elsewhere in the manual. There is even more information available in the Mutt FAQ and various web pages. See the Mutt homepage for more details.

The keybindings described in this section are the defaults as distributed. Your local system administrator may have altered the defaults for your site. You can always type "?" in any menu to display the current bindings.

The first thing you need to do is invoke Mutt, simply by typing `mutt` at the command line. There are various command-line options, see either the Mutt man page or the reference.

1. Core Concepts

Mutt is a text-based application which interacts with users through different menus which are mostly line-/entry-based or page-based. A line-based menu is the so-called "index" menu (listing all messages of the currently opened folder) or the "alias" menu (allowing you to select recipients from a list). Examples for page-based menus are the "pager" (showing one message at a time) or the "help" menu listing all available key bindings.

The user interface consists of a context sensitive help line at the top, the menu's contents followed by a context sensitive status line and finally the command line. The command line is used to display informational and error messages as well as for prompts and for entering interactive commands.

Mutt is configured through variables which, if the user wants to permanently use a non-default value, are written to configuration files. Mutt supports a rich config file syntax to make even complex configuration files readable and commentable.

Because Mutt allows for customizing almost all key bindings, there are so-called "functions" which can be executed manually (using the command line) or in macros. Macros allow the user to bind a sequence of commands to a single key or a short key sequence instead of repeating a sequence of actions over and over.

Many commands (such as saving or copying a message to another folder) can be applied to a single message or a set of messages (so-called “tagged” messages). To help selecting messages, Mutt provides a rich set of message patterns (such as recipients, sender, body contents, date sent/received, etc.) which can be combined into complex expressions using the boolean *and* and *or* operations as well as negating. These patterns can also be used to (for example) search for messages or to limit the index to show only matching messages.

Mutt supports a “hook” concept which allows the user to execute arbitrary configuration commands and functions in certain situations such as entering a folder, starting a new message or replying to an existing one. These hooks can be used to highly customize Mutt's behavior including managing multiple identities, customizing the display for a folder or even implementing auto-archiving based on a per-folder basis and much more.

Besides an interactive mode, Mutt can also be used as a command-line tool only send messages. It also supports a `mailx(1)`-compatible interface, see Table 9.1, “Command line options” for a complete list of command-line options.

2. Screens and Menus

2.1. Index

The index is the screen that you usually see first when you start Mutt. It gives an overview over your emails in the currently opened mailbox. By default, this is your system mailbox. The information you see in the index is a list of emails, each with its number on the left, its flags (new email, important email, email that has been forwarded or replied to, tagged email, ...), the date when email was sent, its sender, the email size, and the subject. Additionally, the index also shows thread hierarchies: when you reply to an email, and the other person replies back, you can see the other person's email in a "sub-tree" below. This is especially useful for personal email between a group of people or when you've subscribed to mailing lists.

2.2. Pager

The pager is responsible for showing the email content. On the top of the pager you have an overview over the most important email headers like the sender, the recipient, the subject, and much more information. How much information you actually see depends on your configuration, which we'll describe below.

Below the headers, you see the email body which usually contains the message. If the email contains any attachments, you will see more information about them below the email body, or, if the attachments are text files, you can view them directly in the pager.

To give the user a good overview, it is possible to configure Mutt to show different things in the pager with different colors. Virtually everything that can be described with a regular expression can be colored, e.g. URLs, email addresses or smileys.

2.3. File Browser

The file browser is the interface to the local or remote file system. When selecting a mailbox to open, the

browser allows custom sorting of items, limiting the items shown by a regular expression and a freely adjustable format of what to display in which way. It also allows for easy navigation through the file system when selecting file(s) to attach to a message, select multiple files to attach and many more.

2.4. Help

The help screen is meant to offer a quick help to the user. It lists the current configuration of key bindings and their associated commands including a short description, and currently unbound functions that still need to be associated with a key binding (or alternatively, they can be called via the Mutt command prompt).

2.5. Compose Menu

The compose menu features a split screen containing the information which really matter before actually sending a message by mail: who gets the message as what (recipients and who gets what kind of copy). Additionally, users may set security options like deciding whether to sign, encrypt or sign and encrypt a message with/for what keys. Also, it's used to attach messages, to re-edit any attachment including the message itself.

2.6. Alias Menu

The alias menu is used to help users finding the recipients of messages. For users who need to contact many people, there's no need to remember addresses or names completely because it allows for searching, too. The alias mechanism and thus the alias menu also features grouping several addresses by a shorter nickname, the actual alias, so that users don't have to select each single recipient manually.

2.7. Attachment Menu

As will be later discussed in detail, Mutt features a good and stable MIME implementation, that is, it supports sending and receiving messages of arbitrary MIME types. The attachment menu displays a message's structure in detail: what content parts are attached to which parent part (which gives a true tree structure), which type is of what type and what size. Single parts may saved, deleted or modified to offer great and easy access to message's internals.

3. Moving Around in Menus

The most important navigation keys common to line- or entry-based menus are shown in Table 2.1, “Most common navigation keys in entry-based menus” and in Table 2.2, “Most common navigation keys in page-based menus” for page-based menus.

Table 2.1. Most common navigation keys in entry-based menus

Key	Function	Description
j or <Down>	`<next-entry>`	move to the next entry
	`<previous-entry>`	

k or <Up>		move to the previous entry
z or <PageDn>	`<page-down>`	go to the next page
Z or <PageUp>	`<page-up>`	go to the previous page
= or <Home>	`<first-entry>`	jump to the first entry
* or <End>	`<last-entry>`	jump to the last entry
q	`<quit>`	exit the current menu
?	`<help>`	list all keybindings for the current menu

Table 2.2. Most common navigation keys in page-based menus

Key	Function	Description
J or <Return>	`<next-line>`	scroll down one line
<Backspace>	`<previous-line>`	scroll up one line
K, <Space> or <PageDn>	`<next-page>`	move to the next page
- or <PageUp>	`<previous-page>`	move the previous page
<Home>	`<top>`	move to the top
<End>	`<bottom>`	move to the bottom

4. Editing Input Fields

4.1. Introduction

Mutt has a built-in line editor for inputting text, e.g. email addresses or filenames. The keys used to manipulate text input are very similar to those of Emacs. See Table 2.3, “Most common line editor keys” for a full reference of available functions, their default key bindings, and short descriptions.

Table 2.3. Most common line editor keys

Key	Function	Description
^A or <Home>	`<bol>`	move to the start of the line
^B or <Left>	`<backward-char>`	move back one char
Esc B	`<backward-word>`	move back one word
^D or <Delete>	`<delete-char>`	delete the char under the cursor

^E or <End>	`<eol>`	move to the end of the line
^F or <Right>	`<forward-char>`	move forward one char
Esc F	`<forward-word>`	move forward one word
<Tab>	`<complete>`	complete filename or alias
^T	`<complete-query>`	complete address with query
^K	`<kill-eol>`	delete to the end of the line
Esc d	`<kill-eow>`	delete to the end of the word
^W	`<kill-word>`	kill the word in front of the cursor
^U	`<kill-line>`	delete entire line
^V	`<quote-char>`	quote the next typed key
<Up>	`<history-up>`	recall previous string from history
<Down>	`<history-down>`	recall next string from history
<BackSpace>	`<backspace>`	kill the char in front of the cursor
Esc u	`<upcase-word>`	convert word to upper case
Esc l	`<downcase-word>`	convert word to lower case
Esc c	`<capitalize-word>`	capitalize the word
^G	n/a	abort
<Return>	n/a	finish editing

You can remap the *editor* functions using the `bind` command. For example, to make the <Delete> key delete the character in front of the cursor rather than under, you could use:

```
bind editor <delete> backspace
```

4.2. History

Mutt maintains a history for the built-in editor. The number of items is controlled by the $history variable and can be made persistent using an external file specified using $history_file. You may cycle through them at an editor prompt by using the `<history-up>` and/or `<history-down>` commands. But notice that Mutt does not remember the currently entered text, it only cycles through history and wraps around at the end or beginning.

Mutt maintains several distinct history lists, one for each of the following categories:

- `.muttrc` commands
- addresses and aliases
- shell commands
- filenames
- patterns
- everything else

Mutt automatically filters out consecutively repeated items from the history. It also mimics the behavior of some shells by ignoring items starting with a space. The latter feature can be useful in macros to not clobber the history's valuable entries with unwanted entries.

5. Reading Mail

Similar to many other mail clients, there are two modes in which mail is read in Mutt. The first is a list of messages in the mailbox, which is called the "index" menu in Mutt. The second mode is the display of the message contents. This is called the "pager."

The next few sections describe the functions provided in each of these modes.

5.1. The Message Index

Common keys used to navigate through and manage messages in the index are shown in Table 2.4, "Most common message index keys". How messages are presented in the index menu can be customized using the $index_format variable.

Table 2.4. Most common message index keys

Key	Description
c	change to a different mailbox
Esc c	change to a folder in read-only mode
C	copy the current message to another mailbox
Esc C	decode a message and copy it to a folder
Esc s	decode a message and save it to a folder
D	delete messages matching a pattern
d	delete the current message
F	mark as important

l	show messages matching a pattern
N	mark message as new
o	change the current sort method
O	reverse sort the mailbox
q	save changes and exit
s	save-message
T	tag messages matching a pattern
t	toggle the tag on a message
Esc t	toggle tag on entire message thread
U	undelete messages matching a pattern
u	undelete-message
v	view-attachments
x	abort changes and exit
<Return>	display-message
<Tab>	jump to the next new or unread message
@	show the author's full e-mail address
$	save changes to mailbox
/	search
Esc /	search-reverse
^L	clear and redraw the screen
^T	untag messages matching a pattern

In addition to who sent the message and the subject, a short summary of the disposition of each message is printed beside the message number. Zero or more of the "flags" in Table 2.5, "Message status flags" may appear, some of which can be turned on or off using these functions: `<set-flag>` and `<clear-flag>` bound by default to "w" and "W" respectively.

Furthermore, the flags in Table 2.6, "Message recipient flags" reflect who the message is addressed to. They can be customized with the $to_chars variable.

Table 2.5. Message status flags

Flag	Description

D	message is deleted (is marked for deletion)
d	message has attachments marked for deletion
K	contains a PGP public key
N	message is new
O	message is old
P	message is PGP encrypted
r	message has been replied to
S	message is signed, and the signature is successfully verified
s	message is signed
!	message is flagged
*	message is tagged
n	thread contains new messages (only if collapsed)
o	thread contains old messages (only if collapsed)

Table 2.6. Message recipient flags

Flag	Description
+	message is to you and you only
T	message is to you, but also to or CC'ed to others
C	message is CC'ed to you
F	message is from you
L	message is sent to a subscribed mailing list

5.2. The Pager

By default, Mutt uses its built-in pager to display the contents of messages (an external pager such as `less(1)` can be configured, see $pager variable). The pager is very similar to the Unix program `less(1)` though not nearly as featureful.

Table 2.7. Most common pager keys

Key	Description
<Return>	go down one line

<Space>	display the next page (or next message if at the end of a message)
-	go back to the previous page
n	search for next match
S	skip beyond quoted text
T	toggle display of quoted text
?	show keybindings
/	regular expression search
Esc /	backward regular expression search
\	toggle highlighting of search matches
^	jump to the top of the message

In addition to key bindings in Table 2.7, "Most common pager keys", many of the functions from the index menu are also available in the pager, such as `<delete-message>` or `<copy-message>` (this is one advantage over using an external pager to view messages).

Also, the internal pager supports a couple other advanced features. For one, it will accept and translate the "standard" nroff sequences for bold and underline. These sequences are a series of either the letter, backspace ("^H"), the letter again for bold or the letter, backspace, "_" for denoting underline. Mutt will attempt to display these in bold and underline respectively if your terminal supports them. If not, you can use the bold and underline color objects to specify a `color` or mono attribute for them.

Additionally, the internal pager supports the ANSI escape sequences for character attributes. Mutt translates them into the correct color and character settings. The sequences Mutt supports are:

```
\e[Ps;Ps;..Ps;m
```

where *Ps* can be one of the codes shown in Table 2.8, "ANSI escape sequences".

Table 2.8. ANSI escape sequences

Escape code	Description
0	All attributes off
1	Bold on
4	Underline on
5	Blink on
7	Reverse video on

3<*color*>	Foreground color is <*color*> (see Table 2.9, "Color sequences")
4<*color*>	Background color is <*color*> (see Table 2.9, "Color sequences")

Table 2.9. Color sequences

Color code	Color
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

Mutt uses these attributes for handling `text/enriched` messages, and they can also be used by an external autoview script for highlighting purposes.

> ***Note:***
>
> If you change the colors for your display, for example by changing the color associated with color2 for your xterm, then that color will be used instead of green.

> ***Note:***
>
> Note that the search commands in the pager take regular expressions, which are not quite the same as the more complex patterns used by the search command in the index. This is because patterns are used to select messages by criteria whereas the pager already displays a selected message.

5.3. Threaded Mode

So-called "threads" provide a hierarchy of messages where replies are linked to their parent message(s). This organizational form is extremely useful in mailing lists where different parts of the discussion diverge. Mutt displays threads as a tree structure.

In Mutt, when a mailbox is sorted by *threads*, there are a few additional functions available in the *index* and

pager modes as shown in Table 2.10, "Most common thread mode keys".

Table 2.10. Most common thread mode keys

Key	Function	Description
^D	`<delete-thread>`	delete all messages in the current thread
^U	`<undelete-thread>`	undelete all messages in the current thread
^N	`<next-thread>`	jump to the start of the next thread
^P	`<previous-thread>`	jump to the start of the previous thread
^R	`<read-thread>`	mark the current thread as read
Esc d	`<delete-subthread>`	delete all messages in the current subthread
Esc u	`<undelete-subthread>`	undelete all messages in the current subthread
Esc n	`<next-subthread>`	jump to the start of the next subthread
Esc p	`<previous-subthread>`	jump to the start of the previous subthread
Esc r	`<read-subthread>`	mark the current subthread as read
Esc t	`<tag-thread>`	toggle the tag on the current thread
Esc v	`<collapse-thread>`	toggle collapse for the current thread
Esc V	`<collapse-all>`	toggle collapse for all threads
P	`<parent-message>`	jump to parent message in thread

Collapsing a thread displays only the first message in the thread and hides the others. This is useful when threads contain so many messages that you can only see a handful of threads on the screen. See %M in $index_format. For example, you could use "%?M?(#%03M)&(%4l)?" in $index_format to optionally display the number of hidden messages if the thread is collapsed. The `%?<char>?<if-part>&<else-part>?` syntax is explained in detail in format string conditionals.

Technically, every reply should contain a list of its parent messages in the thread tree, but not all do. In these cases, Mutt groups them by subject which can be controlled using the $strict_threads variable.

5.4. Miscellaneous Functions

In addition, the *index* and *pager* menus have these interesting functions:

`<create-alias>` (default: a)

> Creates a new alias based upon the current message (or prompts for a new one). Once editing is complete, an `alias` command is added to the file specified by the $alias_file variable for future use

> ***Note:***
>
> Mutt does not read the $alias_file upon startup so you must explicitly `source` the file.

`<check-traditional-pgp>` (default: Esc P)

This function will search the current message for content signed or encrypted with PGP the "traditional" way, that is, without proper MIME tagging. Technically, this function will temporarily change the MIME content types of the body parts containing PGP data; this is similar to the `<edit-type>` function's effect.

`<edit>` (default: e)

This command (available in the index and pager) allows you to edit the raw current message as it's present in the mail folder. After you have finished editing, the changed message will be appended to the current folder, and the original message will be marked for deletion; if the message is unchanged it won't be replaced.

`<edit-type>` (default: ^E on the attachment menu, and in the pager and index menus; ^T on the compose menu)

This command is used to temporarily edit an attachment's content type to fix, for instance, bogus character set parameters. When invoked from the index or from the pager, you'll have the opportunity to edit the top-level attachment's content type. On the attachment menu, you can change any attachment's content type. These changes are not persistent, and get lost upon changing folders.

Note that this command is also available on the compose menu. There, it's used to fine-tune the properties of attachments you are going to send.

`<enter-command>` (default: ":")

This command is used to execute any command you would normally put in a configuration file. A common use is to check the settings of variables, or in conjunction with macros to change settings on the fly.

`<extract-keys>` (default: ^K)

This command extracts PGP public keys from the current or tagged message(s) and adds them to your PGP public key ring.

`<forget-passphrase>` (default: ^F)

This command wipes the passphrase(s) from memory. It is useful, if you misspelled the passphrase.

`<list-reply>` (default: L)

Reply to the current or tagged message(s) by extracting any addresses which match the regular

expressions given by the `lists or subscribe` commands, but also honor any `Mail-Followup-To` header(s) if the $honor_followup_to configuration variable is set. In addition, the `List-Post` header field is examined for `mailto:` URLs specifying a mailing list address. Using this when replying to messages posted to mailing lists helps avoid duplicate copies being sent to the author of the message you are replying to.

`<pipe-message>` (default: |)

Asks for an external Unix command and pipes the current or tagged message(s) to it. The variables $pipe_decode, $pipe_split, $pipe_sep and $wait_key control the exact behavior of this function.

`<resend-message>` (default: Esc e)

Mutt takes the current message as a template for a new message. This function is best described as "recall from arbitrary folders". It can conveniently be used to forward MIME messages while preserving the original mail structure. Note that the amount of headers included here depends on the value of the $weed variable.

This function is also available from the attachment menu. You can use this to easily resend a message which was included with a bounce message as a `message/rfc822` body part.

`<shell-escape>` (default: !)

Asks for an external Unix command and executes it. The $wait_key can be used to control whether Mutt will wait for a key to be pressed when the command returns (presumably to let the user read the output of the command), based on the return status of the named command. If no command is given, an interactive shell is executed.

`<toggle-quoted>` (default: T)

The pager uses the $quote_regexp variable to detect quoted text when displaying the body of the message. This function toggles the display of the quoted material in the message. It is particularly useful when being interested in just the response and there is a large amount of quoted text in the way.

`<skip-quoted>` (default: S)

This function will go to the next line of non-quoted text which comes after a line of quoted text in the internal pager.

6. Sending Mail

6.1. Introduction

The bindings shown in Table 2.11, “Most common mail sending keys” are available in the *index* and *pager* to start a new message.

Table 2.11. Most common mail sending keys

Key	Function	Description
m	`<compose>`	compose a new message
r	`<reply>`	reply to sender
g	`<group-reply>`	reply to all recipients
L	`<list-reply>`	reply to mailing list address
f	`<forward>`	forward message
b	`<bounce>`	bounce (remail) message
Esc k	`<mail-key>`	mail a PGP public key to someone

Bouncing a message sends the message as-is to the recipient you specify. *Forwarding* a message allows you to add comments or modify the message you are forwarding. These items are discussed in greater detail in the next section "Forwarding and Bouncing Mail."

Mutt will then enter the *compose* menu and prompt you for the recipients to place on the "To:" header field when you hit `m` to start a new message. Next, it will ask you for the "Subject:" field for the message, providing a default if you are replying to or forwarding a message. You again have the chance to adjust recipients, subject, and security settings right before actually sending the message. See also $askcc, $askbcc, $autoedit, $bounce, $fast_reply, and $include for changing how and if Mutt asks these questions.

When replying, Mutt fills these fields with proper values depending on the reply type. The types of replying supported are:

Simple reply

> Reply to the author directly.

Group reply

> Reply to the author as well to all recipients except you; this consults `alternates`.

List reply

> Reply to all mailing list addresses found, either specified via configuration or auto-detected. See Section 12, "Mailing Lists" for details.

After getting recipients for new messages, forwards or replies, Mutt will then automatically start your $editor on the message body. If the $edit_headers variable is set, the headers will be at the top of the message in your editor. Any messages you are replying to will be added in sort order to the message, with appropriate $attribution, $indent_string and $post_indent_string. When forwarding a message, if the $mime_forward variable is unset, a copy of the forwarded message will be included. If you have specified a $signature, it will be appended to the message.

Once you have finished editing the body of your mail message, you are returned to the *compose* menu providing the functions shown in Table 2.12, "Most common compose menu keys" to modify, send or postpone the message.

Table 2.12. Most common compose menu keys

Key	Function	Description
a	`<attach-file>`	attach a file
A	`<attach-message>`	attach message(s) to the message
Esc k	`<attach-key>`	attach a PGP public key
d	`<edit-description>`	edit description on attachment
D	`<detach-file>`	detach a file
t	`<edit-to>`	edit the To field
Esc f	`<edit-from>`	edit the From field
r	`<edit-reply-to>`	edit the Reply-To field
c	`<edit-cc>`	edit the Cc field
b	`<edit-bcc>`	edit the Bcc field
y	`<send-message>`	send the message
s	`<edit-subject>`	edit the Subject
S	`<smime-menu>`	select S/MIME options
f	`<edit-fcc>`	specify an "Fcc" mailbox
p	`<pgp-menu>`	select PGP options
P	`<postpone-message>`	postpone this message until later
q	`<quit>`	quit (abort) sending the message
w	`<write-fcc>`	write the message to a folder
i	`<ispell>`	check spelling (if available on your system)
^F	`<forget-passphrase>`	wipe passphrase(s) from memory

The compose menu is also used to edit the attachments for a message which can be either files or other messages. The `<attach-message>` function to will prompt you for a folder to attach messages from. You can now tag messages in that folder and they will be attached to the message you are sending.

NOTE:

Note that certain operations like composing a new mail, replying, forwarding, etc. are not permitted when you are in that folder. The %r in $status_format will change to a "A" to indicate that you are in attach-message mode.

6.2. Editing the Message Header

When editing the header because of $edit_headers being set, there are a several pseudo headers available which will not be included in sent messages but trigger special Mutt behavior.

6.2.1. Fcc: Pseudo Header

If you specify

`Fcc:` *filename*

as a header, Mutt will pick up *filename* just as if you had used the `<edit-fcc>` function in the *compose* menu. It can later be changed from the compose menu.

6.2.2. Attach: Pseudo Header

You can also attach files to your message by specifying

`Attach:` *filename* [*description*]

where *filename* is the file to attach and *description* is an optional string to use as the description of the attached file. Spaces in filenames have to be escaped using backslash ("\"). The file can be removed as well as more added from the compose menu.

6.2.3. Pgp: Pseudo Header

If you want to use PGP, you can specify

`Pgp:` [`E` | `S` | `S`<*id*>]

"E" selects encryption, "S" selects signing and "S<id>" selects signing with the given key, setting $pgp_sign_as permanently. The selection can later be changed in the compose menu.

6.2.4. In-Reply-To: Header

When replying to messages, the *In-Reply-To:* header contains the Message-Id of the message(s) you reply to. If you remove or modify its value, Mutt will not generate a *References:* field, which allows you to create a new message thread, for example to create a new message to a mailing list without having to enter the mailing list's

address.

If you intend to start a new thread by replying, please make really sure you remove the *In-Reply-To:* header in your editor. Otherwise, though you'll produce a technically valid reply, some netiquette guardians will be annoyed by this so-called “thread hijacking”.

6.3. Sending Cryptographically Signed/Encrypted Messages

If you have told Mutt to PGP or S/MIME encrypt a message, it will guide you through a key selection process when you try to send the message. Mutt will not ask you any questions about keys which have a certified user ID matching one of the message recipients' mail addresses. However, there may be situations in which there are several keys, weakly certified user ID fields, or where no matching keys can be found.

In these cases, you are dropped into a menu with a list of keys from which you can select one. When you quit this menu, or Mutt can't find any matching keys, you are prompted for a user ID. You can, as usually, abort this prompt using `^G`. When you do so, Mutt will return to the compose screen.

Once you have successfully finished the key selection, the message will be encrypted using the selected public keys when sent out.

Most fields of the entries in the key selection menu (see also $pgp_entry_format) have obvious meanings. But some explanations on the capabilities, flags, and validity fields are in order.

The flags sequence (“%f”) will expand to one of the flags in Table 2.13, “PGP key menu flags”.

Table 2.13. PGP key menu flags

Flag	Description
R	The key has been revoked and can't be used.
X	The key is expired and can't be used.
d	You have marked the key as disabled.
c	There are unknown critical self-signature packets.

The capabilities field (“%c”) expands to a two-character sequence representing a key's capabilities. The first character gives the key's encryption capabilities: A minus sign (“-”) means that the key cannot be used for encryption. A dot (“.”) means that it's marked as a signature key in one of the user IDs, but may also be used for encryption. The letter “e” indicates that this key can be used for encryption.

The second character indicates the key's signing capabilities. Once again, a “-” implies “not for signing”, “.” implies that the key is marked as an encryption key in one of the user-ids, and “s” denotes a key which can be used for signing.

Finally, the validity field (“%t”) indicates how well-certified a user-id is. A question mark (“?”) indicates

undefined validity, a minus character ("-") marks an untrusted association, a space character means a partially trusted association, and a plus character ("+") indicates complete validity.

6.4. Sending Format=Flowed Messages

6.4.1. Concept

`format=flowed`-style messages (or `f=f` for short) are `text/plain` messages that consist of paragraphs which a receiver's mail client may reformat to its own needs which mostly means to customize line lengths regardless of what the sender sent. Technically this is achieved by letting lines of a "flowable" paragraph end in spaces except for the last line.

While for text-mode clients like Mutt it's the best way to assume only a standard 80x25 character cell terminal, it may be desired to let the receiver decide completely how to view a message.

6.4.2. Mutt Support

Mutt only supports setting the required `format=flowed` MIME parameter on outgoing messages if the $text_flowed variable is set, specifically it does not add the trailing spaces.

After editing the initial message text and before entering the compose menu, Mutt properly space-stuffs the message. *Space-stuffing* is required by RfC3676 defining `format=flowed` and means to prepend a space to:

- all lines starting with a space
- lines starting with the word "`From`" followed by space
- all lines starting with ">" which is not intended to be a quote character

> ***NOTE:***
>
> Mutt only supports space-stuffing for the first two types of lines but not for the third: It is impossible to safely detect whether a leading > character starts a quote or not. Furthermore, Mutt only applies space-stuffing *once* after the initial edit is finished.

All leading spaces are to be removed by receiving clients to restore the original message prior to further processing.

6.4.3. Editor Considerations

As Mutt provides no additional features to compose `f=f` messages, it's completely up to the user and his editor to produce proper messages. Please consider your editor's documentation if you intend to send `f=f` messages.

Please note that when editing messages from the compose menu several times before really sending a mail, it's up to the user to ensure that the message is properly space-stuffed.

For example, *vim* provides the `w` flag for its `formatoptions` setting to assist in creating `f=f` messages, see `:help fo-table` for details.

7. Forwarding and Bouncing Mail

Bouncing and forwarding let you send an existing message to recipients that you specify. Bouncing a message sends a verbatim copy of a message to alternative addresses as if they were the message's original recipients specified in the Bcc header. Forwarding a message, on the other hand, allows you to modify the message before it is resent (for example, by adding your own comments). Bouncing is done using the `<bounce>` function and forwarding using the `<forward>` function bound to "b" and "f" respectively.

Forwarding can be done by including the original message in the new message's body (surrounded by indicating lines) or including it as a MIME attachment, depending on the value of the $mime_forward variable. Decoding of attachments, like in the pager, can be controlled by the $forward_decode and $mime_forward_decode variables, respectively. The desired forwarding format may depend on the content, therefore $mime_forward is a quadoption which, for example, can be set to "ask-no".

The inclusion of headers is controlled by the current setting of the $weed variable, unless $mime_forward is set.

Editing the message to forward follows the same procedure as sending or replying to a message does.

8. Postponing Mail

At times it is desirable to delay sending a message that you have already begun to compose. When the `<postpone-message>` function is used in the *compose* menu, the body of your message and attachments are stored in the mailbox specified by the $postponed variable. This means that you can recall the message even if you exit Mutt and then restart it at a later time.

Once a message is postponed, there are several ways to resume it. From the command line you can use the "-p" option, or if you compose a new message from the *index* or *pager* you will be prompted if postponed messages exist. If multiple messages are currently postponed, the *postponed* menu will pop up and you can select which message you would like to resume.

> ***Note:***
>
> If you postpone a reply to a message, the reply setting of the message is only updated when you actually finish the message and send it. Also, you must be in the same folder with the message you replied to for the status of the message to be updated.

See also the $postpone quad-option.

Chapter 3. Configuration

Table of Contents

1. Location of Initialization Files

While the default configuration (or "preferences") make Mutt usable right out of the box, it is often desirable to tailor Mutt to suit your own tastes. When Mutt is first invoked, it will attempt to read the "system" configuration file (defaults set by your local system administrator), unless the "-n" command line option is specified. This file is typically `/usr/local/share/mutt/Muttrc` or `/etc/Muttrc`. Mutt will next look for a file named `.muttrc` in your home directory. If this file does not exist and your home directory has a subdirectory named `.mutt`, Mutt tries to load a file named `.mutt/muttrc`.

`.muttrc` is the file where you will usually place your commands to configure Mutt.

In addition, Mutt supports version specific configuration files that are parsed instead of the default files as explained above. For instance, if your system has a `Muttrc-0.88` file in the system configuration directory, and you are running version 0.88 of Mutt, this file will be sourced instead of the `Muttrc` file. The same is true of the user configuration file, if you have a file `.muttrc-0.88.6` in your home directory, when you run Mutt version 0.88.6, it will source this file instead of the default `.muttrc` file. The version number is the same which is visible using the "-v" command line switch or using the `show-version` key (default: V) from the index menu.

2. Syntax of Initialization Files

An initialization file consists of a series of commands. Each line of the file may contain one or more commands. When multiple commands are used, they must be separated by a semicolon (";").

Example 3.1. Multiple configuration commands per line

```
set realname='Mutt user' ; ignore x-
```

The hash mark, or pound sign ("#"), is used as a "comment" character. You can use it to annotate your initialization file. All text after the comment character to the end of the line is ignored.

Example 3.2. Commenting configuration files

```
my_hdr X-Disclaimer: Why are you listening to me? # This is a comment
```

Single quotes ("'") and double quotes ("") can be used to quote strings which contain spaces or other special characters. The difference between the two types of quotes is similar to that of many popular shell programs, namely that a single quote is used to specify a literal string (one that is not interpreted for shell variables or

quoting with a backslash [see next paragraph]), while double quotes indicate a string for which should be evaluated. For example, backticks are evaluated inside of double quotes, but *not* for single quotes.

"\" quotes the next character, just as in shells such as bash and zsh. For example, if want to put quotes ""inside of a string, you can use "\" to force the next character to be a literal instead of interpreted character.

Example 3.3. Escaping quotes in configuration files

```
set realname="Michael \"MuttDude\" Elkins"
```

"\\" means to insert a literal "\" into the line. "\n" and "\r" have their usual C meanings of linefeed and carriage-return, respectively.

A "\" at the end of a line can be used to split commands over multiple lines as it "escapes" the line end, provided that the split points don't appear in the middle of command names. Lines are first concatenated before interpretation so that a multi-line can be commented by commenting out the first line only.

Example 3.4. Splitting long configuration commands over several lines

```
set status_format="some very \
long value split \
over several lines"
```

It is also possible to substitute the output of a Unix command in an initialization file. This is accomplished by enclosing the command in backticks (``). In Example 3.5, "Using external command's output in configuration files", the output of the Unix command "uname -a" will be substituted before the line is parsed. Since initialization files are line oriented, only the first line of output from the Unix command will be substituted.

Example 3.5. Using external command's output in configuration files

```
my_hdr X-Operating-System: `uname -a`
```

Both environment variables and Mutt variables can be accessed by prepending "$" to the name of the variable. For example,

Example 3.6. Using environment variables in configuration files

```
set record=+sent_on_$HOSTNAME
```

will cause Mutt to save outgoing messages to a folder named "sent_on_kremvax" if the environment variable `$HOSTNAME` is set to "kremvax." (See $record for details.)

Mutt expands the variable when it is assigned, not when it is used. If the value of a variable on the right-hand side of an assignment changes after the assignment, the variable on the left-hand side will not be affected.

The commands understood by Mutt are explained in the next paragraphs. For a complete list, see the command reference.

All configuration files are expected to be in the current locale as specified by the $charset variable which doesn't have a default value since it's determined by Mutt at startup. If a configuration file is not encoded in the same character set the $config_charset variable should be used: all lines starting with the next are recoded from $config_charset to $charset.

This mechanism should be avoided if possible as it has the following implications:

- These variables should be set early in a configuration file with $charset preceding $config_charset so Mutt knows what character set to convert to.
- If $config_charset is set, it should be set in each configuration file because the value is global and *not* per configuration file.
- Because Mutt first recodes a line before it attempts to parse it, a conversion introducing question marks or other characters as part of errors (unconvertable characters, transliteration) may introduce syntax errors or silently change the meaning of certain tokens (e.g. inserting question marks into regular expressions).

3. Address Groups

Usage:

```
group [ -group name ...] { -rx expr ... | -addr expr ... }
ungroup [ -group name ...] { * | -rx expr ... | -addr expr ... }
```

Mutt supports grouping addresses logically into named groups. An address or address pattern can appear in several groups at the same time. These groups can be used in patterns (for searching, limiting and tagging) and in hooks by using group patterns. This can be useful to classify mail and take certain actions depending on in what groups the message is. For example, the mutt user's mailing list would fit into the categories “mailing list” and “mutt-related”. Using `send-hook`, the sender can be set to a dedicated one for writing mailing list messages, and the signature could be set to a mutt-related one for writing to a mutt list — for other lists, the list sender setting still applies but a different signature can be selected. Or, given a group only containing recipients known to accept encrypted mail, “auto-encryption” can be achieved easily.

