notmuch

Release 0.30

Jul 12, 2020

Contents

Contents:

CHAPTER 1

notmuch

1.1 SYNOPSIS

notmuch [option ...] **command** [arg ...]

1.2 DESCRIPTION

Notmuch is a command-line based program for indexing, searching, reading, and tagging large collections of email messages.

This page describes how to get started using notmuch from the command line, and gives a brief overview of the commands available. For more information on e.g. **notmuch show** consult the **notmuch-show(1)** man page, also accessible via **notmuch help show**

The quickest way to get started with Notmuch is to simply invoke the `notmuch` command with no arguments, which will interactively guide you through the process of indexing your mail.

1.3 NOTE

While the command-line program `notmuch` provides powerful functionality, it does not provide the most convenient interface for that functionality. More sophisticated interfaces are expected to be built on top of either the command-line interface, or more likely, on top of the notmuch library interface. See https://notmuchmail.org for more about alternate interfaces to notmuch. The emacs-based interface to notmuch (available under **emacs/** in the Notmuch source distribution) is probably the most widely used at this time.

1.4 OPTIONS

Supported global options for `notmuch` include

--help [command-name] Print a synopsis of available commands and exit. With an optional command name, show the man page for that subcommand.

--version Print the installed version of notmuch, and exit.

--config=FILE Specify the configuration file to use. This overrides any configuration file specified by ${NOTMUCH_CONFIG}.

--uuid=HEX Enforce that the database UUID (a unique identifier which persists until e.g. the database is compacted) is HEX; exit with an error if it is not. This is useful to detect rollover in modification counts on messages. You can find this UUID using e.g. `notmuch count --lastmod`

All global options except `--config` can also be specified after the command. For example, `notmuch subcommand --uuid=HEX` is equivalent to `notmuch --uuid=HEX subcommand`.

1.5 COMMANDS

1.5.1 SETUP

The **notmuch setup** command is used to configure Notmuch for first use, (or to reconfigure it later).

The setup command will prompt for your full name, your primary email address, any alternate email addresses you use, and the directory containing your email archives. Your answers will be written to a configuration file in ${NOTMUCH_CONFIG} (if set) or ${HOME}/.notmuch-config . This configuration file will be created with descriptive comments, making it easy to edit by hand later to change the configuration. Or you can run **notmuch setup** again to change the configuration.

The mail directory you specify can contain any number of sub-directories and should primarily contain only files with individual email messages (eg. maildir or mh archives are perfect). If there are other, non-email files (such as indexes maintained by other email programs) then notmuch will do its best to detect those and ignore them.

Mail storage that uses mbox format, (where one mbox file contains many messages), will not work with notmuch. If that's how your mail is currently stored, it is recommended you first convert it to maildir format with a utility such as mb2md before running **notmuch setup .**

Invoking `notmuch` with no command argument will run **setup** if the setup command has not previously been completed.

1.5.2 OTHER COMMANDS

Several of the notmuch commands accept search terms with a common syntax. See **notmuch-search-terms**(7) for more details on the supported syntax.

The **search**, **show**, **address** and **count** commands are used to query the email database.

The **reply** command is useful for preparing a template for an email reply.

The **tag** command is the only command available for manipulating database contents.

The **dump** and **restore** commands can be used to create a textual dump of email tags for backup purposes, and to restore from that dump.

The **config** command can be used to get or set settings in the notmuch configuration file.

1.5.3 CUSTOM COMMANDS

If the given command is not known to notmuch, notmuch tries to execute the external **notmuch-<subcommand>** in ${PATH} instead. This allows users to have their own notmuch related tools to be run via the notmuch command. By design, this does not allow notmuch's own commands to be overridden using external commands.

1.5.4 OPTION SYNTAX

All options accepting an argument can be used with '=' or ':' as a separator. Except for boolean options (which would be ambiguous), a space can also be used as a separator. The following are all equivalent:

```
notmuch --config=alt-config config get user.name
notmuch --config:alt-config config get user.name
notmuch --config alt-config config get user.name
```

1.6 ENVIRONMENT

The following environment variables can be used to control the behavior of notmuch.

NOTMUCH_CONFIG Specifies the location of the notmuch configuration file. Notmuch will use ${HOME}/.notmuch-config if this variable is not set.

NOTMUCH_TALLOC_REPORT Location to write a talloc memory usage report. See **talloc_enable_leak_report_full** in **talloc(3)** for more information.

NOTMUCH_DEBUG_QUERY If set to a non-empty value, the notmuch library will print (to stderr) Xapian queries it constructs.

1.7 SEE ALSO

notmuch-address(1), **notmuch-compact(1)**, **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-properties(7)**, **notmuch-reindex(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)**

The notmuch website: **https://notmuchmail.org**

1.8 CONTACT

Feel free to send questions, comments, or kudos to the notmuch mailing list <notmuch@notmuchmail.org> . Subscription is not required before posting, but is available from the notmuchmail.org website.

Real-time interaction with the Notmuch community is available via IRC (server: irc.freenode.net, channel: #notmuch).

