The Mutt E-Mail Client

Michael Elkins

The Mutt E-Mail Client
by Michael Elkins

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

Table of Contents

1. Introduction 1

Mutt Home Page 1
Mailing Lists 1
Getting Mutt 1
Mutt Online Resources 1
Contributing to Mutt 2
Typographical Conventions 2
Copyright 2

2. Getting Started 4

Core Concepts 4
Screens and Menus 4
Index 5
Pager 5
File Browser 5
Sidebar 5
Help 5
Compose Menu 6
Alias Menu 6
Attachment Menu 6
List Menu 6
Moving Around in Menus 6
Editing Input Fields 7
Introduction 7
Buffy Cycle 8
History 9
Reading Mail 9
The Message Index 9
The Pager 11
Threaded Mode 13
Miscellaneous Functions 14
Sending Mail 16
Introduction 16
Editing the Message Header 19
Fcc: Pseudo Header 19
Attach: Pseudo Header 19
Pgp: Pseudo Header 19
In-Reply-To: Header 19
Sending Cryptographically Signed/Encrypted Messages 20
Sending Format=Flowed Messages 20
Concept 21
Mutt Support 21
Editor Considerations 21
Reformatting 21
Background Editing 22
Forwarding and Bouncing Mail 22

Postponing Mail 23
Encryption and Signing 23
OpenPGP Configuration 24
S/MIME Configuration 24
3. Configuration 25
Location of Initialization Files 25
Starter Muttrc 25
Syntax of Initialization Files 25
Address Groups 27
Defining/Using Aliases 28
Changing the Default Key Bindings 29
Terminal Keybindings 31
Enter versus Return 32
Changing the current working directory 32
Defining Aliases for Character Sets 32
Setting Variables Based Upon Mailbox 33
Keyboard Macros 33
Using Color and Mono Video Attributes 34
Message Header Display 36
Header Display 37
Selecting Headers 37
Ordering Displayed Headers 37
Alternative Addresses 38
Mailing Lists 39
Using Multiple Spool Mailboxes 40
Monitoring Incoming Mail 40
User-Defined Headers 41
Specify Default Save Mailbox 42
Specify Default Fcc: Mailbox When Composing 42
Specify Default Save Filename and Default Fcc: Mailbox at Once 43
Change Settings Based Upon Message Recipients 43
Change Settings Before Formatting a Message 44
Choosing the Cryptographic Key of the Recipient 44
Dynamically Changing $index_format using Patterns 45
Adding Key Sequences to the Keyboard Buffer 45
Executing Functions 46
Message Scoring 46
Spam Detection 47
Setting and Querying Variables 48
Variable Types 49
Commands 49
User-Defined Variables 50
Introduction 50
Examples 51
Type Conversions 51
Reading Initialization Commands From Another File 52
Removing Hooks 53

Format Strings 53
Basic usage 53
Conditionals 54
Filters 54
Padding 54
Bytes size display 55
Control allowed header fields in a mailto: URL 55
4. Advanced Usage 57
Character Set Handling 57
Regular Expressions 57
Patterns: Searching, Limiting and Tagging 60
Pattern Modifier 60
Simple Searches 63
Nesting and Boolean Operators 64
Searching by Date 65
Absolute Dates 65
Relative Dates 66
Marking Messages 66
Using Tags 67
Using Hooks 67
Message Matching in Hooks 68
Mailbox Matching in Hooks 68
Managing the Environment 69
External Address Queries 69
Mailbox Formats 70
Mailbox Shortcuts 71
Handling Mailing Lists 71
Display Munging 73
New Mail Detection 74
How New Mail Detection Works 74
Polling For New Mail 74
Monitoring New Mail 75
Calculating Mailbox Message Counts 75
Editing Threads 75
Linking Threads 76
Breaking Threads 76
Delivery Status Notification (DSN) Support 76
Start a WWW Browser on URLs 76
Echoing Text 77
Message Composition Flow 77
Batch Composition Flow 78
Using MuttLisp (EXPERIMENTAL) 78
Running a command generated by MuttLisp 79
Interpolating MuttLisp in a Command Argument 79
MuttLisp Syntax 79
MuttLisp Functions 80
concat 80

quote 80
equal 80
not 81
and 81
or 81
if 82
Examples 82
Miscellany 83
5. Mutt's MIME Support 85
Using MIME in Mutt 85
MIME Overview 85
Viewing MIME Messages in the Pager 85
The Attachment Menu 86
Viewing Attachments 86
The Compose Menu 87
MIME Type Configuration with `mime.types` 87
MIME Viewer Configuration with Mailcap 88
The Basics of the Mailcap File 89
Secure Use of Mailcap 90
Advanced Mailcap Usage 90
Optional Fields 90
Search Order 92
Command Expansion 93
Example Mailcap Files 93
MIME Autoview 94
MIME Multipart/Alternative 95
Attachment Searching and Counting 96
MIME Lookup 98
6. Optional Features 99
General Notes 99
Enabling/Disabling Features 99
URL Syntax 99
SSL/TLS Support 100
STARTTLS 100
Tunnel 100
POP3 Support 100
IMAP Support 101
The IMAP Folder Browser 101
Authentication 102
SMTP Support 102
OAUTHBEARER Support 103
XOAUTH2 Support 103
Managing Multiple Accounts 103
Local Caching 104
Header Caching 104
Body Caching 105
Cache Directories 105

Maintenance 105
Exact Address Generation 105
Sending Anonymous Messages via Mixmaster 106
Sidebar 106
Introduction 106
Variables 106
Functions 107
Commands 107
Colors 108
Sort 108
See Also 108
Compressed Folders Feature 109
Introduction 109
Commands 109
Read from compressed mailbox 110
Write to a compressed mailbox 110
Append to a compressed mailbox 111
Empty Files 111
Security 111
Autocrypt 112
Requirements 112
First Run 112
Compose Menu 113
Account Management 114
Alternative Key and Keyring Strategies 114
7. Security Considerations 116
Passwords 116
Temporary Files 116
Information Leaks 116
`mailto:`-style Links 116
External Applications 117
8. Performance Tuning 118
Reading and Writing Mailboxes 118
Reading Messages from Remote Folders 118
Searching and Limiting 118
9. Reference 120
Command-Line Options 120
Configuration Commands 122
Configuration Variables 126
abort_noattach 126
abort_noattach_regexp 126
abort_nosubject 127
abort_unmodified 127
alias_file 127
alias_format 127
allow_8bit 128
allow_ansi 128

arrow_cursor 128
ascii_chars 128
askbcc 129
askcc 129
assumed_charset 129
attach_charset 129
attach_format 130
attach_save_charset_convert 130
attach_save_dir 131
attach_sep 131
attach_split 131
attribution 131
attribution_locale 131
auto_subscribe 132
auto_tag 132
autocrypt 132
autocrypt_acct_format 132
autocrypt_dir 133
autocrypt_reply 133
autoedit 133
background_edit 133
background_confirm_quit 134
background_format 134
beep 134
beep_new 134
bounce 135
bounce_delivered 135
braille_friendly 135
browser_abbreviate_mailboxes 135
browser_sticky_cursor 136
certificate_file 136
change_folder_next 136
charset 136
check_mbox_size 137
check_new 137
collapse_unread 137
compose_confirm_detach_first 137
compose_format 138
config_charset 138
confirmappend 138
confirmcreate 138
connect_timeout 139
content_type 139
copy 139
copy_decode_weed 139
count_alternatives 139
cursor_overlay 140
crypt_autoencrypt 140

crypt_autopgp 140
crypt_autosign 140
crypt_autosmime 141
crypt_confirmhook 141
crypt_opportunistic_encrypt 141
crypt_opportunistic_encrypt_strong_keys 141
crypt_protected_headers_read 142
crypt_protected_headers_save 142
crypt_protected_headers_subject 142
crypt_protected_headers_write 143
crypt_replyencrypt 143
crypt_replysign 143
crypt_replysignencrypted 143
crypt_timestamp 143
crypt_use_gpgme 144
crypt_use_pka 144
crypt_verify_sig 144
date_format 144
default_hook 145
delete 145
delete_untag 145
digest_collapse 145
display_filter 145
dotlock_program 146
dsn_notify 146
dsn_return 146
duplicate_threads 147
edit_headers 147
editor 147
encode_from 147
entropy_file 148
envelope_from_address 148
error_history 148
escape 148
fast_reply 148
fcc_attach 149
fcc_before_send 149
fcc_clear 149
fcc_delimiter 149
flag_safe 150
folder 150
folder_format 150
followup_to 151
force_name 151
forward_attachments 151
forward_attribution_intro 151
forward_attribution_trailer 152
forward_decode 152

forward_decrypt 152
forward_edit 152
forward_format 153
forward_quote 153
from 153
gecos_mask 153
hdrs 153
header 154
header_cache 154
header_cache_compress 154
header_cache_pagesize 154
header_color_partial 154
help 155
hidden_host 155
hide_limited 155
hide_missing 155
hide_thread_subject 155
hide_top_limited 156
hide_top_missing 156
history 156
history_file 156
history_remove_dups 156
honor_disposition 157
honor_followup_to 157
hostname 157
idn_decode 157
idn_encode 158
ignore_linear_white_space 158
ignore_list_reply_to 158
imap_authenticators 158
imap_check_subscribed 159
imap_condstore 159
imap_deflate 159
imap_delim_chars 159
imap_fetch_chunk_size 160
imap_headers 160
imap_idle 160
imap_keepalive 160
imap_list_subscribed 161
imap_login 161
imap_oauth_refresh_command 161
imap_pass 161
imap_passive 161
imap_peek 162
imap_pipeline_depth 162
imap_poll_timeout 162
imap_qresync 162
imap_reconnect_sleep 162

imap_reconnect_tries 163
imap_servernoise 163
imap_user 163
implicit_autoview 163
include 163
include_encrypted 164
include_onlyfirst 164
indent_string 164
index_format 164
ispell 167
keep_flagged 167
local_date_header 167
mail_check 167
mail_check_recent 168
mail_check_stats 168
mail_check_stats_interval 168
mailcap_path 168
mailcap_sanitize 169
maildir_header_cache_verify 169
maildir_trash 169
maildir_check_cur 169
mark_macro_prefix 169
mark_old 170
markers 170
mask 170
mbox 170
mbox_type 170
menu_context 171
menu_move_off 171
menu_scroll 171
message_cache_clean 171
message_cachedir 171
message_format 172
message_id_format 172
meta_key 173
metoo 173
mh_purge 173
mh_seq_flagged 173
mh_seq_replied 173
mh_seq_unseen 174
mime_forward 174
mime_forward_decode 174
mime_forward_rest 174
mime_type_query_command 174
mime_type_query_first 175
mix_entry_format 175
mixmaster 175
move 176

muttlisp_inline_eval 176
narrow_tree 176
net_inc 176
new_mail_command 176
pager 176
pager_context 177
pager_format 177
pager_index_lines 177
pager_skip_quoted_context 178
pager_stop 178
pattern_format 178
pgp_auto_decode 178
pgp_autoinline 179
pgp_check_exit 179
pgp_check_gpg_decrypt_status_fd 179
pgp_clearsign_command 179
pgp_decode_command 180
pgp_decrypt_command 180
pgp_decryption_okay 180
pgp_default_key 181
pgp_encrypt_only_command 181
pgp_encrypt_sign_command 181
pgp_entry_format 181
pgp_export_command 182
pgp_getkeys_command 182
pgp_good_sign 182
pgp_ignore_subkeys 182
pgp_import_command 183
pgp_list_pubring_command 183
pgp_list_secring_command 183
pgp_long_ids 184
pgp_mime_auto 184
pgp_replyinline 184
pgp_retainable_sigs 184
pgp_self_encrypt 185
pgp_show_unusable 185
pgp_sign_as 185
pgp_sign_command 185
pgp_sort_keys 185
pgp_strict_enc 186
pgp_timeout 186
pgp_use_gpg_agent 186
pgp_verify_command 186
pgp_verify_key_command 187
pipe_decode 187
pipe_decode_weed 187
pipe_sep 187
pipe_split 187

pop_auth_try_all 188
pop_authenticators 188
pop_checkinterval 188
pop_delete 188
pop_host 189
pop_last 189
pop_oauth_refresh_command 189
pop_pass 189
pop_reconnect 189
pop_user 190
post_indent_string 190
postpone 190
postponed 190
postpone_encrypt 190
postpone_encrypt_as 191
preconnect 191
print 191
print_command 191
print_decode 192
print_decode_weed 192
print_split 192
prompt_after 192
query_command 192
query_format 193
quit 193
quote_regexp 193
read_inc 194
read_only 194
realname 194
recall 194
record 195
reflow_space_quotes 195
reflow_text 195
reflow_wrap 195
reply_regexp 196
reply_self 196
reply_to 196
resolve 197
resume_draft_files 197
resume_edited_draft_files 197
reverse_alias 197
reverse_name 198
reverse_realname 198
rfc2047_parameters 198
save_address 199
save_empty 199
save_history 199
save_name 199

send_group_reply_to 199
score 200
score_threshold_delete 200
score_threshold_flag 200
score_threshold_read 200
search_context 200
send_charset 201
send_multipart_alternative 201
send_multipart_alternative_filter 201
sendmail 201
sendmail_wait 202
shell 202
sidebar_delim_chars 202
sidebar_divider_char 203
sidebar_folder_indent 203
sidebar_format 203
sidebar_indent_string 204
sidebar_new_mail_only 204
sidebar_next_new_wrap 204
sidebar_relative_shortpath_indent 204
sidebar_short_path 205
sidebar_sort_method 205
sidebar_use_mailbox_shortcuts 206
sidebar_visible 206
sidebar_width 206
sig_dashes 206
sig_on_top 207
signature 207
simple_search 207
size_show_bytes 207
size_show_fractions 207
size_show_mb 208
size_units_on_left 208
sleep_time 208
smart_wrap 208
smileys 208
pgp_mime_signature_filename 209
pgp_mime_signature_description 209
smime_ask_cert_label 209
smime_ca_location 209
smime_certificates 209
smime_decrypt_command 210
smime_decrypt_use_default_key 210
smime_default_key 210
smime_encrypt_command 211
smime_encrypt_with 211
smime_get_cert_command 211
smime_get_cert_email_command 211

smime_get_signer_cert_command 212
smime_import_cert_command 212
smime_is_default 212
smime_keys 212
smime_pkcs7_default_smime_type 213
smime_pk7out_command 213
smime_self_encrypt 213
smime_sign_as 213
smime_sign_command 214
smime_sign_digest_alg 214
smime_sign_opaque_command 214
smime_timeout 214
smime_verify_command 214
smime_verify_opaque_command 215
smtp_authenticators 215
smtp_oauth_refresh_command 215
smtp_pass 215
smtp_url 216
socket_receive_timeout 216
socket_send_timeout 216
sort 216
sort_alias 217
sort_aux 217
sort_browser 217
sort_browser_mailboxes 218
sort_re 218
sort_thread_groups 219
spam_separator 219
spoolfile 219
ssl_ca_certificates_file 219
ssl_client_cert 220
ssl_force_tls 220
ssl_min_dh_prime_bits 220
ssl_starttls 220
ssl_use_sslv2 220
ssl_use_sslv3 221
ssl_use_tlsv1 221
ssl_use_tlsv1_1 221
ssl_use_tlsv1_2 221
ssl_use_tlsv1_3 221
ssl_usesystemcerts 222
ssl_verify_dates 222
ssl_verify_host 222
ssl_verify_host_override 222
ssl_verify_partial_chains 222
ssl_ciphers 223
status_chars 223
status_format 223

status_on_top 225
strict_threads 225
suspend 225
text_flowed 225
thorough_search 226
thread_received 226
tilde 226
time_inc 226
timeout 227
tmpdir 227
tmpdraftdir 227
to_chars 227
trash 228
ts_icon_format 228
ts_enabled 228
ts_status_format 228
tunnel 228
tunnel_is_secure 229
uncollapse_jump 229
uncollapse_new 229
use_8bitmime 229
use_domain 230
use_envelope_from 230
use_from 230
use_ipv6 230
user_agent 230
visual 231
wait_key 231
weed 231
wrap 231
wrap_headers 232
wrap_search 232
wrapmargin 232
write_bcc 232
write_inc 233
Functions 233
Generic Menu 233
Index Menu 235
Pager Menu 239
Alias Menu 243
Query Menu 244
Attachment Menu 244
Compose Menu 245
Postpone Menu 247
Browser Menu 248
Pgp Menu 248
Smime Menu 249
Mixmaster Menu 249

Editor Menu 249
Autocrypt Account Menu 251
List Menu 251
10. Miscellany 252
Acknowledgements 252
About This Document 253

List of Tables

1-1. Typographical conventions for special terms 2
2-1. Most common navigation keys in entry-based menus 6
2-2. Most common navigation keys in page-based menus 7
2-3. Most common line editor keys 7
2-4. Most common message index keys 9
2-5. Message status flags 10
2-6. Message recipient flags 11
2-7. Most common pager keys 11
2-8. ANSI escape sequences 12
2-9. Color sequences 12
2-10. Most common thread mode keys 13
2-11. Special Thread Characters 14
2-12. Most common mail sending keys 16
2-13. Most common compose menu keys 18
2-14. PGP key menu flags 20
3-1. Symbolic key names 31
4-1. POSIX regular expression character classes 58
4-2. Regular expression repetition operators 59
4-3. GNU regular expression extensions 60
4-4. Pattern modifiers 60
4-5. Simple search keywords 64
4-6. Date units 65
4-7. Relative date units 66
4-8. Mailbox shortcuts 71
5-1. Supported MIME types 88
6-1. Sidebar Variables 107
6-2. Sidebar Functions 107
6-3. Sidebar Colors 108
6-4. Sidebar Sort 108
6-5. Not all Hooks are Required 109
9-1. Command line options 120
9-2. Default Generic Menu Bindings 233
9-3. Default Index Menu Bindings 235
9-4. Default Pager Menu Bindings 239
9-5. Default Alias Menu Bindings 244
9-6. Default Query Menu Bindings 244
9-7. Default Attachment Menu Bindings 244
9-8. Default Compose Menu Bindings 245
9-9. Default Postpone Menu Bindings 247
9-10. Default Browser Menu Bindings 248
9-11. Default Pgp Menu Bindings 248
9-12. Default Smime Menu Bindings 249
9-13. Default Mixmaster Menu Bindings 249
9-14. Default Editor Menu Bindings 249
9-15. Default Autocrypt Account Menu Bindings 251
9-16. Default List Menu Bindings 251

Chapter 1. Introduction

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.

Mutt Home Page

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

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`> — users help users
- <`mutt-dev-request@mutt.org`> — patches, bug reports, feature requests

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

NOTE: You MUST be subscribed to a list in order to post to it. This is not to make your life harder, but to reduce SPAM and/or UCE. The mailing list software being used is GNU Mailman, with its implied deficiencies.

Getting Mutt

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

For version control access, please refer to the Mutt development site (https://gitlab.com/muttmua/mutt).

Mutt Online Resources

Bug Tracking System

The official Mutt bug tracking system (not for feature requests) can be found at https://gitlab.com/muttmua/mutt/issues

Wiki

An (unofficial) wiki can be found at https://gitlab.com/muttmua/mutt/wikis/home.

IRC

For the IRC user community, visit channel *#mutt* on irc.libera.chat (https://libera.chat/).

USENET

For USENET, see the newsgroup comp.mail.mutt (news:comp.mail.mutt).

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 https://gitlab.com/muttmua/mutt for more details.

Typographical Conventions

This section lists typographical conventions followed throughout this manual. See table Table 1-1 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.

Copyright

Mutt is Copyright © 1996-2026 Michael R. Elkins <me@mutt.org> and others.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Chapter 2. Getting Started

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 (http://www.mutt.org/) 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.

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 to send messages. It also supports a `mailx(1)`-compatible interface, see Table 9-1 for a complete list of command-line options.

Screens and Menus

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.

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.

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.

Some mail systems can nest mail folders inside other mail folders. The normal open entry commands in mutt will open the mail folder and you can't see the sub-folders. If you instead use the `<descend-directory>` function it will go into the directory and not open it as a mail directory.

Sidebar

The Sidebar shows a list of all your mailboxes. The list can be turned on and off, it can be themed and the list style can be configured.

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).

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.

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.

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.

List Menu

The list menu assists with operations on mailing lists. RFC 2369 defines several interactions with mailing lists and list memberships that can be specified within the email message: subscribe, unsubscribe, contact the list owner, etc. When you invoke the list menu, these interactions are made accessible as menu options.

Moving Around in Menus

The most important navigation keys common to line- or entry-based menus are shown in Table 2-1 and in Table 2-2 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
k or <Up>	`<previous-entry>`	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

Editing Input Fields

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 the Section called *Editor Menu* in Chapter 9 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
<Space>	`<buffy-cycle>`	cycle among incoming mailboxes
^D or <Delete>	`<delete-char>`	delete the char under the cursor

Key	Function	Description
^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, alias, or label
^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
^R	<history-search>	use current input to search 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

^G is the generic "abort" key in Mutt. In addition to the line editor, it can also be used to abort prompts. Generally, typing ^G at a confirmation prompt or line editor should abort the entire action.