The `group` command is used to directly add either addresses or regular expressions to the specified group or groups. The different categories of arguments to the `group` command can be in any order. The flags `-rx` and `-addr` specify what the following strings (that cannot begin with a hyphen) should be interpreted as: either a regular expression or an email address, respectively.

These address groups can also be created implicitly by the `alias`, `lists`, `subscribe` and `alternates`

commands by specifying the optional `-group` option. For example,

```
alternates -group me address1 address2
alternates -group me -group work address3
```

would create a group named “me” which contains all your addresses and a group named “work” which contains only your work address *address3*. Besides many other possibilities, this could be used to automatically mark your own messages in a mailing list folder as read or use a special signature for work-related messages.

The `ungroup` command is used to remove addresses or regular expressions from the specified group or groups. The syntax is similar to the `group` command, however the special character * can be used to empty a group of all of its contents. As soon as a group gets empty because all addresses and regular expressions have been removed, it'll internally be removed, too (i.e. there cannot be an empty group). When removing regular expressions from a group, the pattern must be specified exactly as given to the `group` command or `-group` argument.

4. Defining/Using Aliases

Usage:

```
alias [ -group name ...] key address [ address ...]
unalias [ -group name ...] { * | key ... }
```

It's usually very cumbersome to remember or type out the address of someone you are communicating with. Mutt allows you to create “aliases” which map a short string to a full address.

> ***Note:***
>
> If you want to create an alias for more than one address, you *must* separate the addresses with a comma (“,”).

The optional `-group` argument to `alias` causes the aliased address(es) to be added to the named *group*.

To remove an alias or aliases (“*” means all aliases):

```
alias muttdude me@cs.hmc.edu (Michael Elkins)
alias theguys manny, moe, jack
```

Unlike other mailers, Mutt doesn't require aliases to be defined in a special file. The `alias` command can appear anywhere in a configuration file, as long as this file is `sourced`. Consequently, you can have multiple alias files, or you can have all aliases defined in your `.muttrc`.

On the other hand, the `<create-alias>` function can use only one file, the one pointed to by the $alias_file

variable (which is `~/.muttrc` by default). This file is not special either, in the sense that Mutt will happily append aliases to any file, but in order for the new aliases to take effect you need to explicitly `source` this file too.

Example 3.7. Configuring external alias files

```
source /usr/local/share/Mutt.aliases
source ~/.mail_aliases
set alias_file=~/.mail_aliases
```

To use aliases, you merely use the alias at any place in Mutt where Mutt prompts for addresses, such as the *To:* or *Cc:* prompt. You can also enter aliases in your editor at the appropriate headers if you have the $edit_headers variable set.

In addition, at the various address prompts, you can use the tab character to expand a partial alias to the full alias. If there are multiple matches, Mutt will bring up a menu with the matching aliases. In order to be presented with the full list of aliases, you must hit tab without a partial alias, such as at the beginning of the prompt or after a comma denoting multiple addresses.

In the alias menu, you can select as many aliases as you want with the `select-entry` key (default: <Return>), and use the *exit* key (default: q) to return to the address prompt.

5. Changing the Default Key Bindings

Usage:

`bind` *map key function*

This command allows you to change the default key bindings (operation invoked when pressing a key).

map specifies in which menu the binding belongs. Multiple maps may be specified by separating them with commas (no additional whitespace is allowed). The currently defined maps are:

generic

> This is not a real menu, but is used as a fallback for all of the other menus except for the pager and editor modes. If a key is not defined in another menu, Mutt will look for a binding to use in this menu. This allows you to bind a key to a certain function in multiple menus instead of having multiple `bind` statements to accomplish the same task.

alias

> The alias menu is the list of your personal aliases as defined in your `.muttrc`. It is the mapping from a short alias name to the full email address(es) of the recipient(s).

attach

The attachment menu is used to access the attachments on received messages.

browser

The browser is used for both browsing the local directory structure, and for listing all of your incoming mailboxes.

editor

The editor is used to allow the user to enter a single line of text, such as the *To* or *Subject* prompts in the `compose` menu.

index

The index is the list of messages contained in a mailbox.

compose

The compose menu is the screen used when sending a new message.

pager

The pager is the mode used to display message/attachment data, and help listings.

pgp

The pgp menu is used to select the OpenPGP keys used to encrypt outgoing messages.

smime

The smime menu is used to select the OpenSSL certificates used to encrypt outgoing messages.

postpone

The postpone menu is similar to the index menu, except is used when recalling a message the user was composing, but saved until later.

query

The query menu is the browser for results returned by $query_command.

mix

The mixmaster screen is used to select remailer options for outgoing messages (if Mutt is compiled with Mixmaster support).

key is the key (or key sequence) you wish to bind. To specify a control character, use the sequence *\Cx*, where *x* is the letter of the control character (for example, to specify control-A use "\Ca"). Note that the case of *x* as well as *\C* is ignored, so that *\CA*, *\Ca*, *\cA* and *\ca* are all equivalent. An alternative form is to specify

the key as a three digit octal number prefixed with a "\" (for example *\177* is equivalent to *\c?*). In addition, *key* may be a symbolic name as shown in Table 3.1, "Symbolic key names".

Table 3.1. Symbolic key names

Symbolic name	Meaning
\t	tab
<tab>	tab
<backtab>	backtab / shift-tab
\r	carriage return
\n	newline
\e	escape
<esc>	escape
<up>	up arrow
<down>	down arrow
<left>	left arrow
<right>	right arrow
<pageup>	Page Up
<pagedown>	Page Down
<backspace>	Backspace
<delete>	Delete
<insert>	Insert
<enter>	Enter
<return>	Return
<home>	Home
<end>	End
<space>	Space bar
<f1>	function key 1
<f10>	function key 10

key does not need to be enclosed in quotes unless it contains a space (" ") or semi-colon (";").

function specifies which action to take when *key* is pressed. For a complete list of functions, see the reference. Note that the `bind` expects *function* to be specified without angle brackets.

The special function `<noop>` unbinds the specified key sequence.

6. Defining Aliases for Character Sets

Usage:

```
charset-hook alias charset
iconv-hook charset local-charset
```

The `charset-hook` command defines an alias for a character set. This is useful to properly display messages which are tagged with a character set name not known to Mutt.

The `iconv-hook` command defines a system-specific name for a character set. This is helpful when your systems character conversion library insists on using strange, system-specific names for character sets.

7. Setting Variables Based Upon Mailbox

Usage:

```
folder-hook [!]regexp command
```

It is often desirable to change settings based on which mailbox you are reading. The `folder-hook` command provides a method by which you can execute any configuration command. *regexp* is a regular expression specifying in which mailboxes to execute *command* before loading. If a mailbox matches multiple `folder-hooks`, they are executed in the order given in the `.muttrc`.

> ***Note:***
>
> If you use the "!" shortcut for $spoolfile at the beginning of the pattern, you must place it inside of double or single quotes in order to distinguish it from the logical *not* operator for the expression.

> ***Note:***
>
> Settings are *not* restored when you leave the mailbox. For example, a command action to perform is to change the sorting method based upon the mailbox being read:
>
> ```
> folder-hook mutt "set sort=threads"
> ```
>
> However, the sorting method is not restored to its previous value when reading a different mailbox. To specify a *default* command, use the pattern "." before other `folder-hooks` adjusting a value on a per-folder basis because `folder-hooks` are evaluated in the order given in the configuration file.

The following example will set the sort variable to `date-sent` for all folders but to `threads` for all folders containing "mutt" in their name.

Example 3.8. Setting sort method based on mailbox name

```
folder-hook . "set sort=date-sent"
folder-hook mutt "set sort=threads"
```

8. Keyboard Macros

Usage:

`macro` *menu key sequence* [*description*]

Macros are useful when you would like a single key to perform a series of actions. When you press *key* in menu *menu*, Mutt will behave as if you had typed *sequence*. So if you have a common sequence of commands you type, you can create a macro to execute those commands with a single key or fewer keys.

menu is the map which the macro will be bound in. Multiple maps may be specified by separating multiple menu arguments by commas. Whitespace may not be used in between the menu arguments and the commas separating them.

key and *sequence* are expanded by the same rules as the key bindings with some additions. The first is that control characters in *sequence* can also be specified as ^*x*. In order to get a caret ("^") you need to use ^^. Secondly, to specify a certain key such as *up* or to invoke a function directly, you can use the format <*key name*> and <*function name*>. For a listing of key names see the section on key bindings. Functions are listed in the reference.

The advantage with using function names directly is that the macros will work regardless of the current key bindings, so they are not dependent on the user having particular key definitions. This makes them more robust and portable, and also facilitates defining of macros in files used by more than one user (e.g., the system Muttrc).

Optionally you can specify a descriptive text after *sequence*, which is shown in the help screens if they contain a description.

> ***Note:***
>
> Macro definitions (if any) listed in the help screen(s), are silently truncated at the screen width, and are not wrapped.

9. Using Color and Mono Video Attributes

Usage:

```
color object foreground background
color { header | body } foreground background regexp
color index foreground background pattern
uncolor { index | header | body } { * | pattern ... }
```

If your terminal supports color, you can spice up Mutt by creating your own color scheme. To define the color of an object (type of information), you must specify both a foreground color *and* a background color (it is not possible to only specify one or the other).

header and *body* match *regexp* in the header/body of a message, *index* matches *pattern* (see Section 3, "Patterns: Searching, Limiting and Tagging") in the message index.

object can be one of:

- attachment
- bold (highlighting bold patterns in the body of messages)
- error (error messages printed by Mutt)
- hdrdefault (default color of the message header in the pager)
- indicator (arrow or bar used to indicate the current item in a menu)
- markers (the "+" markers at the beginning of wrapped lines in the pager)
- message (informational messages)
- normal
- quoted (text matching $quote_regexp in the body of a message)
- quoted1, quoted2, ..., quoted*N* (higher levels of quoting)
- search (highlighting of words in the pager)
- signature
- status (mode lines used to display info about the mailbox or message)
- tilde (the "~" used to pad blank lines in the pager)
- tree (thread tree drawn in the message index and attachment menu)

- underline (highlighting underlined patterns in the body of messages)

foreground and *background* can be one of the following:

- white
- black
- green
- magenta
- blue
- cyan
- yellow
- red
- default
- color*x*

foreground can optionally be prefixed with the keyword `bright` to make the foreground color boldfaced (e.g., `brightred`).

If your terminal supports it, the special keyword *default* can be used as a transparent color. The value *brightdefault* is also valid. If Mutt is linked against the *S-Lang* library, you also need to set the `$COLORFGBG` environment variable to the default colors of your terminal for this to work; for example (for Bourne-like shells):

```
set COLORFGBG="green;black"
export COLORFGBG
```

Note:

The *S-Lang* library requires you to use the *lightgray* and *brown* keywords instead of *white* and *yellow* when setting this variable.

Note:

The `uncolor` command can be applied to the index, header and body objects only. It removes entries from the list. You *must* specify the same pattern specified in the `color` command for it to be removed. The pattern "*" is a special token which means to clear the color list of all entries.

Mutt also recognizes the keywords *color0*, *color1*, ..., *colorN-1* (*N* being the number of colors supported by your terminal). This is useful when you remap the colors for your display (for example by changing the color associated with *color2* for your xterm), since color names may then lose their normal meaning.

If your terminal does not support color, it is still possible change the video attributes through the use of the "mono" command. Usage:

```
mono object attribute
mono { header | body } attribute regexp
mono index attribute pattern
unmono { index | header | body } { * | pattern ... }
```

For *object*, see the `color` command. *attribute* can be one of the following:

- none
- bold
- underline
- reverse
- standout

10. Message Header Display

10.1. Header Display

When displaying a message in the pager, Mutt folds long header lines at $wrap columns. Though there're precise rules about where to break and how, Mutt always folds headers using a tab for readability. (Note that the sending side is not affected by this, Mutt tries to implement standards compliant folding.)

10.2. Selecting Headers

Usage:

```
ignore pattern [ pattern ...]
unignore { * | pattern ... }
```

Messages often have many header fields added by automatic processing systems, or which may not seem useful to display on the screen. This command allows you to specify header fields which you don't normally want to see in the pager.

You do not need to specify the full header field name. For example, "ignore content-" will ignore all header fields that begin with the pattern "content-". "ignore *" will ignore all headers.

To remove a previously added token from the list, use the “unignore” command. The “unignore” command will make Mutt display headers with the given pattern. For example, if you do “ignore x-” it is possible to “unignore x-mailer”.

“unignore *” will remove all tokens from the ignore list.

Example 3.9. Header weeding

```
# Sven's draconian header weeding
ignore *
unignore from date subject to cc
unignore organization organisation x-mailer: x-newsreader: x-mailing-list:
unignore posted-to:
```

10.3. Ordering Displayed Headers

Usage:

```
hdr_order header [ header ...]
unhdr_order { * | header ... }
```

With the `hdr_order` command you can specify an order in which Mutt will attempt to present these headers to you when viewing messages.

“`unhdr_order` *” will clear all previous headers from the order list, thus removing the header order effects set by the system-wide startup file.

Example 3.10. Configuring header display order

```
hdr_order From Date: From: To: Cc: Subject:
```

11. Alternative Addresses

Usage:

```
alternates [ -group name ...] regexp [ regexp ...]
unalternates [ -group name ...] { * | regexp ... }
```

With various functions, Mutt will treat messages differently, depending on whether you sent them or whether you received them from someone else. For instance, when replying to a message that you sent to a different party, Mutt will automatically suggest to send the response to the original message's recipients — responding to yourself won't make much sense in many cases. (See $reply_to.)

Many users receive e-mail under a number of different addresses. To fully use Mutt's features here, the

program must be able to recognize what e-mail addresses you receive mail under. That's the purpose of the `alternates` command: It takes a list of regular expressions, each of which can identify an address under which you receive e-mail.

As addresses are matched using regular expressions and not exact strict comparisons, you should make sure you specify your addresses as precise as possible to avoid mismatches. For example, if you specify:

```
alternates user@example
```

Mutt will consider "`some-user@example`" as being your address, too which may not be desired. As a solution, in such cases addresses should be specified as:

```
alternates '^user@example$'
```

The `-group` flag causes all of the subsequent regular expressions to be added to the named group.

The `unalternates` command can be used to write exceptions to `alternates` patterns. If an address matches something in an `alternates` command, but you nonetheless do not think it is from you, you can list a more precise pattern under an `unalternates` command.

To remove a regular expression from the `alternates` list, use the `unalternates` command with exactly the same *regexp*. Likewise, if the *regexp* for an `alternates` command matches an entry on the `unalternates` list, that `unalternates` entry will be removed. If the *regexp* for `unalternates` is "*", *all entries* on `alternates` will be removed.

12. Mailing Lists

Usage:

```
lists [ -group name ...] regexp [ regexp ...]
unlists { * | regexp ... }
subscribe [ -group name ...] regexp [ regexp ...]
unsubscribe { * | regexp ... }
```

Mutt has a few nice features for handling mailing lists. In order to take advantage of them, you must specify which addresses belong to mailing lists, and which mailing lists you are subscribed to. Mutt also has limited support for auto-detecting mailing lists: it supports parsing `mailto:` links in the common `List-Post:` header which has the same effect as specifying the list address via the `lists` command (except the group feature). Once you have done this, the `<list-reply>` function will work for all known lists. Additionally, when you send a message to a subscribed list, Mutt will add a Mail-Followup-To header to tell other users' mail user agents not to send copies of replies to your personal address.

> ***Note:***
>
> The Mail-Followup-To header is a non-standard extension which is not supported by all mail user

agents. Adding it is not bullet-proof against receiving personal CCs of list messages. Also note that the generation of the Mail-Followup-To header is controlled by the $followup_to configuration variable since it's common practice on some mailing lists to send Cc upon replies (which is more a group- than a list-reply).

More precisely, Mutt maintains lists of patterns for the addresses of known and subscribed mailing lists. Every subscribed mailing list is known. To mark a mailing list as known, use the `list` command. To mark it as subscribed, use `subscribe`.

You can use regular expressions with both commands. To mark all messages sent to a specific bug report's address on Debian's bug tracking system as list mail, for instance, you could say

```
subscribe [0-9]+.*@bugs.debian.org
```

as it's often sufficient to just give a portion of the list's e-mail address.

Specify as much of the address as you need to to remove ambiguity. For example, if you've subscribed to the Mutt mailing list, you will receive mail addressed to `mutt-users@mutt.org`. So, to tell Mutt that this is a mailing list, you could add `lists mutt-users@` to your initialization file. To tell Mutt that you are subscribed to it, add `subscribe mutt-users` to your initialization file instead. If you also happen to get mail from someone whose address is `mutt-users@example.com`, you could use `lists ^mutt-users@mutt\\.org$` or `subscribe ^mutt-users@mutt\\.org$` to match only mail from the actual list.

The `-group` flag adds all of the subsequent regular expressions to the named address group in addition to adding to the specified address list.

The “unlists” command is used to remove a token from the list of known and subscribed mailing-lists. Use “unlists *” to remove all tokens.

To remove a mailing list from the list of subscribed mailing lists, but keep it on the list of known mailing lists, use `unsubscribe`.

13. Using Multiple Spool Mailboxes

Usage:

```
mbox-hook [!]pattern mailbox
```

This command is used to move read messages from a specified mailbox to a different mailbox automatically when you quit or change folders. *pattern* is a regular expression specifying the mailbox to treat as a “spool” mailbox and *mailbox* specifies where mail should be saved when read.

Unlike some of the other *hook* commands, only the *first* matching pattern is used (it is not possible to save read mail in more than a single mailbox).

14. Monitoring Incoming Mail

Usage:

```
mailboxes mailbox [ mailbox ...]
unmailboxes { * | mailbox ... }
```

This command specifies folders which can receive mail and which will be checked for new messages periodically.

folder can either be a local file or directory (Mbox/Mmdf or Maildir/Mh). If Mutt was built with POP and/or IMAP support, *folder* can also be a POP/IMAP folder URL. The URL syntax is described in Section 1.2, "URL Syntax", POP and IMAP are described in Section 3, "POP3 Support" and Section 4, "IMAP Support" respectively.

Mutt provides a number of advanced features for handling (possibly many) folders and new mail within them, please refer to Section 10, "New Mail Detection" for details (including in what situations and how often Mutt checks for new mail).

The "unmailboxes" command is used to remove a token from the list of folders which receive mail. Use "unmailboxes *" to remove all tokens.

> ***Note:***
>
> The folders in the `mailboxes` command are resolved when the command is executed, so if these names contain shortcut characters (such as "=" and "!"), any variable definition that affects these characters (like $folder and $spoolfile) should be set before the `mailboxes` command. If none of these shortcuts are used, a local path should be absolute as otherwise Mutt tries to find it relative to the directory from where Mutt was started which may not always be desired.

15. User-Defined Headers

Usage:

```
my_hdr string
unmy_hdr { * | field ... }
```

The `my_hdr` command allows you to create your own header fields which will be added to every message you send and appear in the editor if $edit_headers is set.

For example, if you would like to add an "Organization:" header field to all of your outgoing messages, you can put the command something like shown in Example 3.11, "Defining custom headers" in your `.muttrc`.

Example 3.11. Defining custom headers

```
my_hdr Organization: A Really Big Company, Anytown, USA
```

> ***Note:***
>
> Space characters are *not* allowed between the keyword and the colon (":"). The standard for electronic mail (RFC2822) says that space is illegal there, so Mutt enforces the rule.

If you would like to add a header field to a single message, you should either set the $edit_headers variable, or use the `<edit-headers>` function (default: "E") in the compose menu so that you can edit the header of your message along with the body.

To remove user defined header fields, use the `unmy_hdr` command. You may specify an asterisk ("*") to remove all header fields, or the fields to remove. For example, to remove all "To" and "Cc" header fields, you could use:

```
unmy_hdr to cc
```

16. Specify Default Save Mailbox

Usage:

```
save-hook [!]pattern mailbox
```

This command is used to override the default mailbox used when saving messages. *mailbox* will be used as the default if the message matches *pattern*, see Message Matching in Hooks for information on the exact format.

To provide more flexibility and good defaults, Mutt applies the expandos of $index_format to *mailbox* after it was expanded.

Example 3.12. Using %-expandos in `save-hook`

```
# default: save all to ~/Mail/<author name>
save-hook . ~/Mail/%F

# save from me@turing.cs.hmc.edu and me@cs.hmc.edu to $folder/elkins
save-hook me@(turing\\.)?cs\\.hmc\\.edu$ +elkins

# save from aol.com to $folder/spam
save-hook aol\\.com$ +spam
```

Also see the `fcc-save-hook` command.

17. Specify Default Fcc: Mailbox When Composing

Usage:

```
fcc-hook [!]pattern mailbox
```

This command is used to save outgoing mail in a mailbox other than $record. Mutt searches the initial list of message recipients for the first matching *regexp* and uses *mailbox* as the default Fcc: mailbox. If no match is found the message will be saved to $record mailbox.

To provide more flexibility and good defaults, Mutt applies the expandos of $index_format to *mailbox* after it was expanded.

See Message Matching in Hooks for information on the exact format of *pattern*.

```
fcc-hook [@.]aol\\.com$ +spammers
```

...will save a copy of all messages going to the aol.com domain to the `+spammers' mailbox by default. Also see the `fcc-save-hook` command.

18. Specify Default Save Filename and Default Fcc: Mailbox at Once

Usage:

```
fcc-save-hook [!]pattern mailbox
```

This command is a shortcut, equivalent to doing both a `fcc-hook` and a `save-hook` with its arguments, including %-expansion on *mailbox* according to $index_format.

19. Change Settings Based Upon Message Recipients

Usage:

```
reply-hook [!]pattern command
send-hook [!]pattern command
send2-hook [!]pattern command
```

These commands can be used to execute arbitrary configuration commands based upon recipients of the message. *pattern* is used to match the message, see Message Matching in Hooks for details. *command* is executed when *pattern* matches.

`reply-hook` is matched against the message you are *replying to*, instead of the message you are *sending*. `send-hook` is matched against all messages, both *new* and *replies*.

Note:

`reply-hooks` are matched *before* the `send-hook`, *regardless* of the order specified in the user's configuration file.

`send2-hook` is matched every time a message is changed, either by editing it, or by using the compose menu to change its recipients or subject. `send2-hook` is executed after `send-hook`, and can, e.g., be used to set parameters such as the $sendmail variable depending on the message's sender address.

For each type of `send-hook` or `reply-hook`, when multiple matches occur, commands are executed in the order they are specified in the `.muttrc` (for that type of hook).

Example: `send-hook mutt "set mime_forward signature=''"`

Another typical use for this command is to change the values of the $attribution, $signature and $locale variables in order to change the language of the attributions and signatures based upon the recipients.

Note:

`send-hook`'s are only executed once after getting the initial list of recipients. Adding a recipient after replying or editing the message will not cause any `send-hook` to be executed, similarly if $autoedit is set (as then the initial list of recipients is empty). Also note that `my_hdr` commands which modify recipient headers, or the message's subject, don't have any effect on the current message when executed from a `send-hook`.

20. Change Settings Before Formatting a Message

Usage:

```
message-hook [!]pattern command
```

This command can be used to execute arbitrary configuration commands before viewing or formatting a message based upon information about the message. *command* is executed if the *pattern* matches the message to be displayed. When multiple matches occur, commands are executed in the order they are specified in the `.muttrc`.

See Message Matching in Hooks for information on the exact format of *pattern*.

Example:

```
message-hook ~A 'set pager=builtin'
message-hook '~f freshmeat-news' 'set pager="less \"+/^  subject: .*\""'
```

21. Choosing the Cryptographic Key of the Recipient

Usage:

```
crypt-hook pattern keyid
```

When encrypting messages with PGP/GnuPG or OpenSSL, you may want to associate a certain key with a given e-mail address automatically, either because the recipient's public key can't be deduced from the destination address, or because, for some reasons, you need to override the key Mutt would normally use. The `crypt-hook` command provides a method by which you can specify the ID of the public key to be used when encrypting messages to a certain recipient.

The meaning of *keyid* is to be taken broadly in this context: You can either put a numerical key ID here, an e-mail address, or even just a real name.

22. Adding Key Sequences to the Keyboard Buffer

Usage:

```
push string
```

This command adds the named string to the keyboard buffer. The string may contain control characters, key names and function names like the sequence string in the macro command. You may use it to automatically run a sequence of commands at startup, or when entering certain folders. For example, Example 3.13, "Embedding `push` in `folder-hook`" shows how to automatically collapse all threads when entering a folder.

Example 3.13. Embedding `push` in `folder-hook`

```
folder-hook . 'push <collapse-all>'
```

For using functions like shown in the example, it's important to use angle brackets ("<" and ">") to make Mutt recognize the input as a function name. Otherwise it will simulate individual just keystrokes, i.e. "`push collapse-all`" would be interpreted as if you had typed "c", followed by "o", followed by "l", ..., which is not desired and may lead to very unexpected behavior.

Keystrokes can be used, too, but are less portable because of potentially changed key bindings. With default bindings, this is equivalent to the above example:

```
folder-hook . 'push \eV'
```

because it simulates that Esc+V was pressed (which is the default binding of `<collapse-all>`).

23. Executing Functions

Usage:

```
exec function [ function ...]
```

This command can be used to execute any function. Functions are listed in the function reference. "`exec function`" is equivalent to "`push <function>`".

24. Message Scoring

Usage:

```
score pattern value
unscore { * | pattern ... }
```

The `score` commands adds *value* to a message's score if *pattern* matches it. *pattern* is a string in the format described in the patterns section (note: For efficiency reasons, patterns which scan information not available in the index, such as `~b`, `~B` or `~h`, may not be used). *value* is a positive or negative integer. A message's final score is the sum total of all matching `score` entries. However, you may optionally prefix *value* with an equal sign ("=") to cause evaluation to stop at a particular entry if there is a match. Negative final scores are rounded up to 0.

The `unscore` command removes score entries from the list. You *must* specify the same pattern specified in the `score` command for it to be removed. The pattern "*" is a special token which means to clear the list of all score entries.

25. Spam Detection

Usage:

```
spam pattern format
nospam { * | pattern }
```

Mutt has generalized support for external spam-scoring filters. By defining your spam patterns with the `spam` and `nospam` commands, you can *limit*, *search*, and *sort* your mail based on its spam attributes, as determined by the external filter. You also can display the spam attributes in your index display using the `%H` selector in the $index_format variable. (Tip: try `%?H?[%H] ?` to display spam tags only when they are defined for a given message.)

Your first step is to define your external filter's spam patterns using the `spam` command. *pattern* should be a regular expression that matches a header in a mail message. If any message in the mailbox matches this regular expression, it will receive a "spam tag" or "spam attribute" (unless it also matches a `nospam` pattern — see below.) The appearance of this attribute is entirely up to you, and is governed by the *format* parameter. *format* can be any static text, but it also can include back-references from the *pattern* expression. (A regular expression "back-reference" refers to a sub-expression contained within parentheses.) `%1` is replaced with the first back-reference in the regex, `%2` with the second, etc.

To match spam tags, mutt needs the corresponding header information which is always the case for local and POP folders but not for IMAP in the default configuration. Depending on the spam header to be analyzed, $imap_headers may need to be adjusted.

If you're using multiple spam filters, a message can have more than one spam-related header. You can define `spam` patterns for each filter you use. If a message matches two or more of these patterns, and the $spam_separator variable is set to a string, then the message's spam tag will consist of all the *format* strings joined together, with the value of $spam_separator separating them.

For example, suppose one uses DCC, SpamAssassin, and PureMessage, then the configuration might look like in Example 3.14, "Configuring spam detection".

Example 3.14. Configuring spam detection

```
spam "X-DCC-.*-Metrics:.*(....)=many"          "90+/DCC-%1"
spam "X-Spam-Status: Yes"                      "90+/SA"
spam "X-PerlMX-Spam: .*Probability=([0-9]+)%" "%1/PM"
set spam_separator=", "
```

If then a message is received that DCC registered with "many" hits under the "Fuz2" checksum, and that PureMessage registered with a 97% probability of being spam, that message's spam tag would read `90+/DCC-Fuz2, 97/PM`. (The four characters before "=many" in a DCC report indicate the checksum used — in this case, "Fuz2".)

If the $spam_separator variable is unset, then each spam pattern match supersedes the previous one. Instead of getting joined *format* strings, you'll get only the last one to match.

The spam tag is what will be displayed in the index when you use `%H` in the $index_format variable. It's also the string that the `~H` pattern-matching expression matches against for `<search>` and `<limit>` functions. And it's what sorting by spam attribute will use as a sort key.

That's a pretty complicated example, and most people's actual environments will have only one spam filter. The simpler your configuration, the more effective Mutt can be, especially when it comes to sorting.

Generally, when you sort by spam tag, Mutt will sort *lexically* — that is, by ordering strings alphanumerically. However, if a spam tag begins with a number, Mutt will sort numerically first, and lexically only when two numbers are equal in value. (This is like UNIX's `sort -n`.) A message with no spam attributes at all — that is, one that didn't match *any* of your `spam` patterns — is sorted at lowest priority. Numbers are sorted next, beginning with 0 and ranging upward. Finally, non-numeric strings are sorted, with "a" taking lower priority than "z". Clearly, in general, sorting by spam tags is most effective when you can coerce your filter to give you a raw number. But in case you can't, Mutt can still do something useful.

The `nospam` command can be used to write exceptions to `spam` patterns. If a header pattern matches something in a `spam` command, but you nonetheless do not want it to receive a spam tag, you can list a more precise pattern under a `nospam` command.

If the *pattern* given to `nospam` is exactly the same as the *pattern* on an existing `spam` list entry, the effect will be to remove the entry from the spam list, instead of adding an exception. Likewise, if the *pattern* for a `spam` command matches an entry on the `nospam` list, that nospam entry will be removed. If the *pattern* for `nospam` is "*", *all entries on both lists* will be removed. This might be the default action if you use `spam` and `nospam` in conjunction with a `folder-hook`.

You can have as many `spam` or `nospam` commands as you like. You can even do your own primitive `spam` detection within Mutt — for example, if you consider all mail from `MAILER-DAEMON` to be spam, you can use a `spam` command like this:

```
spam "^From: .*MAILER-DAEMON"       "999"
```

26. Setting and Querying Variables

26.1. Variable Types

Mutt supports these types of configuration variables:

boolean

A boolean expression, either "yes" or "no".

number

A signed integer number in the range -32768 to 32767.

string

Arbitrary text.

path

A specialized string for representing paths including support for mailbox shortcuts (see Section 8, "Mailbox Shortcuts") as well as tilde ("~") for a user's home directory and more.

quadoption

Like a boolean but triggers a prompt when set to "ask-yes" or "ask-no" with "yes" and "no" preselected respectively.

sort order

A specialized string allowing only particular words as values depending on the variable.

regular expression

A regular expression, see Section 2, "Regular Expressions" for an introduction.

folder magic

Specifies the type of folder to use: *mbox*, *mmdf*, *mh* or *maildir*. Currently only used to determine the type for newly created folders.

e-mail address

An e-mail address either with or without realname. The older "`user@example.org (Joe User)`" form is supported but strongly deprecated.

user-defined

Arbitrary text, see Section 26.3, "User-Defined Variables" for details.

26.2. Commands

The following commands are available to manipulate and query variables:

Usage:

```
set { [ no | inv ] variable | variable=value } [...]
toggle variable [ variable ...]
unset variable [ variable ...]
reset variable [ variable ...]
```

This command is used to set (and unset) configuration variables. There are four basic types of variables: boolean, number, string and quadoption. *boolean* variables can be *set* (true) or *unset* (false). *number* variables can be assigned a positive integer value. *string* variables consist of any number of printable characters and must be enclosed in quotes if they contain spaces or tabs. You may also use the escape sequences "\n" and "\t" for newline and tab, respectively. *quadoption* variables are used to control whether or not to be prompted for certain actions, or to specify a default action. A value of *yes* will cause the action to be carried out automatically as if you had answered yes to the question. Similarly, a value of *no* will cause the action to be carried out as if you had answered "no." A value of *ask-yes* will cause a prompt with a default answer of "yes" and *ask-no* will provide a default answer of "no."

Prefixing a variable with "no" will unset it. Example: `set noaskbcc`.

For *boolean* variables, you may optionally prefix the variable name with `inv` to toggle the value (on or off). This is useful when writing macros. Example: `set invsmart_wrap`.

The `toggle` command automatically prepends the `inv` prefix to all specified variables.

The `unset` command automatically prepends the `no` prefix to all specified variables.

Using the `<enter-command>` function in the *index* menu, you can query the value of a variable by prefixing the name of the variable with a question mark:

```
set ?allow_8bit
```

The question mark is actually only required for boolean and quadoption variables.

The `reset` command resets all given variables to the compile time defaults (hopefully mentioned in this manual). If you use the command `set` and prefix the variable with "&" this has the same behavior as the `reset` command.

With the `reset` command there exists the special variable "all", which allows you to reset all variables to their system defaults.

26.3. User-Defined Variables

26.3.1. Introduction

Along with the variables listed in the Configuration variables section, Mutt supports user-defined variables with names starting with `my_` as in, for example, `my_cfgdir`.

The `set` command either creates a custom `my_` variable or changes its value if it does exist already. The `unset` and `reset` commands remove the variable entirely.

Since user-defined variables are expanded in the same way that environment variables are (except for the shell-escape command and backtick expansion), this feature can be used to make configuration files more readable.