CHAPTER 2

notmuch-address

2.1 SYNOPSIS

notmuch address [*option* ...] <*search-term*> ...

2.2 DESCRIPTION

Search for messages matching the given search terms, and display the addresses from them. Duplicate addresses are filtered out.

See **notmuch-search-terms(7)** for details of the supported syntax for <search-terms>.

Supported options for **address** include

`--format=`(json|sexp|text|text0) Presents the results in either JSON, S-Expressions, newline character separated plain-text (default), or null character separated plain-text (compatible with **xargs(1)** -0 option where available).

`--format-version=N` Use the specified structured output format version. This is intended for programs that invoke **notmuch(1)** internally. If omitted, the latest supported version will be used.

`--output=(sender|recipients|count|address)` Controls which information appears in the output. This option can be given multiple times to combine different outputs. When neither `--output=sender` nor `--output=recipients` is given, `--output=sender` is implied.

sender Output all addresses from the *From* header.

Note: Searching for **sender** should be much faster than searching for **recipients**, because sender addresses are cached directly in the database whereas other addresses need to be fetched from message files.

recipients Output all addresses from the *To*, *Cc* and *Bcc* headers.

count Print the count of how many times was the address encountered during search.

Note: With this option, addresses are printed only after the whole search is finished. This may take long time.

address Output only the email addresses instead of the full mailboxes with names and email addresses. This option has no effect on the JSON or S-Expression output formats.

`--deduplicate=(no|mailbox|address)` Control the deduplication of results.

no Output all occurrences of addresses in the matching messages. This is not applicable with `--output=count`.

mailbox Deduplicate addresses based on the full, case sensitive name and email address, or mailbox. This is effectively the same as piping the `--deduplicate=no` output to **sort | uniq**, except for the order of results. This is the default.

address Deduplicate addresses based on the case insensitive address part of the mailbox. Of all the variants (with different name or case), print the one occurring most frequently among the matching messages. If `--output=count` is specified, include all variants in the count.

`--sort=`(newest-first|oldest-first) This option can be used to present results in either chronological order (**oldest-first**) or reverse chronological order (**newest-first**).

By default, results will be displayed in reverse chronological order, (that is, the newest results will be displayed first).

However, if either `--output=count` or `--deduplicate=address` is specified, this option is ignored and the order of the results is unspecified.

`--exclude=(true|false)` A message is called "excluded" if it matches at least one tag in search.exclude_tags that does not appear explicitly in the search terms. This option specifies whether to omit excluded messages in the search process.

The default value, **true**, prevents excluded messages from matching the search terms.

false allows excluded messages to match search terms and appear in displayed results.

2.3 EXIT STATUS

This command supports the following special exit status codes

`20` The requested format version is too old.

`21` The requested format version is too new.

2.4 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)**, **notmuch-search(1)**

CHAPTER 3

notmuch-compact

3.1 SYNOPSIS

notmuch compact [–quiet] [–backup=<*directory*>]

3.2 DESCRIPTION

The **compact** command can be used to compact the notmuch database. This can both reduce the space required by the database and improve lookup performance.

The compacted database is built in a temporary directory and is later moved into the place of the origin database. The original uncompacted database is discarded, unless the `--backup=`<directory> option is used.

Note that the database write lock will be held during the compaction process (which may be quite long) to protect data integrity.

Supported options for **compact** include

`--backup=`<directory> Save the current database to the given directory before replacing it with the compacted database. The backup directory must not exist and it must reside on the same mounted filesystem as the current database.

`--quiet` Do not report database compaction progress to stdout.

3.3 ENVIRONMENT

The following environment variables can be used to control the behavior of notmuch.

NOTMUCH_CONFIG Specifies the location of the notmuch configuration file. Notmuch will use ${HOME}/.notmuch-config if this variable is not set.

3.4 SEE ALSO

notmuch(1), **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)**

CHAPTER 4

notmuch-config

4.1 SYNOPSIS

notmuch config get <*section*>.<*item*>

notmuch config set <*section*>.<*item*> [*value* . . .]

notmuch config list

4.2 DESCRIPTION

The **config** command can be used to get or set settings in the notmuch configuration file and corresponding database.

Items marked **[STORED IN DATABASE]** are only in the database. They should not be placed in the configuration file, and should be accessed programmatically as described in the SYNOPSIS above.

get The value of the specified configuration item is printed to stdout. If the item has multiple values (it is a list), each value is separated by a newline character.

set The specified configuration item is set to the given value. To specify a multiple-value item (a list), provide each value as a separate command-line argument.

If no values are provided, the specified configuration item will be removed from the configuration file.

list Every configuration item is printed to stdout, each on a separate line of the form:

```
section.item=value
```

No additional whitespace surrounds the dot or equals sign characters. In a multiple-value item (a list), the values are separated by semicolon characters.

The available configuration items are described below.

database.path The top-level directory where your mail currently exists and to where mail will be delivered in the future. Files should be individual email messages. Notmuch will store its database within a sub-directory of the path configured here named `.notmuch`.

Default: `$MAILDIR` variable if set, otherwise `$HOME/mail`.

user.name Your full name.