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
```

Buffy Cycle

The <buffy-cycle> function key binding is enabled in the prompt for <change-folder> or <change-folder-readonly>. Typing the key will cycle through mailboxes with new mail.

In other prompts for a mailbox or a file (such as for saving an attachment), the key binding instead invokes the <complete> function.

In either case, if you need to type the key literally (e.g. you need to enter a file name with a space in it), use the <quote-char> function. For example, with the default key bindings, by typing "^V" and then "<Space>".

In all other input fields, the <buffy-cycle> key binding is ignored; typing the key simply adds it to the input field text.

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 and $save_history. You may cycle through them at an editor prompt by using the <history-up> and/or <history-down> commands. Mutt will remember the currently entered text as you cycle through history, and will wrap around to the initial entry line.

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

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

Mutt automatically filters out consecutively repeated items from the history. If $history_remove_dups is set, all repeated items are removed 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.

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.

The Message Index

Common keys used to navigate through and manage messages in the index are shown in Table 2-4. 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

Key	Description
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 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 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)

Flag	Description
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

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

Key	Description
/	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, 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.

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)
4<*color*>	Background color is <*color*> (see Table 2-9)
38;5;<*color*>	Foreground color is an 8-bit <*color*>
48;5;<*color*>	Background color is an 8-bit <*color*>

Table 2-9. Color sequences

Color code	Color
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta

Color code	Color
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.

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.

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

Key	Function	Description
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

In the *index*, the subject of threaded children messages will be prepended with thread tree characters. By default, the subject itself will not be duplicated unless $hide_thread_subject is unset. Special characters will be added to the thread tree as detailed in Table 2-11.

Table 2-11. Special Thread Characters

Character	Description	Notes
&	hidden message	see $hide_limited and $hide_top_limited
?	missing message	see $hide_missing and $hide_top_missing
*	pseudo thread	see $strict_threads; not displayed when $narrow_tree is set
=	duplicate thread	see $duplicate_threads; not displayed when $narrow_tree is set

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.

Miscellaneous Functions

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

<check-stats>

Calculate statistics for all monitored mailboxes declared using the **mailboxes** command. It will calculate statistics despite $mail_check_stats being unset.

`<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_decode_weed, $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.

`<skip-headers>` (default: H)

This function will skip past the headers of the current message.

`<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.

`<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.

Sending Mail

Introduction

The bindings shown in Table 2-12 are available in the *index* and *pager* to start a new message.

Table 2-12. Most common mail sending keys

Key	Function	Description
m	`<mail>`	compose a new message
r	`<reply>`	reply to sender
g	`<group-reply>`	reply to all recipients
	`<group-chat-reply>`	reply to all recipients preserving To/Cc
L	`<list-reply>`	reply to mailing list address
f	`<forward-message>`	forward message
b	`<bounce-message>`	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; cc all other recipients; consults **alternates** and excludes you.

Group Chat reply

Reply to the author and other recipients in the To list; cc other recipients in the Cc list; consults **alternates** and excludes you.

List reply

Reply to all mailing list addresses found, either specified via configuration or auto-detected. See the Section called *Mailing Lists* in Chapter 3 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; the message body should start on a new line after the existing blank line at the end of headers. 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 2-13 to modify, send or postpone the message.

Table 2-13. 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
F	`<filter-entry>`	filter attachment through a shell command
G	`<get-attachment>`	get a temporary copy of an attachment
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. `<attach-file>` will add an attachment and `<detach-file>` will remove an attachment from the message. `<get-attachment>` will create a temporary copy of the attachment, which can be useful if you intend to modify it before sending. For instance, `<filter-entry>`, which will pipe the attachment through a program and replace the attachment with the output of the pipe. Because of the danger of unintentionally overwriting the attachment, mutt will always prompt to confirm before invoking the program.

The `<attach-message>` function 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.

After exiting the compose menu via `<send-message>`, the message will be sent. If configured and enabled, this can happen via mixmaster or $smtp_url. Otherwise $sendmail will be invoked. Prior to version 1.13, Mutt enabled $write_bcc by default, assuming the MTA would automatically remove a `Bcc:` header as part of delivery. Starting with 1.13, the option is unset by default, but no longer affects the fcc copy of the message.

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.

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.

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.

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 for the duration of the message composition session. The selection can later be changed in the compose menu.

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".

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.

To ensure you can view encrypted messages you have sent, you may wish to set $pgp_self_encrypt and $pgp_default_key for PGP, or $smime_self_encrypt and $smime_default_key for S/MIME.

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 2-14.

Table 2-14. 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.

Sending Format=Flowed Messages

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.

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, 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.

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

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.

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

Reformatting

Mutt has some support for reformatting when viewing and replying to `format=flowed` messages. In order to take advantage of these, $reflow_text must be set.

- Paragraphs are automatically reflowed and wrapped at a width specified by $reflow_wrap.

- In its original format, the quoting style of `format=flowed` messages can be difficult to read, and doesn't intermix well with non-flowed replies. Setting $reflow_space_quotes adds spaces after each level of quoting when in the pager and replying in a non-flowed format (i.e. with $text_flowed unset).
- If $reflow_space_quotes is unset, mutt will still add one trailing space after all the quotes in the pager (but not when replying).

Background Editing

If $editor is set to a graphical editor, or a script such as contrib/bgedit-screen-tmux.sh (https://gitlab.com/muttmua/mutt/tree/master/contrib/bgedit-screen-tmux.sh) if running inside GNU Screen or tmux, you can run the editor in the background by setting $background_edit.

If set, Mutt will display a landing page while the editor runs. When the editor exits, message composition will resume automatically. Alternatively, you can `<exit>` from the landing page, which will return you to the message index. This allows viewing other messages, changing mailboxes, even starting a new message composition session - all while the first editor session is still running.

Backgrounded message composition sessions can be viewed via `<background-compose-menu>` in the index and pager, by default bound to "`B`". If there is only a single backgrounded session, which has already exited, that session will automatically resume. Otherwise the list will be displayed, and a particular session can be selected. $background_format controls the format string used for the menu.

In case the open mailbox is changed while a reply is backgrounded, Mutt keeps track of the original mailbox. After sending, Mutt will attempt to reopen the original mailbox, if needed, and set reply flags appropriately. This won't affect your currently open mailbox, but may make setting flags a bit slower due to the need to reopen the original mailbox behind the scenes.

One complication with backgrounded compose sessions is the config changes caused by send, reply, and folder hooks. These can get triggered by a new message composition session, or by changing folders during a backgrounded session. To help lessen these problems, Mutt takes a snapshot of certain configuration variables and stores them with each editing session when it is backgrounded. When the session is resumed, those stored settings will temporarily be restored, and removed again when the session finishes (or is backgrounded again).

Mutt will save all `boolean` and `quadoption` configuration variables, the current folder (which will be used for ^ mailbox shortcut expansion), along with: $folder, $record, $postponed, $envelope_from_address, $from, $sendmail, $smtp_url, $pgp_sign_as, $smime_sign_as, and $smime_encrypt_with. It's not feasible to backup all variables, but if you believe we've missed an important setting, please let the developers know.

To help prevent forgetting about backgrounded sessions, $background_confirm_quit will prompt before exiting, in addition to $quit. Additionally, the `%B` expando in $status_format displays the number of backgrounded compose sessions.

Background editing is available for most, but not all, message composition in Mutt. Sending from the command line disables background editing, because there is no index to return to.

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-message>` function and forwarding using the `<forward-message>` 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: see $forward_attribution_intro and $forward_attribution_trailer) 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".

Mutt's default ($mime_forward="no" and $forward_decode="yes") is to use standard inline forwarding. In that mode all text-decodable parts are included in the new message body. Other attachments from the original email can also be attached to the new message, based on the quadoption $forward_attachments.

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

Editing the message to forward follows the same procedure as sending or replying to a message does, but can be disabled via the quadoption $forward_edit.

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.

Encryption and Signing

Mutt supports encrypting and signing emails when used interactively. In batch mode, cryptographic operations are disabled, so these options can't be used to sign an email sent via a cron job, for instance.

OpenPGP and S/MIME are enabled in one of two ways: "classic mode" or GPGME. The former invokes external programs to perform the various operations; it is better tested and more flexible, but requires some configuration. The latter uses the GnuPG project's GPGME library.

To enable "classic mode", ensure GPGME is disabled and use the `gpg.rc` or `smime.rc` files that come with mutt. These are typically installed under `/usr/local/share/doc/mutt/samples/`. Source them, either directly or by copying them to your .mutt directory and sourcing them. Sourcing them directly from `/usr/local/share/doc/mutt/samples/` has the benefit of automatically using fixes and security improvements to the command invocations, and is recommended.

```
unset crypt_use_gpgme
source /usr/local/share/doc/mutt/samples/gpg.rc
source /usr/local/share/doc/mutt/samples/smime.rc
```

To use GPGME instead, simply ensure the option is enabled in your .muttrc:

```
set crypt_use_gpgme
```

OpenPGP Configuration

The two most important settings are $pgp_default_key and $pgp_sign_as. To perform encryption, you must set the first variable. If you have a separate signing key, or only have a signing key, then set the second. Most people will only need to set $pgp_default_key.

Starting with version 2.1.0, GnuPG automatically uses an `agent` to prompt for your passphrase. If you are using a version older than that, you'll need to ensure an agent is running (alternatively, you can unset $pgp_use_gpg_agent and Mutt will prompt you for your passphrase). The agent in turn uses a `pinentry` program to display the prompt. There are many different kinds of pinentry programs that can be used: qt, gtk2, gnome3, fltk, and curses. However, Mutt does *not* work properly with the tty pinentry program. Please ensure you have one of the GUI or curses pinentry programs installed and configured to be the default for your system.

S/MIME Configuration

As with OpenPGP, the two most important settings are $smime_default_key and $smime_sign_as. To perform encryption and decryption, you must set the first variable. If you have a separate signing key, or only have a signing key, then set the second. Most people will only need to set $smime_default_key.

In "classic mode", keys and certificates are managed by the `smime_keys` program that comes with Mutt. By default they are stored under `~/.smime/`. (This is set by the `smime.rc` file with $smime_certificates and $smime_keys.) To initialize this directory, use the command "`smime_keys init`" from a shell prompt. The program can be then be used to import and list certificates. You may also want to periodically run "`smime_keys refresh`" to update status flags for your certificates.

Chapter 3. Configuration

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`. If still not found, Mutt will try `$XDG_CONFIG_HOME/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.

Starter Muttrc

Mutt is highly configurable because it's *meant* to be customized to your needs and preferences. However, this configurability can make it difficult when just getting started. A few sample muttrc files come with mutt, under `doc/mutt/samples/`. Among them, sample.muttrc-starter (https://gitlab.com/muttmua/mutt/tree/master/contrib/sample.muttrc-starter) is a basic example config with a few suggested settings and pointers to useful programs.

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, 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`
```

To avoid the output of backticks being parsed, place them inside double quotes. In Example 3-6, the output of the gpg decryption is assigned directly to $imap_pass, so that special characters in the password (e.g.“'”, “#”, “$”) are not parsed and interpreted specially by mutt.

Example 3-6. Preventing the output of backticks from being parsed

```
set imap_pass="`gpg --batch -q --decrypt ~/.mutt/account.gpg`"
```

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

Example 3-7. 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.

If $muttlisp_inline_eval is set, an unquoted parenthesis-enclosed expression will be evaluated as MuttLisp. See the Using MuttLisp section for more details.

Example 3-8. Using MuttLisp expressions

```
set signature = \
  (if (equal $my_name "Kevin McCarthy") ~/kevin.sig ~/other.sig)
```

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).

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.

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 add an alias:

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

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

```
unalias muttdude
unalias *
```

Note: The alias *key* is matched case insensitively when creating (checking for duplicates), removing, or expanding aliases.

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 **source**d. 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 3-9. 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 `tag-entry` key (default: <Space> or t), and use the *exit* key (default: q) to return to the address prompt.

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?*).

You can also use the form <*177*>, which allows octal numbers with an arbitrary number of digits. In addition, *key* may be a symbolic name as shown in Table 3-1.

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
<keypadenter>	Enter key on numeric keypad
<home>	Home
<end>	End
<space>	Space bar
<f1>	function key 1
<f10>	function key 10

The `<what-key>` function can be used to explore keycode and symbolic names for other keys on your keyboard. Executing this function will display information about each key pressed, until terminated by `^G`.

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.

Terminal Keybindings

Some key bindings are controlled by the terminal, and so by default can't be bound inside Mutt. These may include `^C`, `^\`, `^Q`, `^S`, `^Z`, and on BSD/Mac `^Y`. These terminal settings can be viewed and changed using the `stty` program.

"`stty -a`" will list the bound characters (not all of them affect Mutt), and what actions they take when pressed. For example, you may see "`intr = ^C`" in its output. This means typing `^C` will send an interrupt signal. "`quit = ^\`" means typing `^\` (commonly also `^4`) will send a quit signal.

To unbind a key from an action, you invoke "stty action undef". For example, "`stty quit undef`" will unbind `^\` (and `^4`) from sending the quit signal. Once unbound (e.g, by placing that line in your .bashrc, or in a Mutt wrapper script/function) you can use the key sequence in your Mutt bindings.

Enter versus Return

Prior to version 2.2, Mutt used a default ncurses mode ("`nl()`"). This mode maps keyboard input of either `<Enter>` or `<Return>` to the same value, which Mutt interpreted as `<Return>` internally.

However, starting in version 2.2, this mode is turned off, allowing `<Return>` and `<Enter>` to be mapped separately, if desired. The default keyboard mappings set both, but you can override this or create new bindings with one or the other (or both).

Note that in terminal application, such as Mutt, `<Enter>` is the same as "\n" and `^J`; while `<Return>` is the same as "\r" and `^M`.

Changing the current working directory

Usage:

cd *directory*

The `cd` command changes Mutt's current working directory. This affects commands and functions like `source`, `change-folder`, and `save-entry` that use relative paths. Using `cd` without directory changes to your home directory.

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.

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-hook**s, they are executed in the order given in the `.muttrc`.

The regexp parameter has mailbox shortcut expansion performed on the first character. See Mailbox Matching in Hooks for more details.

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-hook**s adjusting a value on a per-folder basis because **folder-hook**s are evaluated in the order given in the configuration file.

Note: The keyboard buffer will not be processed until after all hooks are run; multiple push or exec commands will end up being processed in reverse order.

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 3-10. Setting sort method based on mailbox name

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

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.

Using Color and Mono Video Attributes

Usage:

color *object* [*attribute* ...] *foreground background*
color { header | body } [*attribute* ...] *foreground background regexp*
color index [*attribute* ...] *foreground background pattern*
color compose *composeobject* [*attribute* ...] *foreground background*
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* in the message index. Note that IMAP server-side searches (=b, =B, =h) are not supported for color index patterns.

When $header_color_partial is unset (the default), a *header* matched by *regexp* will have color applied to the entire header. When set, color is applied only to the exact text matched by *regexp*.

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
- prompt
- 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)

composeobject can be one of:

- header
- security_encrypt
- security_sign
- security_both
- security_none

attribute can be one of the following:

- none
- bold
- underline
- reverse
- standout

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

- white
- black
- green
- magenta

- blue
- cyan
- yellow
- red
- default
- color*x*

The color name can optionally be prefixed with the keyword `bright` or `light` to make the color boldfaced or light (e.g., `brightred`). The precise behavior depends on the terminal and its configuration. In particular, the boldfaced/light difference and such background colors may be available only for terminals configured with at least 16 colors, as specified by the `$TERM` environment variable.

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*
mono compose *composeobject attribute*
unmono { index | header | body } { * | *pattern* }

For *object*, *composeobject*, and *attribute*, see the **color** command.

Message Header Display

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.)

Despite not being a real header, Mutt will also display an mbox "From_" line in the pager along with other headers. This line can be manipulated with **ignore/unignore** and **hdr_order/unhdr_order** commands.

Mutt has two modes for displaying the headers. When $weed is set (the default), or toggled on via `<display-toggle-weed>`, mutt unfolds and then refolds header lines using $wrap as described above. Furthermore, the following sections will be applied, allowing the selection and ordering of headers.

When $weed is unset, or toggled off via `<display-toggle-weed>`, mutt will display all headers as they appear in the original email (but still subject to being folded at the width of your terminal). None of the following sections are applied.

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 3-11. 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:
```

The above example will show "From:" headers as well as mbox "From_" lines. To hide the latter, instead use "`unignore from: date subject to cc`" on the second line.

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 3-12. Configuring header display order

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

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.

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 known list and $followup_to is set, Mutt will add a Mail-Followup-To header. For unsubscribed lists, this will include your personal address, ensuring you receive a copy of replies. For subscribed mailing lists, the header will not, telling 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 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**.

All of the mailing list configuration options described so far govern mutt’s knowledge of your list subscriptions and how it presents list information to you. If you have a message from a mailing list, you can also use the list menu (bound to "ESC L" by default) to interact with the message’s list’s list server. This makes it easy to subscribe, unsubscribe, and so on.

Using Multiple Spool Mailboxes

Usage:

mbox-hook *`[!]regexp mailbox`*

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

The regexp parameter has mailbox shortcut expansion performed on the first character. See Mailbox Matching in Hooks for more details.

Note that execution of mbox-hooks is dependent on the $move configuration variable. If set to “no” (the default), mbox-hooks will not be executed.

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

Monitoring Incoming Mail

Usage:

mailboxes [[`-notify` | `-nonotify`] [`-poll` | `-nopoll`] [`-label` *`label`* | `-nolabel`] *`mailbox`*] [...]
unmailboxes { `*` | *`mailbox`* }

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

Use `-nonotify` to disable notifying when new mail arrives. The `-notify` argument can be used to reenable notifying for an existing mailbox. If unspecified: a new mailbox will notify by default, while an existing mailbox will be unchanged.

To disable polling, specify `-nopoll` before the mailbox name. The `-poll` argument can be used to reenable polling for an existing mailbox. If unspecified: a new mailbox will poll by default, while an existing mailbox will be unchanged.

The `-label` argument can be used to specify an alternative label to print in the sidebar or mailbox browser instead of the mailbox path. A label may be removed via the `-nolabel` argument. If unspecified, an existing mailbox label will be unchanged.

mailbox can either be a local file or directory (Mbox/Mmdf or Maildir/Mh). If Mutt was built with POP and/or IMAP support, *mailbox* can also be a POP/IMAP folder URL. The URL syntax is described in the Section called *URL Syntax* in Chapter 6, POP and IMAP are described in the Section called *POP3 Support* in Chapter 6 and the Section called *IMAP Support* in Chapter 6 respectively.

Mutt provides a number of advanced features for handling (possibly many) folders and new mail within them, please refer to the Section called *New Mail Detection* in Chapter 4 for details (including in what situations and how often Mutt checks for new mail). Additionally, $new_mail_command can be used to run a command when new mail is detected.

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.

User-Defined Headers

Usage:

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

The **my_hdr** command allows you to add custom header fields to every message you send. It can also be used as an alternate way to set some standard envelope header fields, see below for more details.

Custom header fields 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 3-13 in your `.muttrc`.

Example 3-13. Defining custom headers

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

The standard envelope header fields To, Cc, Bcc, Subject, From, Reply-To, and Message-ID can also be set via **my_hdr**. For more details about when those are processed, see the Section called *Message*

Composition Flow in Chapter 4. Note that trying to set the value of other Mutt generated header fields is not supported, and may result in an invalid email being generated.

> **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
```

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 3-14. 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.

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 *pattern* 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.

Multiple mailboxes may be specified by separating them with $fcc_delimiter, if set:

```
set fcc_delimiter = ','
fcc-hook 'foo@example\.com$' '+one,+two'
```

Specify Default Save Filename and Default Fcc: Mailbox at Once

Usage:

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

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

Note, however that the fcc-save-hook is not designed to take advantage of multiple mailboxes, as fcc-hook is. For correct behavior, you should use separate fcc and save hooks in that case.

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-hook**s are matched *before* the **send-hook**, *regardless* of the order specified in the user's configuration file. However, you can inhibit **send-hook** in the reply case by using the pattern `'! ~Q'` (*not replied*, see Message Matching in Hooks) in the **send-hook** to tell when **reply-hook** have been executed.

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, $attribution_locale, and $signature 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. They are not executed when resuming a postponed draft. 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**.

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: .*\""'
```

Choosing the Cryptographic Key of the Recipient

Usage:

crypt-hook *regexp 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. You may use multiple crypt-hooks with the same regexp; multiple matching crypt-hooks result in the use of multiple keyids for a recipient. During key selection, Mutt will confirm whether each crypt-hook is to be used (unless the $crypt_confirmhook option is unset). If all crypt-hooks for a recipient are declined, Mutt will use the original recipient address for key selection instead.

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

Dynamically Changing $index_format using Patterns

Usage:

index-format-hook *name [!]pattern format-string*

This command is used to inject format strings dynamically into $index_format based on pattern matching against the current message.

The $index_format expando *%@name@* specifies a placeholder for the injection. Index-format-hooks with the same *name* are matched using *pattern* against the current message. Matching is done in the order specified in the .muttrc, with the first match being used. The hook's *format-string* is then substituted and evaluated.

Because the first match is used, best practice is to put a catch-all *~A* pattern as the last hook. Here is an example showing how to implement dynamic date formatting:

```
set index_format="%4C %-6@date@ %-15.15F %Z (%4c) %s"

index-format-hook  date  "~d<1d"    "%[%H:%M]"
index-format-hook  date  "~d<1m"    "%[%a %d]"
index-format-hook  date  "~d<1y"    "%[%b %d]"
index-format-hook  date  "~A"       "%[%m/%y]"
```

Another example, showing a way to prepend to the subject. Note that without a catch-all ~A pattern, no match results in the expando being replaced with an empty string.