26.3.2. Examples

The following example defines and uses the variable `my_cfgdir` to abbreviate the calls of the `source` command:

Example 3.15. Using user-defined variables for config file readability

```
set my_cfgdir = $HOME/mutt/config

source $my_cfgdir/hooks
source $my_cfgdir/macros
# more source commands...
```

A custom variable can also be used in macros to backup the current value of another variable. In the following example, the value of the $delete is changed temporarily while its original value is saved as `my_delete`. After the macro has executed all commands, the original value of $delete is restored.

Example 3.16. Using user-defined variables for backing up other config option values

```
macro pager ,x '\
<enter-command>set my_delete=$delete<enter>\
```

```
<enter-command>set delete=yes<enter>\
...\
<enter-command>set delete=$my_delete<enter>'
```

Since Mutt expands such values already when parsing the configuration file(s), the value of `$my_delete` in the last example would be the value of $delete exactly as it was at that point during parsing the configuration file. If another statement would change the value for $delete later in the same or another file, it would have no effect on `$my_delete`. However, the expansion can be deferred to runtime, as shown in the next example, when escaping the dollar sign.

Example 3.17. Deferring user-defined variable expansion to runtime

```
macro pager <PageDown> "\
<enter-command> set my_old_pager_stop=\$pager_stop pager_stop<Enter>\
<next-page>\
<enter-command> set pager_stop=\$my_old_pager_stop<Enter>\
<enter-command> unset my_old_pager_stop<Enter>"
```

Note that there is a space between `<enter-command>` and the `set` configuration command, preventing Mutt from recording the `macro`'s commands into its history.

26.4. Type Conversions

Variables are always assigned string values which Mutt parses into its internal representation according to the type of the variable, for example an integer number for numeric types. For all queries (including $-expansion) the value is converted from its internal type back into string. As a result, any variable can be assigned any value given that its content is valid for the target. This also counts for custom variables which are of type string. In case of parsing errors, Mutt will print error messages. Example 3.18, “Type conversions using variables” demonstrates type conversions.

Example 3.18. Type conversions using variables

```
set my_lines = "5"                # value is string "5"
set pager_index_lines = $my_lines # value is integer 5

set my_sort = "date-received"     # value is string "date-received"
set sort = "last-$my_sort"        # value is sort last-date-received

set my_inc = $read_inc            # value is string "10" (default of $read_inc)
set my_foo = $my_inc              # value is string "10"
```

These assignments are all valid. If, however, the value of `$my_lines` would have been “five” (or something else that cannot be parsed into a number), the assignment to `$pager_index_lines` would have produced an error message.

Type conversion applies to all configuration commands which take arguments. But please note that every expanded value of a variable is considered just a single token. A working example is:

```
set my_pattern = "~A"
set my_number = "10"

# same as: score ~A +10
score $my_pattern +$my_number
```

What does *not* work is:

```
set my_mx = "+mailbox1 +mailbox2"
mailboxes $my_mx +mailbox3
```

because the value of `$my_mx` is interpreted as a single mailbox named "+mailbox1 +mailbox2" and not two distinct mailboxes.

27. Reading Initialization Commands From Another File

Usage:

```
source filename
```

This command allows the inclusion of initialization commands from other files. For example, I place all of my aliases in `~/.mail_aliases` so that I can make my `~/.muttrc` readable and keep my aliases private.

If the filename begins with a tilde ("~"), it will be expanded to the path of your home directory.

If the filename ends with a vertical bar ("|"), then *filename* is considered to be an executable program from which to read input (e.g. `source ~/bin/myscript|`).

28. Removing Hooks

Usage:

```
unhook { * | hook-type }
```

This command permits you to flush hooks you have previously defined. You can either remove all hooks by giving the "*" character as an argument, or you can remove all hooks of a specific type by saying something like `unhook send-hook`.

29. Format Strings

29.1. Basic usage

Format strings are a general concept you'll find in several locations through the Mutt configuration, especially in

the $index_format, $pager_format, $status_format, and other related variables. These can be very straightforward, and it's quite possible you already know how to use them.

The most basic format string element is a percent symbol followed by another character. For example, `%s` represents a message's Subject: header in the $index_format variable. The "expandos" available are documented with each format variable, but there are general modifiers available with all formatting expandos, too. Those are our concern here.

Some of the modifiers are borrowed right out of C (though you might know them from Perl, Python, shell, or another language). These are the `[-]m.n` modifiers, as in `%-12.12s`. As with such programming languages, these modifiers allow you to specify the minimum and maximum size of the resulting string, as well as its justification. If the "-" sign follows the percent, the string will be left-justified instead of right-justified. If there's a number immediately following that, it's the minimum amount of space the formatted string will occupy — if it's naturally smaller than that, it will be padded out with spaces. If a decimal point and another number follow, that's the maximum space allowable — the string will not be permitted to exceed that width, no matter its natural size. Each of these three elements is optional, so that all these are legal format strings: `%-12s`, `%4c`, `%.15F` and `%-12.15L`.

Mutt adds some other modifiers to format strings. If you use an equals symbol (=) as a numeric prefix (like the minus above), it will force the string to be centered within its minimum space range. For example, `%=14y` will reserve 14 characters for the %y expansion — that's the X-Label: header, in $index_format. If the expansion results in a string less than 14 characters, it will be centered in a 14-character space. If the X-Label for a message were "test", that expansion would look like " test ".

There are two very little-known modifiers that affect the way that an expando is replaced. If there is an underline ("_") character between any format modifiers (as above) and the expando letter, it will expands in all lower case. And if you use a colon (":"), it will replace all decimal points with underlines.

29.2. Conditionals

Depending on the format string variable, some of its sequences can be used to optionally print a string if their value is nonzero. For example, you may only want to see the number of flagged messages if such messages exist, since zero is not particularly meaningful. To optionally print a string based upon one of the above sequences, the following construct is used:

```
%?<sequence_char>?<optional_string>?
```

where *sequence_char* is an expando, and *optional_string* is the string you would like printed if *sequence_char* is nonzero. *optional_string* may contain other sequences as well as normal text, but you may not nest optional strings.

Here is an example illustrating how to optionally print the number of new messages in a mailbox in $status_format:

```
%?n?%n new messages.?
```

You can also switch between two strings using the following construct:

```
%?<sequence_char>?<if_string>&<else_string>?
```

If the value of *sequence_char* is non-zero, *if_string* will be expanded, otherwise *else_string* will be expanded.

29.3. Filters

Any format string ending in a vertical bar ("|") will be expanded and piped through the first word in the string, using spaces as separator. The string returned will be used for display. If the returned string ends in %, it will be passed through the formatter a second time. This allows the filter to generate a replacement format string including % expandos.

All % expandos in a format string are expanded before the script is called so that:

Example 3.19. Using external filters in format strings

```
set status_format="script.sh '%r %f (%L)'|"
```

will make Mutt expand `%r`, `%f` and `%L` before calling the script. The example also shows that arguments can be quoted: the script will receive the expanded string between the single quotes as the only argument.

A practical example is the `mutt_xtitle` script installed in the `samples` subdirectory of the Mutt documentation: it can be used as filter for $status_format to set the current terminal's title, if supported.

29.4. Padding

In most format strings, Mutt supports different types of padding using special %-expandos:

`%|X`

When this occurs, Mutt will fill the rest of the line with the character `X`. For example, filling the rest of the line with dashes is done by setting:

```
set status_format = "%v on %h: %B: %?n?%n&no? new messages %|-"
```

`%>X`

Since the previous expando stops at the end of line, there must be a way to fill the gap between two items via the `%>X` expando: it puts as many characters `X` in between two items so that the rest of the line will be right-justified. For example, to not put the version string and hostname the above example on the left but on the right and fill the gap with spaces, one might use (note the space after `%>`):

```
set status_format = "%B: %?n?%n&no? new messages %> (%v on %h)"
```

%*X

Normal right-justification will print everything to the left of the %>, displaying padding and whatever lies to the right only if there's room. By contrast, “soft-fill” gives priority to the right-hand side, guaranteeing space to display it and showing padding only if there's still room. If necessary, soft-fill will eat text leftwards to make room for rightward text. For example, to right-justify the subject making sure as much as possible of it fits on screen, one might use (note two spaces after %* : the second ensures there's a space between the truncated right-hand side and the subject):

```
set index_format="%4C %Z %{%b %d} %-15.15L (%?l?%4l&%4c?)%*  %s"
```

Chapter 4. Advanced Usage

Table of Contents

1. Character Set Handling

A “character set” is basically a mapping between bytes and glyphs and implies a certain character encoding scheme. For example, for the ISO 8859 family of character sets, an encoding of 8bit per character is used. For the Unicode character set, different character encodings may be used, UTF-8 being the most popular. In UTF-8, a character is represented using a variable number of bytes ranging from 1 to 4.

Since Mutt is a command-line tool run from a shell, and delegates certain tasks to external tools (such as an editor for composing/editing messages), all of these tools need to agree on a character set and encoding. There exists no way to reliably deduce the character set a plain text file has. Interoperability is gained by the use of well-defined environment variables. The full set can be printed by issuing `locale` on the command line.

Upon startup, Mutt determines the character set on its own using routines that inspect locale-specific environment variables. Therefore, it is generally not necessary to set the `$charset` variable in Mutt. It may even be counter-productive as Mutt uses system and library functions that derive the character set themselves and on which Mutt has no influence. It's safest to let Mutt work out the locale setup itself.

If you happen to work with several character sets on a regular basis, it's highly advisable to use Unicode and an UTF-8 locale. Unicode can represent nearly all characters in a message at the same time. When not using a Unicode locale, it may happen that you receive messages with characters not representable in your locale. When displaying such a message, or replying to or forwarding it, information may get lost possibly rendering the message unusable (not only for you but also for the recipient, this breakage is not reversible as lost information cannot be guessed).

A Unicode locale makes all conversions superfluous which eliminates the risk of conversion errors. It also eliminates potentially wrong expectations about the character set between Mutt and external programs.

The terminal emulator used also must be properly configured for the current locale. Terminal emulators usually do *not* derive the locale from environment variables, they need to be configured separately. If the terminal is incorrectly configured, Mutt may display random and unexpected characters (question marks, octal codes, or just random glyphs), format strings may not work as expected, you may not be abled to enter non-ascii characters, and possible more. Data is always represented using bytes and so a correct setup is very important as to the machine, all character sets “look” the same.

Warning: A mismatch between what system and library functions think the locale is and what Mutt was told what the locale is may make it behave badly with non-ascii input: it will fail at seemingly random places. This warning is to be taken seriously since not only local mail handling may suffer: sent messages may carry wrong character set information the *receiver* has too deal with. The need to set `$charset` directly in most cases points at terminal and environment variable setup problems, not Mutt problems.

A list of officially assigned and known character sets can be found at IANA, a list of locally supported locales can be obtained by running `locale -a`.

2. Regular Expressions

All string patterns in Mutt including those in more complex patterns must be specified using regular expressions (regexp) in the "POSIX extended" syntax (which is more or less the syntax used by egrep and GNU awk). For your convenience, we have included below a brief description of this syntax.

The search is case sensitive if the pattern contains at least one upper case letter, and case insensitive otherwise.

> ***NOTE:***
>
> "\" must be quoted if used for a regular expression in an initialization command: "\\".

A regular expression is a pattern that describes a set of strings. Regular expressions are constructed analogously to arithmetic expressions, by using various operators to combine smaller expressions.

> ***NOTE:***
>
> The regular expression can be enclosed/delimited by either " or ' which is useful if the regular expression includes a white-space character. See Syntax of Initialization Files for more information on " and ' delimiter processing. To match a literal " or ' you must preface it with \ (backslash).

The fundamental building blocks are the regular expressions that match a single character. Most characters, including all letters and digits, are regular expressions that match themselves. Any metacharacter with special meaning may be quoted by preceding it with a backslash.

The period "." matches any single character. The caret "^" and the dollar sign "$" are metacharacters that respectively match the empty string at the beginning and end of a line.

A list of characters enclosed by "[" and "]" matches any single character in that list; if the first character of the list is a caret "^" then it matches any character *not* in the list. For example, the regular expression *[0123456789]* matches any single digit. A range of ASCII characters may be specified by giving the first and last characters, separated by a hyphen "-". Most metacharacters lose their special meaning inside lists. To include a literal "]" place it first in the list. Similarly, to include a literal "^" place it anywhere but first. Finally, to include a literal hyphen "-" place it last.

Certain named classes of characters are predefined. Character classes consist of "[:", a keyword denoting the class, and ":]". The following classes are defined by the POSIX standard in Table 4.1, "POSIX regular expression character classes"

Table 4.1. POSIX regular expression character classes

Character class	Description
[:alnum:]	Alphanumeric characters

[:alpha:]	Alphabetic characters
[:blank:]	Space or tab characters
[:cntrl:]	Control characters
[:digit:]	Numeric characters
[:graph:]	Characters that are both printable and visible. (A space is printable, but not visible, while an “a” is both)
[:lower:]	Lower-case alphabetic characters
[:print:]	Printable characters (characters that are not control characters)
[:punct:]	Punctuation characters (characters that are not letter, digits, control characters, or space characters)
[:space:]	Space characters (such as space, tab and formfeed, to name a few)
[:upper:]	Upper-case alphabetic characters
[:xdigit:]	Characters that are hexadecimal digits

A character class is only valid in a regular expression inside the brackets of a character list.

> ***NOTE:***
>
> Note that the brackets in these class names are part of the symbolic names, and must be included in addition to the brackets delimiting the bracket list. For example, *[[:digit:]]* is equivalent to *[0-9]*.

Two additional special sequences can appear in character lists. These apply to non-ASCII character sets, which can have single symbols (called collating elements) that are represented with more than one character, as well as several characters that are equivalent for collating or sorting purposes:

Collating Symbols

> A collating symbol is a multi-character collating element enclosed in “[.” and “.]”. For example, if “ch” is a collating element, then *[[.ch.]]* is a regexp that matches this collating element, while *[ch]* is a regexp that matches either “c” or “h”.

Equivalence Classes

> An equivalence class is a locale-specific name for a list of characters that are equivalent. The name is enclosed in “[=” and “=]”. For example, the name “e” might be used to represent all of “e” with grave (“è”), “e” with acute (“é”) and “e”. In this case, *[[=e=]]* is a regexp that matches any of: “e” with grave (“è”), “e” with acute (“é”) and “e”.

A regular expression matching a single character may be followed by one of several repetition operators described in Table 4.2, "Regular expression repetition operators".

Table 4.2. Regular expression repetition operators

Operator	Description
?	The preceding item is optional and matched at most once
*	The preceding item will be matched zero or more times
+	The preceding item will be matched one or more times
{n}	The preceding item is matched exactly *n* times
{n,}	The preceding item is matched *n* or more times
{,m}	The preceding item is matched at most *m* times
{n,m}	The preceding item is matched at least *n* times, but no more than *m* times

Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two substrings that respectively match the concatenated subexpressions.

Two regular expressions may be joined by the infix operator "|"; the resulting regular expression matches any string matching either subexpression.

Repetition takes precedence over concatenation, which in turn takes precedence over alternation. A whole subexpression may be enclosed in parentheses to override these precedence rules.

> ***NOTE:***
>
> If you compile Mutt with the included regular expression engine, the following operators may also be used in regular expressions as described in Table 4.3, "GNU regular expression extensions".

Table 4.3. GNU regular expression extensions

Expression	Description
\\y	Matches the empty string at either the beginning or the end of a word
\\B	Matches the empty string within a word
\\<	Matches the empty string at the beginning of a word
\\>	Matches the empty string at the end of a word
\\w	Matches any word-constituent character (letter, digit, or underscore)

\\W	Matches any character that is not word-constituent
\\`	Matches the empty string at the beginning of a buffer (string)
\\'	Matches the empty string at the end of a buffer

Please note however that these operators are not defined by POSIX, so they may or may not be available in stock libraries on various systems.

3. Patterns: Searching, Limiting and Tagging

3.1. Pattern Modifier

Many of Mutt's commands allow you to specify a pattern to match (`limit`, `tag-pattern`, `delete-pattern`, etc.). Table 4.4, "Pattern modifiers" shows several ways to select messages.

Table 4.4. Pattern modifiers

Pattern modifier	Description
~A	all messages
~b *EXPR*	messages which contain *EXPR* in the message body
=b *STRING*	messages which contain *STRING* in the message body. If IMAP is enabled, searches for *STRING* on the server, rather than downloading each message and searching it locally.
~B *EXPR*	messages which contain *EXPR* in the whole message
~c *EXPR*	messages carbon-copied to *EXPR*
%c *GROUP*	messages carbon-copied to any member of *GROUP*
~C *EXPR*	messages either to: or cc: *EXPR*
%C *GROUP*	messages either to: or cc: to any member of *GROUP*
~d [*MIN*]-[*MAX*]	messages with "date-sent" in a Date range
~D	deleted messages
~e *EXPR*	messages which contains *EXPR* in the "Sender" field
%e *GROUP*	messages which contain a member of *GROUP* in the "Sender" field
~E	expired messages
~F	flagged messages

~f *EXPR*	messages originating from *EXPR*
%f *GROUP*	messages originating from any member of *GROUP*
~g	cryptographically signed messages
~G	cryptographically encrypted messages
~h *EXPR*	messages which contain *EXPR* in the message header
~H *EXPR*	messages with a spam attribute matching *EXPR*
~i *EXPR*	messages which match *EXPR* in the “Message-ID” field
~k	messages which contain PGP key material
~L *EXPR*	messages either originated or received by *EXPR*
%L *GROUP*	message either originated or received by any member of *GROUP*
~l	messages addressed to a known mailing list
~m [*MIN*]-[*MAX*]	messages in the range *MIN* to *MAX* *)
~n [*MIN*]-[*MAX*]	messages with a score in the range *MIN* to *MAX* *)
~N	new messages
~O	old messages
~p	messages addressed to you (consults `alternates`)
~P	messages from you (consults `alternates`)
~Q	messages which have been replied to
~r [*MIN*]-[*MAX*]	messages with “date-received” in a Date range
~R	read messages
~s *EXPR*	messages having *EXPR* in the “Subject” field.
~S	superseded messages
~t *EXPR*	messages addressed to *EXPR*
~T	tagged messages
~u	messages addressed to a subscribed mailing list
~U	unread messages

~v	messages part of a collapsed thread.
~V	cryptographically verified messages
~x *EXPR*	messages which contain *EXPR* in the "References" or "In-Reply-To" field
~X [*MIN*]-[*MAX*]	messages with *MIN* to *MAX* attachments *)
~y *EXPR*	messages which contain *EXPR* in the "X-Label" field
~z [*MIN*]-[*MAX*]	messages with a size in the range *MIN* to *MAX* *) **)
~=	duplicated messages (see $duplicate_threads)
~$	unreferenced messages (requires threaded view)
~(*PATTERN*)	messages in threads containing messages matching *PATTERN*, e.g. all threads containing messages from you: ~(~P)

Where *EXPR* is a regular expression, and *GROUP* is an address group.

*) The forms "<[*MAX*]", ">[*MIN*]", "[*MIN*]-" and "-[*MAX*]" are allowed, too.

**) The suffixes "K" and "M" are allowed to specify kilobyte and megabyte respectively.

Special attention has to be payed when using regular expressions inside of patterns. Specifically, Mutt's parser for these patterns will strip one level of backslash ("\"), which is normally used for quoting. If it is your intention to use a backslash in the regular expression, you will need to use two backslashes instead ("\\"). You can force Mutt to treat *EXPR* as a simple string instead of a regular expression by using = instead of ~ in the pattern name. For example, `=b *.*` will find all messages that contain the literal string "*.*". Simple string matches are less powerful than regular expressions but can be considerably faster. This is especially true for IMAP folders, because string matches can be performed on the server instead of by fetching every message. IMAP treats `=h` specially: it must be of the form "header: substring" and will not partially match header names. The substring part may be omitted if you simply wish to find messages containing a particular header without regard to its value.

Patterns matching lists of addresses (notably c, C, p, P and t) match if there is at least one match in the whole list. If you want to make sure that all elements of that list match, you need to prefix your pattern with "^". This example matches all mails which only has recipients from Germany.

Example 4.1. Matching all addresses in address lists

```
^~C \.de$
```

3.2. Simple Searches

Mutt supports two versions of so called “simple searches”. These are issued if the query entered for searching, limiting and similar operations does not seem to contain a valid pattern modifier (i.e. it does not contain one of these characters: “~”, “=” or “%”). If the query is supposed to contain one of these special characters, they must be escaped by prepending a backslash (“\”).

The first type is by checking whether the query string equals a keyword case-insensitively from Table 4.5, “Simple search keywords”: If that is the case, Mutt will use the shown pattern modifier instead. If a keyword would conflict with your search keyword, you need to turn it into a regular expression to avoid matching the keyword table. For example, if you want to find all messages matching “flag” (using $simple_search) but don't want to match flagged messages, simply search for “`[f]lag`”.

Table 4.5. Simple search keywords

Keyword	Pattern modifier
all	~A
.	~A
^	~A
del	~D
flag	~F
new	~N
old	~O
repl	~Q
read	~R
tag	~T
unread	~U

The second type of simple search is to build a complex search pattern using $simple_search as a template. Mutt will insert your query properly quoted and search for the composed complex query.

3.3. Nesting and Boolean Operators

Logical AND is performed by specifying more than one criterion. For example:

```
~t mutt ~f elkins
```

would select messages which contain the word “mutt” in the list of recipients *and* that have the word “elkins” in

the “From” header field.

Mutt also recognizes the following operators to create more complex search patterns:

- ! — logical NOT operator
- | — logical OR operator
- () — logical grouping operator

Here is an example illustrating a complex search pattern. This pattern will select all messages which do not contain “mutt” in the “To” or “Cc” field and which are from “elkins”.

Example 4.2. Using boolean operators in patterns

```
!(~t mutt|~c mutt) ~f elkins
```

Here is an example using white space in the regular expression (note the “"” and “"” delimiters). For this to match, the mail's subject must match the “^Junk +From +Me$” and it must be from either “Jim +Somebody” or “Ed +SomeoneElse”:

```
'~s "^Junk +From +Me$" ~f ("Jim +Somebody"|"Ed +SomeoneElse")'
```

Note:

If a regular expression contains parenthesis, or a vertical bar ("|"), you *must* enclose the expression in double or single quotes since those characters are also used to separate different parts of Mutt's pattern language. For example: `~f "me@(mutt\.org|cs\.hmc\.edu)"` Without the quotes, the parenthesis wouldn't end. This would be separated to two OR'd patterns: *~f me@(mutt\.org* and *cs\.hmc\.edu)*. They are never what you want.

3.4. Searching by Date

Mutt supports two types of dates, *absolute* and *relative*.

3.4.1. Absolute Dates

Dates *must* be in DD/MM/YY format (month and year are optional, defaulting to the current month and year). An example of a valid range of dates is:

```
Limit to messages matching: ~d 20/1/95-31/10
```

If you omit the minimum (first) date, and just specify “-DD/MM/YY”, all messages *before* the given date will

be selected. If you omit the maximum (second) date, and specify "DD/MM/YY-", all messages *after* the given date will be selected. If you specify a single date with no dash ("-"), only messages sent on the given date will be selected.

You can add error margins to absolute dates. An error margin is a sign (+ or -), followed by a digit, followed by one of the units in Table 4.6, "Date units". As a special case, you can replace the sign by a "*" character, which is equivalent to giving identical plus and minus error margins.

Table 4.6. Date units

Unit	Description
y	Years
m	Months
w	Weeks
d	Days

Example: To select any messages two weeks around January 15, 2001, you'd use the following pattern:

```
Limit to messages matching: ~d 15/1/2001*2w
```

3.4.2. Relative Dates

This type of date is relative to the current date, and may be specified as:

- >*offset* for messages older than *offset* units
- <*offset* for messages newer than *offset* units
- =*offset* for messages exactly *offset* units old

offset is specified as a positive number with one of the units from Table 4.6, "Date units".

Example: to select messages less than 1 month old, you would use

```
Limit to messages matching: ~d <1m
```

> ***NOTE:***
>
> All dates used when searching are relative to the *local* time zone, so unless you change the setting of your $index_format to include a `%[...]` format, these are *not* the dates shown in the main index.

4. Using Tags

Sometimes it is desirable to perform an operation on a group of messages all at once rather than one at a time. An example might be to save messages to a mailing list to a separate folder, or to delete all messages with a given subject. To tag all messages matching a pattern, use the `<tag-pattern>` function, which is bound to "shift-T" by default. Or you can select individual messages by hand using the `<tag-message>` function, which is bound to "t" by default. See patterns for Mutt's pattern matching syntax.

Once you have tagged the desired messages, you can use the "tag-prefix" operator, which is the ";" (semicolon) key by default. When the "tag-prefix" operator is used, the *next* operation will be applied to all tagged messages if that operation can be used in that manner. If the $auto_tag variable is set, the next operation applies to the tagged messages automatically, without requiring the "tag-prefix".

In `macros` or `push` commands, you can use the `<tag-prefix-cond>` operator. If there are no tagged messages, Mutt will "eat" the rest of the macro to abort it's execution. Mutt will stop "eating" the macro when it encounters the `<end-cond>` operator; after this operator the rest of the macro will be executed as normal.

5. Using Hooks

A *hook* is a concept found in many other programs which allows you to execute arbitrary commands before performing some operation. For example, you may wish to tailor your configuration based upon which mailbox you are reading, or to whom you are sending mail. In the Mutt world, a *hook* consists of a regular expression or pattern along with a configuration option/command. See:

- `account-hook`
- `charset-hook`
- `crypt-hook`
- `fcc-hook`
- `fcc-save-hook`
- `folder-hook`
- `iconv-hook`
- `mbox-hook`
- `message-hook`
- `reply-hook`
- `save-hook`

- send-hook
- send2-hook

for specific details on each type of *hook* available.

> ***NOTE:***
>
> If a hook changes configuration settings, these changes remain effective until the end of the current Mutt session. As this is generally not desired, a "default" hook needs to be added before all other hooks of that type to restore configuration defaults.

Example 4.3. Specifying a "default" hook

```
send-hook . 'unmy_hdr From:'
send-hook ~C'^b@b\.b$' my_hdr from: c@c.c
```

In Example 4.3, "Specifying a "default" hook", by default the value of $from and $realname is not overridden. When sending messages either To: or Cc: to <b@b.b>, the From: header is changed to <c@c.c>.

5.1. Message Matching in Hooks

Hooks that act upon messages (message-hook, reply-hook, send-hook, send2-hook, save-hook, fcc-hook) are evaluated in a slightly different manner. For the other types of hooks, a regular expression is sufficient. But in dealing with messages a finer grain of control is needed for matching since for different purposes you want to match different criteria.

Mutt allows the use of the search pattern language for matching messages in hook commands. This works in exactly the same way as it would when *limiting* or *searching* the mailbox, except that you are restricted to those operators which match information Mutt extracts from the header of the message (i.e., from, to, cc, date, subject, etc.).

For example, if you wanted to set your return address based upon sending mail to a specific address, you could do something like:

```
send-hook '~t ^me@cs\.hmc\.edu$' 'my_hdr From: Mutt User <user@host>'
```

which would execute the given command when sending mail to *me@cs.hmc.edu*.

However, it is not required that you write the pattern to match using the full searching language. You can still specify a simple *regular expression* like the other hooks, in which case Mutt will translate your pattern into the full language, using the translation specified by the $default_hook variable. The pattern is translated at the time the hook is declared, so the value of $default_hook that is in effect at that time will be used.

6. External Address Queries

Mutt supports connecting to external directory databases such as LDAP, ph/qi, bbdb, or NIS through a wrapper script which connects to Mutt using a simple interface. Using the $query_command variable, you specify the wrapper command to use. For example:

```
set query_command = "mutt_ldap_query.pl %s"
```

The wrapper script should accept the query on the command-line. It should return a one line message, then each matching response on a single line, each line containing a tab separated address then name then some other optional information. On error, or if there are no matching addresses, return a non-zero exit code and a one line error message.

An example multiple response output:

```
Searching database ... 20 entries ... 3 matching:
me@cs.hmc.edu           Michael Elkins  mutt dude
blong@fiction.net       Brandon Long    mutt and more
roessler@does-not-exist.org        Thomas Roessler mutt pgp
```

There are two mechanisms for accessing the query function of Mutt. One is to do a query from the index menu using the `<query>` function (default: Q). This will prompt for a query, then bring up the query menu which will list the matching responses. From the query menu, you can select addresses to create aliases, or to mail. You can tag multiple addresses to mail, start a new query, or have a new query appended to the current responses.

The other mechanism for accessing the query function is for address completion, similar to the alias completion. In any prompt for address entry, you can use the `<complete-query>` function (default: ^T) to run a query based on the current address you have typed. Like aliases, Mutt will look for what you have typed back to the last space or comma. If there is a single response for that query, Mutt will expand the address in place. If there are multiple responses, Mutt will activate the query menu. At the query menu, you can select one or more addresses to be added to the prompt.

7. Mailbox Formats

Mutt supports reading and writing of four different local mailbox formats: mbox, MMDF, MH and Maildir. The mailbox type is auto detected, so there is no need to use a flag for different mailbox types. When creating new mailboxes, Mutt uses the default specified with the $mbox_type variable. A short description of the formats follows.

mbox. This is a widely used mailbox format for UNIX. All messages are stored in a single file. Each message has a line of the form:

```
From me@cs.hmc.edu Fri, 11 Apr 1997 11:44:56 PST
```

to denote the start of a new message (this is often referred to as the “From_” line). The mbox format requires

mailbox locking, is prone to mailbox corruption with concurrently writing clients or misinterpreted From_ lines. Depending on the environment, new mail detection can be unreliable. Mbox folders are fast to open and easy to archive.

MMDF. This is a variant of the *mbox* format. Each message is surrounded by lines containing "^A^A^A^A" (four times control-A's). The same problems as for mbox apply (also with finding the right message separator as four control-A's may appear in message bodies).

MH. A radical departure from *mbox* and *MMDF*, a mailbox consists of a directory and each message is stored in a separate file. The filename indicates the message number (however, this is may not correspond to the message number Mutt displays). Deleted messages are renamed with a comma (",") prepended to the filename. Mutt detects this type of mailbox by looking for either `.mh_sequences` or `.xmhcache` files (needed to distinguish normal directories from MH mailboxes). MH is more robust with concurrent clients writing the mailbox, but still may suffer from lost flags; message corruption is less likely to occur than with mbox/mmdf. It's usually slower to open compared to mbox/mmdf since many small files have to be read (Mutt provides Section 7.1, "Header Caching" to greatly speed this process up). Depending on the environment, MH is not very disk-space efficient.

Maildir. The newest of the mailbox formats, used by the Qmail MTA (a replacement for sendmail). Similar to *MH*, except that it adds three subdirectories of the mailbox: *tmp*, *new* and *cur*. Filenames for the messages are chosen in such a way they are unique, even when two programs are writing the mailbox over NFS, which means that no file locking is needed and corruption is very unlikely. Maildir maybe slower to open without caching in Mutt, it too is not very disk-space efficient depending on the environment. Since no additional files are used for metadata (which is embedded in the message filenames) and Maildir is locking-free, it's easy to sync across different machines using file-level synchronization tools.

8. Mailbox Shortcuts

There are a number of built in shortcuts which refer to specific mailboxes. These shortcuts can be used anywhere you are prompted for a file or mailbox path or in path-related configuration variables. Note that these only work at the beginning of a string.

Table 4.7. Mailbox shortcuts

Shortcut	Refers to...
!	your $spoolfile (incoming) mailbox
>	your $mbox file
<	your $record file
^	the current mailbox
- or !!	the file you've last visited
~	your home directory

= or +	your $folder directory
@alias	to the default save folder as determined by the address of the alias

For example, to store a copy of outgoing messages in the folder they were composed in, a `folder-hook` can be used to set $record:

```
folder-hook . 'set record=^'
```

9. Handling Mailing Lists

Mutt has a few configuration options that make dealing with large amounts of mail easier. The first thing you must do is to let Mutt know what addresses you consider to be mailing lists (technically this does not have to be a mailing list, but that is what it is most often used for), and what lists you are subscribed to. This is accomplished through the use of the `lists and subscribe` commands in your `.muttrc`.

Now that Mutt knows what your mailing lists are, it can do several things, the first of which is the ability to show the name of a list through which you received a message (i.e., of a subscribed list) in the *index* menu display. This is useful to distinguish between personal and list mail in the same mailbox. In the $index_format variable, the expando "%L" will print the string "To <list>" when "list" appears in the "To" field, and "Cc <list>" when it appears in the "Cc" field (otherwise it prints the name of the author).

Often times the "To" and "Cc" fields in mailing list messages tend to get quite large. Most people do not bother to remove the author of the message they reply to from the list, resulting in two or more copies being sent to that person. The `<list-reply>` function, which by default is bound to "L" in the *index* menu and *pager*, helps reduce the clutter by only replying to the known mailing list addresses instead of all recipients (except as specified by `Mail-Followup-To`, see below).

Mutt also supports the `Mail-Followup-To` header. When you send a message to a list of recipients which includes one or several subscribed mailing lists, and if the $followup_to option is set, Mutt will generate a Mail-Followup-To header which contains all the recipients to whom you send this message, but not your address. This indicates that group-replies or list-replies (also known as "followups") to this message should only be sent to the original recipients of the message, and not separately to you - you'll receive your copy through one of the mailing lists you are subscribed to.