Default: `$NAME` variable if set, otherwise read from `/etc/passwd`.

user.primary_email Your primary email address.

Default: `$EMAIL` variable if set, otherwise constructed from the username and hostname of the current machine.

user.other_email A list of other email addresses at which you receive email.

Default: not set.

new.tags A list of tags that will be added to all messages incorporated by **notmuch new**.

Default: `unread;inbox`.

new.ignore A list to specify files and directories that will not be searched for messages by **notmuch new**. Each entry in the list is either:

A file or a directory name, without path, that will be ignored, regardless of the location in the mail store directory hierarchy.

Or:

A regular expression delimited with // that will be matched against the path of the file or directory relative to the database path. Matching files and directories will be ignored. The beginning and end of string must be explicitly anchored. For example, /.*/foo$/ would match "bar/foo" and "bar/baz/foo", but not "foo" or "bar/foobar".

Default: empty list.

search.exclude_tags A list of tags that will be excluded from search results by default. Using an excluded tag in a query will override that exclusion.

Default: empty list. Note that **notmuch-setup(1)** puts `deleted;spam` here when creating new configuration file.

maildir.synchronize_flags If true, then the following maildir flags (in message filenames) will be synchronized with the corresponding notmuch tags:

Flag	Tag
D	draft
F	flagged
P	passed
R	replied
S	unread (added when 'S' flag is not present)

The **notmuch new** command will notice flag changes in filenames and update tags, while the **notmuch tag** and **notmuch restore** commands will notice tag changes and update flags in filenames.

If there have been any changes in the maildir (new messages added, old ones removed or renamed, maildir flags changed, etc.), it is advisable to run **notmuch new** before **notmuch tag** or **notmuch restore** commands to ensure the tag changes are properly synchronized to the maildir flags, as the commands expect the database and maildir to be in sync.

Default: `true`.

index.decrypt [STORED IN DATABASE] Policy for decrypting encrypted messages during indexing. Must be one of: `false`, `auto`, `nostash`, or `true`.

When indexing an encrypted e-mail message, if this variable is set to `true`, notmuch will try to decrypt the message and index the cleartext, stashing a copy of any discovered session keys for the message. If `auto`, it will try to index the cleartext if a stashed session key is already known for the message (e.g. from a previous copy), but will not try to access your secret keys. Use `false` to avoid decrypting even when a stashed session key is already present.

`nostash` is the same as `true` except that it will not stash newly-discovered session keys in the database.

From the command line (i.e. during **notmuch-new(1)**, **notmuch-insert(1)**, or **notmuch-reindex(1)**), the user can override the database's stored decryption policy with the `--decrypt=` option.

Here is a table that summarizes the functionality of each of these policies:

	false	auto	nostash	true
Index cleartext using stashed session keys		X	X	X
Index cleartext using secret keys			X	X
Stash session keys				X
Delete stashed session keys on reindex	X			

Stashed session keys are kept in the database as properties associated with the message. See `session-key` in **notmuch-properties(7)** for more details about how they can be useful.

Be aware that the notmuch index is likely sufficient (and a stashed session key is certainly sufficient) to reconstruct the cleartext of the message itself, so please ensure that the notmuch message index is adequately protected. DO NOT USE `index.decrypt=true` or `index.decrypt=nostash` without considering the security of your index.

Default: `auto`.

index.header.<prefix> [STORED IN DATABASE] Define the query prefix <prefix>, based on a mail header. For example `index.header.List=List-Id` will add a probabilistic prefix `List:` that searches the `List-Id` field. User defined prefixes must not start with 'a'...'z'; in particular adding a prefix with same name as a predefined prefix is not supported. See **notmuch-search-terms(7)** for a list of existing prefixes, and an explanation of probabilistic prefixes.

built_with.<name> Compile time feature <name>. Current possibilities include "retry_lock" (configure option, included by default). (since notmuch 0.30, "compact" and "field_processor" are always included.)

query.<name> [STORED IN DATABASE] Expansion for named query called <name>. See **notmuch-search-terms(7)** for more information about named queries.

4.3 ENVIRONMENT

The following environment variables can be used to control the behavior of notmuch.

NOTMUCH_CONFIG Specifies the location of the notmuch configuration file. Notmuch will use ${HOME}/.notmuch-config if this variable is not set.

4.4 SEE ALSO

notmuch(1), **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-properties(7)**,

notmuch-show(1), **notmuch-tag(1)**

CHAPTER 5

notmuch-count

5.1 SYNOPSIS

notmuch count [*option* . . .] <*search-term*> . . .

5.2 DESCRIPTION

Count messages matching the search terms.

The number of matching messages (or threads) is output to stdout.

With no search terms, a count of all messages (or threads) in the database will be displayed.

See **notmuch-search-terms(7)** for details of the supported syntax for <search-terms>.

Supported options for **count** include

`--output=(messages|threads|files)`

> **messages** Output the number of matching messages. This is the default.
>
> **threads** Output the number of matching threads.
>
> **files** Output the number of files associated with matching messages. This may be bigger than the number of matching messages due to duplicates (i.e. multiple files having the same message-id).