```
set index_format="%4C %@subj_flags@%s"

index-format-hook  subj_flags  "~f boss@example.com"    "** BOSS ** "
index-format-hook  subj_flags  "~f spouse@example.com"  ":-) "
```

Adding Key Sequences to the Keyboard Buffer

Usage:

push *string*

This command adds the named string to the beginning of 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 3-15 shows how to automatically collapse all threads when entering a folder.

Example 3-15. 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>`).

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>`".

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`, `~h`, `~M`, or `~X` 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.

Scoring occurs as the messages are read in, before the mailbox is sorted. Because of this, patterns which depend on threading, such as ~=, ~$, and ~(), will not work by default. A workaround is to push the scoring command in a folder hook. This will cause the mailbox to be rescored after it is opened and input starts being processed:

```
folder-hook . 'push "<enter-command>score ~= 10<enter>"'
```

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.)

Note: the value displayed by `%H` and searched by `~H` is stored in the header cache. Mutt isn't smart enough to invalidate a header cache entry based on changing `spam` rules, so if you aren't seeing correct `%H` values, try temporarily turning off the header cache. If that fixes the problem, then once your spam rules are set to your liking, remove your stale header cache files and turn the header cache back on.

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 3-16.

Example 3-16. 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"
```

Setting and Querying Variables

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.

number (long)

A signed integer number in the range -2147483648 to 2147483647.

string

Arbitrary text.

path

A specialized string for representing paths including support for mailbox shortcuts (see the Section called *Mailbox Shortcuts* in Chapter 4) 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 the Section called *Regular Expressions* in Chapter 4 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 the Section called *User-Defined Variables* for details.

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.

User-Defined Variables

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.

Examples

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

Example 3-17. 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 3-18. 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 3-19. 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.

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 3-20 demonstrates type conversions.

Example 3-20. 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.

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|`).

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`.

Format Strings

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.

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.

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 3-21. 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.

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"
```

Bytes size display

Various format strings contain expandos that display the size of messages in bytes. This includes `%s` in $attach_format, `%l` in $compose_format, `%s` in $folder_format, `%c` in $index_format, and %l and %L in $status_format. There are four configuration variables that can be used to customize how the numbers are displayed.

$size_show_bytes will display the number of bytes when the size is < 1 kilobyte. When unset, kilobytes will be displayed instead.

$size_show_mb will display the number of megabytes when the size is >= 1 megabyte. When unset, kilobytes will be displayed instead (which could be a large number).

$size_show_fractions, will display numbers with a single decimal place for values from 0 to 10 kilobytes, and 1 to 10 megabytes.

$size_units_on_left will display the unit ("K" or "M") to the left of the number, instead of the right if unset.

These variables also affect size display in a few other places, such as progress indicators and attachment delimiters in the pager.

Control allowed header fields in a mailto: URL

Usage:

mailto_allow { * | *header-field* }
unmailto_allow { * | *header-field* }

As a security measure, Mutt will only add user-approved header fields from a `mailto:` URL. This is necessary since Mutt will handle certain header fields, such as `Attach:`, in a special way. The `mailto_allow` and `unmailto_allow` commands allow the user to modify the list of approved headers.

Mutt initializes the default list to contain the `Subject` and `Body` header fields, which are the only requirement specified by the `mailto:` specification in RFC2368, along with `Cc`, `In-Reply-To`, and `References`, to support mailing list URLs.

Chapter 4. Advanced Usage

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 (http://www.iana.org/assignments/character-sets), a list of locally supported locales can be obtained by running `locale -a`.

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

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

Character class	Description
[: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.

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

Operator	Description
{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.

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.

Patterns: Searching, Limiting and Tagging

Pattern Modifier

Many of Mutt's commands allow you to specify a pattern to match (`limit`, `tag-pattern`, `delete-pattern`, etc.). Table 4-4 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*	If IMAP is enabled, like ~b but 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 ***)
=B *STRING*	If IMAP is enabled, like ~B but searches for *STRING* on the server, rather than downloading each message and searching it locally.
~c *EXPR*	messages carbon-copied to *EXPR*
%c *GROUP*	messages carbon-copied to any member of *GROUP*
~C *EXPR*	messages to:, cc:, or bcc: *EXPR*
%C *GROUP*	messages to:, cc:, or bcc: 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 *STRING*	If IMAP is enabled, like ~h but searches for *STRING* on the server, rather than downloading each message and searching it locally; *STRING* must be of the form “header: substring” (see below).
~H *EXPR*	messages with a spam attribute matching *EXPR*
~i *EXPR*	messages which match *EXPR* in the “Message-ID” field

Pattern modifier	Description
~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* *)
~M *EXPR*	messages which contain a mime Content-Type matching *EXPR* ***)
~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 $from, **alternates**, and local account/hostname information)
~P	messages from you (consults $from, **alternates**, and local account/hostname information)
~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)

Pattern modifier	Description
~<(*PATTERN*)	messages whose immediate parent matches *PATTERN*, e.g. replies to your messages: ~<(~P)
~>(*PATTERN*)	messages having an immediate child matching *PATTERN*, e.g. messages you replied to: ~>(~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.

***) These patterns read each message in, and can therefore be much slower. Over IMAP this will entail downloading each message. They can not be used for message scoring, and it is recommended to avoid using them for index coloring.

Special attention has to be paid 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.

For IMAP folders, string matches `=b`, `=B`, and `=h` will 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$
```

You can restrict address pattern matching to aliases that you have defined with the "@" modifier. This example matches messages whose recipients are all from Germany, and who are known to your alias list.

Example 4-2. Matching restricted to aliases

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

To match any defined alias, use a regular expression that matches any string. This example matches messages whose senders are known aliases.

Example 4-3. Matching any defined alias

```
@~f .
```

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: 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.

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 4-4. 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.

Searching by Date

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

Absolute Dates

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

```
Limit to messages matching: ~d 20/1/95-31/10
Limit to messages matching: ~d 19950120-19951031
```

If you omit the minimum (first) date, and just specify “-DD/MM/YY” or “-YYYYMMDD”, 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. 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
```

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 4-7.

Table 4-7. Relative date units

Unit	Description
y	Years
m	Months
w	Weeks
d	Days
H	Hours
M	Minutes
S	Seconds

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.

Marking Messages

There are times that it's useful to ask Mutt to "remember" which message you're currently looking at, while you move elsewhere in your mailbox. You can do this with the "mark-message" operator, which is bound to the "~" key by default. Press this key to enter an identifier for the marked message. When you want to return to this message, press "'" and the name that you previously entered.

(Message marking is really just a shortcut for defining a macro that returns you to the current message by searching for its Message-ID. You can choose a different prefix by setting the $mark_macro_prefix variable.)

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 **macro**s 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.

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**
- **index-format-hook**
- **mbox-hook**
- **message-hook**
- **reply-hook**
- **save-hook**
- **send-hook**
- **send2-hook**

for specific details on each type of *hook* available. Also see Message Composition Flow for an overview of the composition process.

> **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 4-5. Specifying a "default" hook

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

In Example 4-5, 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>`.

Message Matching in Hooks

Hooks that act upon messages (**message-hook**, **reply-hook**, **send-hook**, **send2-hook**, **save-hook**, **fcc-hook**, **index-format-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.

Mailbox Matching in Hooks

Hooks that match against mailboxes (**folder-hook**, **mbox-hook**) apply both regular expression syntax as well as mailbox shortcut expansion on the regexp parameter. There is some overlap between these, so special attention should be paid to the first character of the regexp.

```
# Here, ^ will expand to "the current mailbox" not "beginning of string":
folder-hook ^/home/user/Mail/bar "set sort=threads"
```

```
# If you want ^ to be interpreted as "beginning of string", one workaround
# is to enclose the regexp in parenthesis:
folder-hook (^/home/user/Mail/bar) "set sort=threads"

# This will expand to the default save folder for the alias "imap.example.com", which
# is probably not what you want:
folder-hook @imap.example.com "set sort=threads"

# A workaround is to use parenthesis or a backslash:
folder-hook (@imap.example.com) "set sort=threads"
folder-hook '\@imap.example.com' "set sort=threads"
```

Keep in mind that mailbox shortcut expansion on the regexp parameter takes place when the hook is initially parsed, not when the hook is matching against a mailbox. When Mutt starts up and is reading the .muttrc, some mailbox shortcuts may not be usable. For example, the "current mailbox" shortcut, ^, will expand to an empty string because no mailbox has been opened yet. Mutt will issue an error for this case or if the mailbox shortcut results in an empty regexp.

Managing the Environment

You can alter the environment that Mutt passes on to its child processes using the “setenv” and “unsetenv” operators. (N.B. These follow Mutt-style syntax, not shell-style!) You can also query current environment values by prefixing a “?” character.

```
setenv TERM vt100
setenv ORGANIZATION "The Mutt Development Team"
unsetenv DISPLAY
setenv ?LESS
```

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.

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 the Section called *Header Caching* in Chapter 6 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.

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 4-8. 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=^'
```

Note: the current mailbox shortcut, "^", has no value in some cases. No mailbox is opened when Mutt is invoked to send an email from the command-line. In interactive mode, Mutt reads the muttrc before opening the mailbox, so immediate expansion won't work as expected either. This can be an issue when trying to directly assign to $record, but also affects the fcc-hook mailbox, which is expanded immediately too. The folder-hook example above works because the command is executed later, when the folder-hook fires.

Note: the $record shortcut "<" is substituted without any regard to multiple mailboxes and $fcc_delimiter. If you use multiple Fcc mailboxes, and also want to use the "<" mailbox shortcut, it might be better to set $record to the primary mailbox and use a fcc-hook to set all mailboxes during message composition.

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`. Alternatively or additionally, you can set $auto_subscribe to automatically subscribe addresses found in a `List-Post` header.

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 known mailing lists, and if the $followup_to option is set, Mutt will generate a Mail-Followup-To header. If any of the recipients are subscribed mailing lists, this header will contain 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. If none of the recipients are subscribed mailing lists, the header will also contain your address, ensuring you receive a copy of replies.

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.

While looking at an email message from a mailing list in the index or pager, you can interact with the list server in the ways defined by RFC 2369, provided the email message specifies how to do so. Invoke the list menu (bound to "ESC L" by default) to see what options are available for a given message. Common options are:

- Post to the list
- Contact the list owner

- Subscribe to the list
- Unsubscribe from the list
- Get help from the list server
- Get list archive information

Note that many list servers only specify some of these options.

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.

You can change or delete the “X-Label:” field within Mutt using the “edit-label” command, bound to the “y” key by default. This works for tagged messages, too. While in the edit-label function, pressing the <complete> binding (TAB, by default) will perform completion against all labels currently in use.

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.

Display Munging

Working within the confines of a console or terminal window, it is often useful to be able to modify certain information elements in a non-destructive way -- to change how they display, without changing the stored value of the information itself. This is especially so of message subjects, which may often be polluted with extraneous metadata that either is reproduced elsewhere, or is of secondary interest.

subjectrx *pattern replacement*
unsubjectrx { * | *pattern* }

`subjectrx` specifies a regular expression “pattern” which, if detected in a message subject, causes the subject to be replaced with the “replacement” value. The replacement is subject to substitutions in the same way as for the spam command: `%L` for the text to the left of the match, `%R` for text to the right of the match, and `%1` for the first subgroup in the match (etc). If you simply want to erase the match, set it to “%L%R”. Any number of `subjectrx` commands may coexist.

Note this well: the “replacement” value replaces the entire subject, not just the match!

`unsubjectrx` removes a given subjectrx from the substitution list. If `*` is used as the pattern, all substitutions will be removed.

Example 4-6. Subject Munging

```
# Erase [rt #12345] tags from Request Tracker (RT) e-mails
subjectrx '\[rt #[0-9]+\] *' '%L%R'
```

```
# Servicedesk is another RT that sends more complex subjects.
# Keep the ticket number.
subjectrx '\[servicedesk #([0-9]+)\] ([^.]+)\.([^.]+) - (new|open|pending|update) - ' '%L[#

# Strip out annoying [listname] prefixes in subjects
subjectrx '\[[^]]*\]:? *' '%L%R'
```

New Mail Detection

Mutt supports setups with multiple folders, allowing all of them to be monitored for new mail (see the Section called *Monitoring Incoming Mail* in Chapter 3 for details).

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.

The $mail_check_recent option changes whether Mutt will notify you of new mail in an already visited mailbox. When set (the default) it will only notify you of new mail received since the last time you opened the mailbox. When unset, Mutt will notify you of any new mail in the mailbox.

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 (excepting those specified with the

`-nopoll` flag). 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, or from the index/pager via the `<browse-mailboxes>` function.

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).

Monitoring New Mail

When the *Inotify* mechanism for monitoring of files is supported (Linux only) and not disabled at compilation time, Mutt immediately notifies about new mail for all folders configured via the **mailboxes** command (excepting those specified with the `-nopoll` flag). Dependent on mailbox format also added *old* mails are tracked (not for Maildir).

No configuration variables are available. Trace output is given when debugging is enabled via command line option `-d3`. The lower level 2 only shows errors, the higher level 5 all including raw Inotify events.

> **Note:** Getting events about new mail is limited to the capabilities of the underlying mechanism. *Inotify* only reports local changes, i. e. new mail notification works for mails delivered by an agent on the same machine as Mutt, but not when delivered remotely on a network file system as NFS. Also the monitoring handles might fail in rare conditions, so you better don't completely rely on this feature.

Calculating Mailbox Message Counts

If $mail_check_stats is set, Mutt will periodically calculate the unread, flagged, and total message counts for each mailbox watched by the **mailboxes** command. (Note: IMAP mailboxes only support unread and total counts). This calculation takes place at the same time as new mail polling, but is controlled by a separate timer: $mail_check_stats_interval.

The sidebar can display these message counts. See $sidebar_format.

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.

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.

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.

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.

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

https://github.com/sigpipe/urlview and the configuration commands:

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

Echoing Text

Usage:

echo *message*

You can print messages to the message window using the "echo" command. This might be useful after a macro finishes executing. After printing the message, echo will pause for the number of seconds specified by $sleep_time.

```
echo "Sourcing muttrc file"

unset confirmappend
macro index ,a "<save-message>=archive<enter><enter-command>echo 'Saved to archive'<enter>"
```

Message Composition Flow

This is a brief overview of the steps Mutt takes during message composition. It also shows the order and timing of hook execution.

- Reply envelope settings. $reverse_name processing. To, Cc, Subject, References header defaults.
- my_hdr processing for To, Cc, Bcc, Subject headers.
- Prompts for To, Cc, Bcc, Subject headers. See $askcc, $askbcc, $fast_reply.
- From header setting. Note: this is so send-hooks below can match ~P, but From is re-set further below in case a send-hook changes the value.
- reply-hook
- send-hook
- From header setting.
- my_hdr processing for From, Reply-To, Message-ID and user-defined headers. The To, Cc, Bcc, and Subject headers are ignored at this stage.
- Message body and signature generation.
- send2-hook
- $realname part of From header setting.
- $editor invocation for the message.
- send2-hook

- Cryptographic settings.
- fcc-hook. Fcc setting.
- Compose menu. Note: send2-hook is evaluated each time the headers are changed.
- $send_multipart_alternative generation.
- Message encryption and signing. Key selection.
- Fcc saving if $fcc_before_send is set. (Note the variable documentation for caveats of Fcc'ing before sending.)
- Message sending.
- Fcc saving if $fcc_before_send is unset (the default). Note: prior to version 1.12, the Fcc was saved before sending the message. It is now by default saved afterwards, but if the saving fails, the user is prompted.

Batch Composition Flow

In batch mode, Mutt performs less steps than interactive mode. Encryption and Signing are not supported.

- my_hdr processing for To, Cc, Bcc headers. (Subject is not processed.)
- From header setting. Note: this is so send-hooks below can match ~P, but From is re-set further below in case a send-hook changes the value.
- send-hook
- From header setting.
- my_hdr processing for From, Reply-To, Message-ID and user-defined headers. The To, Cc, Bcc, Subject, and Return-Path headers are ignored at this stage.
- Message body is copied from stdin. $signature is not appended in batch mode.
- send2-hook
- $realname part of From header setting.
- fcc-hook. Fcc setting.
- $send_multipart_alternative generation.
- Fcc saving if $fcc_before_send is set. (Note the variable documentation for caveats of Fcc'ing before sending.)
- Message sending.
- Fcc saving if $fcc_before_send is unset (the default). Note: prior to version 1.12, the Fcc was saved before sending the message. It is now by default saved afterwards, but if the saving fails, the user is prompted.

Using MuttLisp (EXPERIMENTAL)

MuttLisp is a Lisp-like enhancement for the Mutt configuration file. It is currently experimental, meaning new releases may change or break syntax. MuttLisp is not a real language, and is not meant to be an alternative to macros. The features are purposely minimal, with the actual work still being done by Mutt commands.

There are two ways to invoke MuttLisp: via the `run` command, or interpolated as a command argument.

Running a command generated by MuttLisp

Usage:

run *MuttLisp*

The `run` command evaluates the MuttLisp argument. The output of the MuttLisp is then executed as a Mutt command, as if it were typed in the muttrc instead.

```
run (concat "set my_name = '" \
      (or $ENV_NAME "Test User") "'")

  ==> generates and runs the line:
      set my_name = 'Test User'
```

This will set the Mutt User-Defined Variable `$my_name` to either the environment variable $ENV_NAME, if defined, or else "Test User".

Interpolating MuttLisp in a Command Argument

The second way of running is directly as a command argument. An unquoted parenthesis expression will be evaluated, and the result substituted as the argument.

To avoid breaking existing configurations, this is disabled by default. It can be enabled by setting $muttlisp_inline_eval. Before doing so, you should review your Mutt configuration to ensure you don't have any bare parenthesis expressions elsewhere, such as the regexp parameter of a folder-hook. These can typically be surrounded by single or double-quotes to prevent being evaluated as MuttLisp.

```
set my_name = (or $ENV_NAME "Test User")
```

The result of the MuttLisp is directly assigned as the argument. It isn't reinterpreted, so there is no need for the outer quotes. This is in contrast with the run command, where the output is reinterpreted by the muttrc parser.

MuttLisp Syntax

MuttLisp was inspired by Lisp, and so follows the same basic syntax. All statements are surrounded by parenthesis. The first argument inside the parenthesis is a function to invoke. The remaining arguments are passed as parameters.

The arguments to functions are read and evaluated using muttrc syntax. This means Mutt variables or environment variables can be passed directly, or interpolated inside a double-quoted string.

Although the arguments to a function are evaluated, the result of the function call is not.

```
echo (concat '$' 'spoolfile')
  ==> $spoolfile
```

MuttLisp has no types - everything is stored and evaluated as a string, just as with the muttrc. True is defined as a non-empty string, and false as the empty string.

The muttrc is evaluated line by line, and MuttLisp is similarly constrained. Input can be continued on more than one line by placing a backslash at the end of the line.

MuttLisp Functions

concat

Combines all arguments into a single string.

```
echo (concat one two three)
  ==> onetwothree
```

quote

Prevents interpretation of the list. Note that the list must still obey MuttLisp syntax: single quotes, double quotes, backticks, and parenthesis are still parsed prior to `quote` running and must be matching.

```
echo (quote one two three)
  ==> one two three

echo (quote $spoolfile)
  ==> $spoolfile

echo (quote (one two three))
  ==> (one two three)
```

equal

Performs a case-sensitive comparison of each argument. Stops evaluating arguments when it finds the first one that is not equal. Returns "t" if they are all equal, and the empty string if not.

```
echo (equal one one)
  ==> "t"

echo (equal one `echo one`)
  ==> "t"

echo (equal one one two `echo three`)
```

```
  ==> ""
  note: `echo three` does not execute.

echo (equal "one two" `echo one two`)
  ==> ""
  note: backticks generate two arguments "one" and "two"

echo (equal "one two" "`echo one two`")
  ==> "t"
  note: backticks inside double quotes generates a single argument: "one two"
```

not

Accepts a single argument only. Returns "t" if the argument evaluates to the empty string. Otherwise returns the empty string.

```
echo (not one)
  ==> ""

echo (not "")
  ==> "t"

echo (not (equal one two))
  ==> "t"
```

and

Returns the first argument that evaluates to the empty string. Otherwise returns the last argument, or "t" if there are no arguments.

```
echo (and one two)
  ==> "two"

echo (and "" two `echo three`)
  ==> ""
  note: `echo three` does not execute.

echo (and)
  ==> "t"
```

or

Returns the first argument that evaluates to a non-empty string. Otherwise returns the empty string.

```
echo (or one two)
  ==> "one"

echo (or "" two `echo three`)
  ==> "two"
```

```
  note: `echo three` does not execute.

echo (or)
  ==> ""
```

if

Requires 2 or 3 arguments. The first is a conditional. If it evaluates to "true" (a non-empty string), the second argument is evaluated and returned. Otherwise the third argument is evaluated and returned.

```
echo (if a one two)
  ==> "one"

echo (if "" one two)
  ==> "two"

set spoolfile = "/var/mail/user"
echo (if (equal $spoolfile "/var/mail/user") yes no)
  ==> "yes"
```

Note that boolean configuration variables evaluate to the strings "yes" or "no". You can see the value of other kinds of configuration variables using the echo command.

```
unset allow_ansi
echo $allow_ansi
  ===> "no"

# the correct way to test a boolean:
echo (if (equal $allow_ansi "yes") "set" "unset")
  ===> "unset"

# the incorrect way to test a boolean:
echo (if $allow_ansi "set" "unset")
  ===> "set"
```

Examples

It's important to remember that function arguments are evaluated, but the result is not. Also, the result of an interpolated command argument is used directly, and needs no quoting.

```
# A three-way toggle of $index_format:

set muttlisp_inline_eval
set my_idx1 = "one"
set my_idx2 = "two"
set my_idx3 = "three"
set index_format = $my_idx1

macro index i '<enter-command>set index_format =  \
```

```
  (or                                                \
    (if (equal $index_format $my_idx1) $my_idx2)     \
    (if (equal $index_format $my_idx2) $my_idx3)     \
    $my_idx1) \
<enter>'
```

The output of the run command is re-evaluated by the muttrc parser. So it's important to pay more attention to quoting issues when generating the command string below.

```
# Conditionally set up background editing in tmux or GNU Screen:

run \
  (if (or $STY $TMUX)                                \
    (concat                                          \
      'set background_edit;'                         \
      'set editor = "bgedit-screen-tmux.sh vim"') \
    (concat                                          \
      'unset background_edit;'                       \
      'set editor = "vim"'))
```

Because backticks are evaluated by MuttLisp too, we need to use the run command below and pay close attention to quoting.

```
# Use a Mutt variable inside backticks.

set spoolfile = "/var/mail/testuser"

# This will generate and then run the command string:
#   set my_var = "`~/bin/myscript.sh /var/mail/testuser`"
run                                              \
  (concat                                        \
     'set my_var = "`~/bin/myscript.sh '  \
     $spoolfile                                  \
     '`"')
```

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

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.