Conversely, when group-replying or list-replying to a message which has a `Mail-Followup-To` header, Mutt will respect this header if the $honor_followup_to configuration variable is set. Using list-reply will in this case also make sure that the reply goes to the mailing list, even if it's not specified in the list of recipients in the `Mail-Followup-To`.

> ***Note:***
>
> When header editing is enabled, you can create a `Mail-Followup-To` header manually. Mutt will only auto-generate this header if it doesn't exist when you send the message.

The other method some mailing list admins use is to generate a "Reply-To" field which points back to the mailing list address rather than the author of the message. This can create problems when trying to reply directly to the author in private, since most mail clients will automatically reply to the address given in the "Reply-To" field. Mutt uses the $reply_to variable to help decide which address to use. If set to *ask-yes* or *ask-no*, you will be prompted as to whether or not you would like to use the address given in the "Reply-To" field, or reply directly to the address given in the "From" field. When set to *yes*, the "Reply-To" field will be used when present.

The "X-Label:" header field can be used to further identify mailing lists or list subject matter (or just to annotate messages individually). The $index_format variable's "%y" and "%Y" expandos can be used to expand "X-Label:" fields in the index, and Mutt's pattern-matcher can match regular expressions to "X-Label:" fields with the "~y" selector. "X-Label:" is not a standard message header field, but it can easily be inserted by procmail and other mail filtering agents.

Lastly, Mutt has the ability to sort the mailbox into threads. A thread is a group of messages which all relate to the same subject. This is usually organized into a tree-like structure where a message and all of its replies are represented graphically. If you've ever used a threaded news client, this is the same concept. It makes dealing with large volume mailing lists easier because you can easily delete uninteresting threads and quickly find topics of value.

10. New Mail Detection

Mutt supports setups with multiple folders, allowing all of them to be monitored for new mail (see Section 14, "Monitoring Incoming Mail" for details).

10.1. How New Mail Detection Works

For Mbox and Mmdf folders, new mail is detected by comparing access and/or modification times of files: Mutt assumes a folder has new mail if it wasn't accessed after it was last modified. Utilities like `biff` or `frm` or any other program which accesses the mailbox might cause Mutt to never detect new mail for that mailbox if they do not properly reset the access time. Other possible causes of Mutt not detecting new mail in these folders are backup tools (updating access times) or filesystems mounted without access time update support (for Linux systems, see the `relatime` option).

> ***Note:***
>
> Contrary to older Mutt releases, it now maintains the new mail status of a folder by properly resetting the access time if the folder contains at least one message which is neither read, nor deleted, nor marked as old.

In cases where new mail detection for Mbox or Mmdf folders appears to be unreliable, the $check_mbox_size option can be used to make Mutt track and consult file sizes for new mail detection instead which won't work

for size-neutral changes.

New mail for Maildir is assumed if there is one message in the `new/` subdirectory which is not marked deleted (see $maildir_trash). For MH folders, a mailbox is considered having new mail if there's at least one message in the "unseen" sequence as specified by $mh_seq_unseen.

Mutt does not poll POP3 folders for new mail, it only periodically checks the currently opened folder (if it's a POP3 folder).

For IMAP, by default Mutt uses recent message counts provided by the server to detect new mail. If the $imap_idle option is set, it'll use the IMAP IDLE extension if advertised by the server.

10.2. Polling For New Mail

When in the index menu and being idle (also see $timeout), Mutt periodically checks for new mail in all folders which have been configured via the `mailboxes` command. The interval depends on the folder type: for local/IMAP folders it consults $mail_check and $pop_checkinterval for POP folders.

Outside the index menu the directory browser supports checking for new mail using the `<check-new>` function which is unbound by default. Pressing TAB will bring up a menu showing the files specified by the `mailboxes` command, and indicate which contain new messages. Mutt will automatically enter this mode when invoked from the command line with the `-y` option.

For the pager, index and directory browser menus, Mutt contains the `<buffy-list>` function (bound to "." by default) which will print a list of folders with new mail in the command line at the bottom of the screen.

For the index, by default Mutt displays the number of mailboxes with new mail in the status bar, please refer to the $status_format variable for details.

When changing folders, Mutt fills the prompt with the first folder from the mailboxes list containing new mail (if any), pressing `<Space>` will cycle through folders with new mail. The (by default unbound) function `<next-unread-mailbox>` in the index can be used to immediately open the next folder with unread mail (if any).

11. Editing Threads

Mutt has the ability to dynamically restructure threads that are broken either by misconfigured software or bad behavior from some correspondents. This allows to clean your mailboxes from these annoyances which make it hard to follow a discussion.

11.1. Linking Threads

Some mailers tend to "forget" to correctly set the "In-Reply-To:" and "References:" headers when replying to a message. This results in broken discussions because Mutt has not enough information to guess the correct threading. You can fix this by tagging the reply, then moving to the parent message and using the `<link-threads>` function (bound to & by default). The reply will then be connected to this parent message.

You can also connect multiple children at once, tagging them and using the <tag-prefix> command (";") or the $auto_tag option.

11.2. Breaking Threads

On mailing lists, some people are in the bad habit of starting a new discussion by hitting "reply" to any message from the list and changing the subject to a totally unrelated one. You can fix such threads by using the <break-thread> function (bound by default to #), which will turn the subthread starting from the current message into a whole different thread.

12. Delivery Status Notification (DSN) Support

RFC1894 defines a set of MIME content types for relaying information about the status of electronic mail messages. These can be thought of as "return receipts."

To support DSN, there are two variables. $dsn_notify is used to request receipts for different results (such as failed message, message delivered, etc.). $dsn_return requests how much of your message should be returned with the receipt (headers or full message).

When using $sendmail for mail delivery, you need to use either Berkeley sendmail 8.8.x (or greater) a MTA supporting DSN command line options compatible to Sendmail: The -N and -R options can be used by the mail client to make requests as to what type of status messages should be returned. Please consider your MTA documentation whether DSN is supported.

For SMTP delivery using $smtp_url, it depends on the capabilities announced by the server whether Mutt will attempt to request DSN or not.

13. Start a WWW Browser on URLs

If a message contains URLs, it is efficient to get a menu with all the URLs and start a WWW browser on one of them. This functionality is provided by the external urlview program which can be retrieved at ftp://ftp.mutt.org/mutt/contrib/ and the configuration commands:

```
macro index \cb |urlview\n
macro pager \cb |urlview\n
```

14. Miscellany

This section documents various features that fit nowhere else.

Address normalization

> Mutt normalizes all e-mail addresses to the simplest form possible. If an address contains a realname, the form *Joe User <joe@example.com>* is used and the pure e-mail address without angle brackets

otherwise, i.e. just *joe@example.com*.

This normalization affects all headers Mutt generates including aliases.

Initial folder selection

The folder Mutt opens at startup is determined as follows: the folder specified in the `$MAIL` environment variable if present. Otherwise, the value of `$MAILDIR` is taken into account. If that isn't present either, Mutt takes the user's mailbox in the mailspool as determined at compile-time (which may also reside in the home directory). The $spoolfile setting overrides this selection. Highest priority has the mailbox given with the `-f` command line option.

Chapter 5. Mutt's MIME Support

Table of Contents

Quite a bit of effort has been made to make Mutt the premier text-mode MIME MUA. Every effort has been made to provide the functionality that the discerning MIME user requires, and the conformance to the standards wherever possible. When configuring Mutt for MIME, there are two extra types of configuration files which Mutt uses. One is the `mime.types` file, which contains the mapping of file extensions to IANA MIME types. The other is the `mailcap` file, which specifies the external commands to use for handling specific MIME types.

1. Using MIME in Mutt

1.1. MIME Overview

MIME is short for “Multipurpose Internet Mail Extension” and describes mechanisms to internationalize and structure mail messages. Before the introduction of MIME, messages had a single text part and were limited to us-ascii header and content. With MIME, messages can have attachments (and even attachments which itself have attachments and thus form a tree structure), nearly arbitrary characters can be used for sender names, recipients and subjects.

Besides the handling of non-ascii characters in message headers, to Mutt the most important aspect of MIME are so-called MIME types. These are constructed using a *major* and *minor* type separated by a forward slash. These specify details about the content that follows. Based upon these, Mutt decides how to handle this part. The most popular major type is “`text`” with minor types for plain text, HTML and various other formats. Major types also exist for images, audio, video and of course general application data (e.g. to separate cryptographically signed data with a signature, send office documents, and in general arbitrary binary data). There's also the `multipart` major type which represents the root of a subtree of MIME parts. A list of supported MIME types can be found in Table 5.1, “Supported MIME types”.

MIME also defines a set of encoding schemes for transporting MIME content over the network: `7bit`, `8bit`, `quoted-printable`, `base64` and `binary`. There're some rules when to choose what for encoding headers and/or body (if needed), and Mutt will in general make a good choice.

Mutt does most of MIME encoding/decoding behind the scenes to form messages conforming to MIME on the sending side. On reception, it can be flexibly configured as to how what MIME structure is displayed (and if it's displayed): these decisions are based on the content's MIME type. There are three areas/menus in dealing with MIME: the pager (while viewing a message), the attachment menu and the compose menu.

1.2. Viewing MIME Messages in the Pager

When you select a message from the index and view it in the pager, Mutt decodes as much of a message as possible to a text representation. Mutt internally supports a number of MIME types, including the `text` major type (with all minor types), the `message/rfc822` (mail messages) type and some `multipart` types. In addition, it recognizes a variety of PGP MIME types, including PGP/MIME and `application/pgp`.

Mutt will denote attachments with a couple lines describing them. These lines are of the form:

```
[-- Attachment #1: Description --]
[-- Type: text/plain, Encoding: 7bit, Size: 10000 --]
```

Where the *Description* is the description or filename given for the attachment, and the *Encoding* is one of the already mentioned content encodings.

If Mutt cannot deal with a MIME type, it will display a message like:

```
[-- image/gif is unsupported (use 'v' to view this part) --]
```

1.3. The Attachment Menu

The default binding for `<view-attachments>` is “v”, which displays the attachment menu for a message. The

attachment menu displays a list of the attachments in a message. From the attachment menu, you can save, print, pipe, delete, and view attachments. You can apply these operations to a group of attachments at once, by tagging the attachments and by using the `<tag-prefix>` operator. You can also reply to the current message from this menu, and only the current attachment (or the attachments tagged) will be quoted in your reply. You can view attachments as text, or view them using the mailcap viewer definition (the mailcap mechanism is explained later in detail).

Finally, you can apply the usual message-related functions (like `<resend-message>`, and the `<reply>` and `<forward>` functions) to attachments of type `message/rfc822`.

See table Table 9.7, "Default Attachment Menu Bindings" for all available functions.

1.4. The Compose Menu

The compose menu is the menu you see before you send a message. It allows you to edit the recipient list, the subject, and other aspects of your message. It also contains a list of the attachments of your message, including the main body. From this menu, you can print, copy, filter, pipe, edit, compose, review, and rename an attachment or a list of tagged attachments. You can also modifying the attachment information, notably the type, encoding and description.

Attachments appear as follows by default:

```
- 1 [text/plain, 7bit, 1K]           /tmp/mutt-euler-8082-0 <no description>
  2 [applica/x-gunzip, base64, 422K] ~/src/mutt-0.85.tar.gz <no description>
```

The "-" denotes that Mutt will delete the file after sending (or postponing, or canceling) the message. It can be toggled with the `<toggle-unlink>` command (default: u). The next field is the MIME content-type, and can be changed with the `<edit-type>` command (default: ^T). The next field is the encoding for the attachment, which allows a binary message to be encoded for transmission on 7bit links. It can be changed with the `<edit-encoding>` command (default: ^E). The next field is the size of the attachment, rounded to kilobytes or megabytes. The next field is the filename, which can be changed with the `<rename-file>` command (default: R). The final field is the description of the attachment, and can be changed with the `<edit-description>` command (default: d). See $attach_format for a full list of available expandos to format this display to your needs.

2. MIME Type Configuration with `mime.types`

To get most out of MIME, it's important that a MIME part's content type matches the content as closely as possible so that the recipient's client can automatically select the right viewer for the content. However, there's no reliable for Mutt to know how to detect every possible file type. Instead, it uses a simple plain text mapping file that specifies what file extension corresponds to what MIME type. This file is called `mime.types`.

When you add an attachment to your mail message, Mutt searches your personal `mime.types` file at `$HOME/.mime.types`, and then the system `mime.types` file at `/usr/local/share/mutt/mime.types` or `/etc/mime.types`

Each line starts with the full MIME type, followed by a space and space-separated list of file extensions. For example you could use:

Example 5.1. `mime.types`

```
application/postscript          ps eps
application/pgp                 pgp
audio/x-aiff                    aif aifc aiff
```

A sample `mime.types` file comes with the Mutt distribution, and should contain most of the MIME types you are likely to use.

If Mutt can not determine the MIME type by the extension of the file you attach, it will look at the file. If the file is free of binary information, Mutt will assume that the file is plain text, and mark it as `text/plain`. If the file contains binary information, then Mutt will mark it as `application/octet-stream`. You can change the MIME type that Mutt assigns to an attachment by using the `<edit-type>` command from the compose menu (default: ^T), see Table 5.1, "Supported MIME types" for supported major types. Mutt recognizes all of these if the appropriate entry is found in the `mime.types` file. Non-recognized mime types should only be used if the recipient of the message is likely to be expecting such attachments.

Table 5.1. Supported MIME types

MIME major type	Standard	Description
`application`	yes	General application data
`audio`	yes	Audio data
`image`	yes	Image data
`message`	yes	Mail messages, message status information
`model`	yes	VRML and other modeling data
`multipart`	yes	Container for other MIME parts
`text`	yes	Text data
`video`	yes	Video data
`chemical`	no	Mostly molecular data

MIME types are not arbitrary, they need to be assigned by IANA.

3. MIME Viewer Configuration with Mailcap

Mutt supports RFC 1524 MIME Configuration, in particular the Unix specific format specified in Appendix A

of RFC 1524. This file format is commonly referred to as the “mailcap” format. Many MIME compliant programs utilize the mailcap format, allowing you to specify handling for all MIME types in one place for all programs. Programs known to use this format include Firefox, lynx and metamail.

In order to handle various MIME types that Mutt doesn't have built-in support for, it parses a series of external configuration files to find an external handler. The default search string for these files is a colon delimited list containing the following files:

1. `$HOME/.mailcap`
2. `$PKGDATADIR/mailcap`
3. `$SYSCONFDIR/mailcap`
4. `/etc/mailcap`
5. `/usr/etc/mailcap`
6. `/usr/local/etc/mailcap`

where `$HOME` is your home directory. The `$PKGDATADIR` and the `$SYSCONFDIR` directories depend on where Mutt is installed: the former is the default for shared data, the latter for system configuration files.

The default search path can be obtained by running the following command:

```
mutt -nF /dev/null -Q mailcap_path
```

In particular, the metamail distribution will install a mailcap file, usually as `/usr/local/etc/mailcap`, which contains some baseline entries.

3.1. The Basics of the Mailcap File

A mailcap file consists of a series of lines which are comments, blank, or definitions.

A comment line consists of a # character followed by anything you want.

A blank line is blank.

A definition line consists of a content type, a view command, and any number of optional fields. Each field of a definition line is divided by a semicolon “;” character.

The content type is specified in the MIME standard “type/subtype” notation. For example, `text/plain`, `text/html`, `image/gif`, etc. In addition, the mailcap format includes two formats for wildcards, one using the special “*” subtype, the other is the implicit wild, where you only include the major type. For example, `image/*`, or `video` will match all image types and video types, respectively.

The view command is a Unix command for viewing the type specified. There are two different types of

commands supported. The default is to send the body of the MIME message to the command on stdin. You can change this behavior by using `%s` as a parameter to your view command. This will cause Mutt to save the body of the MIME message to a temporary file, and then call the view command with the `%s` replaced by the name of the temporary file. In both cases, Mutt will turn over the terminal to the view program until the program quits, at which time Mutt will remove the temporary file if it exists. This means that mailcap does *not* work out of the box with programs which detach themselves from the terminal right after starting, like `open` on Mac OS X. In order to nevertheless use these programs with mailcap, you probably need custom shell scripts.

So, in the simplest form, you can send a `text/plain` message to the external pager more on standard input:

```
text/plain; more
```

Or, you could send the message as a file:

```
text/plain; more %s
```

Perhaps you would like to use lynx to interactively view a `text/html` message:

```
text/html; lynx %s
```

In this case, lynx does not support viewing a file from standard input, so you must use the `%s` syntax.

> ***NOTE:***
>
> *Some older versions of lynx contain a bug where they will check the mailcap file for a viewer for `text/html`. They will find the line which calls lynx, and run it. This causes lynx to continuously spawn itself to view the object.*

On the other hand, maybe you don't want to use lynx interactively, you just want to have it convert the `text/html` to `text/plain`, then you can use:

```
text/html; lynx -dump %s | more
```

Perhaps you wish to use lynx to view `text/html` files, and a pager on all other text formats, then you would use the following:

```
text/html; lynx %s
text/*; more
```

3.2. Secure Use of Mailcap

The interpretation of shell meta-characters embedded in MIME parameters can lead to security problems in general. Mutt tries to quote parameters in expansion of `%s` syntaxes properly, and avoids risky characters by substituting them, see the $mailcap_sanitize variable.

Although Mutt's procedures to invoke programs with mailcap seem to be safe, there are other applications parsing mailcap, maybe taking less care of it. Therefore you should pay attention to the following rules:

Keep the %-expandos away from shell quoting. Don't quote them with single or double quotes. Mutt does this for you, the right way, as should any other program which interprets mailcap. Don't put them into backtick expansions. Be highly careful with evil statements, and avoid them if possible at all. Trying to fix broken behavior with quotes introduces new leaks — there is no alternative to correct quoting in the first place.

If you have to use the %-expandos' values in context where you need quoting or backtick expansions, put that value into a shell variable and reference the shell variable where necessary, as in the following example (using `$charset` inside the backtick expansion is safe, since it is not itself subject to any further expansion):

```
text/test-mailcap-bug; cat %s; copiousoutput; test=charset=%{charset} \
        && test "`echo $charset | tr '[A-Z]' '[a-z]'`" != iso-8859-1
```

3.3. Advanced Mailcap Usage

3.3.1. Optional Fields

In addition to the required content-type and view command fields, you can add semi-colon ";" separated fields to set flags and other options. Mutt recognizes the following optional fields:

copiousoutput

> This flag tells Mutt that the command passes possibly large amounts of text on standard output. This causes Mutt to invoke a pager (either the internal pager or the external pager defined by the pager variable) on the output of the view command. Without this flag, Mutt assumes that the command is interactive. One could use this to replace the pipe to `more` in the `lynx -dump` example in the Basic section:
>
> ```
> text/html; lynx -dump %s ; copiousoutput
> ```
>
> This will cause lynx to format the `text/html` output as `text/plain` and Mutt will use your standard pager to display the results.
>
> Note that when using the built-in pager, *only* entries with this flag will be considered a handler for a MIME type — all other entries will be ignored.

needsterminal

> Mutt uses this flag when viewing attachments with `auto_view`, in order to decide whether it should honor the setting of the $wait_key variable or not. When an attachment is viewed using an interactive program, and the corresponding mailcap entry has a *needsterminal* flag, Mutt will use $wait_key and the exit status of the program to decide if it will ask you to press a key after the external program has exited. In all other situations it will not prompt you for a key.

compose=<command>

This flag specifies the command to use to create a new attachment of a specific MIME type. Mutt supports this from the compose menu.

composetyped=<command>

This flag specifies the command to use to create a new attachment of a specific MIME type. This command differs from the compose command in that Mutt will expect standard MIME headers on the data. This can be used to specify parameters, filename, description, etc. for a new attachment. Mutt supports this from the compose menu.

print=<command>

This flag specifies the command to use to print a specific MIME type. Mutt supports this from the attachment and compose menus.

edit=<command>

This flag specifies the command to use to edit a specific MIME type. Mutt supports this from the compose menu, and also uses it to compose new attachments. Mutt will default to the defined $editor for text attachments.

nametemplate=<template>

This field specifies the format for the file denoted by `%s` in the command fields. Certain programs will require a certain file extension, for instance, to correctly view a file. For instance, lynx will only interpret a file as `text/html` if the file ends in `.html`. So, you would specify lynx as a `text/html` viewer with a line in the mailcap file like:

```
text/html; lynx %s; nametemplate=%s.html
```

test=<command>

This field specifies a command to run to test whether this mailcap entry should be used. The command is defined with the command expansion rules defined in the next section. If the command returns 0, then the test passed, and Mutt uses this entry. If the command returns non-zero, then the test failed, and Mutt continues searching for the right entry. Note that the content-type must match before Mutt performs the test. For example:

```
text/html; firefox -remote 'openURL(%s)' ; test=RunningX
text/html; lynx %s
```

In this example, Mutt will run the program `RunningX` which will return 0 if the X Window manager is running, and non-zero if it isn't. If `RunningX` returns 0, then Mutt will run firefox to display the `text/html` object. If RunningX doesn't return 0, then Mutt will go on to the next entry and use lynx to display the `text/html` object.

3.3.2. Search Order

When searching for an entry in the mailcap file, Mutt will search for the most useful entry for its purpose. For instance, if you are attempting to print an `image/gif`, and you have the following entries in your mailcap file, Mutt will search for an entry with the print command:

```
image/*;        xv %s
image/gif;      ; print= anytopnm %s | pnmtops | lpr; \
                nametemplate=%s.gif
```

Mutt will skip the `image/*` entry and use the `image/gif` entry with the print command.

In addition, you can use this with `auto_view` to denote two commands for viewing an attachment, one to be viewed automatically, the other to be viewed interactively from the attachment menu using the `<view-mailcap>` function (bound to "m" by default). In addition, you can then use the test feature to determine which viewer to use interactively depending on your environment.

```
text/html;      firefox -remote 'openURL(%s)' ; test=RunningX
text/html;      lynx %s; nametemplate=%s.html
text/html;      lynx -dump %s; nametemplate=%s.html; copiousoutput
```

For `auto_view`, Mutt will choose the third entry because of the `copiousoutput` tag. For interactive viewing, Mutt will run the program `RunningX` to determine if it should use the first entry. If the program returns non-zero, Mutt will use the second entry for interactive viewing. The last entry is for inline display in the pager and the `<view-attach>` function in the attachment menu.

Entries with the `copiousoutput` tag should always be specified as the last one per type. For non-interactive use, the last entry will then actually be the first matching one with the tag set. For non-interactive use, only `copiousoutput`-tagged entries are considered. For interactive use, Mutt ignores this tag and treats all entries equally. Therefore, if not specified last, all following entries without this tag would never be considered for `<view-attach>` because the `copiousoutput` before them matched already.

3.3.3. Command Expansion

The various commands defined in the mailcap files are passed to the `/bin/sh` shell using the `system(3)` function. Before the command is passed to `/bin/sh -c`, it is parsed to expand various special parameters with information from Mutt. The keywords Mutt expands are:

%s

> As seen in the basic mailcap section, this variable is expanded to a filename specified by the calling program. This file contains the body of the message to view/print/edit or where the composing program should place the results of composition. In addition, the use of this keyword causes Mutt to not pass the body of the message to the view/print/edit program on stdin.

%t

Mutt will expand `%t` to the text representation of the content type of the message in the same form as the first parameter of the mailcap definition line, i.e. `text/html` or `image/gif`.

%{<parameter>}

Mutt will expand this to the value of the specified parameter from the Content-Type: line of the mail message. For instance, if your mail message contains:

```
Content-Type: text/plain; charset=iso-8859-1
```

then Mutt will expand `%{charset}` to "iso-8859-1". The default metamail mailcap file uses this feature to test the charset to spawn an xterm using the right charset to view the message.

\%

This will be replaced by a literal `%`.

Mutt does not currently support the `%F` and `%n` keywords specified in RFC 1524. The main purpose of these parameters is for multipart messages, which is handled internally by Mutt.

3.4. Example Mailcap Files

This mailcap file is fairly simple and standard:

```
# I'm always running X :)
video/*;        xanim %s > /dev/null
image/*;        xv %s > /dev/null

# I'm always running firefox (if my computer had more memory, maybe)
text/html;      firefox -remote 'openURL(%s)'
```

This mailcap file shows quite a number of examples:

```
# Use xanim to view all videos   Xanim produces a header on startup,
# send that to /dev/null so I don't see it
video/*;        xanim %s > /dev/null

# Send html to a running firefox by remote
text/html;      firefox -remote 'openURL(%s)'; test=RunningFirefox

# If I'm not running firefox but I am running X, start firefox on the
# object
text/html;      firefox %s; test=RunningX

# Else use lynx to view it as text
text/html;      lynx %s

# This version would convert the text/html to text/plain
text/html;      lynx -dump %s; copiousoutput

# I use enscript to print text in two columns to a page
text/*;         more %s; print=enscript -2Gr %s
```

```
# Firefox adds a flag to tell itself to view jpegs internally
image/jpeg;xv %s; x-mozilla-flags=internal

# Use xv to view images if I'm running X
# In addition, this uses the \ to extend the line and set my editor
# for images
image/*;xv %s; test=RunningX; \
        edit=xpaint %s

# Convert images to text using the netpbm tools
image/*;  (anytopnm %s | pnmscale -xysize 80 46 | ppmtopgm | pgmtopbm |
pbmtoascii -1x2 ) 2>&1 ; copiousoutput

# Send excel spreadsheets to my NT box
application/ms-excel; open.pl %s
```

4. MIME Autoview

Usage:

```
auto_view mimetype [ mimetype ...]
unauto_view { * | mimetype ... }
```

In addition to explicitly telling Mutt to view an attachment with the MIME viewer defined in the mailcap file from the attachments menu, Mutt has support for automatically viewing MIME attachments while in the pager.

For this to work, you must define a viewer in the mailcap file which uses the `copiousoutput` option to denote that it is non-interactive. Usually, you also use the entry to convert the attachment to a text representation which you can view in the pager.

You then use the `auto_view` configuration command to list the content-types that you wish to view automatically. For instance, if you set it to:

```
auto_view text/html application/x-gunzip \
  application/postscript image/gif application/x-tar-gz
```

...Mutt would try to find corresponding entries for rendering attachments of these types as text. A corresponding mailcap could look like:

```
text/html;      lynx -dump %s; copiousoutput; nametemplate=%s.html
image/*;        anytopnm %s | pnmscale -xsize 80 -ysize 50 | ppmtopgm | \
                pgmtopbm | pbmtoascii ; copiousoutput
application/x-gunzip;   gzcat; copiousoutput
application/x-tar-gz; gunzip -c %s | tar -tf - ; copiousoutput
application/postscript; ps2ascii %s; copiousoutput
```

`unauto_view` can be used to remove previous entries from the `auto_view` list. This can be used with `message-hook` to autoview messages based on size, etc. "`unauto_view *`" will remove all previous entries.

5. MIME Multipart/Alternative

The `multipart/alternative` container type only has child MIME parts which represent the same content in an alternative way. This is often used to send HTML messages which contain an alternative plain text representation.

Mutt has some heuristics for determining which attachment of a `multipart/alternative` type to display:

1. First, Mutt will check the `alternative_order` list to determine if one of the available types is preferred. It consists of a number of MIME types in order, including support for implicit and explicit wildcards. For example:

   ```
   alternative_order text/enriched text/plain text \
     application/postscript image/*
   ```

2. Next, Mutt will check if any of the types have a defined `auto_view`, and use that.

3. Failing that, Mutt will look for any text type.

4. As a last attempt, Mutt will look for any type it knows how to handle.

To remove a MIME type from the `alternative_order` list, use the `unalternative_order` command.

6. Attachment Searching and Counting

If you ever lose track of attachments in your mailboxes, Mutt's attachment-counting and -searching support might be for you. You can make your message index display the number of qualifying attachments in each message, or search for messages by attachment count. You also can configure what kinds of attachments qualify for this feature with the `attachments` and `unattachments` commands.

In order to provide this information, Mutt needs to fully MIME-parse all messages affected first. This can slow down operation especially for remote mail folders such as IMAP because all messages have to be downloaded first regardless whether the user really wants to view them or not though using Section 7.2, “Body Caching” usually means to download the message just once.

The syntax is:

```
attachments { + | - }disposition mime-type
unattachments { + | - }disposition mime-type
attachments ?
```

disposition is the attachment's Content-Disposition type — either `inline` or `attachment`. You can abbreviate this to `I` or `A`.

Disposition is prefixed by either a “+” symbol or a “-” symbol. If it's a “+”, you're saying that you want to allow this disposition and MIME type to qualify. If it's a “-”, you're saying that this disposition and MIME type is an

exception to previous "+" rules. There are examples below of how this is useful.

mime-type is the MIME type of the attachment you want the command to affect. A MIME type is always of the format `major/minor`, where `major` describes the broad category of document you're looking at, and `minor` describes the specific type within that category. The major part of mime-type must be literal text (or the special token "*"), but the minor part may be a regular expression. (Therefore, "`*/.*`" matches any MIME type.)

The MIME types you give to the `attachments` directive are a kind of pattern. When you use the `attachments` directive, the patterns you specify are added to a list. When you use `unattachments`, the pattern is removed from the list. The patterns are not expanded and matched to specific MIME types at this time — they're just text in a list. They're only matched when actually evaluating a message.

Some examples might help to illustrate. The examples that are not commented out define the default configuration of the lists.

Example 5.2. Attachment counting

```
# Removing a pattern from a list removes that pattern literally. It
# does not remove any type matching the pattern.
#
#  attachments   +A */.*
#  attachments   +A image/jpeg
#  unattachments +A */.*
#
# This leaves "attached" image/jpeg files on the allowed attachments
# list. It does not remove all items, as you might expect, because the
# second */.* is not a matching expression at this time.
#
# Remember: "unattachments" only undoes what "attachments" has done!
# It does not trigger any matching on actual messages.

# Qualify any MIME part with an "attachment" disposition, EXCEPT for
# text/x-vcard and application/pgp parts. (PGP parts are already known
# to mutt, and can be searched for with ~g, ~G, and ~k.)
#
# I've added x-pkcs7 to this, since it functions (for S/MIME)
# analogously to PGP signature attachments. S/MIME isn't supported
# in a stock mutt build, but we can still treat it specially here.
#

attachments   +A */.*
attachments   -A text/x-vcard application/pgp.*
attachments   -A application/x-pkcs7-.*


# Discount all MIME parts with an "inline" disposition, unless they're
# text/plain. (Why inline a text/plain part unless it's external to the
# message flow?)

attachments   +I text/plain


# These two lines make Mutt qualify MIME containers.  (So, for example,
```

```
# a message/rfc822 forward will count as an attachment.)  The first
# line is unnecessary if you already have "attach-allow */.*", of
# course.  These are off by default!  The MIME elements contained
# within a message/* or multipart/* are still examined, even if the
# containers themselves don't qualify.

#attachments  +A message/.* multipart/.*
#attachments  +I message/.* multipart/.*


## You probably don't really care to know about deleted attachments.
attachments   -A message/external-body
attachments   -I message/external-body
```

Entering the command "`attachments` ?" as a command will list your current settings in Muttrc format, so that it can be pasted elsewhere.

7. MIME Lookup

Usage:

```
mime-lookup mimetype [ mimetype ...]
unmime-lookup { * | mimetype ... }
```

Mutt's `mime_lookup` list specifies a list of MIME types that should *not* be treated according to their mailcap entry. This option is designed to deal with binary types such as `application/octet-stream`. When an attachment's MIME type is listed in `mime_lookup`, then the extension of the filename will be compared to the list of extensions in the `mime.types` file. The MIME type associated with this extension will then be used to process the attachment according to the rules in the mailcap file and according to any other configuration options (such as `auto_view`) specified. Common usage would be:

```
mime_lookup application/octet-stream application/X-Lotus-Manuscript
```

In addition, the `unmime_lookup` command may be used to disable this feature for any particular MIME type if it had been set, for example, in a global `.muttrc`.

Chapter 6. Optional Features

Table of Contents

1. General Notes

1.1. Enabling/Disabling Features

Mutt supports several of optional features which can be enabled or disabled at compile-time by giving the *configure* script certain arguments. These are listed in the "Optional features" section of the *configure --help* output.

Which features are enabled or disabled can later be determined from the output of `mutt -v`. If a compile option starts with "+" it is enabled and disabled if prefixed with "-". For example, if Mutt was compiled using GnuTLS for encrypted communication instead of OpenSSL, `mutt -v` would contain:

```
-USE_SSL_OPENSSL +USE_SSL_GNUTLS
```

1.2. URL Syntax

Mutt optionally supports the IMAP, POP3 and SMTP protocols which require to access servers using URLs. The canonical syntax for specifying URLs in Mutt is (an item enclosed in `[]` means it is optional and may be omitted):

```
proto[s]://[username[:password]@]server[:port][/path]
```

proto is the communication protocol: `imap` for IMAP, `pop` for POP3 and `smtp` for SMTP. If "s" for "secure communication" is appended, Mutt will attempt to establish an encrypted communication using SSL or TLS.

Since all protocols supported by Mutt support/require authentication, login credentials may be specified in the URL. This has the advantage that multiple IMAP, POP3 or SMTP servers may be specified (which isn't possible using, for example, $imap_user). The username may contain the "@" symbol being used by many mail

systems as part of the login name. The special characters "/" (`%2F`), ":" (`%3A`) and "%" (`%25`) have to be URL-encoded in usernames using the `%`-notation.

A password can be given, too but is not recommended if the URL is specified in a configuration file on disk.