`--exclude=(true|false)` Specify whether to omit messages matching search.exclude_tags from the count (the default) or not.

`--batch` Read queries from a file (stdin by default), one per line, and output the number of matching messages (or threads) to stdout, one per line. On an empty input line the count of all messages (or threads) in the database will be output. This option is not compatible with specifying search terms on the command line.

`--lastmod` Append lastmod (counter for number of database updates) and UUID to the output. lastmod values are only comparable between databases with the same UUID.

`--input=<filename>` Read input from given file, instead of from stdin. Implies `--batch`.

5.3 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)**

CHAPTER 6

notmuch-dump

6.1 SYNOPSIS

notmuch dump [–gzip] [–format=(batch-tag|sup)] [–output=<*file*>] [–] [<*search-term*> . . .]

6.2 DESCRIPTION

Dump tags for messages matching the given search terms.

Output is to the given filename, if any, or to stdout.

These tags are the only data in the notmuch database that can't be recreated from the messages themselves. The output of notmuch dump is therefore the only critical thing to backup (and much more friendly to incremental backup than the native database files.)

See **notmuch-search-terms(7)** for details of the supported syntax for <search-terms>. With no search terms, a dump of all messages in the database will be generated. A `--` argument instructs notmuch that the remaining arguments are search terms.

Supported options for **dump** include

`--gzip` Compress the output in a format compatible with **gzip(1)**.

`--format=(sup|batch-tag)` Notmuch restore supports two plain text dump formats, both with one message-id per line, followed by a list of tags.

batch-tag The default **batch-tag** dump format is intended to more robust against malformed message-ids and tags containing whitespace or non-**ascii(7)** characters. Each line has the form:

```
+<*encoded-tag*\ > +<*encoded-tag*\ > ... -- id:<*quoted-message-id*\ >
```

Tags are hex-encoded by replacing every byte not matching the regex **[A-Za-z0-9@=.,_+-]** with **%nn** where nn is the two digit hex encoding. The message ID is a valid Xapian query, quoted using Xapian boolean term quoting rules: if the ID contains whitespace or a close paren or starts with a double quote, it must be enclosed in double quotes and double quotes inside the ID must be doubled. The astute reader will notice this is a special case of the batch input format for **notmuch-tag(1)**; note that the single message-id query is mandatory for **notmuch-restore(1)**.

sup The **sup** dump file format is specifically chosen to be compatible with the format of files produced by sup-dump. So if you've previously been using sup for mail, then the **notmuch restore** command provides you a way to import all of your tags (or labels as sup calls them). Each line has the following form:

```
<*message-id*\ > **(** <*tag*\ > ... **)**
```

with zero or more tags are separated by spaces. Note that (malformed) message-ids may contain arbitrary non-null characters. Note also that tags with spaces will not be correctly restored with this format.

`--include=(config|properties|tags)` Control what kind of metadata is included in the output.

config Output configuration data stored in the database. Each line starts with "#@ ", followed by a space separated key-value pair. Both key and value are hex encoded if needed.

properties Output per-message (key,value) metadata. Each line starts with "#= ", followed by a message id, and a space separated list of key=value pairs. Ids, keys and values are hex encoded if needed. See **notmuch-properties(7)** for more details.

tags Output per-message boolean metadata, namely tags. See *format* above for description of the output.

The default is to include all available types of data. The option can be specified multiple times to select some subset. As of version 3 of the dump format, there is a header line of the following form:

```
#notmuch-dump <*format*>:<*version*> <*included*>
```

where <*included*> is a comma separated list of the above options.

`--output=`<filename> Write output to given file instead of stdout.

6.3 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-properties(7)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)**

CHAPTER 7

notmuch-emacs-mua

7.1 SYNOPSIS

notmuch emacs-mua [options ...] [<to-address> ... | <mailto-url>]

7.2 DESCRIPTION

Start composing an email in the Notmuch Emacs UI with the specified subject, recipients, and message body, or mailto: URL.

Supported options for **emacs-mua** include

`-h, --help` Display help.

`-s, --subject`=<subject> Specify the subject of the message.

`--to`=<to-address> Specify a recipient (To).

`-c, --cc`=<cc-address> Specify a carbon-copy (Cc) recipient.

`-b, --bcc`=<bcc-address> Specify a blind-carbon-copy (Bcc) recipient.

`-i, --body`=<file> Specify a file to include into the body of the message.

`--hello` Go to the Notmuch hello screen instead of the message composition window if no message composition parameters are given.

`--no-window-system` Even if a window system is available, use the current terminal.

`--client` Use **emacsclient**, rather than **emacs**. For **emacsclient** to work, you need an already running Emacs with a server, or use `--auto-daemon`.

`--auto-daemon` Automatically start Emacs in daemon mode, if the Emacs server is not running. Applicable with `--client`. Implies `--create-frame`.

`--create-frame` Create a new frame instead of trying to use the current Emacs frame. Applicable with `--client`. This will be required when Emacs is running (or automatically started with `--auto-daemon`) in daemon mode.

`--print` Output the resulting elisp to stdout instead of evaluating it.