Using MIME in Mutt

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.

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.

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) --]
```

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-message>` functions) to attachments of type `message/rfc822`.

See table Table 9-7 for all available functions.

Viewing Attachments

There are four(!) ways of viewing attachments, so the functions deserve some extra explanation.

`<view-mailcap>` (default keybinding: m)

This will use the first matching mailcap entry.

If no matching mailcap entries are found, it will abort with an error message.

`<view-attach>` (default keybinding: <Enter>)

Mutt will display internally supported MIME types (see the Section called *Viewing MIME Messages in the Pager*) in the pager. This will respect auto_view settings, to determine whether to use a `copiousoutput` mailcap entry or just directly display the attachment.

Other MIME types will use the first matching mailcap entry.

If no matching mailcap entries are found, the attachment will be displayed in the pager as raw text.

`<view-pager>`

Mutt will use the first matching `copiousoutput` mailcap entry to display the attachment in the pager (regardless of auto_view settings).

If no matching mailcap entries are found, the attachment will be displayed in the pager as raw text.

<view-text> (default keybinding: T)

The attachment will always be displayed in the pager as raw text.

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.

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 way 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 run the command specified in $mime_type_query_command. If that command is not specified, Mutt 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 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 (http://www.iana.org/assignments/media-types/).

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.

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
```

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
```

Advanced Mailcap Usage

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.

Mutt will set the `COLUMNS` environment variable to the width of the pager. Some programs make use of this environment variable automatically. Others provide a command line argument that can use this to set the output width:

```
text/html; lynx -dump -width ${COLUMNS:-80} %s; copiousoutput
```

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.

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.

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.

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
```

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.

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 first for `text/enriched`, followed by `text/plain`, and finally `text/html`.
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.

Generating `multipart/alternative` content is supported via the $send_multipart_alternative quadoption and $send_multipart_alternative_filter filter script. The composed `text/plain` content will be piped to the filter script's stdin. The output from the filter script should be the generated mime type of the content, a blank line, and the content. For example:

```
text/html

<html>
<body>
Content in html format
</body>
</html>
```

A preview of the alternative can be viewed in the compose menu using the functions `<view-alt>` (bound to "v"), `<view-alt-text>` (bound to "Esc v"), `<view-alt-mailcap>` (bound to "V"), and `<view-alt-pager>` (unbound). See the Section called *Viewing Attachments* for a discussion of the differences between these viewing functions.

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 the Section called *Body Caching* in Chapter 6 usually means to download the message just once.

By default, Mutt will not search inside `multipart/alternative` containers. This can be changed via the $count_alternatives configuration variable.

The syntax is:

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

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

The first part of a message or multipart group, if inline, is counted separately than other inline parts. Specify `root` or `R` for *disposition* to count these as attachments. If this first part is of type multipart/alternative, note that its top-level inline parts are also counted via `root` *disposition* (if $count_alternatives is set).

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 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/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.

Entering the command "**unattachments** *" as a command will Clear all attachment settings.

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

General Notes

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
```

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/
```

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”.

STARTTLS

When non-secure URL protocols `imap://`, `pop://`, and `smtp://` are used, the initial connection to the server will be unencrypted. `STARTTLS` can be used to negotiate an encrypted connection after the initial unencrypted connection and exchange.

Two configuration variables control Mutt’s behavior with `STARTTLS`. $ssl_starttls will initiate `STARTTLS` if the server advertises support for it. $ssl_force_tls will always try to initiate it, whether the server advertises support or not.

Mutt *highly recommends* setting $ssl_force_tls unless you need to connect to an unencrypted server. It’s possible for an attacker to spoof interactions during the initial connection and hide support for `STARTTLS`. The only way to prevent these attacks is by forcing `STARTTLS` with the $ssl_force_tls configuration variable.

Tunnel

When connecting through a $tunnel and $tunnel_is_secure is set (the default), Mutt will assume the connection to the server through the pipe is already secured. Mutt will ignore $ssl_starttls and $ssl_force_tls, behaving as if TLS has already been negotiated.

When $tunnel_is_secure is unset, Mutt will respect the values of $ssl_starttls and $ssl_force_tls. It is *highly recommended* to set $ssl_force_tls in this case, to force `STARTTLS` negotiation. Note that doing so will prevent connection to an IMAP server configured for preauthentication (`PREAUTH`). If you use this configuration, it is recommended to use a secure tunnel.

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 the Section called *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 the Section called *Header Caching* and the Section called *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.

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 the Section called *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.

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. A mailbox name with a trailing delimiter (usually "/" or ".") indicates 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).

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).

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 the Section called *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.

OAUTHBEARER Support

Preliminary OAUTH support for IMAP, POP, and SMTP is provided via external scripts.

At least for Gmail, you can use the `oauth2.py` script from Google's gmail-oauth2-tools: https://github.com/google/gmail-oauth2-tools/blob/master/python/oauth2.py

You'll need to get your own oauth client credentials for Gmail here: https://console.developers.google.com/apis/credentials

Then, you'd use `oauth2.py` with `--generate_oauth2_token` to get a refresh token, and configure mutt with:

```
set imap_authenticators="oauthbearer"
set imap_oauth_refresh_command="/path/to/oauth2.py --quiet --user=[email_address]\
    --client_id=[client_id] --client_secret=[client_secret]\
    --refresh_token=[refresh_token]"
```

Substitute pop or smtp for imap in the above example to configure for those.

An alternative script is contrib/mutt_oauth2.py (https://gitlab.com/muttmua/mutt/tree/master/contrib/mutt_oauth2.py) script. For more details see contrib/mutt_oauth2.py.README (https://gitlab.com/muttmua/mutt/tree/master/contrib/mutt_oauth2.py.README).

XOAUTH2 Support

Support for the deprecated XOAUTH2 protocol is also available. To enable this, add "xoauth2" to the $imap_authenticators, $pop_authenticators, or $smtp_authenticators config variables. XOAUTH2 uses the same refresh command configuration variables as OAUTHBEARER: $imap_oauth_refresh_command, $pop_oauth_refresh_command, and $smtp_oauth_refresh_command. Those will need to be set to a script to generate the appropriate XOAUTH2 token.

Managing Multiple Accounts

Usage:

account-hook *regexp 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 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 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.

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).

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, kyotocabinet, lmdb, 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. When pointing to a directory, be sure to create the directory in advance, or Mutt will interpret it as a file to be created.

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.

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.

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.

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.

Note: If the full address contains non-ascii characters, or sequences that require RFC 2047 encoding, Mutt reverts to writing out the normalized "Name <user@host>" form, in order to generate legal output.

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.

Sidebar

Introduction

The Sidebar shows a list of all your mailboxes. The list can be turned on and off, it can be themed and the list style can be configured.

Variables

Table 6-1. Sidebar Variables

Name	Type	Default
sidebar_delim_chars	string	/.
sidebar_divider_char	string	|
sidebar_folder_indent	boolean	no
sidebar_format	string	%B%* %n
sidebar_indent_string	string	(two spaces)
sidebar_new_mail_only	boolean	no
sidebar_next_new_wrap	boolean	no
sidebar_short_path	boolean	no
sidebar_sort_method	enum	unsorted
sidebar_visible	boolean	no
sidebar_width	number	20

Functions

Sidebar adds the following functions to Mutt. By default, none of them are bound to keys.

Table 6-2. Sidebar Functions

Menus	Function	Description
index,pager	<sidebar-next>	Move the highlight to next mailbox
index,pager	<sidebar-next-new>	Move the highlight to next mailbox with new mail
index,pager	<sidebar-open>	Open highlighted mailbox
index,pager	<sidebar-page-down>	Scroll the Sidebar down 1 page
index,pager	<sidebar-page-up>	Scroll the Sidebar up 1 page
index,pager	<sidebar-prev>	Move the highlight to previous mailbox
index,pager	<sidebar-prev-new>	Move the highlight to previous mailbox with new mail
index,pager	<sidebar-toggle-visible>	Make the Sidebar (in)visible

Commands

sidebar_whitelist *mailbox* [*mailbox* ...]

unsidebar_whitelist { * | *mailbox* }

This command specifies mailboxes that will always be displayed in the sidebar, even if $sidebar_new_mail_only is set and the mailbox does not contain new mail.

The “unsidebar_whitelist” command is used to remove a mailbox from the list of whitelisted mailboxes. Use “unsidebar_whitelist *” to remove all mailboxes.

Colors

Table 6-3. Sidebar Colors

Name	Default Color	Description
`sidebar_divider`	default	The dividing line between the Sidebar and the Index/Pager panels
`sidebar_flagged`	default	Mailboxes containing flagged mail
`sidebar_highlight`	underline	Cursor to select a mailbox
`sidebar_indicator`	mutt `indicator`	The mailbox open in the Index panel
`sidebar_new`	default	Mailboxes containing new mail
`sidebar_spoolfile`	default	Mailbox that receives incoming mail

If the `sidebar_indicator` color isn’t set, then the default Mutt indicator color will be used (the color used in the index panel).

Sort

Table 6-4. Sidebar Sort

Sort	Description
`alpha`	Alphabetically by path or label
`count`	Total number of messages
`flagged`	Number of flagged messages
`name`	Alphabetically by path or label
`new`	Number of unread messages
`path`	Alphabetically by path (ignores label)
`unread`	Number of unread messages
`unsorted`	Do not resort the paths

See Also

- Regular Expressions
- Patterns
- Color command

Compressed Folders Feature

Introduction

The Compressed Folder patch allows Mutt to read mailbox files that are compressed. But it isn't limited to compressed files. It works well with encrypted files, too. In fact, if you can create a program/script to convert to and from your format, then Mutt can read it.

The patch adds three hooks to Mutt: `open-hook`, `close-hook` and `append-hook`. They define commands to: uncompress a file; compress a file; append messages to an already compressed file.

There are some examples of both compressed and encrypted files, later. For now, the documentation will just concentrate on compressed files.

Commands

open-hook *`pattern shell-command`*
close-hook *`pattern shell-command`*
append-hook *`pattern shell-command`*

The shell-command must contain two placeholders for filenames: `%f` and `%t`. These represent "from" and "to" filenames. These placeholders should be placed inside single-quotes to prevent unintended shell expansions.

If you need the exact string "%f" or "%t" in your command, simply double up the "%" character, e.g. "%%f" or "%%t".

Table 6-5. Not all Hooks are Required

Open	Close	Append	Effect	Useful if
Open	-	-	Folder is readonly	The folder is just a backup

Open	Close	Append	Effect	Useful if
Open	Close	-	Folder is read/write, but the entire folder must be written if anything is changed	Your compression format doesn't support appending
Open	Close	Append	Folder is read/write and emails can be efficiently added to the end	Your compression format supports appending
Open	-	Append	Folder is readonly, but can be appended to	You want to store emails, but never change them

Note: The command:

- should return a non-zero exit status on failure
- should not delete any files

Read from compressed mailbox

`open-hook regexp shell-command`

If Mutt is unable to open a file, it then looks for `open-hook` that matches the filename.

If your compression program doesn't have a well-defined extension, then you can use `.` as the regexp.

Example 6-3. Example of open-hook

```
open-hook '\.gz$' "gzip -cd '%f' > '%t'"
```

- Mutt finds a file, "example.gz", that it can't read
- Mutt has an `open-hook` whose regexp matches the filename: `\.gz$`
- Mutt uses the command `gzip -cd` to create a temporary file that it *can* read

Write to a compressed mailbox

`close-hook regexp shell-command`

When Mutt has finished with a compressed mail folder, it will look for a matching `close-hook` to recompress the file. This hook is optional.

Note: If the folder has not been modified, the `close-hook` will not be called.

Example 6-4. Example of close-hook

```
close-hook '\.gz$' "gzip -c '%t' > '%f'"
```

- Mutt has finished with a folder, "example.gz", that it opened with `open-hook`
- The folder has been modified
- Mutt has a `close-hook` whose regexp matches the filename: `\.gz$`
- Mutt uses the command `gzip -c` to create a new compressed file

Append to a compressed mailbox

```
append-hook regexp shell-command
```

When Mutt wants to append an email to a compressed mail folder, it will look for a matching `append-hook`. This hook is optional.

Using the `append-hook` will save time, but Mutt won't be able to determine the type of the mail folder inside the compressed file.

Mutt will *assume* the type to be that of the `$mbox_type` variable. Mutt also uses this type for temporary files.

Mutt will only use the `append-hook` for existing files. The `close-hook` will be used for empty, or missing files.

> **Note:** If your command writes to stdout, it is vital that you use `>>` in the "append-hook". If not, data will be lost.

Example 6-5. Example of append-hook

```
append-hook '\.gz$' "gzip -c '%t' >> '%f'"
```

- Mutt wants to append an email to a folder, "example.gz", that it opened with `open-hook`
- Mutt has an `append-hook` whose regexp matches the filename: `\.gz$`
- Mutt knows the mailbox type from the `$mbox` variable
- Mutt uses the command `gzip -c` to append to an existing compressed file

Empty Files

Mutt assumes that an empty file is not compressed. In this situation, unset $save_empty, so that the compressed file will be removed if you delete all of the messages.

Security

Encrypted files are decrypted into temporary files which are stored in the $tmpdir directory. This could be a security risk.

Autocrypt

Mutt can be compiled with Autocrypt support by running `configure` with the `--enable-autocrypt` flag. Autocrypt provides easy to use, passive protection against data collection. Keys are distributed via an `Autocrypt:` header added to emails. It does *not* protect against active adversaries, and so should not be considered a substitute for normal encryption via your keyring, using key signing and the web of trust to verify identities. With an understanding of these limitations, Autocrypt still provides an easy way to minimize cleartext emails sent between common correspondents, without having to explicitly exchange keys. More information can be found at https://autocrypt.org/.

Requirements

Autocrypt requires support for ECC cryptography, and Mutt by default will generate ECC keys. Therefore GnuPG 2.1 or greater is required. Additionally, Mutt's Autocrypt implementation uses GPGME and requires at least version 1.8.0.

Account and peer information is stored in a sqlite3 database, and so Mutt must be configured with the `--with-sqlite3` flag when autocrypt is enabled.

It is highly recommended Mutt be configured `--with-idn` or `--with-idn2` so that Autocrypt can properly deal with international domain names.

While Mutt uses GPGME for Autocrypt, normal keyring operations can still be performed via classic mode (i.e. with $crypt_use_gpgme unset). However, to avoid unnecessary prompts, it is recommended gpg not be configured in `loopback pinentry` mode, and that $pgp_use_gpg_agent remain set (the default).

First Run

To enable Autocrypt, set $autocrypt, and if desired change the value of $autocrypt_dir in your muttrc. The first time Mutt is run after that, you will be prompted to create $autocrypt_dir. Mutt will then automatically create an sqlite3 database and GPG keyring in that directory. Note since these files should be considered private, Mutt will create this directory with mode `700`. If you create the directory manually, you should do the same.

Mutt recommends keeping the $autocrypt_dir directory set differently from your GnuPG keyring directory (e.g. `~/.gnupg`). Keys are automatically imported into the keyring from `Autocrypt:` headers. Compared to standard "web of trust" keys, Autocrypt keys are somewhat ephemeral, and the autocrypt database is used to track when keys change or fall out of use. Having these keys mixed in with your normal keyring will make it more difficult to use features such as $crypt_opportunistic_encrypt and Autocrypt at the same time.

The $autocrypt_dir variable is not designed to be changed while Mutt is running. The database is created (if necessary) and connected to during startup. Changing the variable can result in a situation where Mutt is looking in one place for the database and a different place for the GPG keyring, resulting in strange behavior.

Once the directory, keyring, and database are created, Mutt will ask whether you would like to create an account. In order to use Autocrypt, each sending address needs an account. As a convenience you can create an account during the first run. If you would like to add additional accounts later, this can be done via the `<autocrypt-acct-menu>` function in the index, by default bound to `A`.

Account creation will first ask you for an email address. Next, it will ask whether you want to create a new key or select an existing key. (Note key selection takes place from the $autocrypt_dir keyring, which will normally be empty during first run). Finally, it will ask whether this address should prefer encryption or not. Autocrypt 1.1 allows automatically enabling encryption if *both* sender and receiver have set "prefer encryption". Otherwise, you will need to manually enable autocrypt encryption in the compose menu. For more details, see the compose menu section below.

After optionally creating an account, Mutt will prompt you to scan mailboxes for Autocrypt headers. This step occurs because header cached messages are not re-scanned for Autocrypt headers. Scanning during this step will temporarily disable the header cache while opening each mailbox. If you wish to do this manually later, you can simulate the same thing by unsetting $header_cache and opening a mailbox.

A final technical note: the first run process takes place between reading the muttrc and opening the initial mailbox. Some muttrc files will push macros to be run after opening the mailbox. To prevent this from interfering with the first run prompts, Mutt disables all macros during the first run.

Compose Menu

When enabled, Autocrypt will add a line to the compose menu with two fields: `Autocrypt:` and `Recommendation:`.

The `Autocrypt:` field shows whether the message will be encrypted by Autocrypt when sent. It has two values: `Encrypt` and `Off`. `Encrypt` can be enabled using the `<autocrypt-menu>` function, by default bound to `o`.

The `Recommendation:` field shows the output of the Autocrypt recommendation engine. This can have one of five values:

- `Off` means the engine is disabled. This can happen if the From address doesn't have an autocrypt account, or if the account has been manually disabled.
- `No` means one or more recipients are missing an autocrypt key, or the key found is unusable (i.e. expired, revoked, disabled, invalid, or not usable for encryption.)
- `Discouraged` means a key was found for every recipient, but the engine is not confident the message will be decryptable by the recipient. This can happen if the key hasn't been used recently (compared to their last seen email).

  It can also happen if the key wasn't seen first-hand from the sender. Autocrypt has a feature where recipient keys can be included in group-encrypted emails. This allows you to reply to a conversation where you don't have a key first-hand from one of the other recipients. However, those keys are not trusted as much as from first-hand emails, so the engine warns you with a `Discouraged` status.

- `Available` means a key was found for every recipient, and the engine believes all keys are recent and seen from the recipient first hand. However, either you or one of the recipients chose not to specify "prefer encryption".
- `Yes` is the same as `Available`, with the addition that you and all recipients have specified "prefer encryption". This value will automatically enable encryption, unless you have manually switched it off or enabled regular encryption or signing via the `<pgp-menu>`.

As mentioned above the `<autocrypt-menu>` function, by default bound to `o`, can be used to change the `Encrypt:` field value. `(e)ncrypt` will toggle encryption on. `(c)lear` will toggle encryption off. If either of these are chosen, the field will remain in that state despite what the `Recommendation:` field shows. Lastly, `(a)utomatic` will set the value based on the recommendation engine's output.

Autocrypt encryption defers to normal encryption or signing. *Anything* that enables normal encryption or signing will cause autocrypt encryption to turn off. The only exception is when replying to an autocrypt-encrypted email (i.e. an email decrypted from the $autocrypt_dir keyring). Then, if $autocrypt_reply is *set*, autocrypt mode will be forced on, overriding the settings $crypt_autosign, $crypt_autoencrypt, $crypt_replyencrypt, $crypt_replysign, $crypt_replysignencrypted, and $crypt_opportunistic_encrypt.

When postponing a message, autocrypt will respect $postpone_encrypt, but will use the autocrypt account key to encrypt the message. Be sure to set $postpone_encrypt to ensure postponed messages marked for autocrypt encryption are encrypted.