If no port number is given, Mutt will use the system's default for the given protocol (usually consulting `/etc/services`).

The optional path is only relevant for IMAP and ignored elsewhere.

Example 6.1. URLs

```
pops://host/
imaps://user@host/INBOX/Sent
smtp://user@host:587/
```

2. SSL/TLS Support

If Mutt is compiled with IMAP, POP3 and/or SMTP support, it can also be compiled with support for SSL or TLS using either OpenSSL or GnuTLS (by running the *configure* script with the *--enable-ssl*=... option for OpenSSL or *--enable-gnutls*=... for GnuTLS). Mutt can then attempt to encrypt communication with remote servers if these protocols are suffixed with "s" for "secure communication".

3. POP3 Support

If Mutt is compiled with POP3 support (by running the *configure* script with the *--enable-pop* flag), it has the ability to work with mailboxes located on a remote POP3 server and fetch mail for local browsing.

Remote POP3 servers can be accessed using URLs with the `pop` protocol for unencrypted and `pops` for encrypted communication, see Section 1.2, "URL Syntax" for details.

Polling for new mail is more expensive over POP3 than locally. For this reason the frequency at which Mutt will check for mail remotely can be controlled by the $pop_checkinterval variable, which defaults to every 60 seconds.

POP is read-only which doesn't allow for some features like editing messages or changing flags. However, using Section 7.1, "Header Caching" and Section 7.2, "Body Caching" Mutt simulates the new/old/read flags as well as flagged and replied. Mutt applies some logic on top of remote messages but cannot change them so that modifications of flags are lost when messages are downloaded from the POP server (either by Mutt or other tools).

Another way to access your POP3 mail is the `<fetch-mail>` function (default: G). It allows to connect to $pop_host, fetch all your new mail and place it in the local $spoolfile. After this point, Mutt runs exactly as if the mail had always been local.

NOTE:

If you only need to fetch all messages to a local mailbox you should consider using a specialized program, such as `fetchmail(1)`, `getmail(1)` or similar.

4. IMAP Support

If Mutt was compiled with IMAP support (by running the *configure* script with the *--enable-imap* flag), it has the ability to work with folders located on a remote IMAP server.

You can access the remote inbox by selecting the folder by its URL (see Section 1.2, "URL Syntax" for details) using the `imap` or `imaps` protocol. Alternatively, a pine-compatible notation is also supported, i.e. `{[username@]imapserver[:port][/ssl]}path/to/folder`

Note that not all servers use "/" as the hierarchy separator. Mutt should correctly notice which separator is being used by the server and convert paths accordingly.

When browsing folders on an IMAP server, you can toggle whether to look at only the folders you are subscribed to, or all folders with the *toggle-subscribed* command. See also the $imap_list_subscribed variable.

Polling for new mail on an IMAP server can cause noticeable delays. So, you'll want to carefully tune the $mail_check and $timeout variables. Reasonable values are:

```
set mail_check=90
set timeout=15
```

with relatively good results even over slow modem lines.

NOTE:

Note that if you are using mbox as the mail store on UW servers prior to v12.250, the server has been reported to disconnect a client if another client selects the same folder.

4.1. The IMAP Folder Browser

As of version 1.2, Mutt supports browsing mailboxes on an IMAP server. This is mostly the same as the local file browser, with the following differences:

- In lieu of file permissions, Mutt displays the string "IMAP", possibly followed by the symbol "+", indicating that the entry contains both messages and subfolders. On Cyrus-like servers folders will often contain both messages and subfolders.

- For the case where an entry can contain both messages and subfolders, the selection key (bound to `enter` by default) will choose to descend into the subfolder view. If you wish to view the messages in that folder, you must use `view-file` instead (bound to `space` by default).

- You can create, delete and rename mailboxes with the `<create-mailbox>`, `<delete-mailbox>`, and `<rename-mailbox>` commands (default bindings: `C`, `d` and `r`, respectively). You may also `<subscribe>` and `<unsubscribe>` to mailboxes (normally these are bound to `s` and `u`, respectively).

4.2. Authentication

Mutt supports four authentication methods with IMAP servers: SASL, GSSAPI, CRAM-MD5, and LOGIN (there is a patch by Grant Edwards to add NTLM authentication for you poor exchange users out there, but it has yet to be integrated into the main tree). There is also support for the pseudo-protocol ANONYMOUS, which allows you to log in to a public IMAP server without having an account. To use ANONYMOUS, simply make your username blank or "anonymous".

SASL is a special super-authenticator, which selects among several protocols (including GSSAPI, CRAM-MD5, ANONYMOUS, and DIGEST-MD5) the most secure method available on your host and the server. Using some of these methods (including DIGEST-MD5 and possibly GSSAPI), your entire session will be encrypted and invisible to those teeming network snoops. It is the best option if you have it. To use it, you must have the Cyrus SASL library installed on your system and compile Mutt with the *--with-sasl* flag.

Mutt will try whichever methods are compiled in and available on the server, in the following order: SASL, ANONYMOUS, GSSAPI, CRAM-MD5, LOGIN.

There are a few variables which control authentication:

- $imap_user - controls the username under which you request authentication on the IMAP server, for all authenticators. This is overridden by an explicit username in the mailbox path (i.e. by using a mailbox name of the form `{user@host}`).

- $imap_pass - a password which you may preset, used by all authentication methods where a password is needed.

- $imap_authenticators - a colon-delimited list of IMAP authentication methods to try, in the order you wish to try them. If specified, this overrides Mutt's default (attempt everything, in the order listed above).

5. SMTP Support

Besides supporting traditional mail delivery through a sendmail-compatible program, Mutt supports delivery through SMTP if it was configured and built with `--enable-smtp`.

If the configuration variable $smtp_url is set, Mutt will contact the given SMTP server to deliver messages; if it is unset, Mutt will use the program specified by $sendmail.

For details on the URL syntax, please see Section 1.2, "URL Syntax".

The built-in SMTP support supports encryption (the `smtps` protocol using SSL or TLS) as well as SMTP authentication using SASL. The authentication mechanisms for SASL are specified in $smtp_authenticators defaulting to an empty list which makes Mutt try all available methods from most-secure to least-secure.

6. Managing Multiple Accounts

Usage:

```
account-hook pattern command
```

If you happen to have accounts on multiple IMAP, POP and/or SMTP servers, you may find managing all the authentication settings inconvenient and error-prone. The `account-hook` command may help. This hook works like `folder-hook` but is invoked whenever Mutt needs to access a remote mailbox (including inside the folder browser), not just when you open the mailbox. This includes (for example) polling for new mail, storing Fcc messages and saving messages to a folder. As a consequence, `account-hook` should only be used to set connection-related settings such as passwords or tunnel commands but not settings such as sender address or name (because in general it should be considered unpredictable which `account-hook` was last used).

Some examples:

```
account-hook . 'unset imap_user; unset imap_pass; unset tunnel'
account-hook imap://host1/ 'set imap_user=me1 imap_pass=foo'
account-hook imap://host2/ 'set tunnel="ssh host2 /usr/libexec/imapd"'
account-hook smtp://user@host3/ 'set tunnel="ssh host3 /usr/libexec/smtpd"'
```

To manage multiple accounts with, for example, different values of $record or sender addresses, `folder-hook` has to be be used together with the `mailboxes` command.

Example 6.2. Managing multiple accounts

```
mailboxes imap://user@host1/INBOX
folder-hook imap://user@host1/ 'set folder=imap://host1/ ; set record=+INBOX/Sent'

mailboxes imap://user@host2/INBOX
folder-hook imap://user@host2/ 'set folder=imap://host2/ ; set record=+INBOX/Sent'
```

In example Example 6.2, “Managing multiple accounts” the folders are defined using `mailboxes` so Mutt polls them for new mail. Each `folder-hook` triggers when one mailbox below each IMAP account is opened and sets $folder to the account's root folder. Next, it sets $record to the *INBOX/Sent* folder below the newly set $folder. Please notice that the value the “+” mailbox shortcut refers to depends on the *current* value of $folder and therefore has to be set separately per account. Setting other values like $from or $signature is analogous to setting $record.

7. Local Caching

Mutt contains two types of local caching: *(1)* the so-called "header caching" and *(2)* the so-called "body caching" which are both described in this section.

Header caching is optional as it depends on external libraries, body caching is always enabled if Mutt is compiled with POP and/or IMAP support as these use it (body caching requires no external library).

7.1. Header Caching

Mutt provides optional support for caching message headers for the following types of folders: IMAP, POP, Maildir and MH. Header caching greatly speeds up opening large folders because for remote folders, headers usually only need to be downloaded once. For Maildir and MH, reading the headers from a single file is much faster than looking at possibly thousands of single files (since Maildir and MH use one file per message.)

Header caching can be enabled via the configure script and the *--enable-hcache* option. It's not turned on by default because external database libraries are required: one of tokyocabinet, qdbm, gdbm or bdb must be present.

If enabled, $header_cache can be used to either point to a file or a directory. If set to point to a file, one database file for all folders will be used (which may result in lower performance), but one file per folder if it points to a directory.

7.2. Body Caching

Both cache methods can be combined using the same directory for storage (and for IMAP/POP even provide meaningful file names) which simplifies manual maintenance tasks.

In addition to caching message headers only, Mutt can also cache whole message bodies. This results in faster display of messages for POP and IMAP folders because messages usually have to be downloaded only once.

For configuration, the variable $message_cachedir must point to a directory. There, Mutt will create a hierarchy of subdirectories named like the account and mailbox path the cache is for.

7.3. Cache Directories

For using both, header and body caching, $header_cache and $message_cachedir can be safely set to the same value.

In a header or body cache directory, Mutt creates a directory hierarchy named like: `proto:user@hostname` where `proto` is either "pop" or "imap." Within there, for each folder, Mutt stores messages in single files and header caches in files with the ".hcache" extension. All files can be removed as needed if the consumed disk space becomes an issue as Mutt will silently fetch missing items again. Pathnames are always stored in UTF-8 encoding.

For Maildir and MH, the header cache files are named after the MD5 checksum of the path.

7.4. Maintenance

Mutt does not (yet) support maintenance features for header cache database files so that files have to be removed in case they grow too big. It depends on the database library used for header caching whether disk space freed by removing messages is re-used.

For body caches, Mutt can keep the local cache in sync with the remote mailbox if the $message_cache_clean variable is set. Cleaning means to remove messages from the cache which are no longer present in the mailbox which only happens when other mail clients or instances of Mutt using a different body cache location delete messages (Mutt itself removes deleted messages from the cache when syncing a mailbox). As cleaning can take a noticeable amount of time, it should not be set in general but only occasionally.

8. Exact Address Generation

Mutt supports the "Name <user@host>" address syntax for reading and writing messages, the older "user@host (Name)" syntax is only supported when reading messages. The *--enable-exact-address* switch can be given to configure to build it with write-support for the latter syntax. `EXACT_ADDRESS` in the output of `mutt -v` indicates whether it's supported.

9. Sending Anonymous Messages via Mixmaster

You may also have compiled Mutt to co-operate with Mixmaster, an anonymous remailer. Mixmaster permits you to send your messages anonymously using a chain of remailers. Mixmaster support in Mutt is for mixmaster version 2.04 or later.

To use it, you'll have to obey certain restrictions. Most important, you cannot use the `Cc` and `Bcc` headers. To tell Mutt to use mixmaster, you have to select a remailer chain, using the mix function on the compose menu.

The chain selection screen is divided into two parts. In the (larger) upper part, you get a list of remailers you may use. In the lower part, you see the currently selected chain of remailers.

You can navigate in the chain using the `<chain-prev>` and `<chain-next>` functions, which are by default bound to the left and right arrows and to the `h` and `l` keys (think vi keyboard bindings). To insert a remailer at the current chain position, use the `<insert>` function. To append a remailer behind the current chain position, use `<select-entry>` or `<append>`. You can also delete entries from the chain, using the corresponding function. Finally, to abandon your changes, leave the menu, or `<accept>` them pressing (by default) the `Return` key.

Note that different remailers do have different capabilities, indicated in the %c entry of the remailer menu lines (see $mix_entry_format). Most important is the "middleman" capability, indicated by a capital "M": This means that the remailer in question cannot be used as the final element of a chain, but will only forward messages to other mixmaster remailers. For details on the other capabilities, please have a look at the mixmaster documentation.

Chapter 7. Security Considerations

Table of Contents

First of all, Mutt contains no security holes included by intention but may contain unknown security holes. As a consequence, please run Mutt only with as few permissions as possible. Especially, do not run Mutt as the super user.

When configuring Mutt, there're some points to note about secure setups so please read this chapter carefully.

1. Passwords

Although Mutt can be told the various passwords for accounts, please never store passwords in configuration files. Besides the fact that the system's operator can always read them, you could forget to mask it out when reporting a bug or asking for help via a mailing list. Even worse, your mail including your password could be archived by internet search engines, mail-to-news gateways etc. It may already be too late before you notice your mistake.

2. Temporary Files

Mutt uses many temporary files for viewing messages, verifying digital signatures, etc. As long as being used, these files are visible by other users and maybe even readable in case of misconfiguration. Also, a different location for these files may be desired which can be changed via the $tmpdir variable.

3. Information Leaks

3.1. Message-Id: headers

Message-Id: headers contain a local part that is to be created in a unique fashion. In order to do so, Mutt will “leak” some information to the outside world when sending messages: the generation of this header includes a step counter which is increased (and rotated) with every message sent. In a longer running mutt session, others can make assumptions about your mailing habits depending on the number of messages sent. If this is not desired, the header can be manually provided using $edit_headers (though not recommended).

3.2. `mailto:`-style Links

As Mutt be can be set up to be the mail client to handle `mailto:` style links in websites, there're security

considerations, too. Arbitrary header fields can be embedded in these links which could override existing header fields or attach arbitrary files using the Attach: pseudoheader. This may be problematic if the $edit-headers variable is *unset*, i.e. the user doesn't want to see header fields while editing the message and doesn't pay enough attention to the compose menu's listing of attachments.

For example, following a link like

```
mailto:joe@host?Attach=~/.gnupg/secring.gpg
```

will send out the user's private gnupg keyring to `joe@host` if the user doesn't follow the information on screen carefully enough.

4. External Applications

Mutt in many places has to rely on external applications or for convenience supports mechanisms involving external applications.

One of these is the `mailcap` mechanism as defined by RfC1524. Details about a secure use of the mailcap mechanisms is given in Section 3.2, “Secure Use of Mailcap”.

Besides the mailcap mechanism, Mutt uses a number of other external utilities for operation, for example to provide crypto support, in backtick expansion in configuration files or format string filters. The same security considerations apply for these as for tools involved via mailcap.

Chapter 8. Performance Tuning

Table of Contents

1. Reading and Writing Mailboxes

Mutt's performance when reading mailboxes can be improved in two ways:

1. For remote folders (IMAP and POP) as well as folders using one-file-per message storage (Maildir and MH), Mutt's performance can be greatly improved using header caching. using a single database per folder.

2. Mutt provides the $read_inc and $write_inc variables to specify at which rate to update progress counters. If these values are too low, Mutt may spend more time on updating the progress counter than it spends on actually reading/writing folders.

   For example, when opening a maildir folder with a few thousand messages, the default value for

$read_inc may be too low. It can be tuned on on a folder-basis using folder-hooks:

```
# use very high $read_inc to speed up reading hcache'd maildirs
folder-hook . 'set read_inc=1000'
# use lower value for reading slower remote IMAP folders
folder-hook ^imap 'set read_inc=100'
# use even lower value for reading even slower remote POP folders
folder-hook ^pop 'set read_inc=1'
```

These settings work on a per-message basis. However, as messages may greatly differ in size and certain operations are much faster than others, even per-folder settings of the increment variables may not be desirable as they produce either too few or too much progress updates. Thus, Mutt allows to limit the number of progress updates per second it'll actually send to the terminal using the $time_inc variable.

2. Reading Messages from Remote Folders

Reading messages from remote folders such as IMAP an POP can be slow especially for large mailboxes since Mutt only caches a very limited number of recently viewed messages (usually 10) per session (so that it will be gone for the next session.)

To improve performance and permanently cache whole messages, please refer to Mutt's so-called body caching for details.

3. Searching and Limiting

When searching mailboxes either via a search or a limit action, for some patterns Mutt distinguishes between regular expression and string searches. For regular expressions, patterns are prefixed with "~" and with "=" for string searches.

Even though a regular expression search is fast, it's several times slower than a pure string search which is noticeable especially on large folders. As a consequence, a string search should be used instead of a regular expression search if the user already knows enough about the search pattern.

For example, when limiting a large folder to all messages sent to or by an author, it's much faster to search for the initial part of an e-mail address via `=Luser@` instead of `~Luser@`. This is especially true for searching message bodies since a larger amount of input has to be searched.

As for regular expressions, a lower case string search pattern makes Mutt perform a case-insensitive search except for IMAP (because for IMAP Mutt performs server-side searches which don't support case-insensitivity).

Chapter 9. Reference

Table of Contents

1. Command-Line Options

Running `mutt` with no arguments will make Mutt attempt to read your spool mailbox. However, it is possible to read other mailboxes and to send messages from the command line as well.

Table 9.1. Command line options

Option	Description
-A	expand an alias
-a	attach a file to a message
-b	specify a blind carbon-copy (BCC) address
-c	specify a carbon-copy (Cc) address
-D	print the value of all Mutt variables to stdout
-e	specify a config command to be run after initialization files are read
-f	specify a mailbox to load
-F	specify an alternate file to read initialization commands
-h	print help on command line options
-H	specify a draft file from which to read a header and body
-i	specify a file to include in a message composition
-m	specify a default mailbox type
-n	do not read the system Muttrc
-p	recall a postponed message
-Q	query a configuration variable
-R	open mailbox in read-only mode
-s	specify a subject (enclose in quotes if it contains spaces)
-v	show version number and compile-time definitions

-x	simulate the mailx(1) compose mode
-y	show a menu containing the files specified by the `mailboxes` command
-z	exit immediately if there are no messages in the mailbox
-Z	open the first folder with new message, exit immediately if none

To read messages in a mailbox

```
mutt [-nz] [-F muttrc ] [-m type ] [-f mailbox ]
```

To compose a new message

```
mutt [-n] [-F muttrc ] [-c address ] [-i filename ] [-s subject ] [ -a file [...] -- ] address | mailto_url ...
```

Mutt also supports a "batch" mode to send prepared messages. Simply redirect input from the file you wish to send. For example,

```
mutt -s "data set for run #2" professor@bigschool.edu < ~/run2.dat
```

will send a message to <professor@bigschool.edu> with a subject of "data set for run #2". In the body of the message will be the contents of the file "~/run2.dat".

All files passed with -a *file* will be attached as a MIME part to the message. To attach a single or several files, use "--" to separate files and recipient addresses:

```
mutt -a image.png -- some@one.org
```

or

```
mutt -a *.png -- some@one.org
```

> ***Note:***
>
> The -a option must be last in the option list.

In addition to accepting a list of email addresses, Mutt also accepts a URL with the `mailto:` schema as specified in RFC2368. This is useful when configuring a web browser to launch Mutt when clicking on mailto links.

```
mutt mailto:some@one.org?subject=test&cc=other@one.org
```

2. Configuration Commands

The following are the commands understood by Mutt:

- account-hook *pattern command*
- alias [-group *name* ...] *key address* [*address* ...]
  unalias [-group *name* ...] { * | *key* ... }
- alternates [-group *name* ...] *regexp* [*regexp* ...]
  unalternates [-group *name* ...] { * | *regexp* ... }
- alternative-order *mimetype* [*mimetype* ...]
  unalternative-order { * | *mimetype* ... }
- attachments *{ + | - }disposition mime-type*
  unattachments *{ + | - }disposition mime-type*
- auto_view *mimetype* [*mimetype* ...]
  unauto_view { * | *mimetype* ... }
- bind *map key function*
- charset-hook *alias charset*
- iconv-hook *charset local-charset*
- color *object foreground background*
  color { header | body } *foreground background regexp*
  color index *foreground background pattern*
  uncolor { index | header | body } { * | *pattern* ... }
- crypt-hook *pattern keyid*
- exec *function* [*function* ...]
- fcc-hook *[!]pattern mailbox*
- fcc-save-hook *[!]pattern mailbox*
- folder-hook *[!]regexp command*
- group [-group *name* ...] { -rx *expr* ... | -addr *expr* ... }
  ungroup [-group *name* ...] { * | -rx *expr* ... | -addr *expr* ... }
- hdr_order *header* [*header* ...]
  unhdr_order { * | *header* ... }

- ignore *pattern* [*pattern* ...]
  unignore { * | *pattern* ... }

- lists [-group *name*] *regexp* [*regexp* ...]
  unlists [-group *name* ...] { * | *regexp* ... }

- macro *menu key sequence* [*description*]

- mailboxes *mailbox* [*mailbox* ...]
  unmailboxes { * | *mailbox* ... }

- mbox-hook *[!]pattern mailbox*

- message-hook *[!]pattern command*

- mime-lookup *mimetype* [*mimetype* ...]
  unmime-lookup { * | *mimetype* ... }

- mono *object attribute*
  mono { header | body } *attribute regexp*
  mono index *attribute pattern*
  unmono { index | header | body } { * | *pattern* ... }

- my_hdr *string*
  unmy_hdr { * | *field* ... }

- push *string*

- save-hook *[!]pattern mailbox*

- score *pattern value*
  unscore { * | *pattern* ... }

- reply-hook *[!]pattern command*

- send-hook *[!]pattern command*

- send2-hook *[!]pattern command*

- set { [no | inv] *variable* | *variable=value* } [...]
  toggle *variable* [*variable* ...]
  unset *variable* [*variable* ...]
  reset *variable* [*variable* ...]

- source *filename*

- spam *pattern format*
  nospam { * | *pattern* }

- subscribe [-group *name* ...] *regexp* [*regexp* ...]
  unsubscribe [-group *name* ...] { * | *regexp* ... }

- unhook { * | *hook-type* }

3. Configuration Variables

3.1. abort_nosubject

Type: quadoption
Default: ask-yes

If set to *yes*, when composing messages and no subject is given at the subject prompt, composition will be aborted. If set to *no*, composing messages with no subject given at the subject prompt will never be aborted.

3.2. abort_unmodified

Type: quadoption
Default: yes

If set to *yes*, composition will automatically abort after editing the message body if no changes are made to the file (this check only happens after the *first* edit of the file). When set to *no*, composition will never be aborted.

3.3. alias_file

Type: path
Default: "`~/.muttrc`"

The default file in which to save aliases created by the <create-alias> function. Entries added to this file are encoded in the character set specified by $config_charset if it is *set* or the current character set otherwise.

Note: Mutt will not automatically source this file; you must explicitly use the "source" command for it to be executed in case this option points to a dedicated alias file.

The default for this option is the currently used muttrc file, or "~/.muttrc" if no user muttrc was found.

3.4. alias_format

Type: string
Default: "`%4n %2f %t %-10a  %r`"

Specifies the format of the data displayed for the "alias" menu. The following `printf(3)`-style sequences are available:

%a	alias name

%f	flags - currently, a “d” for an alias marked for deletion
%n	index number
%r	address which alias expands to
%t	character which indicates if the alias is tagged for inclusion

3.5. allow_8bit

Type: boolean
Default: yes

Controls whether 8-bit data is converted to 7-bit using either Quoted- Printable or Base64 encoding when sending mail.

3.6. allow_ansi

Type: boolean
Default: no

Controls whether ANSI color codes in messages (and color tags in rich text messages) are to be interpreted. Messages containing these codes are rare, but if this option is *set*, their text will be colored accordingly. Note that this may override your color choices, and even present a security problem, since a message could include a line like

```
[-- PGP output follows ...
```

and give it the same color as your attachment color (see also $crypt_timestamp).

3.7. arrow_cursor

Type: boolean
Default: no

When *set*, an arrow (“->”) will be used to indicate the current entry in menus instead of highlighting the whole line. On slow network or modem links this will make response faster because there is less that has to be redrawn on the screen when moving to the next or previous entries in the menu.

3.8. ascii_chars

Type: boolean
Default: no

If *set*, Mutt will use plain ASCII characters when displaying thread and attachment trees, instead of the default *ACS* characters.

3.9. askbcc

Type: boolean
Default: no

If *set*, Mutt will prompt you for blind-carbon-copy (Bcc) recipients before editing an outgoing message.

3.10. askcc

Type: boolean
Default: no

If *set*, Mutt will prompt you for carbon-copy (Cc) recipients before editing the body of an outgoing message.

3.11. assumed_charset

Type: string
Default: (empty)

This variable is a colon-separated list of character encoding schemes for messages without character encoding indication. Header field values and message body content without character encoding indication would be assumed that they are written in one of this list. By default, all the header fields and message body without any charset indication are assumed to be in “us-ascii”.

For example, Japanese users might prefer this:

```
set assumed_charset="iso-2022-jp:euc-jp:shift_jis:utf-8"
```

However, only the first content is valid for the message body.

3.12. attach_charset

Type: string
Default: (empty)

This variable is a colon-separated list of character encoding schemes for text file attachments. Mutt uses this setting to guess which encoding files being attached are encoded in to convert them to a proper character set given in $send_charset.

If *unset*, the value of $charset will be used instead. For example, the following configuration would work for Japanese text handling:

```
set attach_charset="iso-2022-jp:euc-jp:shift_jis:utf-8"
```

Note: for Japanese users, “iso-2022-*” must be put at the head of the value as shown above if included.

3.13. attach_format

Type: string
Default: “`%u%D%I %t%4n %T%.40d%> [%.7m/%.10M, %.6e%?C?, %C?, %s] `”

This variable describes the format of the “attachment” menu. The following `printf(3)`-style sequences are understood:

%C	charset
%c	requires charset conversion (“n” or “c”)
%D	deleted flag
%d	description
%e	MIME content-transfer-encoding
%f	filename
%I	disposition (“I” for inline, “A” for attachment)
%m	major MIME type
%M	MIME subtype
%n	attachment number
%Q	“Q”, if MIME part qualifies for attachment counting
%s	size
%t	tagged flag
%T	graphic tree characters
%u	unlink (=to delete) flag
%X	number of qualifying MIME parts in this part and its children (please see the “attachments” section for possible speed effects)
%>X	right justify the rest of the string and pad with character “X”
%|X	pad to the end of the line with character “X”
%*X	soft-fill with character “X” as pad

For an explanation of “soft-fill”, see the $index_format documentation.

3.14. attach_sep

Type: string
Default: “\n”

The separator to add between attachments when operating (saving, printing, piping, etc) on a list of tagged attachments.

3.15. attach_split

Type: boolean
Default: yes

If this variable is *unset*, when operating (saving, printing, piping, etc) on a list of tagged attachments, Mutt will concatenate the attachments and will operate on them as a single attachment. The $attach_sep separator is added after each attachment. When *set*, Mutt will operate on the attachments one by one.

3.16. attribution

Type: string
Default: "`On %d, %n wrote:`"

This is the string that will precede a message which has been included in a reply. For a full listing of defined `printf(3)`-like sequences see the section on $index_format.

3.17. auto_tag

Type: boolean
Default: no

When *set*, functions in the *index* menu which affect a message will be applied to all tagged messages (if there are any). When unset, you must first use the `<tag-prefix>` function (bound to ";" by default) to make the next function apply to all tagged messages.

3.18. autoedit

Type: boolean
Default: no

When *set* along with $edit_headers, Mutt will skip the initial send-menu (prompting for subject and recipients) and allow you to immediately begin editing the body of your message. The send-menu may still be accessed once you have finished editing the body of your message.

Note: when this option is *set*, you cannot use send-hooks that depend on the recipients when composing a new (non-reply) message, as the initial list of recipients is empty.

Also see $fast_reply.

3.19. beep

Type: boolean

Default: yes

When this variable is *set*, mutt will beep when an error occurs.

3.20. beep_new

Type: boolean
Default: no

When this variable is *set*, mutt will beep whenever it prints a message notifying you of new mail. This is independent of the setting of the $beep variable.

3.21. bounce

Type: quadoption
Default: ask-yes

Controls whether you will be asked to confirm bouncing messages. If set to *yes* you don't get asked if you want to bounce a message. Setting this variable to *no* is not generally useful, and thus not recommended, because you are unable to bounce messages.

3.22. bounce_delivered

Type: boolean
Default: yes

When this variable is *set*, mutt will include Delivered-To headers when bouncing messages. Postfix users may wish to *unset* this variable.

3.23. braille_friendly

Type: boolean
Default: no

When this variable is *set*, mutt will place the cursor at the beginning of the current line in menus, even when the $arrow_cursor variable is *unset*, making it easier for blind persons using Braille displays to follow these menus. The option is *unset* by default because many visual terminals don't permit making the cursor invisible.

3.24. certificate_file

Type: path
Default: "`~/.mutt_certificates`"

This variable specifies the file where the certificates you trust are saved. When an unknown certificate is encountered, you are asked if you accept it or not. If you accept it, the certificate can also be saved in this file and further connections are automatically accepted.

You can also manually add CA certificates in this file. Any server certificate that is signed with one of these CA certificates is also automatically accepted.

Example:

```
set certificate_file=~/.mutt/certificates
```

3.25. charset

Type: string
Default: (empty)

Character set your terminal uses to display and enter textual data. It is also the fallback for $send_charset.

Upon startup Mutt tries to derive this value from environment variables such as `$LC_CTYPE` or `$LANG`.

Note: It should only be set in case Mutt isn't able to determine the character set used correctly.

3.26. check_mbox_size

Type: boolean
Default: no

When this variable is *set*, mutt will use file size attribute instead of access time when checking for new mail in mbox and mmdf folders.

This variable is *unset* by default and should only be enabled when new mail detection for these folder types is unreliable or doesn't work.

Note that enabling this variable should happen before any "mailboxes" directives occur in configuration files regarding mbox or mmdf folders because mutt needs to determine the initial new mail status of such a mailbox by performing a fast mailbox scan when it is defined. Afterwards the new mail status is tracked by file size changes.

3.27. check_new

Type: boolean
Default: yes

Note: this option only affects *maildir* and *MH* style mailboxes.

When *set*, Mutt will check for new mail delivered while the mailbox is open. Especially with MH mailboxes, this operation can take quite some time since it involves scanning the directory and checking each file to see if it has already been looked at. If this variable is *unset*, no check for new mail is performed while the mailbox is open.

3.28. collapse_unread

Type: boolean
Default: yes

When *unset*, Mutt will not collapse a thread if it contains any unread messages.

3.29. compose_format

Type: string
Default: "`-- Mutt: Compose  [Approx. msg size: %l   Atts: %a]%>-`"

Controls the format of the status line displayed in the "compose" menu. This string is similar to $status_format, but has its own set of `printf(3)`-like sequences:

%a	total number of attachments
%h	local hostname
%l	approximate size (in bytes) of the current message
%v	Mutt version string

See the text describing the $status_format option for more information on how to set $compose_format.

3.30. config_charset

Type: string
Default: (empty)

When defined, Mutt will recode commands in rc files from this encoding to the current character set as specified by $charset and aliases written to $alias_file from the current character set.

Please note that if setting $charset it must be done before setting $config_charset.

Recoding should be avoided as it may render unconvertable characters as question marks which can lead to undesired side effects (for example in regular expressions).

3.31. confirmappend

Type: boolean
Default: yes

When *set*, Mutt will prompt for confirmation when appending messages to an existing mailbox.

3.32. confirmcreate

Type: boolean
Default: yes

When *set*, Mutt will prompt for confirmation when saving messages to a mailbox which does not yet exist before creating it.

3.33. connect_timeout

Type: number
Default: 30

Causes Mutt to timeout a network connection (for IMAP, POP or SMTP) after this many seconds if the connection is not able to be established. A negative value causes Mutt to wait indefinitely for the connection attempt to succeed.

3.34. content_type

Type: string
Default: "`text/plain`"

Sets the default Content-Type for the body of newly composed messages.

3.35. copy

Type: quadoption
Default: yes

This variable controls whether or not copies of your outgoing messages will be saved for later references. Also see $record, $save_name, $force_name and "fcc-hook".

3.36. crypt_autoencrypt

Type: boolean
Default: no

Setting this variable will cause Mutt to always attempt to PGP encrypt outgoing messages. This is probably only useful in connection to the "send-hook" command. It can be overridden by use of the pgp menu, when encryption is not required or signing is requested as well. If $smime_is_default is *set*, then OpenSSL is used instead to create S/MIME messages and settings can be overridden by use of the smime menu instead. (Crypto only)

3.37. crypt_autopgp

Type: boolean
Default: yes

This variable controls whether or not mutt may automatically enable PGP encryption/signing for messages. See also $crypt_autoencrypt, $crypt_replyencrypt, $crypt_autosign, $crypt_replysign and $smime_is_default.

3.38. crypt_autosign

Type: boolean
Default: no

Setting this variable will cause Mutt to always attempt to cryptographically sign outgoing messages. This can be overridden by use of the pgp menu, when signing is not required or encryption is requested as well. If $smime_is_default is *set*, then OpenSSL is used instead to create S/MIME messages and settings can be overridden by use of the smime menu instead of the pgp menu. (Crypto only)

3.39. crypt_autosmime

Type: boolean
Default: yes

This variable controls whether or not mutt may automatically enable S/MIME encryption/signing for messages. See also $crypt_autoencrypt, $crypt_replyencrypt, $crypt_autosign, $crypt_replysign and $smime_is_default.