The supported positional parameters and short options are a compatible subset of the **mutt** MUA command-line options. The options and positional parameters modifying the message can't be combined with the mailto: URL.

Options may be specified multiple times.

7.3 ENVIRONMENT VARIABLES

EMACS Name of emacs command to invoke. Defaults to "emacs".

EMACSCLIENT Name of emacsclient command to invoke. Defaults to "emacsclient".

7.4 SEE ALSO

notmuch(1), **emacsclient(1)**, **mutt(1)**

CHAPTER 8

notmuch-hooks

8.1 SYNOPSIS

$DATABASEDIR/.notmuch/hooks/*

8.2 DESCRIPTION

Hooks are scripts (or arbitrary executables or symlinks to such) that notmuch invokes before and after certain actions. These scripts reside in the .notmuch/hooks directory within the database directory and must have executable permissions.

The currently available hooks are described below.

pre-new This hook is invoked by the **new** command before scanning or importing new messages into the database. If this hook exits with a non-zero status, notmuch will abort further processing of the **new** command.

Typically this hook is used for fetching or delivering new mail to be imported into the database.

post-new This hook is invoked by the **new** command after new messages have been imported into the database and initial tags have been applied. The hook will not be run if there have been any errors during the scan or import.

Typically this hook is used to perform additional query-based tagging on the imported messages.

post-insert This hook is invoked by the **insert** command after the message has been delivered, added to the database, and initial tags have been applied. The hook will not be run if there have been any errors during the message delivery; what is regarded as successful delivery depends on the `--keep` option.

Typically this hook is used to perform additional query-based tagging on the delivered messages.

8.3 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)**

CHAPTER 9

notmuch-insert

9.1 SYNOPSIS

notmuch insert [option . . .] [+<*tag*>|-<*tag*> . . .]

9.2 DESCRIPTION

notmuch insert reads a message from standard input and delivers it into the maildir directory given by configuration option **database.path**, then incorporates the message into the notmuch database. It is an alternative to using a separate tool to deliver the message then running **notmuch new** afterwards.

The new message will be tagged with the tags specified by the **new.tags** configuration option, then by operations specified on the command-line: tags prefixed by '+' are added while those prefixed by '-' are removed.

If the new message is a duplicate of an existing message in the database (it has same Message-ID), it will be added to the maildir folder and notmuch database, but the tags will not be changed.

The **insert** command supports hooks. See **notmuch-hooks(5)** for more details on hooks.

Option arguments must appear before any tag operation arguments. Supported options for **insert** include

--folder=<folder> Deliver the message to the specified folder, relative to the top-level directory given by the value of **database.path**. The default is the empty string, which means delivering to the top-level directory.

--create-folder Try to create the folder named by the `--folder` option, if it does not exist. Otherwise the folder must already exist for mail delivery to succeed.

--keep Keep the message file if indexing fails, and keep the message indexed if applying tags or maildir flag synchronization fails. Ignore these errors and return exit status 0 to indicate successful mail delivery.

--no-hooks Prevent hooks from being run.

--world-readable When writing mail to the mailbox, allow it to be read by users other than the current user. Note that this does not override umask. By default, delivered mail is only readable by the current user.

--decrypt=(true|nostash|auto|false) If `true` and the message is encrypted, try to decrypt the message while indexing, stashing any session keys discovered. If `auto`, and notmuch already knows about a session key for the message, it will try decrypting using that session key but will not try to access the user's secret keys. If decryption is successful, index the cleartext itself. Either way, the message is always stored to disk in its original form (ciphertext).

`nostash` is the same as `true` except that it will not stash newly-discovered session keys in the database.

Be aware that the index is likely sufficient (and a stashed session key is certainly sufficient) to reconstruct the cleartext of the message itself, so please ensure that the notmuch message index is adequately protected. DO NOT USE `--decrypt=true` or `--decrypt=nostash` without considering the security of your index.

See also `index.decrypt` in **notmuch-config(1)**.

9.3 EXIT STATUS

This command returns exit status 0 on successful mail delivery, non-zero otherwise. The default is to indicate failed mail delivery on any errors, including message file delivery to the filesystem, message indexing to Notmuch database, changing tags, and synchronizing tags to maildir flags. The `--keep` option may be used to settle for successful message file delivery.

This command supports the following special exit status code for errors most likely to be temporary in nature, e.g. failure to get a database write lock.

75 (EX_TEMPFAIL) A temporary failure occurred; the user is invited to retry.

The exit status of the **post-insert** hook does not affect the exit status of the **insert** command.

9.4 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)**

CHAPTER 10

notmuch-new

10.1 SYNOPSIS

notmuch new [options]

10.2 DESCRIPTION

Find and import any new messages to the database.

The **new** command scans all sub-directories of the database, performing full-text indexing on new messages that are found. Each new message will automatically be tagged with both the **inbox** and **unread** tags.

You should run **notmuch new** once after first running **notmuch setup** to create the initial database. The first run may take a long time if you have a significant amount of mail (several hundred thousand messages or more). Subsequently, you should run **notmuch new** whenever new mail is delivered and you wish to incorporate it into the database. These subsequent runs will be much quicker than the initial run.