Account Management

The Autocrypt Account Menu is available from the index via `<autocrypt-acct-menu>`, by default bound to `A`. See Autocrypt Account Menu for the list of functions and their default keybindings.

In this menu, you can create new accounts, delete accounts, toggle an account active/inactive, and toggle the "prefer encryption" flag for an account.

Deleting an account only removes the account from the database. The GPG key is kept, to ensure you still have the ability to read past encrypted emails.

The Autocrypt 1.1 "Setup Message" feature is not available yet, but will be added in the future.

Alternative Key and Keyring Strategies

Mutt by default partitions Autocrypt from normal keyring encryption/signing. It does this by using a separate GPG keyring (in $autocrypt_dir) and creating a new ECC key in that keyring for accounts. There are good reasons for doing this by default. It keeps random keys found inside email headers out of your normal keyring. ECC keys are compact and better suited for email headers. Autocrypt key selection is completely different from "web of trust" key selection, based on last-seen rules as opposed to trust and validity. It also allows Mutt to distinguish Autocrypt encrypted emails from regular encrypted emails, and set the mode appropriately when replying to each type of email.

Still, some users may want to use an existing key from their normal keyring for Autocrypt too. There are two ways this can be accomplished. The *recommended* way is to set $autocrypt_dir to your normal keyring directory (e.g. `~/.gnupg`). During account creation, choosing "(s)elect existing GPG key" will then list and allow selecting your existing key for the new account.

An alternative is to copy your key over to the Autocrypt keyring, but there is a severe downside. Mutt *first* tries to decrypt messages using the Autocrypt keyring, and if that fails tries the normal keyring second. This means all encrypted emails to that key will be decrypted, and have signatures verified from, the Autocrypt keyring. Keys signatures and web of trust from your normal keyring will no longer show up in signatures when decrypting.

For that reason, if you want to use an existing key from your normal keyring, it is recommended to just set $autocrypt_dir to `~/.gnupg`. This allows "web of trust" to show an appropriate signature message for verified messages. Autocrypt header keys will be imported into your keyring, but if you don't want them mixed you should strongly consider using a separate autocrypt key and keyring instead.

Both methods have a couple additional caveats:

- Replying to an Autocrypt decrypted message by default forces Autocrypt mode on. By sharing the same key, all replies will then start in Autocrypt mode, even if a message wasn't sent by one of your Autocrypt peers. $autocrypt_reply can be *unset* to allow manual control of the mode when replying.
- When Mutt creates an account from a GPG key, it exports the public key, base64 encodes it, and stores that value in the sqlite3 database. The value is then used in the Autocrypt header added to outgoing emails. The ECC keys Mutt creates don't change, but if you use external keys that expire, when you resign to extend the expiration you will need to recreate the Autocrypt account using the account menu. Otherwise the Autocrypt header will contain the old expired exported keydata.

Chapter 7. Security Considerations

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.

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.

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 and $tmpdraftdir variables.

Information Leaks

`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.

To prevent these issues, Mutt by default only accepts the `Subject`, `Body`, `Cc`, `In-Reply-To`, and `References` headers. Allowed headers can be adjusted with the **mailto_allow** and **unmailto_allow** commands.

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 the Section called *Secure Use of Mailcap* in Chapter 5.

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

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 a folder-basis using **folder-hook**s:

   ```
   # 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.

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 and headers, please refer to body caching and header caching for details.

Additionally, it may be worth trying some of Mutt's experimental features. $imap_qresync (which requires header caching) can provide a huge speed boost opening mailboxes if your IMAP server supports it. $imap_deflate enables compression, which can also noticeably reduce download time for large mailboxes and messages.

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

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	log debugging output to ~/.muttdebug0 if mutt was compiled with +DEBUG; it can range from -5 to 5 and affects verbosity. A value of 0 disables debugging. A value less than zero disables automatic log file rotation. A value of 2 is recommended for most diagnostics.
-D	print the value of all Mutt variables to stdout
-E	edit the draft (-H) or include (-i) file
-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

Option	Description
-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 [-En] [-F *muttrc*] [-c *address*] [-Hi *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".

An include file passed with -i will be used as the body of the message. When combined with -E, the include file will be directly edited during message composition. The file will be modified regardless of whether the message is sent or aborted.

A draft file passed with -H will be used as the initial header and body for the message. Multipart messages can be used as a draft file, and are processed the same in interactive and batch mode; they are not passed through untouched. For example, encrypted draft files will be decrypted. When combined with -E, the draft file will be updated to the final state of the message after composition, regardless of whether the message is sent, aborted, or even postponed. Note that if the message is sent encrypted or signed, the draft file will be saved that way too.

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
```

Configuration Commands

The following are the commands understood by Mutt:

•

account-hook *regexp 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*
attachments ?
unattachments *

•

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

•

bind *map key function*

•

cd *directory*

•

charset-hook *alias charset*

•

iconv-hook *charset local-charset*

•

color *object* [*attribute* ...] *foreground background*
color { header | body } [*attribute* ...] *foreground background regexp*
color index [*attribute* ...] *foreground background pattern*
color compose *composeobject* [*attribute* ...] *foreground background*
uncolor { index | header | body } { * | *pattern* }

•

crypt-hook *regexp keyid*

•

echo *message*

•

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* }

•

index-format-hook *name [!]pattern format-string*

•

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

•

macro *menu key sequence* [*description*]

•

mailboxes [[-notify | -nonotify] [-poll | -nopoll] [-label *label* | -nolabel] *mailbox*] [...]
unmailboxes { * | *mailbox* }

•

mailto_allow { * | *header-field* }
unmailto_allow { * | *header-field* }

•

mbox-hook *[!]regexp mailbox*

•

message-hook *[!]pattern command*

•

mime_lookup *mimetype* [*mimetype* ...]
unmime_lookup { * | *mimetype* }

•

mono *object attribute*
mono { header | body } *attribute regexp*
mono index *attribute pattern*
mono compose *composeobject attribute*
unmono { index | header | body } { * | *pattern* }

•

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

•

push *string*

•

reply-hook *[!]pattern command*

•

run *MuttLisp*

•

save-hook *[!]pattern mailbox*

•

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

•

send-hook *[!]pattern command*

•

send2-hook *[!]pattern command*

•

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

•

setenv *[?]variable* [*value*]
unsetenv *variable*

•

sidebar_whitelist *mailbox* [*mailbox* ...]

unsidebar_whitelist { * | *mailbox* }

•

source *filename*

•

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

•

subjectrx *pattern replacement*
unsubjectrx { * | *pattern* }

•

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

•

unhook { * | *hook-type* }

Configuration Variables

abort_noattach

Type: quadoption
Default: no

When the body of the message matches $abort_noattach_regexp and there are no attachments, this quadoption controls whether to abort sending the message.

abort_noattach_regexp

Type: regular expression
Default: "attach"

Specifies a regular expression to match against the body of the message, to determine if an attachment was mentioned but mistakenly forgotten. If it matches, $abort_noattach will be consulted to determine if message sending will be aborted.

Like other regular expressions in Mutt, the search is case sensitive if the pattern contains at least one upper case letter, and case insensitive otherwise.

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.

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.

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.

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

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.

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).

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.

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.

askbcc

Type: boolean
Default: no

If *set*, Mutt will prompt you for blind-carbon-copy (Bcc) recipients before editing an outgoing message.

askcc

Type: boolean
Default: no

If *set*, Mutt will prompt you for carbon-copy (Cc) recipients before editing the body of an outgoing message.

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.

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.

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 (if none, falls back to %F)
%e	MIME content-transfer-encoding
%F	filename in content-disposition header (if none, falls back to %f)
%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 (see formatstrings-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.

attach_save_charset_convert

Type: quadoption
Default: ask-yes

When saving received text-type attachments, this quadoption prompts to convert the character set if the encoding of the attachment (or $assumed_charset if none is specified) differs from charset.

attach_save_dir

Type: path
Default: (empty)

The default directory to save attachments from the “attachment” menu. If it doesn’t exist, Mutt will prompt to create the directory before saving.

If the path is invalid (e.g. not a directory, or cannot be chdir’ed to), Mutt will fall back to using the current directory.

attach_sep

Type: string
Default: “`\n`”

The separator to add between attachments when operating (saving, printing, piping, etc) on a list of tagged attachments.

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.

attribution

Type: string (localized)
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.

attribution_locale

Type: string
Default: (empty)

The locale used by `strftime(3)` to format dates in the attribution string. Legal values are the strings your system accepts for the locale environment variable `$LC_TIME`.

This variable is to allow the attribution date format to be customized by recipient or folder using hooks. By default, Mutt will use your locale environment, so there is no need to set this except to override that default.

auto_subscribe

Type: boolean
Default: no

When *set*, Mutt assumes the presence of a List-Post header means the recipient is subscribed to the list. Unless the mailing list is in the "unsubscribe" or "unlist" lists, it will be added to the "subscribe" list. Parsing and checking these things slows header reading down, so this option is disabled by default.

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.

autocrypt

Type: boolean
Default: no

When *set*, enables autocrypt, which provides passive encryption protection with keys exchanged via headers. See "autocryptdoc" for more details. (Autocrypt only)

autocrypt_acct_format

Type: string
Default: "`%4n %-30a %20p %10s`"

This variable describes the format of the "autocrypt account" menu. The following `printf(3)`-style sequences are understood

%a	email address
%k	gpg keyid
%n	current entry number
%p	prefer-encrypt flag
%s	status flag (active/inactive)

(Autocrypt only)

autocrypt_dir

Type: path
Default: "`~/.mutt/autocrypt`"

This variable sets where autocrypt files are stored, including the GPG keyring and sqlite database. See "autocryptdoc" for more details. (Autocrypt only)

autocrypt_reply

Type: boolean
Default: yes

When *set*, replying to an autocrypt email automatically enables autocrypt in the reply. You may want to unset this if you're using the same key for autocrypt as normal web-of-trust, so that autocrypt isn't forced on for all encrypted replies. (Autocrypt only)

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.

background_edit

Type: boolean
Default: no

When *set*, Mutt will run $editor in the background during message composition. A landing page will display, waiting for the $editor to exit. The landing page may be exited, allowing perusal of the mailbox, or even for other messages to be composed. Backgrounded sessions may be returned to via the `<background-compose-menu>` function.

For background editing to work properly, $editor must be set to an editor that does not try to use the Mutt terminal: for example a graphical editor, or a script launching (and waiting for) the editor in another Gnu Screen window.

For more details, see “bgedit” ("Background Editing" in the manual).

background_confirm_quit

Type: boolean
Default: yes

When *set*, if there are any background edit sessions, you will be prompted to confirm exiting Mutt, in addition to the $quit prompt.

background_format

Type: string
Default: “`%10S %7p %s`”

This variable describes the format of the “background compose” menu. The following `printf(3)`-style sequences are understood:

%i	parent message id (for replies and forwarded messages)
%n	the running number on the menu
%p	pid of the $editor process
%r	comma separated list of “To:” recipients
%R	comma separated list of “Cc:” recipients
%s	subject of the message
%S	status of the $editor process: running/finished

beep

Type: boolean
Default: yes

When this variable is *set*, mutt will beep when an error occurs.

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.

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.

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.

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.

browser_abbreviate_mailboxes

Type: boolean
Default: yes

When this variable is *set*, mutt will abbreviate mailbox names in the browser mailbox list, using '~' and '=' shortcuts.

The default "alpha" setting of $sort_browser uses locale-based sorting (using strcoll(3)), which ignores some punctuation. This can lead to some situations where the order doesn't make intuitive sense. In those cases, it may be desirable to *unset* this variable.

browser_sticky_cursor

Type: boolean
Default: yes

When this variable is *set*, the browser will attempt to keep the cursor on the same mailbox when performing various functions. These include moving up a directory, toggling between mailboxes and directory listing, creating/renaming a mailbox, toggling subscribed mailboxes, and entering a new mask.