3.40. crypt_replyencrypt

Type: boolean
Default: yes

If *set*, automatically PGP or OpenSSL encrypt replies to messages which are encrypted. (Crypto only)

3.41. crypt_replysign

Type: boolean
Default: no

If *set*, automatically PGP or OpenSSL sign replies to messages which are signed.

Note: this does not work on messages that are encrypted *and* signed! (Crypto only)

3.42. crypt_replysignencrypted

Type: boolean
Default: no

If *set*, automatically PGP or OpenSSL sign replies to messages which are encrypted. This makes sense in combination with $crypt_replyencrypt, because it allows you to sign all messages which are automatically encrypted. This works around the problem noted in $crypt_replysign, that mutt is not able to find out whether an encrypted message is also signed. (Crypto only)

3.43. crypt_timestamp

Type: boolean
Default: yes

If *set*, mutt will include a time stamp in the lines surrounding PGP or S/MIME output, so spoofing such lines is more difficult. If you are using colors to mark these lines, and rely on these, you may *unset* this setting. (Crypto only)

3.44. crypt_use_gpgme

Type: boolean
Default: no

This variable controls the use of the GPGME-enabled crypto backends. If it is *set* and Mutt was built with gpgme support, the gpgme code for S/MIME and PGP will be used instead of the classic code. Note that you need to set this option in .muttrc; it won't have any effect when used interactively.

3.45. crypt_use_pka

Type: boolean
Default: no

Controls whether mutt uses PKA (see http://www.g10code.de/docs/pka-intro.de.pdf) during signature verification (only supported by the GPGME backend).

3.46. crypt_verify_sig

Type: quadoption
Default: yes

If *"yes"*, always attempt to verify PGP or S/MIME signatures. If *"ask-*"*, ask whether or not to verify the signature. If *"no"*, never attempt to verify cryptographic signatures. (Crypto only)

3.47. date_format

Type: string
Default: "`!%a, %b %d, %Y at %I:%M:%S%p %Z`"

This variable controls the format of the date printed by the "%d" sequence in $index_format. This is passed to the `strftime(3)` function to process the date, see the man page for the proper syntax.

Unless the first character in the string is a bang ("!"), the month and week day names are expanded according to the locale specified in the variable $locale. If the first character in the string is a bang, the bang is discarded, and the month and week day names in the rest of the string are expanded in the *C* locale (that is in US English).

3.48. default_hook

Type: string
Default: "`~f %s !~P | (~P ~C %s)`"

This variable controls how "message-hook", "reply-hook", "send-hook", "send2-hook", "save-hook", and "fcc-hook" will be interpreted if they are specified with only a simple regexp, instead of a matching pattern. The hooks are expanded when they are declared, so a hook will be interpreted according to the value of this variable at the time the hook is declared.

The default value matches if the message is either from a user matching the regular expression given, or if it is from you (if the from address matches "alternates") and is to or cc'ed to a user matching the given regular expression.

3.49. delete

Type: quadoption
Default: ask-yes

Controls whether or not messages are really deleted when closing or synchronizing a mailbox. If set to *yes*, messages marked for deleting will automatically be purged without prompting. If set to *no*, messages marked for deletion will be kept in the mailbox.

3.50. delete_untag

Type: boolean
Default: yes

If this option is *set*, mutt will untag messages when marking them for deletion. This applies when you either explicitly delete a message, or when you save it to another folder.

3.51. digest_collapse

Type: boolean
Default: yes

If this option is *set*, mutt's received-attachments menu will not show the subparts of individual messages in a multipart/digest. To see these subparts, press "v" on that menu.

3.52. display_filter

Type: path
Default: (empty)

When set, specifies a command used to filter messages. When a message is viewed it is passed as standard input to $display_filter, and the filtered message is read from the standard output.

3.53. dotlock_program

Type: path
Default: "`/usr/local/bin/mutt_dotlock`"

Contains the path of the `mutt_dotlock(8)` binary to be used by mutt.

3.54. dsn_notify

Type: string
Default: (empty)

This variable sets the request for when notification is returned. The string consists of a comma separated list (no spaces!) of one or more of the following: *never*, to never request notification, *failure*, to request notification on transmission failure, *delay*, to be notified of message delays, *success*, to be notified of successful transmission.

Example:

```
set dsn_notify="failure,delay"
```

Note: when using $sendmail for delivery, you should not enable this unless you are either using Sendmail 8.8.x or greater or a MTA providing a `sendmail(1)`-compatible interface supporting the `-N` option for DSN. For SMTP delivery, DSN support is auto-detected so that it depends on the server whether DSN will be used or not.

3.55. dsn_return

Type: string
Default: (empty)

This variable controls how much of your message is returned in DSN messages. It may be set to either *hdrs* to return just the message header, or *full* to return the full message.

Example:

```
set dsn_return=hdrs
```

Note: when using $sendmail for delivery, you should not enable this unless you are either using Sendmail 8.8.x or greater or a MTA providing a `sendmail(1)`-compatible interface supporting the `-R` option for DSN. For SMTP delivery, DSN support is auto-detected so that it depends on the server whether DSN will be used or not.

3.56. duplicate_threads

Type: boolean

Default: yes

This variable controls whether mutt, when $sort is set to *threads*, threads messages with the same Message-Id together. If it is *set*, it will indicate that it thinks they are duplicates of each other with an equals sign in the thread tree.

3.57. edit_headers

Type: boolean
Default: no

This option allows you to edit the header of your outgoing messages along with the body of your message.

Note that changes made to the References: and Date: headers are ignored for interoperability reasons.

3.58. editor

Type: path
Default: (empty)

This variable specifies which editor is used by mutt. It defaults to the value of the `$VISUAL`, or `$EDITOR`, environment variable, or to the string "vi" if neither of those are set.

The `$editor` string may contain a *%s* escape, which will be replaced by the name of the file to be edited. If the *%s* escape does not appear in `$editor`, a space and the name to be edited are appended.

The resulting string is then executed by running

```
sh -c 'string'
```

where *string* is the expansion of `$editor` described above.

3.59. encode_from

Type: boolean
Default: no

When *set*, mutt will quoted-printable encode messages when they contain the string "From " (note the trailing space) in the beginning of a line. This is useful to avoid the tampering certain mail delivery and transport agents tend to do with messages (in order to prevent tools from misinterpreting the line as a mbox message separator).

3.60. entropy_file

Type: path
Default: (empty)

The file which includes random data that is used to initialize SSL library functions.

3.61. envelope_from_address

Type: e-mail address
Default: (empty)

Manually sets the *envelope* sender for outgoing messages. This value is ignored if $use_envelope_from is *unset*.

3.62. escape

Type: string
Default: "~"

Escape character to use for functions in the built-in editor.

3.63. fast_reply

Type: boolean
Default: no

When *set*, the initial prompt for recipients and subject are skipped when replying to messages, and the initial prompt for subject is skipped when forwarding messages.

Note: this variable has no effect when the $autoedit variable is *set*.

3.64. fcc_attach

Type: quadoption
Default: yes

This variable controls whether or not attachments on outgoing messages are saved along with the main body of your message.

3.65. fcc_clear

Type: boolean
Default: no

When this variable is *set*, FCCs will be stored unencrypted and unsigned, even when the actual message is encrypted and/or signed. (PGP only)

3.66. folder

Type: path

Default: "`~/Mail`"

Specifies the default location of your mailboxes. A "+" or "=" at the beginning of a pathname will be expanded to the value of this variable. Note that if you change this variable (from the default) value you need to make sure that the assignment occurs *before* you use "+" or "=" for any other variables since expansion takes place when handling the "mailboxes" command.

3.67. folder_format

Type: string
Default: "`%2C %t %N %F %2l %-8.8u %-8.8g %8s %d %f`"

This variable allows you to customize the file browser display to your personal taste. This string is similar to $index_format, but has its own set of `printf(3)`-like sequences:

%C	current file number
%d	date/time folder was last modified
%D	date/time folder was last modified using $date_format.
%f	filename ("/" is appended to directory names, "@" to symbolic links and "*" to executable files)
%F	file permissions
%g	group name (or numeric gid, if missing)
%l	number of hard links
%N	N if folder has new mail, blank otherwise
%s	size in bytes
%t	"*" if the file is tagged, blank otherwise
%u	owner name (or numeric uid, if missing)
%>X	right justify the rest of the string and pad with character "X"
%|X	pad to the end of the line with character "X"
%*X	soft-fill with character "X" as pad

For an explanation of "soft-fill", see the $index_format documentation.

3.68. followup_to

Type: boolean
Default: yes

Controls whether or not the "Mail-Followup-To:" header field is generated when sending mail. When *set*, Mutt

will generate this field when you are replying to a known mailing list, specified with the "subscribe" or "lists" commands.

This field has two purposes. First, preventing you from receiving duplicate copies of replies to messages which you send to mailing lists, and second, ensuring that you do get a reply separately for any messages sent to known lists to which you are not subscribed.

The header will contain only the list's address for subscribed lists, and both the list address and your own email address for unsubscribed lists. Without this header, a group reply to your message sent to a subscribed list will be sent to both the list and your address, resulting in two copies of the same email for you.

3.69. force_name

Type: boolean
Default: no

This variable is similar to $save_name, except that Mutt will store a copy of your outgoing message by the username of the address you are sending to even if that mailbox does not exist.

Also see the $record variable.

3.70. forward_decode

Type: boolean
Default: yes

Controls the decoding of complex MIME messages into `text/plain` when forwarding a message. The message header is also RFC2047 decoded. This variable is only used, if $mime_forward is *unset*, otherwise $mime_forward_decode is used instead.

3.71. forward_decrypt

Type: boolean
Default: yes

Controls the handling of encrypted messages when forwarding a message. When *set*, the outer layer of encryption is stripped off. This variable is only used if $mime_forward is *set* and $mime_forward_decode is *unset*. (PGP only)

3.72. forward_edit

Type: quadoption
Default: yes

This quadoption controls whether or not the user is automatically placed in the editor when forwarding messages. For those who always want to forward with no modification, use a setting of "no".

3.73. forward_format

Type: string
Default: "`[%a: %s]`"

This variable controls the default subject when forwarding a message. It uses the same format sequences as the $index_format variable.

3.74. forward_quote

Type: boolean
Default: no

When *set*, forwarded messages included in the main body of the message (when $mime_forward is *unset*) will be quoted using $indent_string.

3.75. from

Type: e-mail address
Default: (empty)

When *set*, this variable contains a default from address. It can be overridden using "my_hdr" (including from a "send-hook") and $reverse_name. This variable is ignored if $use_from is *unset*.

This setting defaults to the contents of the environment variable `$EMAIL`.

3.76. gecos_mask

Type: regular expression
Default: "`^[^,]*`"

A regular expression used by mutt to parse the GECOS field of a password entry when expanding the alias. The default value will return the string up to the first "," encountered. If the GECOS field contains a string like "lastname, firstname" then you should set it to "`.*`".

This can be useful if you see the following behavior: you address an e-mail to user ID "stevef" whose full name is "Steve Franklin". If mutt expands "stevef" to ""Franklin" stevef@foo.bar" then you should set the $gecos_mask to a regular expression that will match the whole name so mutt will expand "Franklin" to "Franklin, Steve".

3.77. hdrs

Type: boolean
Default: yes

When *unset*, the header fields normally added by the "my_hdr" command are not created. This variable *must*

be unset before composing a new message or replying in order to take effect. If *set*, the user defined header fields are added to every new message.

3.78. header

Type: boolean
Default: no

When *set*, this variable causes Mutt to include the header of the message you are replying to into the edit buffer. The $weed setting applies.

3.79. header_cache

Type: path
Default: (empty)

This variable points to the header cache database. If pointing to a directory Mutt will contain a header cache database file per folder, if pointing to a file that file will be a single global header cache. By default it is *unset* so no header caching will be used.

Header caching can greatly improve speed when opening POP, IMAP MH or Maildir folders, see "caching" for details.

3.80. header_cache_compress

Type: boolean
Default: yes

When mutt is compiled with qdbm or tokyocabinet as header cache backend, this option determines whether the database will be compressed. Compression results in database files roughly being one fifth of the usual diskspace, but the decompression can result in a slower opening of cached folder(s) which in general is still much faster than opening non header cached folders.

3.81. header_cache_pagesize

Type: string
Default: "`16384`"

When mutt is compiled with either gdbm or bdb4 as the header cache backend, this option changes the database page size. Too large or too small values can waste space, memory, or CPU time. The default should be more or less optimal for most use cases.

3.82. help

Type: boolean

Default: yes

When *set*, help lines describing the bindings for the major functions provided by each menu are displayed on the first line of the screen.

Note: The binding will not be displayed correctly if the function is bound to a sequence rather than a single keystroke. Also, the help line may not be updated if a binding is changed while Mutt is running. Since this variable is primarily aimed at new users, neither of these should present a major problem.

3.83. hidden_host

Type: boolean
Default: no

When *set*, mutt will skip the host name part of $hostname variable when adding the domain part to addresses. This variable does not affect the generation of Message-IDs, and it will not lead to the cut-off of first-level domains.

3.84. hide_limited

Type: boolean
Default: no

When *set*, mutt will not show the presence of messages that are hidden by limiting, in the thread tree.

3.85. hide_missing

Type: boolean
Default: yes

When *set*, mutt will not show the presence of missing messages in the thread tree.

3.86. hide_thread_subject

Type: boolean
Default: yes

When *set*, mutt will not show the subject of messages in the thread tree that have the same subject as their parent or closest previously displayed sibling.

3.87. hide_top_limited

Type: boolean
Default: no

When *set*, mutt will not show the presence of messages that are hidden by limiting, at the top of threads in the

thread tree. Note that when $hide_limited is *set*, this option will have no effect.

3.88. hide_top_missing

Type: boolean
Default: yes

When *set*, mutt will not show the presence of missing messages at the top of threads in the thread tree. Note that when $hide_missing is *set*, this option will have no effect.

3.89. history

Type: number
Default: 10

This variable controls the size (in number of strings remembered) of the string history buffer per category. The buffer is cleared each time the variable is set.

3.90. history_file

Type: path
Default: "`~/.mutthistory`"

The file in which Mutt will save its history.

3.91. honor_disposition

Type: boolean
Default: no

When *set*, Mutt will not display attachments with a disposition of "attachment" inline even if it could render the part to plain text. These MIME parts can only be viewed from the attachment menu.

If *unset*, Mutt will render all MIME parts it can properly transform to plain text.

3.92. honor_followup_to

Type: quadoption
Default: yes

This variable controls whether or not a Mail-Followup-To header is honored when group-replying to a message.

3.93. hostname

Type: string

Default: (empty)

Specifies the fully-qualified hostname of the system mutt is running on containing the host's name and the DNS domain it belongs to. It is used as the domain part (after "@") for local email addresses as well as Message-Id headers.

Its value is determined at startup as follows: If the node's name as returned by the `uname(3)` function contains the hostname and the domain, these are used to construct $hostname. If there is no domain part returned, Mutt will look for a "domain" or "search" line in `/etc/resolv.conf` to determine the domain. Optionally, Mutt can be compiled with a fixed domain name in which case a detected one is not used.

Also see $use_domain and $hidden_host.

3.94. ignore_linear_white_space

Type: boolean
Default: no

This option replaces linear-white-space between encoded-word and text to a single space to prevent the display of MIME-encoded "Subject:" field from being divided into multiple lines.

3.95. ignore_list_reply_to

Type: boolean
Default: no

Affects the behavior of the `<reply>` function when replying to messages from mailing lists (as defined by the "subscribe" or "lists" commands). When *set*, if the "Reply-To:" field is set to the same value as the "To:" field, Mutt assumes that the "Reply-To:" field was set by the mailing list to automate responses to the list, and will ignore this field. To direct a response to the mailing list when this option is *set*, use the `<list-reply>` function; `<group-reply>` will reply to both the sender and the list.

3.96. imap_authenticators

Type: string
Default: (empty)

This is a colon-delimited list of authentication methods mutt may attempt to use to log in to an IMAP server, in the order mutt should try them. Authentication methods are either "login" or the right side of an IMAP "AUTH=xxx" capability string, e.g. "digest-md5", "gssapi" or "cram-md5". This option is case-insensitive. If it's *unset* (the default) mutt will try all available methods, in order from most-secure to least-secure.

Example:

```
set imap_authenticators="gssapi:cram-md5:login"
```

Note: Mutt will only fall back to other authentication methods if the previous methods are unavailable. If a method is available but authentication fails, mutt will not connect to the IMAP server.

3.97. imap_check_subscribed

Type: boolean
Default: no

When *set*, mutt will fetch the set of subscribed folders from your server on connection, and add them to the set of mailboxes it polls for new mail just as if you had issued individual "mailboxes" commands.

3.98. imap_delim_chars

Type: string
Default: "`/.`"

This contains the list of characters which you would like to treat as folder separators for displaying IMAP paths. In particular it helps in using the "=" shortcut for your *folder* variable.

3.99. imap_headers

Type: string
Default: (empty)

Mutt requests these header fields in addition to the default headers ("Date:", "From:", "Subject:", "To:", "Cc:", "Message-Id:", "References:", "Content-Type:", "Content-Description:", "In-Reply-To:", "Reply-To:", "Lines:", "List-Post:", "X-Label:") from IMAP servers before displaying the index menu. You may want to add more headers for spam detection.

Note: This is a space separated list, items should be uppercase and not contain the colon, e.g. "X-BOGOSITY X-SPAM-STATUS" for the "X-Bogosity:" and "X-Spam-Status:" header fields.

3.100. imap_idle

Type: boolean
Default: no

When *set*, mutt will attempt to use the IMAP IDLE extension to check for new mail in the current mailbox. Some servers (dovecot was the inspiration for this option) react badly to mutt's implementation. If your connection seems to freeze up periodically, try unsetting this.

3.101. imap_keepalive

Type: number
Default: 300

This variable specifies the maximum amount of time in seconds that mutt will wait before polling open IMAP connections, to prevent the server from closing them before mutt has finished with them. The default is well within the RFC-specified minimum amount of time (30 minutes) before a server is allowed to do this, but in practice the RFC does get violated every now and then. Reduce this number if you find yourself getting disconnected from your IMAP server due to inactivity.

3.102. imap_list_subscribed

Type: boolean
Default: no

This variable configures whether IMAP folder browsing will look for only subscribed folders or all folders. This can be toggled in the IMAP browser with the `<toggle-subscribed>` function.

3.103. imap_login

Type: string
Default: (empty)

Your login name on the IMAP server.

This variable defaults to the value of $imap_user.

3.104. imap_pass

Type: string
Default: (empty)

Specifies the password for your IMAP account. If *unset*, Mutt will prompt you for your password when you invoke the `<imap-fetch-mail>` function or try to open an IMAP folder.

Warning: you should only use this option when you are on a fairly secure machine, because the superuser can read your muttrc even if you are the only one who can read the file.

3.105. imap_passive

Type: boolean
Default: yes

When *set*, mutt will not open new IMAP connections to check for new mail. Mutt will only check for new mail over existing IMAP connections. This is useful if you don't want to be prompted to user/password pairs on mutt invocation, or if opening the connection is slow.

3.106. imap_peek

Type: boolean

Default: yes

When *set*, mutt will avoid implicitly marking your mail as read whenever you fetch a message from the server. This is generally a good thing, but can make closing an IMAP folder somewhat slower. This option exists to appease speed freaks.

3.107. imap_pipeline_depth

Type: number
Default: 15

Controls the number of IMAP commands that may be queued up before they are sent to the server. A deeper pipeline reduces the amount of time mutt must wait for the server, and can make IMAP servers feel much more responsive. But not all servers correctly handle pipelined commands, so if you have problems you might want to try setting this variable to 0.

Note: Changes to this variable have no effect on open connections.

3.108. imap_servernoise

Type: boolean
Default: yes

When *set*, mutt will display warning messages from the IMAP server as error messages. Since these messages are often harmless, or generated due to configuration problems on the server which are out of the users' hands, you may wish to suppress them at some point.

3.109. imap_user

Type: string
Default: (empty)

The name of the user whose mail you intend to access on the IMAP server.

This variable defaults to your user name on the local machine.

3.110. implicit_autoview

Type: boolean
Default: no

If set to “yes”, mutt will look for a mailcap entry with the “`copiousoutput`” flag set for *every* MIME attachment it doesn't have an internal viewer defined for. If such an entry is found, mutt will use the viewer defined in that entry to convert the body part to text form.

3.111. include

Type: quadoption
Default: ask-yes

Controls whether or not a copy of the message(s) you are replying to is included in your reply.

3.112. include_onlyfirst

Type: boolean
Default: no

Controls whether or not Mutt includes only the first attachment of the message you are replying.

3.113. indent_string

Type: string
Default: "> "

Specifies the string to prepend to each line of text quoted in a message to which you are replying. You are strongly encouraged not to change this value, as it tends to agitate the more fanatical netizens.

The value of this option is ignored if $text_flowed is set, too because the quoting mechanism is strictly defined for format=flowed.

This option is a format string, please see the description of $index_format for supported `printf(3)`-style sequences.

3.114. index_format

Type: string
Default: "`%4C %Z %{%b %d} %-15.15L (%?l?%4l&%4c?) %s`"

This variable allows you to customize the message index display to your personal taste.

"Format strings" are similar to the strings used in the C function `printf(3)` to format output (see the man page for more details). The following sequences are defined in Mutt:

%a	address of the author
%A	reply-to address (if present; otherwise: address of author)
%b	filename of the original message folder (think mailbox)
%B	the list to which the letter was sent, or else the folder name (%b).
%c	number of characters (bytes) in the message
%C	current message number
%d	date and time of the message in the format specified by $date_format converted to sender's time

	zone
%D	date and time of the message in the format specified by $date_format converted to the local time zone
%e	current message number in thread
%E	number of messages in current thread
%f	sender (address + real name), either From: or Return-Path:
%F	author name, or recipient name if the message is from you
%H	spam attribute(s) of this message
%i	message-id of the current message
%l	number of lines in the message (does not work with maildir, mh, and possibly IMAP folders)
%L	If an address in the “To:” or “Cc:” header field matches an address defined by the users “subscribe” command, this displays "To <list-name>", otherwise the same as %F.
%m	total number of message in the mailbox
%M	number of hidden messages if the thread is collapsed.
%N	message score
%n	author's real name (or address if missing)
%O	original save folder where mutt would formerly have stashed the message: list name or recipient name if not sent to a list
%P	progress indicator for the built-in pager (how much of the file has been displayed)
%s	subject of the message
%S	status of the message (“N”/“D”/“d”/“!”/“r”/*)
%t	“To:” field (recipients)
%T	the appropriate character from the $to_chars string
%u	user (login) name of the author
%v	first name of the author, or the recipient if the message is from you
%X	number of attachments (please see the “attachments” section for possible speed effects)
%y	“X-Label:” field, if present
%Y	“X-Label:” field, if present, and *(1)* not at part of a thread tree, *(2)* at the top of a thread, or *(3)* “X-Label:” is different from preceding message's “X-Label:”.
%Z	message status flags

%{fmt}	the date and time of the message is converted to sender's time zone, and “fmt” is expanded by the library function `strftime(3)`; a leading bang disables locales
%[fmt]	the date and time of the message is converted to the local time zone, and “fmt” is expanded by the library function `strftime(3)`; a leading bang disables locales
%(fmt)	the local date and time when the message was received. “fmt” is expanded by the library function `strftime(3)`; a leading bang disables locales
%<fmt>	the current local time. “fmt” is expanded by the library function `strftime(3)`; a leading bang disables locales.
%>X	right justify the rest of the string and pad with character “X”
%|X	pad to the end of the line with character “X”
%*X	soft-fill with character “X” as pad

“Soft-fill” deserves some explanation: Normal right-justification will print everything to the left of the “%>”, displaying padding and whatever lies to the right only if there's room. By contrast, soft-fill gives priority to the right-hand side, guaranteeing space to display it and showing padding only if there's still room. If necessary, soft-fill will eat text leftwards to make room for rightward text.

Note that these expandos are supported in “save-hook”, “fcc-hook” and “fcc-save-hook”, too.

3.115. ispell

Type: path
Default: “`ispell`”

How to invoke ispell (GNU's spell-checking software).

3.116. keep_flagged

Type: boolean
Default: no

If *set*, read messages marked as flagged will not be moved from your spool mailbox to your $mbox mailbox, or as a result of a “mbox-hook” command.

3.117. locale

Type: string
Default: “`C`”

The locale used by `strftime(3)` to format dates. Legal values are the strings your system accepts for the locale environment variable `$LC_TIME`.

3.118. mail_check

Type: number
Default: 5

This variable configures how often (in seconds) mutt should look for new mail. Also see the $timeout variable.

3.119. mail_check_recent

Type: boolean
Default: yes

When *set*, Mutt will only notify you about new mail that has been received since the last time you opened the mailbox. When *unset*, Mutt will notify you if any new mail exists in the mailbox, regardless of whether you have visited it recently.

When *$mark_old* is set, Mutt does not consider the mailbox to contain new mail if only old messages exist.

3.120. mailcap_path

Type: string
Default: (empty)

This variable specifies which files to consult when attempting to display MIME bodies not directly supported by Mutt.

3.121. mailcap_sanitize

Type: boolean
Default: yes

If *set*, mutt will restrict possible characters in mailcap % expandos to a well-defined set of safe characters. This is the safe setting, but we are not sure it doesn't break some more advanced MIME stuff.

DON'T CHANGE THIS SETTING UNLESS YOU ARE REALLY SURE WHAT YOU ARE DOING!

3.122. maildir_header_cache_verify

Type: boolean
Default: yes

Check for Maildir unaware programs other than mutt having modified maildir files when the header cache is in use. This incurs one `stat(2)` per message every time the folder is opened (which can be very slow for NFS folders).

3.123. maildir_trash

Type: boolean
Default: no

If *set*, messages marked as deleted will be saved with the maildir trashed flag instead of unlinked. **Note:** this only applies to maildir-style mailboxes. Setting it will have no effect on other mailbox types.

3.124. mark_old

Type: boolean
Default: yes

Controls whether or not mutt marks *new* **unread** messages as *old* if you exit a mailbox without reading them. With this option *set*, the next time you start mutt, the messages will show up with an "O" next to them in the index menu, indicating that they are old.

3.125. markers

Type: boolean
Default: yes

Controls the display of wrapped lines in the internal pager. If set, a "+" marker is displayed at the beginning of wrapped lines.

Also see the $smart_wrap variable.

3.126. mask

Type: regular expression
Default: "`!^\.[^.]`"

A regular expression used in the file browser, optionally preceded by the *not* operator "!". Only files whose names match this mask will be shown. The match is always case-sensitive.

3.127. mbox

Type: path
Default: "`~/mbox`"

This specifies the folder into which read mail in your $spoolfile folder will be appended.

Also see the $move variable.

3.128. mbox_type

Type: folder magic
Default: mbox

The default mailbox type used when creating new folders. May be any of “mbox”, “MMDF”, “MH” and “Maildir”. This is overridden by the `-m` command-line option.

3.129. menu_context

Type: number
Default: 0

This variable controls the number of lines of context that are given when scrolling through menus. (Similar to $pager_context.)

3.130. menu_move_off

Type: boolean
Default: yes

When *unset*, the bottom entry of menus will never scroll up past the bottom of the screen, unless there are less entries than lines. When *set*, the bottom entry may move off the bottom.

3.131. menu_scroll

Type: boolean
Default: no

When *set*, menus will be scrolled up or down one line when you attempt to move across a screen boundary. If *unset*, the screen is cleared and the next or previous page of the menu is displayed (useful for slow links to avoid many redraws).

3.132. message_cache_clean

Type: boolean
Default: no

If *set*, mutt will clean out obsolete entries from the message cache when the mailbox is synchronized. You probably only want to set it every once in a while, since it can be a little slow (especially for large folders).

3.133. message_cachedir

Type: path
Default: (empty)

Set this to a directory and mutt will cache copies of messages from your IMAP and POP servers here. You are free to remove entries at any time.

When setting this variable to a directory, mutt needs to fetch every remote message only once and can perform regular expression searches as fast as for local folders.

Also see the $message_cache_clean variable.

3.134. message_format

Type: string
Default: "`%s`"

This is the string displayed in the "attachment" menu for attachments of type `message/rfc822`. For a full listing of defined `printf(3)`-like sequences see the section on $index_format.

3.135. meta_key

Type: boolean
Default: no

If *set*, forces Mutt to interpret keystrokes with the high bit (bit 8) set as if the user had pressed the Esc key and whatever key remains after having the high bit removed. For example, if the key pressed has an ASCII value of `0xf8`, then this is treated as if the user had pressed Esc then "x". This is because the result of removing the high bit from `0xf8` is `0x78`, which is the ASCII character "x".

3.136. metoo

Type: boolean
Default: no

If *unset*, Mutt will remove your address (see the "alternates" command) from the list of recipients when replying to a message.

3.137. mh_purge

Type: boolean
Default: no

When *unset*, mutt will mimic mh's behavior and rename deleted messages to *,<old file name>* in mh folders instead of really deleting them. This leaves the message on disk but makes programs reading the folder ignore it. If the variable is *set*, the message files will simply be deleted.

This option is similar to $maildir_trash for Maildir folders.

3.138. mh_seq_flagged

Type: string
Default: "`flagged`"

The name of the MH sequence used for flagged messages.

3.139. mh_seq_replied

Type: string
Default: "`replied`"

The name of the MH sequence used to tag replied messages.

3.140. mh_seq_unseen

Type: string
Default: "`unseen`"

The name of the MH sequence used for unseen messages.

3.141. mime_forward

Type: quadoption
Default: no

When *set*, the message you are forwarding will be attached as a separate `message/rfc822` MIME part instead of included in the main body of the message. This is useful for forwarding MIME messages so the receiver can properly view the message as it was delivered to you. If you like to switch between MIME and not MIME from mail to mail, set this variable to "ask-no" or "ask-yes".

Also see $forward_decode and $mime_forward_decode.

3.142. mime_forward_decode

Type: boolean
Default: no

Controls the decoding of complex MIME messages into `text/plain` when forwarding a message while $mime_forward is *set*. Otherwise $forward_decode is used instead.

3.143. mime_forward_rest

Type: quadoption
Default: yes

When forwarding multiple attachments of a MIME message from the attachment menu, attachments which cannot be decoded in a reasonable manner will be attached to the newly composed message if this option is *set*.

3.144. mix_entry_format

Type: string
Default: "`%4n %c %-16s %a`"

This variable describes the format of a remailer line on the mixmaster chain selection screen. The following `printf(3)`-like sequences are supported:

%n	The running number on the menu.
%c	Remailer capabilities.
%s	The remailer's short name.
%a	The remailer's e-mail address.

3.145. mixmaster

Type: path
Default: "`mixmaster`"

This variable contains the path to the Mixmaster binary on your system. It is used with various sets of parameters to gather the list of known remailers, and to finally send a message through the mixmaster chain.

3.146. move

Type: quadoption
Default: no

Controls whether or not Mutt will move read messages from your spool mailbox to your $mbox mailbox, or as a result of a "mbox-hook" command.

3.147. narrow_tree

Type: boolean
Default: no

This variable, when *set*, makes the thread tree narrower, allowing deeper threads to fit on the screen.

3.148. net_inc

Type: number
Default: 10

Operations that expect to transfer a large amount of data over the network will update their progress every $net_inc kilobytes. If set to 0, no progress messages will be displayed.

See also $read_inc, $write_inc and $net_inc.

3.149. pager

Type: path
Default: "`builtin`"

This variable specifies which pager you would like to use to view messages. The value "builtin" means to use the built-in pager, otherwise this variable should specify the pathname of the external pager you would like to use.

Using an external pager may have some disadvantages: Additional keystrokes are necessary because you can't call mutt functions directly from the pager, and screen resizes cause lines longer than the screen width to be badly formatted in the help menu.

3.150. pager_context

Type: number
Default: 0

This variable controls the number of lines of context that are given when displaying the next or previous page in the internal pager. By default, Mutt will display the line after the last one on the screen at the top of the next page (0 lines of context).

This variable also specifies the amount of context given for search results. If positive, this many lines will be given before a match, if 0, the match will be top-aligned.

3.151. pager_format

Type: string
Default: "`-%Z- %C/%m: %-20.20n   %s%*  -- (%P)`"

This variable controls the format of the one-line message "status" displayed before each message in either the internal or an external pager. The valid sequences are listed in the $index_format section.

3.152. pager_index_lines

Type: number
Default: 0

Determines the number of lines of a mini-index which is shown when in the pager. The current message, unless near the top or bottom of the folder, will be roughly one third of the way down this mini-index, giving the reader the context of a few messages before and after the message. This is useful, for example, to determine how many messages remain to be read in the current thread. One of the lines is reserved for the status bar from the index, so a setting of 6 will only show 5 lines of the actual index. A value of 0 results in no index being shown. If the number of messages in the current folder is less than $pager_index_lines, then the index will only use as many lines as it needs.

3.153. pager_stop

Type: boolean
Default: no

When *set*, the internal-pager will **not** move to the next message when you are at the end of a message and invoke the `<next-page>` function.

3.154. pgp_auto_decode

Type: boolean
Default: no

If *set*, mutt will automatically attempt to decrypt traditional PGP messages whenever the user performs an operation which ordinarily would result in the contents of the message being operated on. For example, if the user displays a pgp-traditional message which has not been manually checked with the `<check-traditional-pgp>` function, mutt will automatically check the message for traditional pgp.