Invoking `notmuch` with no command argument will run **new** if **notmuch setup** has previously been completed, but **notmuch new** has not previously been run.

notmuch new updates tags according to maildir flag changes if the **maildir.synchronize_flags** configuration option is enabled. See **notmuch-config(1)** for details.

The **new** command supports hooks. See **notmuch-hooks(5)** for more details on hooks.

Supported options for **new** include

`--no-hooks` Prevents hooks from being run.

`--quiet` Do not print progress or results.

`--verbose` Print file names being processed. Ignored when combined with `--quiet`.

`--decrypt=(true|nostash|auto|false)` If `true`, when encountering an encrypted message, try to decrypt it while indexing, and stash any discovered session keys. If `auto`, try to use any session key already known to belong to this message, but do not attempt to use the user's secret keys. If decryption is successful, index the cleartext of the message.

Be aware that the index is likely sufficient (and the session key is certainly sufficient) to reconstruct the cleartext of the message itself, so please ensure that the notmuch message index is adequately protected. DO NOT USE `--decrypt=true` or `--decrypt=nostash` without considering the security of your index.

See also `index.decrypt` in **notmuch-config(1)**.

`--full-scan` By default notmuch-new uses directory modification times (mtimes) to optimize the scanning of directories for new mail. This option turns that optimization off.

10.3 EXIT STATUS

This command supports the following special exit status code

`75 (EX_TEMPFAIL)` A temporary failure occurred; the user is invited to retry.

10.4 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)**

CHAPTER 11

notmuch-properties

11.1 SYNOPSIS

notmuch count property:<*key*>=<*value*>

notmuch search property:<*key*>=<*value*>

notmuch show property:<*key*>=<*value*>

notmuch reindex property:<*key*>=<*value*>

notmuch tag +<*tag*> **property:**<*key*>=<*value*>

notmuch dump –include=properties

notmuch restore –include=properties

11.2 DESCRIPTION

Several notmuch commands can search for, modify, add or remove properties associated with specific messages. Properties are key/value pairs, and a message can have more than one key/value pair for the same key.

While users can select based on a specific property in their search terms with the prefix **property:**, the notmuch command-line interface does not provide mechanisms for modifying properties directly to the user.

Instead, message properties are expected to be set and used programmatically, according to logic in notmuch itself, or in extensions to it.

Extensions to notmuch which make use of properties are encouraged to report the specific properties used to the upstream notmuch project, as a way of avoiding collisions in the property namespace.

11.3 CONVENTIONS

Any property with a key that starts with "index." will be removed (and possibly re-set) upon reindexing (see **notmuch-reindex(1)**).

11.4 MESSAGE PROPERTIES

The following properties are set by notmuch internally in the course of its normal activity.

index.decryption If a message contains encrypted content, and notmuch tries to decrypt that content during indexing, it will add the property `index.decryption=success` when the cleartext was successfully indexed. If notmuch attempts to decrypt any part of a message during indexing and that decryption attempt fails, it will add the property `index.decryption=failure` to the message.

Note that it's possible for a single message to have both `index.decryption=success` and `index.decryption=failure`. Consider an encrypted e-mail message that contains another encrypted e-mail message as an attachment – if the outer message can be decrypted, but the attached part cannot, then both properties will be set on the message as a whole.

If notmuch never tried to decrypt an encrypted message during indexing (which is the default, see `index.decrypt` in **notmuch-config(1)**), then this property will not be set on that message.

session-key

When **notmuch-show(1)** or **nomtuch-reply** encounters a message with an encrypted part, if notmuch finds a `session-key` property associated with the message, it will try that stashed session key for decryption.

If you do not want to use any stashed session keys that might be present, you should pass those programs `--decrypt=false`.

Using a stashed session key with "notmuch show" will speed up rendering of long encrypted threads. It also allows the user to destroy the secret part of any expired encryption-capable subkey while still being able to read any retained messages for which they have stashed the session key. This enables truly deletable e-mail, since (once the session key and asymmetric subkey are both destroyed) there are no keys left that can be used to decrypt any copy of the original message previously stored by an adversary.

However, access to the stashed session key for an encrypted message permits full byte-for-byte reconstruction of the cleartext message. This includes attachments, cryptographic signatures, and other material that cannot be reconstructed from the index alone.

See `index.decrypt` in **notmuch-config(1)** for more details about how to set notmuch's policy on when to store session keys.

The session key should be in the ASCII text form produced by GnuPG. For OpenPGP, that consists of a decimal representation of the hash algorithm used (identified by number from RFC 4880, e.g. 9 means AES-256) followed by a colon, followed by a hexadecimal representation of the algorithm-specific key. For example, an AES-128 key might be stashed in a notmuch property as: `session-key=7:14B16AF65536C28AF209828DFE34C9E0`.

index.repaired

Some messages arrive in forms that are confusing to view; they can be mangled by mail transport agents, or the sending mail user agent may structure them in a way that is confusing. If notmuch knows how to both detect and repair such a problematic message, it will do so during indexing.