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
```

(OpenSSL and GnuTLS only)

change_folder_next

Type: boolean
Default: no

When this variable is *set*, the <change-folder> function mailbox suggestion will start at the next folder in your "mailboxes" list, instead of starting at the first folder in the list.

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.

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.

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.

collapse_unread

Type: boolean
Default: yes

When *unset*, Mutt will not collapse a thread if it contains any unread messages.

compose_confirm_detach_first

Type: boolean
Default: yes

When *set*, Mutt will prompt for confirmation when trying to use `<detach-file>` on the first entry in the compose menu. This is to help prevent irreversible loss of the typed message by accidentally hitting 'D' in the menu.

Note: Mutt only prompts for the first entry. It doesn't keep track of which message is the typed message if the entries are reordered, or if the first entry was already deleted.

compose_format

Type: string (localized)
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 (see formatstrings-size)
%v	Mutt version string

See the text describing the $status_format option for more information on how to set $compose_format.

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).

confirmappend

Type: boolean
Default: yes

When *set*, Mutt will prompt for confirmation when appending messages to an existing mailbox.

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.

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.

content_type

Type: string
Default: "`text/plain`"

Sets the default Content-Type for the body of newly composed messages.

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".

copy_decode_weed

Type: boolean
Default: no

Controls whether Mutt will weed headers when invoking the `<decode-copy>` or `<decode-save>` functions.

count_alternatives

Type: boolean

Default: no

When *set*, Mutt will recurse inside multipart/alternatives while performing attachment searching and counting (see attachments).

Traditionally, multipart/alternative parts have simply represented different encodings of the main content of the email. Unfortunately, some mail clients have started to place email attachments inside one of alternatives. Setting this will allow Mutt to find and count matching attachments hidden there, and include them in the index via %X or through ~X pattern matching.

cursor_overlay

Type: boolean
Default: no

When *set*, Mutt will overlay the indicator, tree, sidebar_highlight, and sidebar_indicator colors onto the currently selected line. This will allow `default` colors in those to be overridden, and for attributes to be merged between the layers.

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)

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.

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)

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.

crypt_confirmhook

Type: boolean
Default: yes

If set, then you will be prompted for confirmation of keys when using the *crypt-hook* command. If unset, no such confirmation prompt will be presented. This is generally considered unsafe, especially where typos are concerned.

crypt_opportunistic_encrypt

Type: boolean
Default: no

Setting this variable will cause Mutt to automatically enable and disable encryption, based on whether all message recipient keys can be located by Mutt.

When this option is enabled, Mutt will enable/disable encryption each time the TO, CC, and BCC lists are edited. If $edit_headers is set, Mutt will also do so each time the message is edited.

While this is set, encryption can't be manually enabled/disabled. The pgp or smime menus provide a selection to temporarily disable this option for the current message.

If $crypt_autoencrypt or $crypt_replyencrypt enable encryption for a message, this option will be disabled for that message. It can be manually re-enabled in the pgp or smime menus. (Crypto only)

crypt_opportunistic_encrypt_strong_keys

Type: boolean
Default: no

When set, this modifies the behavior of $crypt_opportunistic_encrypt to only search for "strong keys", that is, keys with full validity according to the web-of-trust algorithm. A key with marginal or no validity will not enable opportunistic encryption.

For S/MIME, the behavior depends on the backend. Classic S/MIME will filter for certificates with the 't' (trusted) flag in the .index file. The GPGME backend will use the same filters as with OpenPGP, and depends on GPGME's logic for assigning the GPGME_VALIDITY_FULL and GPGME_VALIDITY_ULTIMATE validity flag.

crypt_protected_headers_read

Type: boolean
Default: yes

When set, Mutt will display protected headers in the pager, and will update the index and header cache with revised headers. Protected headers are stored inside the encrypted or signed part of an an email, to prevent disclosure or tampering. For more information see https://github.com/autocrypt/protected-headers. Currently Mutt only supports the Subject header.

Encrypted messages using protected headers often substitute the exposed Subject header with a dummy value (see $crypt_protected_headers_subject). Mutt will update its concept of the correct subject **after** the message is opened, i.e. via the `<display-message>` function. If you reply to a message before opening it, Mutt will end up using the dummy Subject header, so be sure to open such a message first. (Crypto only)

crypt_protected_headers_save

Type: boolean
Default: no

When $crypt_protected_headers_read is set, and a message with a protected Subject is opened, Mutt will save the updated Subject into the header cache by default. This allows searching/limiting based on the protected Subject header if the mailbox is re-opened, without having to re-open the message each time. However, for mbox/mh mailbox types, or if header caching is not set up, you would need to re-open the message each time the mailbox was reopened before you could see or search/limit on the protected subject again.

When this variable is set, Mutt additionally saves the protected Subject back **in the clear-text message headers**. This provides better usability, but with the tradeoff of reduced security. The protected Subject header, which may have previously been encrypted, is now stored in clear-text in the message headers. Copying the message elsewhere, via Mutt or external tools, could expose this previously encrypted data. Please make sure you understand the consequences of this before you enable this variable. (Crypto only)

crypt_protected_headers_subject

Type: string

Default: “. . .”

When $crypt_protected_headers_write is set, and the message is marked for encryption, this will be substituted into the Subject field in the message headers. To prevent a subject from being substituted, unset this variable, or set it to the empty string. (Crypto only)

crypt_protected_headers_write

Type: boolean
Default: no

When set, Mutt will generate protected headers for signed and encrypted emails. Protected headers are stored inside the encrypted or signed part of an an email, to prevent disclosure or tampering. For more information see https://github.com/autocrypt/protected-headers. Currently Mutt only supports the Subject header. (Crypto only)

crypt_replyencrypt

Type: boolean
Default: yes

If *set*, automatically PGP or OpenSSL encrypt replies to messages which are encrypted. (Crypto only)

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)

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)

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)

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.

Note that the GPGME backend does not support creating old-style inline (traditional) PGP encrypted or signed messages (see $pgp_autoinline).

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).

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)

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. 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).

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.

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.

This option is ignored for maildir-style mailboxes when $maildir_trash is set.

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.

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.

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.

dotlock_program

Type: path
Default: "`/usr/bin/mutt_dotlock`"

Contains the path of the `mutt_dotlock(1)` binary to be used by mutt.

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.

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.

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.

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.

Although the compose menu may have localized header labels, the labels passed to your editor will be standard RFC 2822 headers, (e.g. To:, Cc:, Subject:). Headers added in your editor must also be RFC 2822 headers, or one of the pseudo headers listed in "edit-header". Mutt will not understand localized header labels, just as it would not when parsing an actual email.

Note that changes made to the References: and Date: headers are ignored for interoperability reasons.

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.

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).

entropy_file

Type: path
Default: (empty)

The file which includes random data that is used to initialize SSL library functions. (OpenSSL only)

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*.

error_history

Type: number
Default: 30

This variable controls the size (in number of strings remembered) of the error messages displayed by mutt. These can be shown with the `<error-history>` function. The history is cleared each time this variable is set.

escape

Type: string
Default: "~"

Escape character to use for functions in the built-in editor.

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*.

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.

Note: $fcc_before_send forces the default (set) behavior of this option.

fcc_before_send

Type: boolean
Default: no

When this variable is *set*, FCCs will occur before sending the message. Before sending, the message cannot be manipulated, so it will be stored the exact same as sent: $fcc_attach and $fcc_clear will be ignored (using their default values).

When *unset*, the default, FCCs will occur after sending. Variables $fcc_attach and $fcc_clear will be respected, allowing it to be stored without attachments or encryption/signing if desired.

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.

Note: $fcc_before_send forces the default (unset) behavior of this option. (PGP only)

See also $pgp_self_encrypt, $smime_self_encrypt.

fcc_delimiter

Type: string
Default: (empty)

When specified, this allows the ability to Fcc to more than one mailbox. The fcc value will be split by this delimiter and Mutt will evaluate each part as a mailbox separately.

See $record, “fcc-hook”, and “fcc-save-hook”.

flag_safe

Type: boolean
Default: no

If set, flagged messages cannot be deleted.

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.

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
%m	number of messages in the mailbox *
%n	number of unread messages in the mailbox *
%N	N if mailbox has new mail, blank otherwise
%s	size in bytes (see formatstrings-size)
%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.

* = can be optionally printed if nonzero

%m, %n, and %N only work for monitored mailboxes. %m requires $mail_check_stats to be set. %n requires $mail_check_stats to be set (except for IMAP mailboxes).

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.

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.

forward_attachments

Type: quadoption
Default: ask-yes

When forwarding inline (i.e. $mime_forward *unset* or answered with “no” and $forward_decode *set*), attachments which cannot be decoded in a reasonable manner will be attached to the newly composed message if this quadoption is *set* or answered with “yes”.

forward_attribution_intro

Type: string (localized)
Default: "`----- Forwarded message from %f -----`"

This is the string that will precede a message which has been forwarded in the main body of a message (when $mime_forward is unset). For a full listing of defined `printf(3)`-like sequences see the section on $index_format. See also $attribution_locale.

forward_attribution_trailer

Type: string (localized)
Default: "`----- End forwarded message -----`"

This is the string that will follow a message which has been forwarded in the main body of a message (when $mime_forward is unset). For a full listing of defined `printf(3)`-like sequences see the section on $index_format. See also $attribution_locale.

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.

forward_decrypt

Type: quadoption
Default: yes

This quadoption controls the handling of encrypted messages when forwarding or attaching a message. When set to or answered "yes", the outer layer of encryption is stripped off.

This variable is used if $mime_forward is *set* and $mime_forward_decode is *unset*. It is also used when attaching a message via `<attach-message>` in the compose menu. (PGP only)

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".

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.

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.

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`.

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".

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.

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.

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. If pointing to a directory, it must be created in advance.

Header caching can greatly improve speed when opening POP, IMAP MH or Maildir folders, see "caching" for details.

header_cache_compress

Type: boolean
Default: yes

When mutt is compiled with qdbm, tokyocabinet, or kyotocabinet 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.

header_cache_pagesize

Type: number (long)
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.

header_color_partial

Type: boolean
Default: no

When *set*, color header regexps behave like color body regexps: color is applied to the exact text matched by the regexp. When *unset*, color is applied to the entire header.

One use of this option might be to apply color to just the header labels.

See “color” for more details.

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.

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.

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.

hide_missing

Type: boolean
Default: yes

When *set*, mutt will not show the presence of missing messages in the thread tree.

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.

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.

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.

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.

history_file

Type: path
Default: "`~/.mutthistory`"

The file in which Mutt will save its history.

Also see $save_history.

history_remove_dups

Type: boolean
Default: no

When *set*, all of the string history will be scanned for duplicates when a new entry is added. Duplicate entries in the $history_file will also be removed when it is periodically compacted.

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.

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.

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: the node’s hostname is first determined by the `uname(3)` function. The domain is then looked up using the `gethostname(2)` and `getaddrinfo(3)` functions. If those calls are unable to determine the domain, the full value returned by uname is used. Optionally, Mutt can be compiled with a fixed domain name in which case a detected one is not used.

Starting in Mutt 2.0, the operations described in the previous paragraph are performed after the muttrc is processed, instead of beforehand. This way, if the DNS operations are creating delays at startup, you can avoid those by manually setting the value in your muttrc.

Also see $use_domain and $hidden_host.

idn_decode

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. (IDN only)

idn_encode

Type: boolean
Default: yes

When *set*, Mutt will encode international domain names using IDN. Unset this if your SMTP server can handle newer (RFC 6531) UTF-8 encoded domains. (IDN only)

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.

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.

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.

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.

imap_condstore

Type: boolean
Default: no

When *set*, mutt will use the CONDSTORE extension (RFC 7162) if advertised by the server. Mutt's current implementation is basic, used only for initial message fetching and flag updates.

For some IMAP servers, enabling this will slightly speed up downloading initial messages. Unfortunately, Gmail is not one those, and displays worse performance when enabled. Your mileage may vary.

imap_deflate

Type: boolean
Default: no

When *set*, mutt will use the COMPRESS=DEFLATE extension (RFC 4978) if advertised by the server.

In general a good compression efficiency can be achieved, which speeds up reading large mailboxes also on fairly good connections.

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.

imap_fetch_chunk_size

Type: number (long)
Default: 0

When set to a value greater than 0, new headers will be downloaded in groups of this many headers per request. If you have a very large mailbox, this might prevent a timeout and disconnect when opening the mailbox, by sending a FETCH per set of this many headers, instead of a single FETCH for all new headers.

imap_headers

Type: string
Default: (empty)

Mutt requests these header fields in addition to the default headers ("Date:", "From:", "Sender:", "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.

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.

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.

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.

imap_login

Type: string
Default: (empty)

Your login name on the IMAP server.

This variable defaults to the value of $imap_user.

imap_oauth_refresh_command

Type: string
Default: (empty)

The command to run to generate an OAUTH refresh token for authorizing your connection to your IMAP server. This command will be run on every connection attempt that uses the OAUTHBEARER authentication mechanism. See “oauth” for details.

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.

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 for user/password pairs on mutt invocation, or if opening the connection is slow.

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.

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.

imap_poll_timeout

Type: number
Default: 15

This variable specifies the maximum amount of time in seconds that mutt will wait for a response when polling IMAP connections for new mail, before timing out and closing the connection. Set to 0 to disable timing out.

imap_qresync

Type: boolean
Default: no

When *set*, mutt will use the QRESYNC extension (RFC 7162) if advertised by the server. Mutt's current implementation is basic, used only for initial message fetching and flag updates.

Note: this feature is currently experimental. If you experience strange behavior, such as duplicate or missing messages please file a bug report to let us know.

imap_reconnect_sleep

Type: number
Default: 5

When mutt fails to reconnect to a lost IMAP server connection, it will sleep this many seconds before trying again.

imap_reconnect_tries

Type: number
Default: 3

When mutt loses its connection with the IMAP server, it will try to reconnect this many times before giving up and closing the connection.

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.

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.

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.

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.

include_encrypted

Type: boolean
Default: no

Controls whether or not Mutt includes separately encrypted attachment contents when replying.

This variable was added to prevent accidental exposure of encrypted contents when replying to an attacker. If a previously encrypted message were attached by the attacker, they could trick an unwary recipient into decrypting and including the message in their reply.

include_onlyfirst

Type: boolean
Default: no

Controls whether or not Mutt includes only the first attachment of the message you are replying.

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, 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.

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). For an explanation of the %? construct, see the $status_format description. 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 (see formatstrings-size)
%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 unprocessed message (may not work with maildir, mh, and 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)
%r	comma separated list of “To:” recipients
%R	comma separated list of “Cc:” recipients
%s	subject of the message
%S	single character status of the message (“N”/“O”/“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	a three character set of message status flags. the first character is new/read/replied flags (“n”/“o”/“r”/“O”/“N”). the second is deleted or encryption flags (“D”/“d”/“S”/“P”/“s”/“K”). the third is either tagged/flagged (“*”/“!”), or one of the characters listed in $to_chars.
%@name@	insert and evaluate format-string from the matching “index-format-hook” command
%{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

Note that for mbox/mmdf, “%l” applies to the unprocessed message, and for maildir/mh, the value comes from the “Lines:” header field when present (the meaning is normally the same). Thus the value depends on the encodings used in the different parts of the message and has little meaning in practice.

“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”, “fcc-save-hook”, and “index-format-hook”.

They are also supported in the configuration variables $attribution, $forward_attribution_intro, $forward_attribution_trailer, $forward_format, $indent_string, $message_format, $pager_format, and $post_indent_string.

ispell

Type: path
Default: “`/usr/bin/hunspell`”

How to invoke ispell (GNU’s spell-checking software).

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.

local_date_header

Type: boolean
Default: yes

If *set*, the date in the Date header of emails that you send will be in your local timezone. If unset a UTC date will be used instead to avoid leaking information about your current location.

mail_check

Type: number
Default: 5

This variable configures how often (in seconds) mutt should look for new mail. Also see the $timeout variable.

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.

mail_check_stats

Type: boolean
Default: no

When *set*, mutt will periodically calculate message statistics of a mailbox while polling for new mail. It will check for unread, flagged, and total message counts. (Note: IMAP mailboxes only support unread and total counts).

Because this operation is more performance intensive, it defaults to *unset*, and has a separate option, $mail_check_stats_interval, to control how often to update these counts.

Message statistics can also be explicitly calculated by invoking the `<check-stats>` function.

mail_check_stats_interval

Type: number
Default: 60

When $mail_check_stats is *set*, this variable configures how often (in seconds) mutt will update message counts.

mailcap_path

Type: string
Default: (empty)

This variable specifies which files to consult when attempting to display MIME bodies not directly supported by Mutt. The default value is generated during startup: see the “mailcap” section of the manual.

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!

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).

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.

maildir_check_cur

Type: boolean
Default: no

If *set*, mutt will poll both the new and cur directories of a maildir folder for new messages. This might be useful if other programs interacting with the folder (e.g. dovecot) are moving new messages to the cur directory. Note that setting this option may slow down polling for new messages in large folders, since mutt has to scan all cur messages.

mark_macro_prefix

Type: string
Default: "'"

Prefix for macros created using mark-message. A new macro automatically generated with *<mark-message>a* will be composed from this prefix and the letter *a*.

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.

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.

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.

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.

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.

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.)

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.

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).

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).

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.

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.

message_id_format

Type: string
Default: "`<%z@%f>`"

This variable describes the format of the Message-ID generated when sending messages. Mutt 2.0 introduced a more compact format, but this variable allows the ability to choose your own format. The value may end in "|" to invoke an external filter. See formatstrings-filters.

Please note that the Message-ID value follows a strict syntax, and you are responsible for ensuring correctness if you change this from the default. In particular, the value must follow the syntax in RFC 5322: "`"<" id-left "@" id-right ">"`". No spaces are allowed, and `id-left` should follow the dot-atom-text syntax in the RFC. The `id-right` should generally be left at %f.

The old Message-ID format can be used by setting this to: "`<%Y%02m%02d%02H%02M%02S.G%c%p@%f>`"

The following `printf(3)`-style sequences are understood:

%c	step counter looping from "A" to "Z"
%d	current day of the month (GMT)
%f	$hostname
%H	current hour using a 24-hour clock (GMT)
%m	current month number (GMT)
%M	current minute of the hour (GMT)
%p	pid of the running mutt process
%r	3 bytes of pseudorandom data encoded in Base64

%S	current second of the minute (GMT)
%x	1 byte of pseudorandom data hex encoded (example: '1b')
%Y	current year using 4 digits (GMT)
%z	4 byte timestamp + 8 bytes of pseudorandom data encoded in Base64

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".

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.

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.

mh_seq_flagged

Type: string
Default: "`flagged`"

The name of the MH sequence used for flagged messages.

mh_seq_replied

Type: string
Default: "`replied`"

The name of the MH sequence used to tag replied messages.

mh_seq_unseen

Type: string
Default: "`unseen`"

The name of the MH sequence used for unseen messages.

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.

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.

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*.

mime_type_query_command

Type: string
Default: (empty)

This specifies a command to run, to determine the mime type of a new attachment when composing a message. Unless $mime_type_query_first is set, this will only be run if the attachment's extension is not found in the mime.types file.

The string may contain a "%s", which will be substituted with the attachment filename. 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 attachment filename to the end of the string.

The command should output a single line containing the attachment's mime type.

Suggested values are "xdg-mime query filetype" or "file -bi".

mime_type_query_first

Type: boolean
Default: no

When *set*, the $mime_type_query_command will be run before the mime.types lookup.

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.

(Mixmaster only)

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. (Mixmaster only)

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.

muttlisp_inline_eval

Type: boolean
Default: no

If *set*, Mutt will evaluate bare parenthesis arguments to commands as MuttLisp expressions.

narrow_tree

Type: boolean
Default: no

This variable, when *set*, makes the thread tree narrower, allowing deeper threads to fit on the screen.

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.

new_mail_command

Type: path
Default: (empty)

If *set*, Mutt will call this command after a new message is received. See the $status_format documentation for the values that can be formatted into this command.

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.

The string may contain a "%s", which will be substituted with the generated message filename. 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 message filename to the end of the string.

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.

When using an external pager, also see $prompt_after which defaults *set*.

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.

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.

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.

pager_skip_quoted_context

Type: number
Default: 0

Determines the number of lines of context to show before the unquoted text when using `<skip-quoted>`. When set to a positive number at most that many lines of the previous quote are displayed. If the previous quote is shorter the whole quote is displayed.

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.

pattern_format

Type: string
Default: "`%2n %-15e  %d`"

This variable describes the format of the "pattern completion" menu. The following `printf(3)`-style sequences are understood:

%d	pattern description
%e	pattern expression
%n	index number

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.

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. The GPGME backend does not support this option.

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)

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)

pgp_check_gpg_decrypt_status_fd

Type: boolean
Default: yes

If *set*, mutt will check the status file descriptor output of $pgp_decrypt_command and $pgp_decode_command for GnuPG status codes indicating successful decryption. This will check for the presence of DECRYPTION_OKAY, absence of DECRYPTION_FAILED, and that all PLAINTEXT occurs between the BEGIN_DECRYPTION and END_DECRYPTION status codes.

If *unset*, mutt will instead match the status fd output against $pgp_decryption_okay. (PGP only)

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)

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 if set, otherwise the value of $pgp_default_key.
%r	One or more key IDs (or fingerprints if available).

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)

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)

pgp_decryption_okay

Type: regular expression
Default: (empty)

If you assign text to this variable, then an encrypted PGP message is only considered successfully decrypted if the output from $pgp_decrypt_command contains the text. This is used to protect against a spoofed encrypted message, with multipart/encrypted headers but containing a block that is not actually encrypted. (e.g. simply signed and ascii armored text).

Note that if $pgp_check_gpg_decrypt_status_fd is set, this variable is ignored. (PGP only)

pgp_default_key

Type: string
Default: (empty)

This is the default key-pair to use for PGP operations. It will be used for encryption (see $postpone_encrypt and $pgp_self_encrypt).

It will also be used for signing unless $pgp_sign_as is set.

The (now deprecated) *pgp_self_encrypt_as* is an alias for this variable, and should no longer be used. (PGP only)

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)

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)

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)

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)

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)

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)

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)

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)

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 --with-fingerprint
```

This format is also generated by the `mutt_pgpring` utility which comes with mutt.

Note: gpg's `fixed-list-mode` option should not be used. It produces a different date format which may result in mutt showing incorrect key generation dates.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. Note that in this case, %r expands to the search string, which is a list of one or more quoted values such as email address, name, or keyid. (PGP only)

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 --with-fingerprint
```

This format is also generated by the `mutt_pgpring` utility which comes with mutt.

Note: gpg's `fixed-list-mode` option should not be used. It produces a different date format which may result in mutt showing incorrect key generation dates.

This is a format string, see the $pgp_decode_command command for possible `printf(3)`-like sequences. Note that in this case, %r expands to the search string, which is a list of one or more quoted values such as email address, name, or keyid. (PGP only)

pgp_long_ids

Type: boolean
Default: yes

If *set*, use 64 bit PGP key IDs, if *unset* use the normal 32 bit key IDs. NOTE: Internally, Mutt has transitioned to using fingerprints (or long key IDs as a fallback). This option now only controls the display of key IDs in the key selection menu and a few other places. (PGP only)

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)

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)

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)

pgp_self_encrypt

Type: boolean
Default: yes

When *set*, PGP encrypted messages will also be encrypted using the key in $pgp_default_key. (PGP only)

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)

pgp_sign_as

Type: string
Default: (empty)

If you have a different key pair to use for signing, you should set this to the signing key. Most people will only need to set $pgp_default_key. It is recommended that you use the keyid form to specify your key (e.g. `0x00112233`). (PGP only)

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)

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)

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)

pgp_timeout

Type: number (long)
Default: 300

The number of seconds after which a cached passphrase will expire if not used. (PGP only)

pgp_use_gpg_agent

Type: boolean
Default: yes

If *set*, mutt expects a `gpg-agent(1)` process will handle private key passphrase prompts. If *unset*, mutt will prompt for the passphrase and pass it via stdin to the pgp command.

Note that as of version 2.1, GnuPG automatically spawns an agent and requires the agent be used for passphrase management. Since that version is increasingly prevalent, this variable now defaults *set*.

Mutt works with a GUI or curses pinentry program. A TTY pinentry should not be used.

If you are using an older version of GnuPG without an agent running, or another encryption program without an agent, you will need to *unset* this variable. (PGP only)

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)

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)

pipe_decode

Type: boolean
Default: no

Used in connection with the `<pipe-message>` function. When *unset*, Mutt will pipe the messages without any preprocessing. When *set*, Mutt will attempt to decode the messages first.

Also see $pipe_decode_weed, which controls whether headers will be weeded when this is *set*.

pipe_decode_weed

Type: boolean
Default: yes

For `<pipe-message>`, when $pipe_decode is set, this further controls whether Mutt will weed headers.

pipe_sep

Type: string
Default: "`\n`"

The separator to add between messages when piping a list of tagged messages to an external Unix command.

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.

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.

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"
```

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.

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.

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.

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.

pop_oauth_refresh_command

Type: string
Default: (empty)

The command to run to generate an OAUTH refresh token for authorizing your connection to your POP server. This command will be run on every connection attempt that uses the OAUTHBEARER authentication mechanism. See "oauth" for details.

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.

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.

pop_user

Type: string
Default: (empty)

Your login name on the POP server.

This variable defaults to your user name on the local machine.

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. For a full listing of defined `printf(3)`-like sequences see the section on $index_format.

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.

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.

postpone_encrypt

Type: boolean
Default: no

When *set*, postponed messages that are marked for encryption will be self-encrypted. Mutt will first try to encrypt using the value specified in $pgp_default_key or $smime_default_key. If those are not set, it will try the deprecated $postpone_encrypt_as. (Crypto only)

postpone_encrypt_as

Type: string
Default: (empty)

This is a deprecated fall-back variable for $postpone_encrypt. Please use $pgp_default_key or $smime_default_key. (Crypto only)

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.

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.

print_command

Type: path

Default: "lpr"

This specifies the command pipe that should be used to print messages.

print_decode

Type: boolean
Default: yes

Used in connection with the <print-message> function. 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.

Also see $print_decode_weed, which controls whether headers will be weeded when this is *set*.

print_decode_weed

Type: boolean
Default: yes

For <print-message>, when $print_decode is set, this further controls whether Mutt will weed headers.

print_split

Type: boolean
Default: no

Used in connection with the <print-message> function. 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.

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.

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.

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.

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.

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.

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.

read_only

Type: boolean
Default: no

If *set*, all folders are opened in read-only mode.

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.

recall

Type: quadoption
Default: ask-yes

Controls whether or not Mutt recalls postponed messages when composing a new message.

Setting this variable to *yes* is not generally useful, and thus not recommended. Note that the `<recall-message>` function can be used to manually recall postponed messages.

Also see $postponed variable.

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. Also see $copy and $write_bcc.

Multiple mailboxes may be specified if $fcc_delimiter is set to a string delimiter.

reflow_space_quotes

Type: boolean
Default: yes

This option controls how quotes from format=flowed messages are displayed in the pager and when replying (with $text_flowed *unset*). When set, this option adds spaces after each level of quote marks, turning ">>>foo" into "> > > foo".

Note: If $reflow_text is *unset*, this option has no effect. Also, this option does not affect replies when $text_flowed is *set*.

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.

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.

reply_regexp

Type: regular expression (localized)
Default: "`^(re)(\\[[0-9]+\\])*:[ \t]*`"

A regular expression used to recognize reply messages when threading and replying. The default value corresponds to the standard Latin "Re:" prefix.

This value may have been localized by the translator for your locale, adding other prefixes that are common in the locale. You can add your own prefixes by appending inside `"^(re)"`. For example: `"^(re|se)"` or `"^(re|aw|se)"`.

The second parenthesized expression matches zero or more bracketed numbers following the prefix, such as `"Re[1]: "`. The initial `"\\["` means a literal left-bracket character. Note the backslash must be doubled when used inside a double quoted string in the muttrc. `"[0-9]+"` means one or more numbers. `"\\]"` means a literal right-bracket. Finally the whole parenthesized expression has a `"*"` suffix, meaning it can occur zero or more times.

The last part matches a colon followed by an optional space or tab. Note `"\t"` is converted to a literal tab inside a double quoted string. If you use a single quoted string, you would have to type an actual tab character, and would need to convert the double-backslashes to single backslashes.

Note: the result of this regexp match against the subject is stored in the header cache. Mutt isn't smart enough to invalidate a header cache entry based on changing $reply_regexp, so if you aren't seeing correct values in the index, try temporarily turning off the header cache. If that fixes the problem, then once the variable is set to your liking, remove your stale header cache files and turn the header cache back on.

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.

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.

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.

resume_draft_files

Type: boolean
Default: no

If *set*, draft files (specified by `-H` on the command line) are processed similarly to when resuming a postponed message. Recipients are not prompted for; send-hooks are not evaluated; no alias expansion takes place; user-defined headers and signatures are not added to the message.

resume_edited_draft_files

Type: boolean
Default: yes

If *set*, draft files previously edited (via `-E -H` on the command line) will have $resume_draft_files automatically set when they are used as a draft file again.

The first time a draft file is saved, mutt will add a header, X-Mutt-Resume-Draft to the saved file. The next time the draft file is read in, if mutt sees the header, it will set $resume_draft_files.

This option is designed to prevent multiple signatures, user-defined headers, and other processing effects from being made multiple times to the draft file.

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.

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 and $reverse_realname.

reverse_realname

Type: boolean
Default: yes

This variable fine-tunes the behavior of the $reverse_name feature.

When it is *unset*, Mutt will remove the real name part of a matching address. This allows the use of the email address without having to also use what the sender put in the real name field.

When it is *set*, Mutt will use the matching address as-is.

In either case, a missing real name will be filled in afterwards using the value of $realname.

rfc2047_parameters

Type: boolean
Default: yes

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.

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.

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.

save_history

Type: number
Default: 0

This variable controls the size of the history (per category) saved in the $history_file file.

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.

send_group_reply_to

Type: boolean
Default: no

This variable controls how group replies are done. When set, all recepients listet in "To:" are set in the "To:" header again, else in the "CC", which is the default.

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.

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.

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".

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.

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.

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.

send_multipart_alternative

Type: quadoption
Default: no

If *set*, Mutt will generate a multipart/alternative container and an alternative part using the filter script specified in $send_multipart_alternative_filter. See the section "MIME Multipart/Alternative" (alternative-order).

Note that enabling multipart/alternative is not compatible with inline PGP encryption. Mutt will prompt to use PGP/MIME in that case.

send_multipart_alternative_filter

Type: path
Default: (empty)

This specifies a filter script, which will convert the main (composed) message of the email to an alternative format. The message will be piped to the filter's stdin. The expected output of the filter is the generated mime type, e.g. text/html, followed by a blank line, and then the converted content. See the section "MIME Multipart/Alternative" (alternative-order).