3.155. pgp_autoinline

Type: boolean
Default: no

This option controls whether Mutt generates old-style inline (traditional) PGP encrypted or signed messages under certain circumstances. This can be overridden by use of the pgp menu, when inline is not required.

Note that Mutt might automatically use PGP/MIME for messages which consist of more than a single MIME part. Mutt can be configured to ask before sending PGP/MIME messages when inline (traditional) would not work.

Also see the $pgp_mime_auto variable.

Also note that using the old-style PGP message format is **strongly deprecated**. (PGP only)

3.156. pgp_check_exit

Type: boolean
Default: yes

If *set*, mutt will check the exit code of the PGP subprocess when signing or encrypting. A non-zero exit code means that the subprocess failed. (PGP only)

3.157. pgp_clearsign_command

Type: string
Default: (empty)

This format is used to create an old-style "clearsigned" PGP message. Note that the use of this format is **strongly deprecated**.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. (PGP only)

3.158. pgp_decode_command

Type: string
Default: (empty)

This format strings specifies a command which is used to decode application/pgp attachments.

The PGP command formats have their own set of `printf(3)`-like sequences:

%p	Expands to PGPPASSFD=0 when a pass phrase is needed, to an empty string otherwise. Note: This may be used with a %? construct.
%f	Expands to the name of a file containing a message.
%s	Expands to the name of a file containing the signature part of a `multipart/signed` attachment when verifying it.
%a	The value of $pgp_sign_as.
%r	One or more key IDs.

For examples on how to configure these formats for the various versions of PGP which are floating around, see the pgp and gpg sample configuration files in the `samples/` subdirectory which has been installed on your system alongside the documentation. (PGP only)

3.159. pgp_decrypt_command

Type: string
Default: (empty)

This command is used to decrypt a PGP encrypted message.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. (PGP only)

3.160. pgp_encrypt_only_command

Type: string
Default: (empty)

This command is used to encrypt a body part without signing it.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. (PGP only)

3.161. pgp_encrypt_sign_command

Type: string
Default: (empty)

This command is used to both sign and encrypt a body part.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. (PGP only)

3.162. pgp_entry_format

Type: string
Default: "`%4n %t%f %4l/0x%k %-4a %2c %u`"

This variable allows you to customize the PGP key selection menu to your personal taste. This string is similar to $index_format, but has its own set of `printf(3)`-like sequences:

%n	number
%k	key id
%u	user id
%a	algorithm
%l	key length
%f	flags
%c	capabilities
%t	trust/validity of the key-uid association
%[<s>]	date of the key where <s> is an `strftime(3)` expression

(PGP only)

3.163. pgp_export_command

Type: string
Default: (empty)

This command is used to export a public key from the user's key ring.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences.

(PGP only)

3.164. pgp_getkeys_command

Type: string
Default: (empty)

This command is invoked whenever Mutt needs to fetch the public key associated with an email address. Of the sequences supported by $pgp_decode_command, %r is the only `printf(3)`-like sequence used with this format. Note that in this case, %r expands to the email address, not the public key ID (the key ID is unknown, which is why Mutt is invoking this command). (PGP only)

3.165. pgp_good_sign

Type: regular expression
Default: (empty)

If you assign a text to this variable, then a PGP signature is only considered verified if the output from $pgp_verify_command contains the text. Use this variable if the exit code from the command is 0 even for bad signatures. (PGP only)

3.166. pgp_ignore_subkeys

Type: boolean
Default: yes

Setting this variable will cause Mutt to ignore OpenPGP subkeys. Instead, the principal key will inherit the subkeys' capabilities. *Unset* this if you want to play interesting key selection games. (PGP only)

3.167. pgp_import_command

Type: string
Default: (empty)

This command is used to import a key from a message into the user's public key ring.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. (PGP only)

3.168. pgp_list_pubring_command

Type: string
Default: (empty)

This command is used to list the public key ring's contents. The output format must be analogous to the one used by

```
gpg --list-keys --with-colons.
```

This format is also generated by the `pgpring` utility which comes with mutt.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. (PGP only)

3.169. pgp_list_secring_command

Type: string
Default: (empty)

This command is used to list the secret key ring's contents. The output format must be analogous to the one used by:

```
gpg --list-keys --with-colons.
```

This format is also generated by the `pgpring` utility which comes with mutt.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. (PGP only)

3.170. pgp_long_ids

Type: boolean
Default: no

If *set*, use 64 bit PGP key IDs, if *unset* use the normal 32 bit key IDs. (PGP only)

3.171. pgp_mime_auto

Type: quadoption
Default: ask-yes

This option controls whether Mutt will prompt you for automatically sending a (signed/encrypted) message using PGP/MIME when inline (traditional) fails (for any reason).

Also note that using the old-style PGP message format is **strongly deprecated**. (PGP only)

3.172. pgp_replyinline

Type: boolean
Default: no

Setting this variable will cause Mutt to always attempt to create an inline (traditional) message when replying to a message which is PGP encrypted/signed inline. This can be overridden by use of the pgp menu, when inline is

not required. This option does not automatically detect if the (replied-to) message is inline; instead it relies on Mutt internals for previously checked/flagged messages.

Note that Mutt might automatically use PGP/MIME for messages which consist of more than a single MIME part. Mutt can be configured to ask before sending PGP/MIME messages when inline (traditional) would not work.

Also see the $pgp_mime_auto variable.

Also note that using the old-style PGP message format is **strongly deprecated**. (PGP only)

3.173. pgp_retainable_sigs

Type: boolean
Default: no

If *set*, signed and encrypted messages will consist of nested `multipart/signed` and `multipart/encrypted` body parts.

This is useful for applications like encrypted and signed mailing lists, where the outer layer (`multipart/encrypted`) can be easily removed, while the inner `multipart/signed` part is retained. (PGP only)

3.174. pgp_show_unusable

Type: boolean
Default: yes

If *set*, mutt will display non-usable keys on the PGP key selection menu. This includes keys which have been revoked, have expired, or have been marked as "disabled" by the user. (PGP only)

3.175. pgp_sign_as

Type: string
Default: (empty)

If you have more than one key pair, this option allows you to specify which of your private keys to use. It is recommended that you use the keyid form to specify your key (e.g. `0x00112233`). (PGP only)

3.176. pgp_sign_command

Type: string
Default: (empty)

This command is used to create the detached PGP signature for a `multipart/signed` PGP/MIME body part.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. (PGP only)

3.177. pgp_sort_keys

Type: sort order
Default: address

Specifies how the entries in the pgp menu are sorted. The following are legal values:

address	sort alphabetically by user id
keyid	sort alphabetically by key id
date	sort by key creation date
trust	sort by the trust of the key

If you prefer reverse order of the above values, prefix it with "reverse-". (PGP only)

3.178. pgp_strict_enc

Type: boolean
Default: yes

If *set*, Mutt will automatically encode PGP/MIME signed messages as quoted-printable. Please note that unsetting this variable may lead to problems with non-verifyable PGP signatures, so only change this if you know what you are doing. (PGP only)

3.179. pgp_timeout

Type: number
Default: 300

The number of seconds after which a cached passphrase will expire if not used. (PGP only)

3.180. pgp_use_gpg_agent

Type: boolean
Default: no

If *set*, mutt will use a possibly-running `gpg-agent(1)` process. (PGP only)

3.181. pgp_verify_command

Type: string

Default: (empty)

This command is used to verify PGP signatures.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. (PGP only)

3.182. pgp_verify_key_command

Type: string
Default: (empty)

This command is used to verify key information from the key selection menu.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. (PGP only)

3.183. pipe_decode

Type: boolean
Default: no

Used in connection with the `<pipe-message>` command. When *unset*, Mutt will pipe the messages without any preprocessing. When *set*, Mutt will weed headers and will attempt to decode the messages first.

3.184. pipe_sep

Type: string
Default: "\n"

The separator to add between messages when piping a list of tagged messages to an external Unix command.

3.185. pipe_split

Type: boolean
Default: no

Used in connection with the `<pipe-message>` function following `<tag-prefix>`. If this variable is *unset*, when piping a list of tagged messages Mutt will concatenate the messages and will pipe them all concatenated. When *set*, Mutt will pipe the messages one by one. In both cases the messages are piped in the current sorted order, and the $pipe_sep separator is added after each message.

3.186. pop_auth_try_all

Type: boolean
Default: yes

If *set*, Mutt will try all available authentication methods. When *unset*, Mutt will only fall back to other authentication methods if the previous methods are unavailable. If a method is available but authentication fails, Mutt will not connect to the POP server.

3.187. pop_authenticators

Type: string
Default: (empty)

This is a colon-delimited list of authentication methods mutt may attempt to use to log in to an POP server, in the order mutt should try them. Authentication methods are either "user", "apop" or any SASL mechanism, e.g. "digest-md5", "gssapi" or "cram-md5". This option is case-insensitive. If this option is *unset* (the default) mutt will try all available methods, in order from most-secure to least-secure.

Example:

```
set pop_authenticators="digest-md5:apop:user"
```

3.188. pop_checkinterval

Type: number
Default: 60

This variable configures how often (in seconds) mutt should look for new mail in the currently selected mailbox if it is a POP mailbox.

3.189. pop_delete

Type: quadoption
Default: ask-no

If *set*, Mutt will delete successfully downloaded messages from the POP server when using the <fetch-mail> function. When *unset*, Mutt will download messages but also leave them on the POP server.

3.190. pop_host

Type: string
Default: (empty)

The name of your POP server for the <fetch-mail> function. You can also specify an alternative port, username and password, i.e.:

```
[pop[s]://][username[:password]@]popserver[:port]
```

where "[...]" denotes an optional part.

3.191. pop_last

Type: boolean
Default: no

If this variable is *set*, mutt will try to use the "`LAST`" POP command for retrieving only unread messages from the POP server when using the `<fetch-mail>` function.

3.192. pop_pass

Type: string
Default: (empty)

Specifies the password for your POP account. If *unset*, Mutt will prompt you for your password when you open a POP mailbox.

Warning: you should only use this option when you are on a fairly secure machine, because the superuser can read your muttrc even if you are the only one who can read the file.

3.193. pop_reconnect

Type: quadoption
Default: ask-yes

Controls whether or not Mutt will try to reconnect to the POP server if the connection is lost.

3.194. pop_user

Type: string
Default: (empty)

Your login name on the POP server.

This variable defaults to your user name on the local machine.

3.195. post_indent_string

Type: string
Default: (empty)

Similar to the $attribution variable, Mutt will append this string after the inclusion of a message which is being replied to.

3.196. postpone

Type: quadoption

Default: ask-yes

Controls whether or not messages are saved in the $postponed mailbox when you elect not to send immediately.

Also see the $recall variable.

3.197. postponed

Type: path
Default: "~/postponed"

Mutt allows you to indefinitely "postpone sending a message" which you are editing. When you choose to postpone a message, Mutt saves it in the mailbox specified by this variable.

Also see the $postpone variable.

3.198. preconnect

Type: string
Default: (empty)

If *set*, a shell command to be executed if mutt fails to establish a connection to the server. This is useful for setting up secure connections, e.g. with `ssh(1)`. If the command returns a nonzero status, mutt gives up opening the server. Example:

```
set preconnect="ssh -f -q -L 1234:mailhost.net:143 mailhost.net \
sleep 20 < /dev/null > /dev/null"
```

Mailbox "foo" on "mailhost.net" can now be reached as "{localhost:1234}foo".

Note: For this example to work, you must be able to log in to the remote machine without having to enter a password.

3.199. print

Type: quadoption
Default: ask-no

Controls whether or not Mutt really prints messages. This is set to "ask-no" by default, because some people accidentally hit "p" often.

3.200. print_command

Type: path
Default: "`lpr`"

This specifies the command pipe that should be used to print messages.

3.201. print_decode

Type: boolean
Default: yes

Used in connection with the `<print-message>` command. If this option is *set*, the message is decoded before it is passed to the external command specified by $print_command. If this option is *unset*, no processing will be applied to the message when printing it. The latter setting may be useful if you are using some advanced printer filter which is able to properly format e-mail messages for printing.

3.202. print_split

Type: boolean
Default: no

Used in connection with the `<print-message>` command. If this option is *set*, the command specified by $print_command is executed once for each message which is to be printed. If this option is *unset*, the command specified by $print_command is executed only once, and all the messages are concatenated, with a form feed as the message separator.

Those who use the `enscript`(1) program's mail-printing mode will most likely want to *set* this option.

3.203. prompt_after

Type: boolean
Default: yes

If you use an *external* $pager, setting this variable will cause Mutt to prompt you for a command when the pager exits rather than returning to the index menu. If *unset*, Mutt will return to the index menu when the external pager exits.

3.204. query_command

Type: path
Default: (empty)

This specifies the command Mutt will use to make external address queries. The string may contain a “%s”, which will be substituted with the query string the user types. Mutt will add quotes around the string substituted for “%s” automatically according to shell quoting rules, so you should avoid adding your own. If no “%s” is found in the string, Mutt will append the user's query to the end of the string. See “query” for more information.

3.205. query_format

Type: string
Default: "`%4c %t %-25.25a %-25.25n %?e?(%e)?`"

This variable describes the format of the "query" menu. The following `printf(3)`-style sequences are understood:

%a	destination address
%c	current entry number
%e	extra information *
%n	destination name
%t	"*" if current entry is tagged, a space otherwise
%>X	right justify the rest of the string and pad with "X"
%|X	pad to the end of the line with "X"
%*X	soft-fill with character "X" as pad

For an explanation of "soft-fill", see the $index_format documentation.

* = can be optionally printed if nonzero, see the $status_format documentation.

3.206. quit

Type: quadoption
Default: yes

This variable controls whether "quit" and "exit" actually quit from mutt. If this option is *set*, they do quit, if it is *unset*, they have no effect, and if it is set to *ask-yes* or *ask-no*, you are prompted for confirmation when you try to quit.

3.207. quote_regexp

Type: regular expression
Default: "`^([ \t]*[|>:}#])+`"

A regular expression used in the internal pager to determine quoted sections of text in the body of a message. Quoted text may be filtered out using the `<toggle-quoted>` command, or colored according to the "color quoted" family of directives.

Higher levels of quoting may be colored differently ("color quoted1", "color quoted2", etc.). The quoting level is determined by removing the last character from the matched text and recursively reapplying the regular expression until it fails to produce a match.

Match detection may be overridden by the $smileys regular expression.

3.208. read_inc

Type: number
Default: 10

If set to a value greater than 0, Mutt will display which message it is currently on when reading a mailbox or when performing search actions such as search and limit. The message is printed after this many messages have been read or searched (e.g., if set to 25, Mutt will print a message when it is at message 25, and then again when it gets to message 50). This variable is meant to indicate progress when reading or searching large mailboxes which may take some time. When set to 0, only a single message will appear before the reading the mailbox.

Also see the $write_inc, $net_inc and $time_inc variables and the "tuning" section of the manual for performance considerations.

3.209. read_only

Type: boolean
Default: no

If *set*, all folders are opened in read-only mode.

3.210. realname

Type: string
Default: (empty)

This variable specifies what "real" or "personal" name should be used when sending messages.

By default, this is the GECOS field from `/etc/passwd`. Note that this variable will *not* be used when the user has set a real name in the $from variable.

3.211. recall

Type: quadoption
Default: ask-yes

Controls whether or not Mutt recalls postponed messages when composing a new message.

Setting this variable to is not generally useful, and thus not recommended.

Also see $postponed variable.

3.212. record

Type: path

Default: "`~/sent`"

This specifies the file into which your outgoing messages should be appended. (This is meant as the primary method for saving a copy of your messages, but another way to do this is using the "my_hdr" command to create a "Bcc:" field with your email address in it.)

The value of *$record* is overridden by the $force_name and $save_name variables, and the "fcc-hook" command.

3.213. reflow_text

Type: boolean
Default: yes

When *set*, Mutt will reformat paragraphs in text/plain parts marked format=flowed. If *unset*, Mutt will display paragraphs unaltered from how they appear in the message body. See RFC3676 for details on the *format=flowed* format.

Also see $reflow_wrap, and $wrap.

3.214. reflow_wrap

Type: number
Default: 78

This variable controls the maximum paragraph width when reformatting text/plain parts when $reflow_text is *set*. When the value is 0, paragraphs will be wrapped at the terminal's right margin. A positive value sets the paragraph width relative to the left margin. A negative value set the paragraph width relative to the right margin.

Also see $wrap.

3.215. reply_regexp

Type: regular expression
Default: "`^(re([\[0-9\]+])*|aw):[ \t]*`"

A regular expression used to recognize reply messages when threading and replying. The default value corresponds to the English "Re:" and the German "Aw:".

3.216. reply_self

Type: boolean
Default: no

If *unset* and you are replying to a message sent by you, Mutt will assume that you want to reply to the recipients of that message rather than to yourself.

Also see the "alternates" command.

3.217. reply_to

Type: quadoption
Default: ask-yes

If *set*, when replying to a message, Mutt will use the address listed in the Reply-to: header as the recipient of the reply. If *unset*, it will use the address in the From: header field instead. This option is useful for reading a mailing list that sets the Reply-To: header field to the list address and you want to send a private message to the author of a message.

3.218. resolve

Type: boolean
Default: yes

When *set*, the cursor will be automatically advanced to the next (possibly undeleted) message whenever a command that modifies the current message is executed.

3.219. reverse_alias

Type: boolean
Default: no

This variable controls whether or not Mutt will display the "personal" name from your aliases in the index menu if it finds an alias that matches the message's sender. For example, if you have the following alias:

```
alias juser abd30425@somewhere.net (Joe User)
```

and then you receive mail which contains the following header:

```
From: abd30425@somewhere.net
```

It would be displayed in the index menu as "Joe User" instead of "abd30425@somewhere.net." This is useful when the person's e-mail address is not human friendly.

3.220. reverse_name

Type: boolean
Default: no

It may sometimes arrive that you receive mail to a certain machine, move the messages to another machine, and reply to some the messages from there. If this variable is *set*, the default *From:* line of the reply messages is built using the address where you received the messages you are replying to **if** that address matches your

“alternates”. If the variable is *unset*, or the address that would be used doesn't match your “alternates”, the *From:* line will use your address on the current machine.

Also see the “alternates” command.

3.221. reverse_realname

Type: boolean
Default: yes

This variable fine-tunes the behavior of the $reverse_name feature. When it is *set*, mutt will use the address from incoming messages as-is, possibly including eventual real names. When it is *unset*, mutt will override any such real names with the setting of the $realname variable.

3.222. rfc2047_parameters

Type: boolean
Default: no

When this variable is *set*, Mutt will decode RFC2047-encoded MIME parameters. You want to set this variable when mutt suggests you to save attachments to files named like:

```
=?iso-8859-1?Q?file=5F=E4=5F991116=2Ezip?=
```

When this variable is *set* interactively, the change won't be active until you change folders.

Note that this use of RFC2047's encoding is explicitly prohibited by the standard, but nevertheless encountered in the wild.

Also note that setting this parameter will *not* have the effect that mutt *generates* this kind of encoding. Instead, mutt will unconditionally use the encoding specified in RFC2231.

3.223. save_address

Type: boolean
Default: no

If *set*, mutt will take the sender's full address when choosing a default folder for saving a mail. If $save_name or $force_name is *set* too, the selection of the Fcc folder will be changed as well.

3.224. save_empty

Type: boolean
Default: yes

When *unset*, mailboxes which contain no saved messages will be removed when closed (the exception is

$spoolfile which is never removed). If *set*, mailboxes are never removed.

Note: This only applies to mbox and MMDF folders, Mutt does not delete MH and Maildir directories.

3.225. save_history

Type: number
Default: 0

This variable controls the size of the history (per category) saved in the $history_file file.

3.226. save_name

Type: boolean
Default: no

This variable controls how copies of outgoing messages are saved. When *set*, a check is made to see if a mailbox specified by the recipient address exists (this is done by searching for a mailbox in the $folder directory with the *username* part of the recipient address). If the mailbox exists, the outgoing message will be saved to that mailbox, otherwise the message is saved to the $record mailbox.

Also see the $force_name variable.

3.227. score

Type: boolean
Default: yes

When this variable is *unset*, scoring is turned off. This can be useful to selectively disable scoring for certain folders when the $score_threshold_delete variable and related are used.

3.228. score_threshold_delete

Type: number
Default: -1

Messages which have been assigned a score equal to or lower than the value of this variable are automatically marked for deletion by mutt. Since mutt scores are always greater than or equal to zero, the default setting of this variable will never mark a message for deletion.

3.229. score_threshold_flag

Type: number
Default: 9999

Messages which have been assigned a score greater than or equal to this variable's value are automatically

marked "flagged".

3.230. score_threshold_read

Type: number
Default: -1

Messages which have been assigned a score equal to or lower than the value of this variable are automatically marked as read by mutt. Since mutt scores are always greater than or equal to zero, the default setting of this variable will never mark a message read.

3.231. search_context

Type: number
Default: 0

For the pager, this variable specifies the number of lines shown before search results. By default, search results will be top-aligned.

3.232. send_charset

Type: string
Default: "`us-ascii:iso-8859-1:utf-8`"

A colon-delimited list of character sets for outgoing messages. Mutt will use the first character set into which the text can be converted exactly. If your $charset is not "iso-8859-1" and recipients may not understand "UTF-8", it is advisable to include in the list an appropriate widely used standard character set (such as "iso-8859-2", "koi8-r" or "iso-2022-jp") either instead of or after "iso-8859-1".

In case the text cannot be converted into one of these exactly, mutt uses $charset as a fallback.

3.233. sendmail

Type: path
Default: "`/usr/sbin/sendmail -oem -oi`"

Specifies the program and arguments used to deliver mail sent by Mutt. Mutt expects that the specified program interprets additional arguments as recipient addresses.

3.234. sendmail_wait

Type: number
Default: 0

Specifies the number of seconds to wait for the $sendmail process to finish before giving up and putting delivery in the background.

Mutt interprets the value of this variable as follows:

>0	number of seconds to wait for sendmail to finish before continuing
0	wait forever for sendmail to finish
<0	always put sendmail in the background without waiting

Note that if you specify a value other than 0, the output of the child process will be put in a temporary file. If there is some error, you will be informed as to where to find the output.

3.235. shell

Type: path
Default: (empty)

Command to use when spawning a subshell. By default, the user's login shell from `/etc/passwd` is used.

3.236. sig_dashes

Type: boolean
Default: yes

If *set*, a line containing "-- " (note the trailing space) will be inserted before your $signature. It is **strongly** recommended that you not *unset* this variable unless your signature contains just your name. The reason for this is because many software packages use "-- \n" to detect your signature. For example, Mutt has the ability to highlight the signature in a different color in the built-in pager.

3.237. sig_on_top

Type: boolean
Default: no

If *set*, the signature will be included before any quoted or forwarded text. It is **strongly** recommended that you do not set this variable unless you really know what you are doing, and are prepared to take some heat from netiquette guardians.

3.238. signature

Type: path
Default: "`~/.signature`"

Specifies the filename of your signature, which is appended to all outgoing messages. If the filename ends with a pipe ("|"), it is assumed that filename is a shell command and input should be read from its standard output.

3.239. simple_search

Type: string
Default: "`~f %s | ~s %s`"

Specifies how Mutt should expand a simple search into a real search pattern. A simple search is one that does not contain any of the "~" pattern operators. See "patterns" for more information on search patterns.

For example, if you simply type "joe" at a search or limit prompt, Mutt will automatically expand it to the value specified by this variable by replacing "%s" with the supplied string. For the default value, "joe" would be expanded to: "~f joe | ~s joe".

3.240. sleep_time

Type: number
Default: 1

Specifies time, in seconds, to pause while displaying certain informational messages, while moving from folder to folder and after expunging messages from the current folder. The default is to pause one second, so a value of zero for this option suppresses the pause.

3.241. smart_wrap

Type: boolean
Default: yes

Controls the display of lines longer than the screen width in the internal pager. If *set*, long lines are wrapped at a word boundary. If *unset*, lines are simply wrapped at the screen edge. Also see the $markers variable.

3.242. smileys

Type: regular expression
Default: "`(>From )|(:[-^]?[][)(><}{|/DP])`"

The *pager* uses this variable to catch some common false positives of $quote_regexp, most notably smileys and not consider a line quoted text if it also matches $smileys. This mostly happens at the beginning of a line.

3.243. smime_ask_cert_label

Type: boolean
Default: yes

This flag controls whether you want to be asked to enter a label for a certificate about to be added to the database or not. It is *set* by default. (S/MIME only)

3.244. smime_ca_location

Type: path

Default: (empty)

This variable contains the name of either a directory, or a file which contains trusted certificates for use with OpenSSL. (S/MIME only)

3.245. smime_certificates

Type: path
Default: (empty)

Since for S/MIME there is no pubring/secring as with PGP, mutt has to handle storage and retrieval of keys by itself. This is very basic right now, and keys and certificates are stored in two different directories, both named as the hash-value retrieved from OpenSSL. There is an index file which contains mailbox-address keyid pairs, and which can be manually edited. This option points to the location of the certificates. (S/MIME only)

3.246. smime_decrypt_command

Type: string
Default: (empty)

This format string specifies a command which is used to decrypt `application/x-pkcs7-mime` attachments.

The OpenSSL command formats have their own set of `printf(3)`-like sequences similar to PGP's:

%f	Expands to the name of a file containing a message.
%s	Expands to the name of a file containing the signature part of a `multipart/signed` attachment when verifying it.
%k	The key-pair specified with $smime_default_key
%c	One or more certificate IDs.
%a	The algorithm used for encryption.
%C	CA location: Depending on whether $smime_ca_location points to a directory or file, this expands to "-CApath $smime_ca_location" or "-CAfile $smime_ca_location".

For examples on how to configure these formats, see the `smime.rc` in the `samples/` subdirectory which has been installed on your system alongside the documentation. (S/MIME only)

3.247. smime_decrypt_use_default_key

Type: boolean
Default: yes

If *set* (default) this tells mutt to use the default key for decryption. Otherwise, if managing multiple certificate-key-pairs, mutt will try to use the mailbox-address to determine the key to use. It will ask you to supply a key,

if it can't find one. (S/MIME only)

3.248. smime_default_key

Type: string
Default: (empty)

This is the default key-pair to use for signing. This must be set to the keyid (the hash-value that OpenSSL generates) to work properly (S/MIME only)

3.249. smime_encrypt_command

Type: string
Default: (empty)

This command is used to create encrypted S/MIME messages.

This is a format string, see the $smime_decrypt_command command for possible `printf(3)`-like sequences. (S/MIME only)

3.250. smime_encrypt_with

Type: string
Default: (empty)

This sets the algorithm that should be used for encryption. Valid choices are “des”, “des3”, “rc2-40”, “rc2-64”, “rc2-128”. If *unset*, “3des” (TripleDES) is used. (S/MIME only)

3.251. smime_get_cert_command

Type: string
Default: (empty)

This command is used to extract X509 certificates from a PKCS7 structure.

This is a format string, see the $smime_decrypt_command command for possible `printf(3)`-like sequences. (S/MIME only)

3.252. smime_get_cert_email_command

Type: string
Default: (empty)

This command is used to extract the mail address(es) used for storing X509 certificates, and for verification purposes (to check whether the certificate was issued for the sender's mailbox).

This is a format string, see the $smime_decrypt_command command for possible `printf(3)`-like sequences. (S/MIME only)

3.253. smime_get_signer_cert_command

Type: string
Default: (empty)

This command is used to extract only the signers X509 certificate from a S/MIME signature, so that the certificate's owner may get compared to the email's "From:" field.

This is a format string, see the $smime_decrypt_command command for possible `printf(3)`-like sequences. (S/MIME only)

3.254. smime_import_cert_command

Type: string
Default: (empty)

This command is used to import a certificate via smime_keys.

This is a format string, see the $smime_decrypt_command command for possible `printf(3)`-like sequences. (S/MIME only)

3.255. smime_is_default

Type: boolean
Default: no

The default behavior of mutt is to use PGP on all auto-sign/encryption operations. To override and to use OpenSSL instead this must be *set*. However, this has no effect while replying, since mutt will automatically select the same application that was used to sign/encrypt the original message. (Note that this variable can be overridden by unsetting $crypt_autosmime.) (S/MIME only)

3.256. smime_keys

Type: path
Default: (empty)

Since for S/MIME there is no pubring/secring as with PGP, mutt has to handle storage and retrieval of keys/certs by itself. This is very basic right now, and stores keys and certificates in two different directories, both named as the hash-value retrieved from OpenSSL. There is an index file which contains mailbox-address keyid pair, and which can be manually edited. This option points to the location of the private keys. (S/MIME only)

3.257. smime_pk7out_command

Type: string
Default: (empty)

This command is used to extract PKCS7 structures of S/MIME signatures, in order to extract the public X509 certificate(s).

This is a format string, see the $smime_decrypt_command command for possible `printf(3)`-like sequences. (S/MIME only)

3.258. smime_sign_command

Type: string
Default: (empty)

This command is used to created S/MIME signatures of type `multipart/signed`, which can be read by all mail clients.

This is a format string, see the $smime_decrypt_command command for possible `printf(3)`-like sequences. (S/MIME only)

3.259. smime_sign_opaque_command

Type: string
Default: (empty)

This command is used to created S/MIME signatures of type `application/x-pkcs7-signature`, which can only be handled by mail clients supporting the S/MIME extension.

This is a format string, see the $smime_decrypt_command command for possible `printf(3)`-like sequences. (S/MIME only)

3.260. smime_timeout

Type: number
Default: 300

The number of seconds after which a cached passphrase will expire if not used. (S/MIME only)

3.261. smime_verify_command

Type: string
Default: (empty)

This command is used to verify S/MIME signatures of type `multipart/signed`.

This is a format string, see the $smime_decrypt_command command for possible `printf(3)`-like sequences.

(S/MIME only)

3.262. smime_verify_opaque_command

Type: string
Default: (empty)

This command is used to verify S/MIME signatures of type `application/x-pkcs7-mime`.

This is a format string, see the $smime_decrypt_command command for possible `printf(3)`-like sequences. (S/MIME only)

3.263. smtp_authenticators

Type: string
Default: (empty)

This is a colon-delimited list of authentication methods mutt may attempt to use to log in to an SMTP server, in the order mutt should try them. Authentication methods are any SASL mechanism, e.g. "digest-md5", "gssapi" or "cram-md5". This option is case-insensitive. If it is "unset" (the default) mutt will try all available methods, in order from most-secure to least-secure.

Example:

```
set smtp_authenticators="digest-md5:cram-md5"
```

3.264. smtp_pass

Type: string
Default: (empty)

Specifies the password for your SMTP account. If *unset*, Mutt will prompt you for your password when you first send mail via SMTP. See $smtp_url to configure mutt to send mail via SMTP.

Warning: you should only use this option when you are on a fairly secure machine, because the superuser can read your muttrc even if you are the only one who can read the file.

3.265. smtp_url

Type: string
Default: (empty)

Defines the SMTP smarthost where sent messages should relayed for delivery. This should take the form of an SMTP URL, e.g.:

```
smtp[s]://[user[:pass]@]host[:port]
```

where “[...]” denotes an optional part. Setting this variable overrides the value of the $sendmail variable.

3.266. sort

Type: sort order
Default: date

Specifies how to sort messages in the “index” menu. Valid values are:

- date or date-sent
- date-received
- from
- mailbox-order (unsorted)
- score
- size
- spam
- subject
- threads
- to

You may optionally use the “reverse-” prefix to specify reverse sorting order (example: “`set sort=reverse-date-sent`”).

3.267. sort_alias

Type: sort order
Default: alias

Specifies how the entries in the “alias” menu are sorted. The following are legal values:

- address (sort alphabetically by email address)
- alias (sort alphabetically by alias name)
- unsorted (leave in order specified in .muttrc)

3.268. sort_aux

Type: sort order

Default: date

When sorting by threads, this variable controls how threads are sorted in relation to other threads, and how the branches of the thread trees are sorted. This can be set to any value that $sort can, except "threads" (in that case, mutt will just use "date-sent"). You can also specify the "last-" prefix in addition to the "reverse-" prefix, but "last-" must come after "reverse-". The "last-" prefix causes messages to be sorted against its siblings by which has the last descendant, using the rest of $sort_aux as an ordering. For instance,

```
set sort_aux=last-date-received
```

would mean that if a new message is received in a thread, that thread becomes the last one displayed (or the first, if you have "`set sort=reverse-threads`".)

Note: For reversed $sort order $sort_aux is reversed again (which is not the right thing to do, but kept to not break any existing configuration setting).

3.269. sort_browser

Type: sort order
Default: alpha

Specifies how to sort entries in the file browser. By default, the entries are sorted alphabetically. Valid values:

- alpha (alphabetically)
- date
- size
- unsorted

You may optionally use the "reverse-" prefix to specify reverse sorting order (example: "`set sort_browser=reverse-date`").

3.270. sort_re

Type: boolean
Default: yes

This variable is only useful when sorting by threads with $strict_threads *unset*. In that case, it changes the heuristic mutt uses to thread messages by subject. With $sort_re *set*, mutt will only attach a message as the child of another message by subject if the subject of the child message starts with a substring matching the setting of $reply_regexp. With $sort_re *unset*, mutt will attach the message whether or not this is the case, as long as the non-$reply_regexp parts of both messages are identical.

3.271. spam_separator

Type: string
Default: “, ”

This variable controls what happens when multiple spam headers are matched: if *unset*, each successive header will overwrite any previous matches value for the spam label. If *set*, each successive match will append to the previous, using this variable's value as a separator.