If it applies a message repair during indexing, it will use the `index.repaired` property to note the type of repair(s) it performed.

`index.repaired=skip-protected-headers-legacy-display` indicates that when indexing the cleartext of an encrypted message, notmuch skipped over a "legacy-display" text/rfc822-headers part that it found in that message, since it was able to index the built-in protected headers directly.

`index.repaired=mixedup` indicates the repair of a "Mixed Up" encrypted PGP/MIME message, a mangling typically produced by Microsoft's Exchange MTA. See https://tools.ietf.org/html/draft-dkg-openpgp-pgpmime-message-mangling for more information.

11.5 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-dump(1)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-reindex(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-show(1)**, ***notmuch-search-terms(7)**

CHAPTER 12

notmuch-reindex

12.1 SYNOPSIS

notmuch reindex [*option* ...] <*search-term*> ...

12.2 DESCRIPTION

Re-index all messages matching the search terms.

See **notmuch-search-terms(7)** for details of the supported syntax for <*search-term*>.

The **reindex** command searches for all messages matching the supplied search terms, and re-creates the full-text index on these messages using the supplied options.

Supported options for **reindex** include

`--decrypt=(true|nostash|auto|false)` If `true`, when encountering an encrypted message, try to decrypt it while reindexing, stashing any session keys discovered. If `auto`, and notmuch already knows about a session key for the message, it will try decrypting using that session key but will not try to access the user's secret keys. If decryption is successful, index the cleartext itself.

`nostash` is the same as `true` except that it will not stash newly-discovered session keys in the database.

If `false`, notmuch reindex will also delete any stashed session keys for all messages matching the search terms.

Be aware that the index is likely sufficient (and a stashed session key is certainly sufficient) to reconstruct the cleartext of the message itself, so please ensure that the notmuch message index is adequately protected. DO NOT USE `--decrypt=true` or `--decrypt=nostash` without considering the security of your index.

See also `index.decrypt` in **notmuch-config(1)**.

12.3 EXAMPLES

A user just received an encrypted message without indexing its cleartext. After reading it (via `notmuch show --decrypt=true`), they decide that they want to index its cleartext so that they can easily find it later and read it without having to have access to their secret keys:

```
notmuch reindex --decrypt=true id:1234567@example.com
```

A user wants to change their policy going forward to start indexing cleartext. But they also want indexed access to the cleartext of all previously-received encrypted messages. Some messages might have already been indexed in the clear (as in the example above). They can ask notmuch to just reindex the not-yet-indexed messages:

```
notmuch config set index.decrypt true
notmuch reindex tag:encrypted and not property:index.decryption=success
```

Later, the user changes their mind, and wants to stop indexing cleartext (perhaps their threat model has changed, or their trust in their index store has been shaken). They also want to clear all of their old cleartext from the index. Note that they compact the database afterward as a workaround for https://trac.xapian.org/ticket/742:

```
notmuch config set index.decrypt false
notmuch reindex property:index.decryption=success
notmuch compact
```

12.4 SEE ALSO

notmuch(1), **notmuch-compact(1)**, **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)**

CHAPTER 13

notmuch-reply

13.1 SYNOPSIS

notmuch reply [option ...] <*search-term*> ...

13.2 DESCRIPTION

Constructs a reply template for a set of messages.

To make replying to email easier, **notmuch reply** takes an existing set of messages and constructs a suitable mail template. Its To: address is set according to the original email in this way: if the Reply-to: header is present and different from any To:/Cc: address it is used, otherwise From: header is used. Unless `--reply-to=sender` is specified, values from the To: and Cc: headers are copied, but not including any of the current user's email addresses (as configured in primary_mail or other_email in the .notmuch-config file) in the recipient list.

It also builds a suitable new subject, including Re: at the front (if not already present), and adding the message IDs of the messages being replied to to the References list and setting the In-Reply-To: field correctly.

Finally, the original contents of the emails are quoted by prefixing each line with '> ' and included in the body.

The resulting message template is output to stdout.

Supported options for **reply** include

`--format`=(default|json|sexp|headers-only)

default Includes subject and quoted message body as an RFC 2822 message.

json Produces JSON output containing headers for a reply message and the contents of the original message. This output can be used by a client to create a reply message intelligently.

sexp Produces S-Expression output containing headers for a reply message and the contents of the original message. This output can be used by a client to create a reply message intelligently.

headers-only Only produces In-Reply-To, References, To, Cc, and Bcc headers.

`--format-version=N` Use the specified structured output format version. This is intended for programs that invoke **notmuch(1)** internally. If omitted, the latest supported version will be used.

`--reply-to=`**(all|sender)**

all (default) Replies to all addresses.

sender Replies only to the sender. If replying to user's own message (Reply-to: or From: header is one of the user's configured email addresses), try To:, Cc:, and Bcc: headers in this order, and copy values from the first that contains something other than only the user's addresses.

`--decrypt=(false|auto|true)`

If `true`, decrypt any MIME encrypted parts found in the selected content (i.e., "multipart/encrypted" parts). Status of the decryption will be reported (currently only supported with `--format=json` and `--format=sexp`), and on successful decryption the multipart/encrypted part will be replaced by the decrypted content.