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. Mutt appends all recipients after adding a `--` delimiter (if not already present). Additional flags, such as for $use_8bitmime, $use_envelope_from, $dsn_notify, or $dsn_return will be added before the delimiter.

Note: This command is invoked differently from most other commands in Mutt. It is tokenized by space, and invoked directly via `execvp(3)` with an array of arguments - so commands or arguments with spaces in them are not supported. The shell is not used to run the command, so shell quoting is also not supported.

See also: $write_bcc.

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.

shell

Type: path
Default: (empty)

Command to use when spawning a subshell. By default, the user's login shell from `/etc/passwd` is used.

sidebar_delim_chars

Type: string
Default: "`/.`"

This contains the list of characters which you would like to treat as folder separators for displaying paths in the sidebar.

Local mail is often arranged in directories: ‘dir1/dir2/mailbox’.

```
set sidebar_delim_chars='/'
```

IMAP mailboxes are often named: ‘folder1.folder2.mailbox’.

```
set sidebar_delim_chars='.'
```

See also: $sidebar_short_path, $sidebar_folder_indent, $sidebar_indent_string.

sidebar_divider_char

Type: string
Default: “`|`”

This specifies the characters to be drawn between the sidebar (when visible) and the other Mutt panels. ASCII and Unicode line-drawing characters are supported.

sidebar_folder_indent

Type: boolean
Default: no

Set this to indent mailboxes in the sidebar.

See also: $sidebar_short_path, $sidebar_indent_string, $sidebar_delim_chars.

sidebar_format

Type: string
Default: “`%B%*  %n`”

This variable allows you to customize the sidebar display. This string is similar to $index_format, but has its own set of `printf(3)`-like sequences:

%B	Name of the mailbox
%S	* Size of mailbox (total number of messages)
%N	* Number of unread messages in the mailbox
%n	N if mailbox has new mail, blank otherwise
%F	* Number of Flagged messages in the mailbox
%!	“!” : one flagged message; “!!” : two flagged messages; “n!” : n flagged messages (for n > 2). Otherwise prints nothing.

%d	* @ Number of deleted messages
%L	* @ Number of messages after limiting
%t	* @ Number of tagged messages
%>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

* = Can be optionally printed if nonzero @ = Only applicable to the current folder

In order to use %S, %N, %F, and %!, $mail_check_stats must be *set*. When thus set, a suggested value for this option is "%B%?F? [%F]?%* %?N?%N/?%S".

sidebar_indent_string

Type: string
Default: “ ”

This specifies the string that is used to indent mailboxes in the sidebar. It defaults to two spaces.

See also: $sidebar_short_path, $sidebar_folder_indent, $sidebar_delim_chars.

sidebar_new_mail_only

Type: boolean
Default: no

When set, the sidebar will only display mailboxes containing new, or flagged, mail.

See also: sidebar_whitelist.

sidebar_next_new_wrap

Type: boolean
Default: no

When set, the `<sidebar-next-new>` command will not stop and the end of the list of mailboxes, but wrap around to the beginning. The `<sidebar-prev-new>` command is similarly affected, wrapping around to the end of the list.

sidebar_relative_shortpath_indent

Type: boolean
Default: no

When set, this option changes how $sidebar_short_path and $sidebar_folder_indent perform shortening and indentation: both will look at the previous sidebar entries and shorten/indent relative to the most recent parent.

An example of this option set/unset for mailboxes listed in this order, with $sidebar_short_path=yes, $sidebar_folder_indent=yes, and $sidebar_indent_string="⟶":

mailbox	set	unset
=a.b	=a.b	⟶b
=a.b.c.d	⟶c.d	⟶⟶⟶d
=a.b.e	⟶e	⟶⟶e

The second line illustrates most clearly. With this option set, =a.b.c.d is shortened relative to =a.b, becoming c.d; it is also indented one place relative to =a.b. With this option unset =a.b.c.d is always shortened to the last part of the mailbox, d and is indented three places, with respect to $folder (represented by '='').

When set, the third line will also be indented and shortened relative to the first line.

sidebar_short_path

Type: boolean
Default: no

By default the sidebar will show the mailbox's path, relative to the $folder variable. Setting sidebar_shortpath=yes will shorten the names relative to the previous name. Here's an example:

shortpath=no	shortpath=yes	shortpath=yes, folderindent=ye indentstr=".."
fruit	fruit	fruit
fruit.apple	apple	..apple
fruit.banana	banana	..banana
fruit.cherry	cherry	..cherry

See also: $sidebar_delim_chars, $sidebar_folder_indent, $sidebar_indent_string.

sidebar_sort_method

Type: sort order
Default: unsorted

Specifies how to sort mailbox entries in the sidebar. By default, the entries are sorted alphabetically. Valid values:

- alpha (alphabetically)
- count (all message count)
- flagged (flagged message count)
- name (alphabetically)
- new (unread message count)
- path (alphabetically)
- unread (unread message count)
- unsorted

You may optionally use the “reverse-” prefix to specify reverse sorting order (example: “`set sidebar_sort_method=reverse-alpha`”).

sidebar_use_mailbox_shortcuts

Type: boolean
Default: no

When set, sidebar mailboxes will be displayed with mailbox shortcut prefixes "=" or "~".

When unset, the sidebar will trim off a matching $folder prefix but otherwise not use mailbox shortcuts.

sidebar_visible

Type: boolean
Default: no

This specifies whether or not to show sidebar. The sidebar shows a list of all your mailboxes.

See also: $sidebar_format, $sidebar_width

sidebar_width

Type: number
Default: 30

This controls the width of the sidebar. It is measured in screen columns. For example: sidebar_width=20 could display 20 ASCII characters, or 10 Chinese characters.

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.

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.

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.

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 modifiers. 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".

size_show_bytes

Type: boolean
Default: no

If *set*, message sizes will display bytes for values less than 1 kilobyte. See formatstrings-size.

size_show_fractions

Type: boolean
Default: yes

If *set*, message sizes will be displayed with a single decimal value for sizes from 0 to 10 kilobytes and 1 to 10 megabytes. See formatstrings-size.

size_show_mb

Type: boolean
Default: yes

If *set*, message sizes will display megabytes for values greater than or equal to 1 megabyte. See formatstrings-size.

size_units_on_left

Type: boolean
Default: no

If *set*, message sizes units will be displayed to the left of the number. See formatstrings-size.

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.

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.

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.

pgp_mime_signature_filename

Type: string
Default: "`signature.asc`"

This option sets the filename used for signature parts in PGP/MIME signed messages.

pgp_mime_signature_description

Type: string
Default: "`Digital signature`"

This option sets the Content-Description used for signature parts in PGP/MIME signed messages.

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)

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)

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)

smime_decrypt_command

Type: string
Default: (empty)

This format string specifies a command which is used to decrypt `application/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.
%d	The message digest algorithm specified with $smime_sign_digest_alg.
%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)

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)

smime_default_key

Type: string
Default: (empty)

This is the default key-pair to use for S/MIME operations, and must be set to the keyid (the hash-value that OpenSSL generates) to work properly.

It will be used for encryption (see $postpone_encrypt and $smime_self_encrypt). If GPGME is enabled, this is the key id displayed by gpgsm.

It will be used for decryption unless $smime_decrypt_use_default_key is *unset*.

It will also be used for signing unless $smime_sign_as is set.

The (now deprecated) *smime_self_encrypt_as* is an alias for this variable, and should no longer be used. (S/MIME only)

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)

smime_encrypt_with

Type: string
Default: “`aes256`”

This sets the algorithm that should be used for encryption. Valid choices are “aes128”, “aes192”, “aes256”, “des”, “des3”, “rc2-40”, “rc2-64”, “rc2-128”. (S/MIME only)

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)

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)

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)

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)

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)

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)

smime_pkcs7_default_smime_type

Type: string
Default: "`signed`"

The application/pkcs7-mime ".p7m" type can contain EnvelopedData (encrypted) or SignedData. Senders should add a "smime-type" parameter to the content type, to help receiving MUAs correctly handle the data. Unfortunately, some clients (e.g. Outlook) don't add this parameter.

This option is used to determine which type to assume when the "smime-type" parameter is missing for ".p7m" file types.

Accepted values are "enveloped" and "signed".

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)

smime_self_encrypt

Type: boolean
Default: yes

When *set*, S/MIME encrypted messages will also be encrypted using the certificate in $smime_default_key. (S/MIME only)

smime_sign_as

Type: string
Default: (empty)

If you have a separate key to use for signing, you should set this to the signing key. Most people will only need to set $smime_default_key. (S/MIME only)

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. NOTE: %c and %k will default to $smime_sign_as if set, otherwise $smime_default_key. (S/MIME only)

smime_sign_digest_alg

Type: string
Default: "`sha256`"

This sets the algorithm that should be used for the signature message digest. Valid choices are "md5", "sha1", "sha224", "sha256", "sha384", "sha512". (S/MIME only)

smime_sign_opaque_command

Type: string
Default: (empty)

This command is used to created S/MIME signatures of type `application/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)

smime_timeout

Type: number (long)
Default: 300

The number of seconds after which a cached passphrase will expire if not used. (S/MIME only)

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)

smime_verify_opaque_command

Type: string
Default: (empty)

This command is used to verify S/MIME signatures of type `application/pkcs7-mime`.

This is a format string, see the $smime_decrypt_command command for possible `printf(3)`-like sequences. (S/MIME only)

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"
```

smtp_oauth_refresh_command

Type: string
Default: (empty)

The command to run to generate an OAUTH refresh token for authorizing your connection to your SMTP server. This command will be run on every connection attempt that uses the OAUTHBEARER authentication mechanism. See “oauth” for details.

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.

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.

Also see $write_bcc.

socket_receive_timeout

Type: number
Default: 0

Causes Mutt to timeout any socket read operation (e.g. SSL_read) after this many seconds. A zero (default) or negative value causes Mutt to wait indefinitely for the read to complete.

socket_send_timeout

Type: number
Default: 0

Causes Mutt to timeout any socket write operation (e.g. SSL_write) after this many seconds. A zero (default) or negative value causes Mutt to wait indefinitely for the write to complete.

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`”).

For values except “threads”, this provides the primary sort method. When two message sort values are equal, $sort_aux will be used for a secondary sort.

When set to “threads”, Mutt threads messages in the index. It uses the variable $sort_thread_groups to sort between threads (at the top/root level), and $sort_aux to sort sub-threads and children.

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)

sort_aux

Type: sort order
Default: date

For non-threaded mode, this provides a secondary sort for messages in the “index” menu, used when the $sort value is equal for two messages.

When sorting by threads, this variable controls 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 sub-thread, that sub-thread becomes the last one displayed.

Note: For reversed-threads $sort order, $sort_aux is reversed again (which is not the right thing to do, but kept to not break any existing configuration setting).

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)
- count
- date
- size
- unread
- unsorted

You may optionally use the "reverse-" prefix to specify reverse sorting order (example: "`set sort_browser=reverse-date`").

sort_browser_mailboxes

Type: sort order
Default: unsorted

Specifies how to sort entries in the mailbox browser. By default, the entries are unsorted, displayed in the same order as listed in the "mailboxes" command. Valid values:

- alpha (alphabetically)
- count
- date
- size
- unread
- unsorted

You may optionally use the "reverse-" prefix to specify reverse sorting order (example: "`set sort_browser_mailboxes=reverse-alpha`").

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.

sort_thread_groups

Type: sort order
Default: aux

When sorting by threads, this variable controls how threads are sorted in relation to other threads (at the top/root level). This can be set to any value that $sort can, except “threads”. 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_thread_groups as an ordering.

For backward compatibility, the default value is “aux”, which means to use $sort_aux for top-level thread sorting too. The value “aux” does not respect “last-” or “reverse-” prefixes, it simply delegates sorting directly to $sort_aux.

Note: For reversed-threads $sort order, $sort_thread_groups is reversed again (which is not the right thing to do, but kept to not break any existing configuration setting).

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.

spoolfile

Type: path
Default: (empty)

If your default mailbox or spool file 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.

Note: Despite the name, this can refer to a local or remote mailbox, e.g., "+INBOX".

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. (GnuTLS only)

Example:

```
set ssl_ca_certificates_file=/etc/ssl/certs/ca-certificates.crt
```

ssl_client_cert

Type: path
Default: (empty)

The file containing a client certificate and its associated private key.

ssl_force_tls

Type: boolean
Default: yes

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.

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. (GnuTLS only)

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.

Note that `STARTTLS` is subject to many kinds of attacks, including the ability of a machine-in-the-middle to suppress the advertising of support. Setting $ssl_force_tls is recommended if you rely on `STARTTLS`.

ssl_use_sslv2

Type: boolean
Default: no

If *set* , Mutt will use SSLv2 when communicating with servers that request it. **N.B. As of 2011, SSLv2 is considered insecure, and using is inadvisable. See https://tools.ietf.org/html/rfc6176 .** (OpenSSL only)

ssl_use_sslv3

Type: boolean
Default: no

If *set* , Mutt will use SSLv3 when communicating with servers that request it. **N.B. As of 2015, SSLv3 is considered insecure, and using it is inadvisable. See https://tools.ietf.org/html/rfc7525 .**

ssl_use_tlsv1

Type: boolean
Default: no

If *set* , Mutt will use TLSv1.0 when communicating with servers that request it. **N.B. As of 2015, TLSv1.0 is considered insecure, and using it is inadvisable. See https://tools.ietf.org/html/rfc7525 .**

ssl_use_tlsv1_1

Type: boolean
Default: no

If *set* , Mutt will use TLSv1.1 when communicating with servers that request it. **N.B. As of 2015, TLSv1.1 is considered insecure, and using it is inadvisable. See https://tools.ietf.org/html/rfc7525 .**

ssl_use_tlsv1_2

Type: boolean
Default: yes

If *set* , Mutt will use TLSv1.2 when communicating with servers that request it.

ssl_use_tlsv1_3

Type: boolean

Default: yes

If *set* , Mutt will use TLSv1.3 when communicating with servers that request it.

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. (OpenSSL only)

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.

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.

ssl_verify_host_override

Type: string
Default: (empty)

Defines an alternate host name to verify the server certificate against. This should not be set unless you are sure what you are doing, but it might be useful for connection to a .onion host without a properly configured host name in the certificate. See $ssl_verify_host.

ssl_verify_partial_chains

Type: boolean
Default: no

This option should not be changed from the default unless you understand what you are doing.

Setting this variable to *yes* will permit verifying partial certification chains, i. e. a certificate chain where not the root, but an intermediate certificate CA, or the host certificate, are marked trusted (in $certificate_file), without marking the root signing CA as trusted.

(OpenSSL 1.0.2b and newer only).

ssl_ciphers

Type: string
Default: (empty)

Contains a colon-separated list of ciphers to use when using SSL. For OpenSSL, see ciphers(1) for the syntax of the string.

For GnuTLS, this option will be used in place of "NORMAL" at the start of the priority string. See gnutls_priority_init(3) for the syntax and more details. (Note: GnuTLS version 2.1.7 or higher is required.)

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).

status_format

Type: string (localized)
Default: "`-%r-Mutt: %f [Msgs:%?M?%M/?%m%?n? New:%n?%?o? Old:%o?%?d? Del:%d?%?F? Flag:%F?%?t? T`

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 *
%B	number of backgrounded editing sessions *
%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 (see formatstrings-size) *
%L	size (in bytes) of the messages shown (i.e., which match the current limit) (see formatstrings-size) *
%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 $status_chars
%R	number of read messages *
%s	current sorting mode ($sort)
%S	current aux sorting method ($sort_aux)
%t	number of tagged messages *
%T	current thread group sorting method ($sort_thread_groups) *
%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.

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.

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.

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`".

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.

The option only controls newly composed messages. Postponed messages, resent messages, and draft messages (via -H on the command line) will use the content-type of the source message.

Note that $indent_string is ignored when this option is *set*.

thorough_search

Type: boolean
Default: yes

Affects the `~b`, `~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.

thread_received

Type: boolean
Default: no

When *set*, mutt uses the date received rather than the date sent to thread messages by subject.

tilde

Type: boolean
Default: no

When *set*, the internal-pager will pad blank lines to the bottom of the screen with a tilde ("~").

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.

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.

tmpdir

Type: path
Default: (empty)

This variable allows you to specify where Mutt will place its temporary files needed for displaying messages. If this variable is not set, the environment variable `$TMPDIR` is used. If `$TMPDIR` is not set then “`/tmp`” is used.

tmpdraftdir

Type: path
Default: (empty)

This variable allows you to specify where Mutt will place its temporary files when composing messages. If this variable is not set, the environment variable `$TMPDIR` is used. If `$TMPDIR` is not set then “`/var/tmp`” is used.

It’s suggested that this be set to a directory whose contents won’t be removed during an unanticipated reboot, so that the files will survive a crash or other unplanned computer shutdown.

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.

trash

Type: path
Default: (empty)

If set, this variable specifies the path of the trash folder where the mails marked for deletion will be moved, instead of being irremediably purged.

NOTE: When you delete a message in the trash folder, it is really deleted, so that you have a way to clean the trash.

ts_icon_format

Type: string (localized)
Default: "`M%?n?AIL&ail?`"

Controls the format of the icon title, as long as "$ts_enabled" is set. This string is identical in formatting to the one used by "$status_format".

ts_enabled

Type: boolean
Default: no

Controls whether mutt tries to set the terminal status line and icon name. Most terminal emulators emulate the status line in the window title.

ts_status_format

Type: string (localized)
Default: "`Mutt with %?m?%m messages&no messages?%?n? [%n NEW]?`"

Controls the format of the terminal status line (or window title), provided that "$ts_enabled" has been set. This string is identical in formatting to the one used by "$status_format".

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.

tunnel_is_secure

Type: boolean
Default: yes

When *set*, Mutt will assume the $tunnel connection does not need STARTTLS to be enabled. It will also allow IMAP PREAUTH server responses inside a tunnel to proceed. This is appropriate if $tunnel uses ssh or directly invokes the server locally.

When *unset*, Mutt will negotiate STARTTLS according to the ssl_starttls and ssl_force_tls variables. If ssl_force_tls is set, Mutt will abort connecting if an IMAP server responds with PREAUTH. This setting is appropriate if $tunnel does not provide security and could be tampered with by attackers.

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.

uncollapse_new

Type: boolean
Default: yes

When *set*, Mutt will automatically uncollapse any collapsed thread that receives a newly delivered message. When *unset*, collapsed threads will remain collapsed. The presence of the newly delivered message will still affect index sorting, though.

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.

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.

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.

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.

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.

user_agent

Type: boolean
Default: no

When *set*, mutt will add a "User-Agent:" header to outgoing messages, indicating which version of mutt was used for composing them.

visual

Type: path
Default: (empty)

Specifies the visual editor to invoke when the "`~v`" command is given in the built-in editor.

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>`, `<print-entry>`, and `<filter-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.

weed

Type: boolean
Default: yes

When *set*, mutt will weed headers when displaying, forwarding, or replying to messages.

Also see $copy_decode_weed, $pipe_decode_weed, $print_decode_weed.

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.

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**.

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.

wrapmargin

Type: number
Default: 0

(DEPRECATED) Equivalent to setting $wrap with a negative value.

write_bcc

Type: boolean
Default: no

Controls whether mutt writes out the "Bcc:" header when preparing messages to be sent. Some MTAs, such as Exim and Courier, do not strip the "Bcc:" header; so it is advisable to leave this unset unless you have a particular need for the header to be in the sent message.

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.

Note this option only affects the sending of messages. Fcc'ed copies of a message will always contain the "Bcc:" header if one exists.

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.

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.

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
<bottom-page>	L	move to the bottom of the page
<check-stats>		calculate message statistics for all mailboxes
<current-bottom>		move entry to bottom of screen
<current-middle>		move entry to middle of screen
<current-top>		move entry to top of screen
<end-cond>		end of conditional execution (noop)
<enter-command>	:	enter a muttrc command
<error-history>		display recent history of error messages
<exit>	q	exit this menu
<first-entry>	<Home>	move to the first entry

Function	Default key	Description
<first-entry>	=	move to the first entry
<half-down>	]	scroll down 1/2 page
<half-up>	[	scroll up 1/2 page
<help>	?	this screen
<jump>	1	jump to an index number
<jump>	2	jump to an index number
<jump>	3	jump to an index number
<jump>	4	jump to an index number
<jump>	5	jump to an index number
<jump>	6	jump to an index number
<jump>	7	jump to an index number
<jump>	8	jump to an index number
<jump>	9	jump to an index number
<last-entry>	<End>	move to the last entry
<last-entry>	*	move to the last entry
<middle-page>	M	move to the middle of the page
<next-entry>	<Down>	move to the next entry
<next-entry>	j	move to the next entry
<next-line>	>	scroll down one line
<next-page>	<Pagedown>	move to the next page
<next-page>	<Right>	move to the next page
<next-page>	z	move to the next page
<previous-entry>	<Up>	move to the previous entry
<previous-entry>	k	move to the previous entry
<previous-line>	<	scroll up one line
<previous-page>	<Left>	move to the previous page
<previous-page>	<Pageup>	move to the previous page
<previous-page>	Z	move to the previous page
<refresh>	^L	clear and redraw the screen
<search>	/	search for a regular expression
<search-next>	n	search for next match
<search-opposite>		search for next match in opposite direction
<search-reverse>	Esc /	search backwards for a regular expression
<select-entry>	<Enter>	select the current entry
<select-entry>	<Keypadenter>	select the current entry
<select-entry>	<Return>	select the current entry
<shell-escape>	!	invoke a command in a subshell

Function	Default key	Description
<tag-entry>	t	tag the current entry
<tag-prefix>	;	apply next function to tagged messages
<tag-prefix-cond>		apply next function ONLY to tagged messages
<top-page>	H	move to the top of the page
<what-key>		display the keycode for a key press

Index Menu

Table 9-3. Default Index Menu Bindings

Function	Default key	Description
<autocrypt-acct-menu>	A	manage autocrypt accounts
<background-compose-menu>	B	list and select backgrounded compose sessions
<bounce-message>	b	remail a message to another user
<break-thread>	#	break the thread in two
<browse-mailboxes>	y	select a new mailbox from the browser
<browse-mailboxes-readonly>		select a new mailbox from the browser in read only mode
<buffy-list>	.	list mailboxes with new mail
<change-folder>	c	open a different folder
<change-folder-readonly>	Esc c	open a different folder in read only mode
<check-traditional-pgp>	Esc P	check for classic PGP
<clear-flag>	W	clear a status flag from a message
<close-all-threads>		collapse all threads
<close-thread>		collapse current thread
<collapse-all>	Esc V	collapse/uncollapse all threads
<collapse-thread>	Esc v	collapse/uncollapse current thread
<compose-to-sender>		compose new message to the current message sender
<copy-message>	C	copy a message to a file/mailbox
<create-alias>	a	create an alias from a message sender

Function	Default key	Description
<decode-copy>	Esc C	make decoded (text/plain) copy
<decode-save>	Esc s	make decoded copy (text/plain) and delete
<decrypt-copy>		make decrypted copy
<decrypt-save>		make decrypted copy and delete
<delete-message>	d	delete the current entry
<delete-pattern>	D	delete messages matching a pattern
<delete-subthread>	Esc d	delete all messages in subthread
<delete-thread>	^D	delete all messages in thread
<display-address>	@	display full address of sender
<display-message>	<Enter>	display a message
<display-message>	<Keypadenter>	display a message
<display-message>	<Return>	display a message
<display-message>	<Space>	display a message
<display-toggle-weed>	h	display message and toggle header weeding
<edit>	e	edit the raw message
<edit-label>	Y	add, change, or delete a message's label
<edit-type>	^E	edit attachment content type
<exit>	x	exit this menu
<extract-keys>	^K	extract supported public keys
<fetch-mail>	G	retrieve mail from POP server
<flag-message>	F	toggle a message's 'important' flag
<forget-passphrase>	^F	wipe passphrase(s) from memory
<forward-message>	f	forward a message with comments
<group-chat-reply>		reply to all recipients preserving To/Cc
<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
<limit>	l	show only messages matching a pattern
<link-threads>	&	link tagged message to the current one

Function	Default key	Description
<list-action>	Esc L	perform mailing list action
<list-reply>	L	reply to specified mailing list
<mail>	m	compose a new mail message
<mail-key>	Esc k	mail a PGP public key
<mark-message>	~	create a hotkey macro for the current message
<next-entry>	J	move to the next entry
<next-new>		jump to the next new message
<next-new-then-unread>	<Tab>	jump to the next new or unread message
<next-subthread>	Esc n	jump to the next subthread
<next-thread>	^N	jump to the next thread
<next-undeleted>	<Down>	move to the next undeleted message
<next-undeleted>	j	move to the next undeleted message
<next-unread>		jump to the next unread message
<next-unread-mailbox>		open next mailbox with new mail
<open-all-threads>		uncollapse all threads
<open-thread>		uncollapse current thread
<parent-message>	P	jump to parent message in thread
<pipe-message>	|	pipe message/attachment to a shell command
<previous-entry>	K	move to the previous entry
<previous-new>		jump to the previous new message
<previous-new-then-unread>	Esc <Tab>	jump to the previous new or unread message
<previous-subthread>	Esc p	jump to previous subthread
<previous-thread>	^P	jump to previous thread
<previous-undeleted>	<Up>	move to the previous undeleted message
<previous-undeleted>	k	move to the previous undeleted message
<previous-unread>		jump to the previous unread message
<print-message>	p	print the current entry
<purge-message>		delete the current entry, bypassing the trash folder

Function	Default key	Description
<query>	Q	query external program for addresses
<quit>	q	save changes to mailbox and quit
<read-subthread>	Esc r	mark the current subthread as read
<read-thread>	^R	mark the current thread as read
<recall-message>	R	recall a postponed message
<reply>	r	reply to a message
<resend-message>	Esc e	use the current message as a template for a new one
<root-message>		jump to root message in thread
<save-message>	s	save message/attachment to a mailbox/file
<set-flag>	w	set a status flag on a message
<show-limit>	Esc l	show currently active limit pattern
<show-version>	V	show the Mutt version number and date
<sidebar-first>		move the highlight to the first mailbox
<sidebar-last>		move the highlight to the last mailbox
<sidebar-next>		move the highlight to next mailbox
<sidebar-next-new>		move the highlight to next mailbox with new mail
<sidebar-open>		open highlighted mailbox
<sidebar-page-down>		scroll the sidebar down 1 page
<sidebar-page-up>		scroll the sidebar up 1 page
<sidebar-prev>		move the highlight to previous mailbox
<sidebar-prev-new>		move the highlight to previous mailbox with new mail
<sidebar-toggle-visible>		make the sidebar (in)visible
<sort-mailbox>	o	sort messages
<sort-reverse>	O	sort messages in reverse order
<sync-mailbox>	$	save changes to mailbox
<tag-pattern>	T	tag messages matching a pattern
<tag-subthread>		tag the current subthread
<tag-thread>	Esc t	tag the current thread

Function	Default key	Description
<toggle-new>	N	toggle a message's 'new' flag
<toggle-write>	%	toggle whether the mailbox will be rewritten
<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
<untag-pattern>	^T	untag messages matching a pattern
<view-attachments>	v	show MIME attachments

Pager Menu

Table 9-4. Default Pager Menu Bindings

Function	Default key	Description
<background-compose-menu>	B	list and select backgrounded compose sessions
<bottom>	<End>	jump to the bottom of the message
<bounce-message>	b	remail a message to another user
<break-thread>	#	break the thread in two
<browse-mailboxes>	y	select a new mailbox from the browser
<browse-mailboxes-readonly>		select a new mailbox from the browser in read only mode
<buffy-list>	.	list mailboxes with new mail
<change-folder>	c	open a different folder
<change-folder-readonly>	Esc c	open a different folder in read only mode
<check-stats>		calculate message statistics for all mailboxes
<check-traditional-pgp>	Esc P	check for classic PGP
<clear-flag>	W	clear a status flag from a message
<compose-to-sender>		compose new message to the current message sender
<copy-message>	C	copy a message to a file/mailbox

Function	Default key	Description
<create-alias>	a	create an alias from a message sender
<decode-copy>	Esc C	make decoded (text/plain) copy
<decode-save>	Esc s	make decoded copy (text/plain) and delete
<decrypt-copy>		make decrypted copy
<decrypt-save>		make decrypted copy and delete
<delete-message>	d	delete the current entry
<delete-subthread>	Esc d	delete all messages in subthread
<delete-thread>	^D	delete all messages in thread
<display-address>	@	display full address of sender
<display-toggle-weed>	h	display message and toggle header weeding
<edit>	e	edit the raw message
<edit-label>	Y	add, change, or delete a message's label
<edit-type>	^E	edit attachment content type
<enter-command>	:	enter a muttrc command
<error-history>		display recent history of error messages
<exit>	i	exit this menu
<exit>	q	exit this menu
<exit>	x	exit this menu
<extract-keys>	^K	extract supported public keys
<flag-message>	F	toggle a message's 'important' flag
<forget-passphrase>	^F	wipe passphrase(s) from memory
<forward-message>	f	forward a message with comments
<group-chat-reply>		reply to all recipients preserving To/Cc
<group-reply>	g	reply to all recipients
<half-down>		scroll down 1/2 page
<half-up>		scroll up 1/2 page
<help>	?	this screen
<imap-fetch-mail>		force retrieval of mail from IMAP server
<imap-logout-all>		logout from all IMAP servers
<jump>	1	jump to an index number

Function	Default key	Description
<jump>	2	jump to an index number
<jump>	3	jump to an index number
<jump>	4	jump to an index number
<jump>	5	jump to an index number
<jump>	6	jump to an index number
<jump>	7	jump to an index number
<jump>	8	jump to an index number
<jump>	9	jump to an index number
<link-threads>	&	link tagged message to the current one
<list-action>	Esc L	perform mailing list action
<list-reply>	L	reply to specified mailing list
<mail>	m	compose a new mail message
<mail-key>	Esc k	mail a PGP public key
<mark-as-new>	N	toggle a message's 'new' flag
<next-entry>	J	move to the next entry
<next-line>	<Enter>	scroll down one line
<next-line>	<Keypadenter>	scroll down one line
<next-line>	<Return>	scroll down one line
<next-new>		jump to the next new message
<next-new-then-unread>	<Tab>	jump to the next new or unread message
<next-page>	<Pagedown>	move to the next page
<next-page>	<Space>	move to the next page
<next-subthread>	Esc n	jump to the next subthread
<next-thread>	^N	jump to the next thread
<next-undeleted>	<Down>	move to the next undeleted message
<next-undeleted>	<Right>	move to the next undeleted message
<next-undeleted>	j	move to the next undeleted message
<next-unread>		jump to the next unread message
<next-unread-mailbox>		open next mailbox with new mail
<parent-message>	P	jump to parent message in thread
<pipe-message>	|	pipe message/attachment to a shell command
<previous-entry>	K	move to the previous entry
<previous-line>	<Backspace>	scroll up one line

Function	Default key	Description
<previous-new>		jump to the previous new message
<previous-new-then-unread>		jump to the previous new or unread message
<previous-page>	<Pageup>	move to the previous page
<previous-page>	-	move to the previous page
<previous-subthread>	Esc p	jump to previous subthread
<previous-thread>	^P	jump to previous thread
<previous-undeleted>	<Left>	move to the previous undeleted message
<previous-undeleted>	<Up>	move to the previous undeleted message
<previous-undeleted>	k	move to the previous undeleted message
<previous-unread>		jump to the previous unread message
<print-message>	p	print the current entry
<purge-message>		delete the current entry, bypassing the trash folder
<quit>	Q	save changes to mailbox and quit
<read-subthread>	Esc r	mark the current subthread as read
<read-thread>	^R	mark the current thread as read
<recall-message>	R	recall a postponed message
<redraw-screen>	^L	clear and redraw the screen
<reply>	r	reply to a message
<resend-message>	Esc e	use the current message as a template for a new one
<root-message>		jump to root message in thread
<save-message>	s	save message/attachment to a mailbox/file
<search>	/	search for a regular expression
<search-next>	n	search for next match
<search-opposite>		search for next match in opposite direction
<search-reverse>	Esc /	search backwards for a regular expression
<search-toggle>	\\	toggle search pattern coloring
<set-flag>	w	set a status flag on a message
<shell-escape>	!	invoke a command in a subshell

Function	Default key	Description
<show-version>	V	show the Mutt version number and date
<sidebar-first>		move the highlight to the first mailbox
<sidebar-last>		move the highlight to the last mailbox
<sidebar-next>		move the highlight to next mailbox
<sidebar-next-new>		move the highlight to next mailbox with new mail
<sidebar-open>		open highlighted mailbox
<sidebar-page-down>		scroll the sidebar down 1 page
<sidebar-page-up>		scroll the sidebar up 1 page
<sidebar-prev>		move the highlight to previous mailbox
<sidebar-prev-new>		move the highlight to previous mailbox with new mail
<sidebar-toggle-visible>		make the sidebar (in)visible
<skip-headers>	H	skip beyond headers
<skip-quoted>	S	skip beyond quoted text
<sort-mailbox>	o	sort messages
<sort-reverse>	O	sort messages in reverse order
<sync-mailbox>	$	save changes to mailbox
<tag-message>	t	tag the current entry
<toggle-quoted>	T	toggle display of quoted text
<toggle-write>	%	toggle whether the mailbox will be rewritten
<top>	<Home>	jump to the top of the message
<top>	^	jump to the top of the message
<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
<what-key>		display the keycode for a key press

Alias Menu

Table 9-5. Default Alias Menu Bindings

Function	Default key	Description
<delete-entry>	d	delete the current entry
<tag-entry>	<Space>	tag the current entry
<undelete-entry>	u	undelete the current entry

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

Attachment Menu

Table 9-7. Default Attachment Menu Bindings

Function	Default key	Description
<bounce-message>	b	remail a message to another user
<check-traditional-pgp>	Esc P	check for classic PGP
<collapse-parts>	v	Toggle display of subparts
<compose-to-sender>		compose new message to the current message sender
<delete-entry>	d	delete the current entry
<display-toggle-weed>	h	display message and toggle header weeding
<edit-type>	^E	edit attachment content type
<extract-keys>	^K	extract supported public keys
<forget-passphrase>	^F	wipe passphrase(s) from memory

Function	Default key	Description
<forward-message>	f	forward a message with comments
<group-chat-reply>		reply to all recipients preserving To/Cc
<group-reply>	g	reply to all recipients
<list-reply>	L	reply to specified mailing list
<pipe-entry>	|	pipe message/attachment to a shell command
<print-entry>	p	print the current entry
<reply>	r	reply to a message
<resend-message>	Esc e	use the current message as a template for a new one
<save-entry>	s	save message/attachment to a mailbox/file
<undelete-entry>	u	undelete the current entry
<view-attach>	<Enter>	view attachment using mailcap entry if necessary
<view-attach>	<Keypadenter>	view attachment using mailcap entry if necessary
<view-attach>	<Return>	view attachment using mailcap entry if necessary
<view-mailcap>	m	force viewing of attachment using mailcap
<view-pager>		view attachment in pager using copiousoutput mailcap entry
<view-text>	T	view attachment as text

Compose Menu

Table 9-8. Default Compose Menu Bindings

Function	Default key	Description
<attach-file>	a	attach file(s) to this message
<attach-key>	Esc k	attach a PGP public key
<attach-message>	A	attach message(s) to this message
<autocrypt-menu>	o	show autocrypt compose menu options
<copy-file>	C	save message/attachment to a mailbox/file

Function	Default key	Description
<detach-file>	D	delete the current entry
<display-toggle-weed>	h	display message and toggle header weeding
<edit-bcc>	b	edit the BCC list
<edit-cc>	c	edit the CC list
<edit-description>	d	edit attachment description
<edit-encoding>	^E	edit attachment transfer-encoding
<edit-fcc>	f	enter a file to save a copy of this message in
<edit-file>	^X e	edit the file to be attached
<edit-from>	Esc f	edit the from field
<edit-headers>	E	edit the message with headers
<edit-message>	e	edit the message
<edit-mime>	m	edit attachment using mailcap entry
<edit-reply-to>	r	edit the Reply-To field
<edit-subject>	s	edit the subject of this message
<edit-to>	t	edit the TO list
<edit-type>	^T	edit attachment content type
<filter-entry>	F	filter attachment through a shell command
<forget-passphrase>	^F	wipe passphrase(s) from memory
<get-attachment>	G	get a temporary copy of an attachment
<ispell>	i	run ispell on the message
<mix>	M	send the message through a mixmaster remailer chain
<move-down>		move attachment down in compose menu list
<move-up>		move attachment up in compose menu list
<new-mime>	n	compose new attachment using mailcap entry
<pgp-menu>	p	show PGP options
<pipe-entry>	|	pipe message/attachment to a shell command
<postpone-message>	P	save this message to send later
<print-entry>	l	print the current entry

Function	Default key	Description
<rename-attachment>	^O	send attachment with a different name
<rename-file>	R	rename/move an attached file
<send-message>	y	send the message
<smime-menu>	S	show S/MIME options
<tag-entry>	T	tag the current entry
<toggle-disposition>	^D	toggle disposition between inline/attachment
<toggle-recode>		toggle recoding of this attachment
<toggle-unlink>	u	toggle whether to delete file after sending it
<update-encoding>	U	update an attachment's encoding info
<view-alt>	v	view multipart/alternative
<view-alt-mailcap>	V	view multipart/alternative using mailcap
<view-alt-pager>		view multipart/alternative in pager using copiousoutput mailcap entry
<view-alt-text>	Esc v	view multipart/alternative as text
<view-attach>	<Enter>	view attachment using mailcap entry if necessary
<view-attach>	<Keypadenter>	view attachment using mailcap entry if necessary
<view-attach>	<Return>	view attachment using mailcap entry if necessary
<view-mailcap>		force viewing of attachment using mailcap
<view-pager>		view attachment in pager using copiousoutput mailcap entry
<view-text>		view attachment as text
<write-fcc>	w	write the message to a folder

Postpone Menu

Table 9-9. Default Postpone Menu Bindings

Function	Default key	Description
<delete-entry>	d	delete the current entry

Function	Default key	Description
<undelete-entry>	u	undelete the current entry

Browser Menu

Table 9-10. Default Browser Menu Bindings

Function	Default key	Description
<buffy-list>	.	list mailboxes with new mail
<change-dir>	c	change directories
<check-new>		check mailboxes for new mail
<create-mailbox>	C	create a new mailbox (IMAP only)
<delete-mailbox>	d	delete the current mailbox (IMAP only)
<descend-directory>		descend into a directory
<display-filename>	@	display the currently selected file's name
<enter-mask>	m	enter a file mask
<rename-mailbox>	r	rename the current mailbox (IMAP only)
<select-new>	N	select a new file in this directory
<sort>	o	sort messages
<sort-reverse>	O	sort messages in reverse order
<subscribe>	s	subscribe to current mailbox (IMAP only)
<toggle-mailboxes>	<Tab>	toggle whether to browse mailboxes or all files
<toggle-subscribed>	T	toggle view all/subscribed mailboxes (IMAP only)
<unsubscribe>	u	unsubscribe from current mailbox (IMAP only)
<view-file>	<Space>	view file

Pgp Menu

Table 9-11. Default Pgp Menu Bindings

Function	Default key	Description

Function	Default key	Description
<verify-key>	c	verify a PGP public key
<view-name>	%	view the key's user id

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

Mixmaster Menu

Table 9-13. Default Mixmaster Menu Bindings

Function	Default key	Description
<accept>	<Enter>	accept the chain constructed
<accept>	<Keypadenter>	accept the chain constructed
<accept>	<Return>	accept the chain constructed
<append>	a	append a remailer to the chain
<chain-next>	<Right>	select the next element of the chain
<chain-next>	l	select the next element of the chain
<chain-prev>	<Left>	select the previous element of the chain
<chain-prev>	h	select the previous element of the chain
<delete>	d	delete a remailer from the chain
<insert>	i	insert a remailer into the chain
<select-entry>	<Space>	select the current entry

Editor Menu

Table 9-14. Default Editor Menu Bindings

Function	Default key	Description

Function	Default key	Description
<backspace>	<Backspace>	delete the char in front of the cursor
<backspace>	<Delete>	delete the char in front of the cursor
<backward-char>	<Left>	move the cursor one character to the left
<backward-char>	^B	move the cursor one character to the left
<backward-word>	Esc b	move the cursor to the beginning of the word
<bol>	<Home>	jump to the beginning of the line
<bol>	^A	jump to the beginning of the line
<buffy-cycle>	<Space>	cycle among incoming mailboxes
<capitalize-word>	Esc c	capitalize the word
<complete>	<Tab>	complete filename or alias
<complete-query>	^T	complete address with query
<delete-char>	^D	delete the char under the cursor
<downcase-word>	Esc l	convert the word to lower case
<eol>	<End>	jump to the end of the line
<eol>	^E	jump to the end of the line
<forward-char>	<Right>	move the cursor one character to the right
<forward-char>	^F	move the cursor one character to the right
<forward-word>	Esc f	move the cursor to the end of the word
<history-down>	<Down>	scroll down through the history list
<history-down>	^N	scroll down through the history list
<history-search>	^R	search through the history list
<history-up>	<Up>	scroll up through the history list
<history-up>	^P	scroll up through the history list
<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
<kill-word>	^W	delete the word in front of the cursor

Function	Default key	Description
<quote-char>	^V	quote the next typed key
<transpose-chars>		transpose character under cursor with previous
<upcase-word>	Esc u	convert the word to upper case

Autocrypt Account Menu

Table 9-15. Default Autocrypt Account Menu Bindings

Function	Default key	Description
<create-account>	c	create a new autocrypt account
<delete-account>	D	delete the current account
<toggle-active>	a	toggle the current account active/inactive
<toggle-prefer-encrypt>	p	toggle the current account prefer-encrypt flag

List Menu

Table 9-16. Default List Menu Bindings

Function	Default key	Description
<list-archive>	a	retrieve list archive information
<list-help>	h	retrieve list help
<list-owner>	o	contact list owner
<list-post>	p	post to mailing list
<list-subscribe>	s	subscribe to mailing list
<list-unsubscribe>	u	unsubscribe from mailing list

Chapter 10. Miscellany

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>

- Kevin J. McCarthy <kevin@8t8.us>
- 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>

About This Document

This document was written in DocBook (http://docbook.sourceforge.net), and then rendered using the Gnome XSLT toolkit (http://xmlsoft.org/XSLT/).