3.272. spoolfile

Type: path
Default: (empty)

If your spool mailbox is in a non-default place where Mutt cannot find it, you can specify its location with this variable. Mutt will initially set this variable to the value of the environment variable `$MAIL` or `$MAILDIR` if either is defined.

3.273. ssl_ca_certificates_file

Type: path
Default: (empty)

This variable specifies a file containing trusted CA certificates. Any server certificate that is signed with one of these CA certificates is also automatically accepted.

Example:

```
set ssl_ca_certificates_file=/etc/ssl/certs/ca-certificates.crt
```

3.274. ssl_client_cert

Type: path
Default: (empty)

The file containing a client certificate and its associated private key.

3.275. ssl_force_tls

Type: boolean
Default: no

If this variable is *set*, Mutt will require that all connections to remote servers be encrypted. Furthermore it will attempt to negotiate TLS even if the server does not advertise the capability, since it would otherwise have to abort the connection anyway. This option supersedes $ssl_starttls.

3.276. ssl_min_dh_prime_bits

Type: number
Default: 0

This variable specifies the minimum acceptable prime size (in bits) for use in any Diffie-Hellman key exchange. A value of 0 will use the default from the GNUTLS library.

3.277. ssl_starttls

Type: quadoption
Default: yes

If *set* (the default), mutt will attempt to use `STARTTLS` on servers advertising the capability. When *unset*, mutt will not attempt to use `STARTTLS` regardless of the server's capabilities.

3.278. ssl_use_sslv2

Type: boolean
Default: no

This variable specifies whether to attempt to use SSLv2 in the SSL authentication process.

3.279. ssl_use_sslv3

Type: boolean
Default: yes

This variable specifies whether to attempt to use SSLv3 in the SSL authentication process.

3.280. ssl_use_tlsv1

Type: boolean
Default: yes

This variable specifies whether to attempt to use TLSv1.0 in the SSL authentication process.

3.281. ssl_use_tlsv1_1

Type: boolean
Default: yes

This variable specifies whether to attempt to use TLSv1.1 in the SSL authentication process.

3.282. ssl_use_tlsv1_2

Type: boolean
Default: yes

This variable specifies whether to attempt to use TLSv1.2 in the SSL authentication process.

3.283. ssl_usesystemcerts

Type: boolean
Default: yes

If set to *yes*, mutt will use CA certificates in the system-wide certificate store when checking if a server certificate is signed by a trusted CA.

3.284. ssl_verify_dates

Type: boolean
Default: yes

If *set* (the default), mutt will not automatically accept a server certificate that is either not yet valid or already expired. You should only unset this for particular known hosts, using the `<account-hook>` function.

3.285. ssl_verify_host

Type: boolean
Default: yes

If *set* (the default), mutt will not automatically accept a server certificate whose host name does not match the host used in your folder URL. You should only unset this for particular known hosts, using the `<account-hook>` function.

3.286. status_chars

Type: string
Default: "`-*%A`"

Controls the characters used by the "%r" indicator in $status_format. The first character is used when the mailbox is unchanged. The second is used when the mailbox has been changed, and it needs to be resynchronized. The third is used if the mailbox is in read-only mode, or if the mailbox will not be written when exiting that mailbox (You can toggle whether to write changes to a mailbox with the `<toggle-write>` operation, bound by default to "%"). The fourth is used to indicate that the current folder has been opened in attach- message mode (Certain operations like composing a new mail, replying, forwarding, etc. are not permitted in this mode).

3.287. status_format

Type: string
Default: "`-%r-Mutt: %f [Msgs:%?M?%M/?%m%?n? New:%n?%?o? Old:%o?%?d? Del:%d?%?F? Flag:%F?%?t? Tag:%t?%?p? Post:%p?%?b? Inc:%b?%?l? %l?]---(%s/%S)-%>-(%P)---`"

Controls the format of the status line displayed in the “index” menu. This string is similar to $index_format, but has its own set of `printf(3)`-like sequences:

%b	number of mailboxes with new mail *
%d	number of deleted messages *
%f	the full pathname of the current mailbox
%F	number of flagged messages *
%h	local hostname
%l	size (in bytes) of the current mailbox *
%L	size (in bytes) of the messages shown (i.e., which match the current limit) *
%m	the number of messages in the mailbox *
%M	the number of messages shown (i.e., which match the current limit) *
%n	number of new messages in the mailbox *
%o	number of old unread messages *
%p	number of postponed messages *
%P	percentage of the way through the index
%r	modified/read-only/won't-write/attach-message indicator, according to $status_chars
%s	current sorting mode ($sort)
%S	current aux sorting method ($sort_aux)
%t	number of tagged messages *
%u	number of unread messages *
%v	Mutt version string
%V	currently active limit pattern, if any *
%>X	right justify the rest of the string and pad with “X”
%|X	pad to the end of the line with “X”
%*X	soft-fill with character “X” as pad

For an explanation of “soft-fill”, see the $index_format documentation.

* = can be optionally printed if nonzero

Some of the above sequences can be used to optionally print a string if their value is nonzero. For example, you may only want to see the number of flagged messages if such messages exist, since zero is not particularly

meaningful. To optionally print a string based upon one of the above sequences, the following construct is used:

```
%?<sequence_char>?<optional_string>?
```

where *sequence_char* is a character from the table above, and *optional_string* is the string you would like printed if *sequence_char* is nonzero. *optional_string* **may** contain other sequences as well as normal text, but you may **not** nest optional strings.

Here is an example illustrating how to optionally print the number of new messages in a mailbox:

```
%?n?%n new messages.?
```

You can also switch between two strings using the following construct:

```
%?<sequence_char>?<if_string>&<else_string>?
```

If the value of *sequence_char* is non-zero, *if_string* will be expanded, otherwise *else_string* will be expanded.

You can force the result of any `printf(3)`-like sequence to be lowercase by prefixing the sequence character with an underscore ("_") sign. For example, if you want to display the local hostname in lowercase, you would use: "`%_h`".

If you prefix the sequence character with a colon (":") character, mutt will replace any dots in the expansion by underscores. This might be helpful with IMAP folders that don't like dots in folder names.

3.288. status_on_top

Type: boolean
Default: no

Setting this variable causes the "status bar" to be displayed on the first line of the screen rather than near the bottom. If $help is *set*, too it'll be placed at the bottom.

3.289. strict_threads

Type: boolean
Default: no

If *set*, threading will only make use of the "In-Reply-To" and "References:" fields when you $sort by message threads. By default, messages with the same subject are grouped together in "pseudo threads.". This may not always be desirable, such as in a personal mailbox where you might have several unrelated messages with the subjects like "hi" which will get grouped together. See also $sort_re for a less drastic way of controlling this behavior.

3.290. suspend

Type: boolean
Default: yes

When *unset*, mutt won't stop when the user presses the terminal's *susp* key, usually "^Z". This is useful if you run mutt inside an xterm using a command like "`xterm -e mutt`".

3.291. text_flowed

Type: boolean
Default: no

When *set*, mutt will generate "format=flowed" bodies with a content type of "`text/plain; format=flowed`". This format is easier to handle for some mailing software, and generally just looks like ordinary text. To actually make use of this format's features, you'll need support in your editor.

Note that $indent_string is ignored when this option is *set*.

3.292. thorough_search

Type: boolean
Default: yes

Affects the `~b` and `~h` search operations described in section "patterns". If *set*, the headers and body/attachments of messages to be searched are decoded before searching. If *unset*, messages are searched as they appear in the folder.

Users searching attachments or for non-ASCII characters should *set* this value because decoding also includes MIME parsing/decoding and possible character set conversions. Otherwise mutt will attempt to match against the raw message received (for example quoted-printable encoded or with encoded headers) which may lead to incorrect search results.

3.293. thread_received

Type: boolean
Default: no

When *set*, mutt uses the date received rather than the date sent to thread messages by subject.

3.294. tilde

Type: boolean
Default: no

When *set*, the internal-pager will pad blank lines to the bottom of the screen with a tilde ("~").

3.295. time_inc

Type: number
Default: 0

Along with $read_inc, $write_inc, and $net_inc, this variable controls the frequency with which progress updates are displayed. It suppresses updates less than $time_inc milliseconds apart. This can improve throughput on systems with slow terminals, or when running mutt on a remote system.

Also see the "tuning" section of the manual for performance considerations.

3.296. timeout

Type: number
Default: 600

When Mutt is waiting for user input either idling in menus or in an interactive prompt, Mutt would block until input is present. Depending on the context, this would prevent certain operations from working, like checking for new mail or keeping an IMAP connection alive.

This variable controls how many seconds Mutt will at most wait until it aborts waiting for input, performs these operations and continues to wait for input.

A value of zero or less will cause Mutt to never time out.

3.297. tmpdir

Type: path
Default: (empty)

This variable allows you to specify where Mutt will place its temporary files needed for displaying and composing messages. If this variable is not set, the environment variable `$TMPDIR` is used. If `$TMPDIR` is not set then "`/tmp`" is used.

3.298. to_chars

Type: string
Default: " `+TCFL`"

Controls the character used to indicate mail addressed to you. The first character is the one used when the mail is *not* addressed to your address. The second is used when you are the only recipient of the message. The third is when your address appears in the "To:" header field, but you are not the only recipient of the message. The fourth character is used when your address is specified in the "Cc:" header field, but you are not the only recipient. The fifth character is used to indicate mail that was sent by *you*. The sixth character is used to indicate when a mail was sent to a mailing-list you subscribe to.

3.299. tunnel

Type: string
Default: (empty)

Setting this variable will cause mutt to open a pipe to a command instead of a raw socket. You may be able to use this to set up preauthenticated connections to your IMAP/POP3/SMTP server. Example:

```
set tunnel="ssh -q mailhost.net /usr/local/libexec/imapd"
```

Note: For this example to work you must be able to log in to the remote machine without having to enter a password.

When set, Mutt uses the tunnel for all remote connections. Please see "account-hook" in the manual for how to use different tunnel commands per connection.

3.300. uncollapse_jump

Type: boolean
Default: no

When *set*, Mutt will jump to the next unread message, if any, when the current thread is *un*collapsed.

3.301. use_8bitmime

Type: boolean
Default: no

Warning: do not set this variable unless you are using a version of sendmail which supports the `-B8BITMIME` flag (such as sendmail 8.8.x) or you may not be able to send mail.

When *set*, Mutt will invoke $sendmail with the `-B8BITMIME` flag when sending 8-bit messages to enable ESMTP negotiation.

3.302. use_domain

Type: boolean
Default: yes

When *set*, Mutt will qualify all local addresses (ones without the "@host" portion) with the value of $hostname. If *unset*, no addresses will be qualified.

3.303. use_envelope_from

Type: boolean
Default: no

When *set*, mutt will set the *envelope* sender of the message. If $envelope_from_address is *set*, it will be used

as the sender address. If *unset*, mutt will attempt to derive the sender from the "From:" header.

Note that this information is passed to sendmail command using the `-f` command line switch. Therefore setting this option is not useful if the $sendmail variable already contains `-f` or if the executable pointed to by $sendmail doesn't support the `-f` switch.

3.304. use_from

Type: boolean
Default: yes

When *set*, Mutt will generate the "From:" header field when sending messages. If *unset*, no "From:" header field will be generated unless the user explicitly sets one using the "my_hdr" command.

3.305. use_idn

Type: boolean
Default: yes

When *set*, Mutt will show you international domain names decoded. Note: You can use IDNs for addresses even if this is *unset*. This variable only affects decoding.

3.306. use_ipv6

Type: boolean
Default: yes

When *set*, Mutt will look for IPv6 addresses of hosts it tries to contact. If this option is *unset*, Mutt will restrict itself to IPv4 addresses. Normally, the default should work.

3.307. user_agent

Type: boolean
Default: yes

When *set*, mutt will add a "User-Agent:" header to outgoing messages, indicating which version of mutt was used for composing them.

3.308. visual

Type: path
Default: (empty)

Specifies the visual editor to invoke when the "~v" command is given in the built-in editor.

3.309. wait_key

Type: boolean
Default: yes

Controls whether Mutt will ask you to press a key after an external command has been invoked by these functions: `<shell-escape>`, `<pipe-message>`, `<pipe-entry>`, `<print-message>`, and `<print-entry>` commands.

It is also used when viewing attachments with "auto_view", provided that the corresponding mailcap entry has a *needsterminal* flag, and the external program is interactive.

When *set*, Mutt will always ask for a key. When *unset*, Mutt will wait for a key only if the external command returned a non-zero status.

3.310. weed

Type: boolean
Default: yes

When *set*, mutt will weed headers when displaying, forwarding, printing, or replying to messages.

3.311. wrap

Type: number
Default: 0

When set to a positive value, mutt will wrap text at $wrap characters. When set to a negative value, mutt will wrap text so that there are $wrap characters of empty space on the right side of the terminal. Setting it to zero makes mutt wrap at the terminal width.

Also see $reflow_wrap.

3.312. wrap_headers

Type: number
Default: 78

This option specifies the number of characters to use for wrapping an outgoing message's headers. Allowed values are between 78 and 998 inclusive.

Note: This option usually shouldn't be changed. RFC5233 recommends a line length of 78 (the default), so **please only change this setting when you know what you're doing**.

3.313. wrap_search

Type: boolean
Default: yes

Controls whether searches wrap around the end.

When *set*, searches will wrap around the first (or last) item. When *unset*, incremental searches will not wrap.

3.314. wrapmargin

Type: number
Default: 0

(DEPRECATED) Equivalent to setting $wrap with a negative value.

3.315. write_bcc

Type: boolean
Default: yes

Controls whether mutt writes out the "Bcc:" header when preparing messages to be sent. Exim users may wish to unset this. If mutt is set to deliver directly via SMTP (see $smtp_url), this option does nothing: mutt will never write out the "Bcc:" header in this case.

3.316. write_inc

Type: number
Default: 10

When writing a mailbox, a message will be printed every $write_inc messages to indicate progress. If set to 0, only a single message will be displayed before writing a mailbox.

Also see the $read_inc, $net_inc and $time_inc variables and the "tuning" section of the manual for performance considerations.

4. Functions

The following is the list of available functions listed by the mapping in which they are available. The default key setting is given, and an explanation of what the function does. The key bindings of these functions can be changed with the bind command.

4.1. Generic Menu

The *generic* menu is not a real menu, but specifies common functions (such as movement) available in all menus except for *pager* and *editor*. Changing settings for this menu will affect the default bindings for all menus (except as noted).

Table 9.2. Default Generic Menu Bindings

Function	Default key	Description

<top-page>	H	move to the top of the page
<next-entry>	j	move to the next entry
<previous-entry>	k	move to the previous entry
<bottom-page>	L	move to the bottom of the page
<refresh>	^L	clear and redraw the screen
<middle-page>	M	move to the middle of the page
<search-next>	n	search for next match
<exit>	q	exit this menu
<tag-entry>	t	tag the current entry
<next-page>	z	move to the next page
<previous-page>	Z	move to the previous page
<last-entry>	*	move to the last entry
<first-entry>	=	move to the first entry
<enter-command>	:	enter a muttrc command
<next-line>	>	scroll down one line
<previous-line>	<	scroll up one line
<half-up>	[	scroll up 1/2 page
<half-down>	]	scroll down 1/2 page
<help>	?	this screen
<tag-prefix>	;	apply next function to tagged messages
<tag-prefix-cond>		apply next function ONLY to tagged messages
<end-cond>		end of conditional execution (noop)
<shell-escape>	!	invoke a command in a subshell
<select-entry>	<Return>	select the current entry
<search>	/	search for a regular expression
<search-reverse>	Esc /	search backwards for a regular expression
<search-opposite>		search for next match in opposite direction
<jump>		jump to an index number

<current-top>		move entry to top of screen
<current-middle>		move entry to middle of screen
<current-bottom>		move entry to bottom of screen
<what-key>		display the keycode for a key press

4.2. Index Menu

Table 9.3. Default Index Menu Bindings

Function	Default key	Description
<create-alias>	a	create an alias from a message sender
<bounce-message>	b	remail a message to another user
<break-thread>	#	break the thread in two
<change-folder>	c	open a different folder
<change-folder-readonly>	Esc c	open a different folder in read only mode
<next-unread-mailbox>		open next mailbox with new mail
<collapse-thread>	Esc v	collapse/uncollapse current thread
<collapse-all>	Esc V	collapse/uncollapse all threads
<copy-message>	C	copy a message to a file/mailbox
<decode-copy>	Esc C	make decoded (text/plain) copy
<decode-save>	Esc s	make decoded copy (text/plain) and delete
<delete-message>	d	delete the current entry
<delete-pattern>	D	delete messages matching a pattern
<delete-thread>	^D	delete all messages in thread
<delete-subthread>	Esc d	delete all messages in subthread
<edit>	e	edit the raw message
<edit-type>	^E	edit attachment content type
<forward-message>	f	forward a message with comments
<flag-message>	F	toggle a message's 'important' flag
<group-reply>	g	reply to all recipients

<fetch-mail>	G	retrieve mail from POP server
<imap-fetch-mail>		force retrieval of mail from IMAP server
<imap-logout-all>		logout from all IMAP servers
<display-toggle-weed>	h	display message and toggle header weeding
<next-undeleted>	j	move to the next undeleted message
<previous-undeleted>	k	move to the previous undeleted message
<limit>	l	show only messages matching a pattern
<link-threads>	&	link tagged message to the current one
<list-reply>	L	reply to specified mailing list
<mail>	m	compose a new mail message
<toggle-new>	N	toggle a message's 'new' flag
<toggle-write>	%	toggle whether the mailbox will be rewritten
<next-thread>	^N	jump to the next thread
<next-subthread>	Esc n	jump to the next subthread
<query>	Q	query external program for addresses
<quit>	q	save changes to mailbox and quit
<reply>	r	reply to a message
<show-limit>	Esc l	show currently active limit pattern
<sort-mailbox>	o	sort messages
<sort-reverse>	O	sort messages in reverse order
<print-message>	p	print the current entry
<previous-thread>	^P	jump to previous thread
<previous-subthread>	Esc p	jump to previous subthread
<recall-message>	R	recall a postponed message
<read-thread>	^R	mark the current thread as read
<read-subthread>	Esc r	mark the current subthread as read
<resend-message>	Esc e	use the current message as a template for a new one
<save-message>	s	save message/attachment to a mailbox/file
<tag-pattern>	T	tag messages matching a pattern

<tag-subthread>		tag the current subthread
<tag-thread>	Esc t	tag the current thread
<untag-pattern>	^T	untag messages matching a pattern
<undelete-message>	u	undelete the current entry
<undelete-pattern>	U	undelete messages matching a pattern
<undelete-subthread>	Esc u	undelete all messages in subthread
<undelete-thread>	^U	undelete all messages in thread
<view-attachments>	v	show MIME attachments
<show-version>	V	show the Mutt version number and date
<set-flag>	w	set a status flag on a message
<clear-flag>	W	clear a status flag from a message
<display-message>	<Return>	display a message
<buffy-list>	.	list mailboxes with new mail
<sync-mailbox>	$	save changes to mailbox
<display-address>	@	display full address of sender
<pipe-message>	|	pipe message/attachment to a shell command
<next-new>		jump to the next new message
<next-new-then-unread>	<Tab>	jump to the next new or unread message
<previous-new>		jump to the previous new message
<previous-new-then-unread>	Esc <Tab>	jump to the previous new or unread message
<next-unread>		jump to the next unread message
<previous-unread>		jump to the previous unread message
<parent-message>	P	jump to parent message in thread
<extract-keys>	^K	extract supported public keys
<forget-passphrase>	^F	wipe passphrase(s) from memory
<check-traditional-pgp>	Esc P	check for classic PGP
<mail-key>	Esc k	mail a PGP public key
<decrypt-copy>		make decrypted copy
<decrypt-save>		make decrypted copy and delete

4.3. Pager Menu

Table 9.4. Default Pager Menu Bindings

Function	Default key	Description
`<break-thread>`	#	break the thread in two
`<create-alias>`	a	create an alias from a message sender
`<bounce-message>`	b	remail a message to another user
`<change-folder>`	c	open a different folder
`<change-folder-readonly>`	Esc c	open a different folder in read only mode
`<next-unread-mailbox>`		open next mailbox with new mail
`<copy-message>`	C	copy a message to a file/mailbox
`<decode-copy>`	Esc C	make decoded (text/plain) copy
`<delete-message>`	d	delete the current entry
`<delete-thread>`	^D	delete all messages in thread
`<delete-subthread>`	Esc d	delete all messages in subthread
`<set-flag>`	w	set a status flag on a message
`<clear-flag>`	W	clear a status flag from a message
`<edit>`	e	edit the raw message
`<edit-type>`	^E	edit attachment content type
`<forward-message>`	f	forward a message with comments
`<flag-message>`	F	toggle a message's 'important' flag
`<group-reply>`	g	reply to all recipients
`<imap-fetch-mail>`		force retrieval of mail from IMAP server
`<imap-logout-all>`		logout from all IMAP servers
`<display-toggle-weed>`	h	display message and toggle header weeding
`<next-undeleted>`	j	move to the next undeleted message
`<next-entry>`	J	move to the next entry
`<previous-undeleted>`	k	move to the previous undeleted message

<previous-entry>	K	move to the previous entry
<link-threads>	&	link tagged message to the current one
<list-reply>	L	reply to specified mailing list
<redraw-screen>	^L	clear and redraw the screen
<mail>	m	compose a new mail message
<mark-as-new>	N	toggle a message's 'new' flag
<search-next>	n	search for next match
<next-thread>	^N	jump to the next thread
<next-subthread>	Esc n	jump to the next subthread
<sort-mailbox>	o	sort messages
<sort-reverse>	O	sort messages in reverse order
<print-message>	p	print the current entry
<previous-thread>	^P	jump to previous thread
<previous-subthread>	Esc p	jump to previous subthread
<quit>	Q	save changes to mailbox and quit
<exit>	q	exit this menu
<reply>	r	reply to a message
<recall-message>	R	recall a postponed message
<read-thread>	^R	mark the current thread as read
<read-subthread>	Esc r	mark the current subthread as read
<resend-message>	Esc e	use the current message as a template for a new one
<save-message>	s	save message/attachment to a mailbox/file
<skip-quoted>	S	skip beyond quoted text
<decode-save>	Esc s	make decoded copy (text/plain) and delete
<tag-message>	t	tag the current entry
<toggle-quoted>	T	toggle display of quoted text
<undelete-message>	u	undelete the current entry
<undelete-subthread>	Esc u	undelete all messages in subthread
<undelete-thread>	^U	undelete all messages in thread

<view-attachments>	v	show MIME attachments
<show-version>	V	show the Mutt version number and date
<search-toggle>	\\	toggle search pattern coloring
<display-address>	@	display full address of sender
<next-new>		jump to the next new message
<pipe-message>	|	pipe message/attachment to a shell command
<help>	?	this screen
<next-page>	<Space>	move to the next page
<previous-page>	-	move to the previous page
<top>	^	jump to the top of the message
<sync-mailbox>	$	save changes to mailbox
<shell-escape>	!	invoke a command in a subshell
<enter-command>	:	enter a muttrc command
<buffy-list>	.	list mailboxes with new mail
<search>	/	search for a regular expression
<search-reverse>	Esc /	search backwards for a regular expression
<search-opposite>		search for next match in opposite direction
<next-line>	<Return>	scroll down one line
<jump>		jump to an index number
<next-unread>		jump to the next unread message
<previous-new>		jump to the previous new message
<previous-unread>		jump to the previous unread message
<half-up>		scroll up 1/2 page
<half-down>		scroll down 1/2 page
<previous-line>		scroll up one line
<bottom>		jump to the bottom of the message
<parent-message>	P	jump to parent message in thread
<check-traditional-pgp>	Esc P	check for classic PGP
<mail-key>	Esc k	mail a PGP public key

<extract-keys>	^K	extract supported public keys
<forget-passphrase>	^F	wipe passphrase(s) from memory
<decrypt-copy>		make decrypted copy
<decrypt-save>		make decrypted copy and delete
<what-key>		display the keycode for a key press

4.4. Alias Menu

Table 9.5. Default Alias Menu Bindings

Function	Default key	Description
<delete-entry>	d	delete the current entry
<undelete-entry>	u	undelete the current entry

4.5. Query Menu

Table 9.6. Default Query Menu Bindings

Function	Default key	Description
<create-alias>	a	create an alias from a message sender
<mail>	m	compose a new mail message
<query>	Q	query external program for addresses
<query-append>	A	append new query results to current results

4.6. Attachment Menu

Table 9.7. Default Attachment Menu Bindings

Function	Default key	Description
<bounce-message>	b	remail a message to another user
<display-toggle-weed>	h	display message and toggle header weeding
<edit-type>	^E	edit attachment content type
<print-entry>	p	print the current entry

<save-entry>	s	save message/attachment to a mailbox/file
<pipe-entry>	|	pipe message/attachment to a shell command
<view-mailcap>	m	force viewing of attachment using mailcap
<reply>	r	reply to a message
<resend-message>	Esc e	use the current message as a template for a new one
<group-reply>	g	reply to all recipients
<list-reply>	L	reply to specified mailing list
<forward-message>	f	forward a message with comments
<view-text>	T	view attachment as text
<view-attach>	<Return>	view attachment using mailcap entry if necessary
<delete-entry>	d	delete the current entry
<undelete-entry>	u	undelete the current entry
<collapse-parts>	v	Toggle display of subparts
<check-traditional-pgp>	Esc P	check for classic PGP
<extract-keys>	^K	extract supported public keys
<forget-passphrase>	^F	wipe passphrase(s) from memory

4.7. Compose Menu

Table 9.8. Default Compose Menu Bindings

Function	Default key	Description
<attach-file>	a	attach file(s) to this message
<attach-message>	A	attach message(s) to this message
<edit-bcc>	b	edit the BCC list
<edit-cc>	c	edit the CC list
<copy-file>	C	save message/attachment to a mailbox/file
<detach-file>	D	delete the current entry
<toggle-disposition>	^D	toggle disposition between inline/attachment
<edit-description>	d	edit attachment description

<edit-message>	e	edit the message
<edit-headers>	E	edit the message with headers
<edit-file>	^X e	edit the file to be attached
<edit-encoding>	^E	edit attachment transfer-encoding
<edit-from>	Esc f	edit the from field
<edit-fcc>	f	enter a file to save a copy of this message in
<filter-entry>	F	filter attachment through a shell command
<get-attachment>	G	get a temporary copy of an attachment
<display-toggle-weed>	h	display message and toggle header weeding
<ispell>	i	run ispell on the message
<print-entry>	l	print the current entry
<edit-mime>	m	edit attachment using mailcap entry
<new-mime>	n	compose new attachment using mailcap entry
<postpone-message>	P	save this message to send later
<edit-reply-to>	r	edit the Reply-To field
<rename-file>	R	rename/move an attached file
<edit-subject>	s	edit the subject of this message
<edit-to>	t	edit the TO list
<edit-type>	^T	edit attachment content type
<write-fcc>	w	write the message to a folder
<toggle-unlink>	u	toggle whether to delete file after sending it
<toggle-recode>		toggle recoding of this attachment
<update-encoding>	U	update an attachment's encoding info
<view-attach>	<Return>	view attachment using mailcap entry if necessary
<send-message>	y	send the message
<pipe-entry>	|	pipe message/attachment to a shell command
<attach-key>	Esc k	attach a PGP public key
<pgp-menu>	p	show PGP options
<forget-passphrase>		

	^F	wipe passphrase(s) from memory
<smime-menu>	S	show S/MIME options
<mix>	M	send the message through a mixmaster remailer chain

4.8. Postpone Menu

Table 9.9. Default Postpone Menu Bindings

Function	Default key	Description
<delete-entry>	d	delete the current entry
<undelete-entry>	u	undelete the current entry

4.9. Browser Menu

Table 9.10. Default Browser Menu Bindings

Function	Default key	Description
<change-dir>	c	change directories
<display-filename>	@	display the currently selected file's name
<enter-mask>	m	enter a file mask
<sort>	o	sort messages
<sort-reverse>	O	sort messages in reverse order
<select-new>	N	select a new file in this directory
<check-new>		check mailboxes for new mail
<toggle-mailboxes>	<Tab>	toggle whether to browse mailboxes or all files
<view-file>	<Space>	view file
<buffy-list>	.	list mailboxes with new mail
<create-mailbox>	C	create a new mailbox (IMAP only)
<delete-mailbox>	d	delete the current mailbox (IMAP only)
<rename-mailbox>	r	rename the current mailbox (IMAP only)
<subscribe>	s	subscribe to current mailbox (IMAP only)
<unsubscribe>	u	unsubscribe from current mailbox (IMAP only)

<toggle-subscribed>	T	toggle view all/subscribed mailboxes (IMAP only)

4.10. Pgp Menu

Table 9.11. Default Pgp Menu Bindings

Function	Default key	Description
<verify-key>	c	verify a PGP public key
<view-name>	%	view the key's user id

4.11. Smime Menu

Table 9.12. Default Smime Menu Bindings

Function	Default key	Description
<verify-key>	c	verify a PGP public key
<view-name>	%	view the key's user id

4.12. Mixmaster Menu

Table 9.13. Default Mixmaster Menu Bindings

Function	Default key	Description
<accept>	<Return>	Accept the chain constructed
<append>	a	Append a remailer to the chain
<insert>	i	Insert a remailer into the chain
<delete>	d	Delete a remailer from the chain
<chain-prev>	<Left>	Select the previous element of the chain
<chain-next>	<Right>	Select the next element of the chain

4.13. Editor Menu

Table 9.14. Default Editor Menu Bindings

Function	Default key	Description
<bol>	^A	jump to the beginning of the line
<backward-char>	^B	move the cursor one character to the left
<backward-word>	Esc b	move the cursor to the beginning of the word
<capitalize-word>	Esc c	capitalize the word
<downcase-word>	Esc l	convert the word to lower case
<upcase-word>	Esc u	convert the word to upper case
<delete-char>	^D	delete the char under the cursor
<eol>	^E	jump to the end of the line
<forward-char>	^F	move the cursor one character to the right
<forward-word>	Esc f	move the cursor to the end of the word
<backspace>	<Backspace>	delete the char in front of the cursor
<kill-eol>	^K	delete chars from cursor to end of line
<kill-eow>	Esc d	delete chars from the cursor to the end of the word
<kill-line>	^U	delete all chars on the line
<quote-char>	^V	quote the next typed key
<kill-word>	^W	delete the word in front of the cursor
<complete>	<Tab>	complete filename or alias
<complete-query>	^T	complete address with query
<buffy-cycle>	<Space>	cycle among incoming mailboxes
<history-up>		scroll up through the history list
<history-down>		scroll down through the history list
<transpose-chars>		transpose character under cursor with previous

Chapter 10. Miscellany

Table of Contents

1. Acknowledgements

Kari Hurtta <kari.hurtta@fmi.fi> co-developed the original MIME parsing code back in the ELM-ME days.

The following people have been very helpful to the development of Mutt:

- Vikas Agnihotri <vikasa@writeme.com>
- Francois Berjon <Francois.Berjon@aar.alcatel-alsthom.fr>
- Aric Blumer <aric@fore.com>
- John Capo <jc@irbs.com>
- David Champion <dgc@uchicago.edu>
- Brendan Cully <brendan@kublai.com>
- Liviu Daia <daia@stoilow.imar.ro>
- Thomas E. Dickey <dickey@herndon4.his.com>
- David DeSimone <fox@convex.hp.com>
- Nickolay N. Dudorov <nnd@wint.itfs.nsk.su>
- Ruslan Ermilov <ru@freebsd.org>
- Edmund Grimley Evans <edmundo@rano.org>
- Michael Finken <finken@conware.de>
- Sven Guckes <guckes@math.fu-berlin.de>
- Lars Hecking <lhecking@nmrc.ie>
- Mark Holloman <holloman@nando.net>
- Andreas Holzmann <holzmann@fmi.uni-passau.de>
- Marco d'Itri <md@linux.it>
- Björn Jacke <bjacke@suse.com>
- Byrial Jensen <byrial@image.dk>
- David Jeske <jeske@igcom.net>

- Christophe Kalt <kalt@hugo.int-evry.fr>
- Tommi Komulainen <Tommi.Komulainen@iki.fi>
- Felix von Leitner (a.k.a “Fefe”) <leitner@math.fu-berlin.de>
- Brandon Long <blong@fiction.net>
- Jimmy Mäkelä <jmy@flashback.net>
- Lars Marowsky-Bree <lmb@pointer.in-minden.de>
- Thomas “Mike” Michlmayr <mike@cosy.sbg.ac.at>
- Andrew W. Nosenko <awn@bcs.zp.ua>
- David O'Brien <obrien@Nuxi.cs.ucdavis.edu>
- Clint Olsen <olsenc@ichips.intel.com>
- Park Myeong Seok <pms@romance.kaist.ac.kr>
- Thomas Parmelan <tom@ankh.fr.eu.org>
- Ollivier Robert <roberto@keltia.freenix.fr>
- Thomas Roessler <roessler@does-not-exist.org>
- Roland Rosenfeld <roland@spinnaker.de>
- Rocco Rutte <pdmef@gmx.net>
- TAKIZAWA Takashi <taki@luna.email.ne.jp>
- Allain Thivillon <Allain.Thivillon@alma.fr>
- Gero Treuner <gero@70t.de>
- Vsevolod Volkov <vvv@lucky.net>
- Ken Weinert <kenw@ihs.com>

2. About This Document

This document was written in DocBook, and then rendered using the Gnome XSLT toolkit.