If `auto`, and a session key is already known for the message, then it will be decrypted, but notmuch will not try to access the user's secret keys.

Use `false` to avoid even automatic decryption.

Non-automatic decryption expects a functioning **gpg-agent(1)** to provide any needed credentials. Without one, the decryption will likely fail.

Default: `auto`

See **notmuch-search-terms(7)** for details of the supported syntax for <search-terms>.

Note: It is most common to use **notmuch reply** with a search string matching a single message, (such as id:<message-id>), but it can be useful to reply to several messages at once. For example, when a series of patches are sent in a single thread, replying to the entire thread allows for the reply to comment on issues found in multiple patches. The default format supports replying to multiple messages at once, but the JSON and S-Expression formats do not.

13.3 EXIT STATUS

This command supports the following special exit status codes

`20` The requested format version is too old.

`21` The requested format version is too new.

13.4 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)**

CHAPTER 14

notmuch-restore

14.1 SYNOPSIS

notmuch restore [–accumulate] [–format=(auto|batch-tag|sup)] [–input=<*filename*>]

14.2 DESCRIPTION

Restores the tags from the given file (see **notmuch dump**).

The input is read from the given filename, if any, or from stdin.

Supported options for **restore** include

`--accumulate` The union of the existing and new tags is applied, instead of replacing each message's tags as they are read in from the dump file.

`--format=(sup|batch-tag|auto)` Notmuch restore supports two plain text dump formats, with each line specifying a message-id and a set of tags. For details of the actual formats, see **notmuch-dump(1)**.

sup The **sup** dump file format is specifically chosen to be compatible with the format of files produced by sup-dump. So if you've previously been using sup for mail, then the **notmuch restore** command provides you a way to import all of your tags (or labels as sup calls them).

batch-tag The **batch-tag** dump format is intended to more robust against malformed message-ids and tags containing whitespace or non-**ascii(7)** characters. See **notmuch-dump(1)** for details on this format.

notmuch restore updates the maildir flags according to tag changes if the **maildir.synchronize_flags** configuration option is enabled. See **notmuch-config(1)** for details.

auto This option (the default) tries to guess the format from the input. For correctly formed input in either supported format, this heuristic, based the fact that batch-tag format contains no parentheses, should be accurate.

`--include=(config|properties|tags)` Control what kind of metadata is restored.

config Restore configuration data to the database. Each configuration line starts with "#@ ", followed by a space separated key-value pair. Both key and value are hex encoded if needed.

properties Restore per-message (key,value) metadata. Each line starts with "#= ", followed by a message id, and a space separated list of key=value pairs. Ids, keys and values are hex encoded if needed. See **notmuch-properties(7)** for more details.

tags Restore per-message metadata, namely tags. See *format* above for more details.

The default is to restore all available types of data. The option can be specified multiple times to select some subset.

`--input=`<filename> Read input from given file instead of stdin.

14.3 GZIPPED INPUT

notmuch restore will detect if the input is compressed in **gzip(1)** format and automatically decompress it while reading. This detection does not depend on file naming and in particular works for standard input.

14.4 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-properties(7)**, **notmuch-reply(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)**

CHAPTER 15

notmuch-search

15.1 SYNOPSIS

notmuch search [*option* . . .] <*search-term*> . . .

15.2 DESCRIPTION

Search for messages matching the given search terms, and display as results the threads containing the matched messages.

The output consists of one line per thread, giving a thread ID, the date of the newest (or oldest, depending on the sort option) matched message in the thread, the number of matched messages and total messages in the thread, the names of all participants in the thread, and the subject of the newest (or oldest) message.

See **notmuch-search-terms(7)** for details of the supported syntax for <search-terms>.

Supported options for **search** include

`--format=`(json|sexp|text|text0) Presents the results in either JSON, S-Expressions, newline character separated plain-text (default), or null character separated plain-text (compatible with **xargs(1)** -0 option where available).

`--format-version=N` Use the specified structured output format version. This is intended for programs that invoke **notmuch(1)** internally. If omitted, the latest supported version will be used.

`--output=(summary|threads|messages|files|tags)`

summary Output a summary of each thread with any message matching the search terms. The summary includes the thread ID, date, the number of messages in the thread (both the number matched and the total number), the authors of the thread and the subject. In the case where a thread contains multiple files for some messages, the total number of files is printed in parentheses (see below for an example).

threads Output the thread IDs of all threads with any message matching the search terms, either one per line (`--format=text`), separated by null characters (`--format=text0`), as a JSON array (`--format=json`), or an S-Expression list (`--format=sexp`).

messages Output the message IDs of all messages matching the search terms, either one per line (`--format=text`), separated by null characters (`--format=text0`), as a JSON array (`--format=json`), or as an S-Expression list (`--format=sexp`).

files Output the filenames of all messages matching the search terms, either one per line (`--format=text`), separated by null characters (`--format=text0`), as a JSON array (`--format=json`), or as an S-Expression list (`--format=sexp`).

Note that each message may have multiple filenames associated with it. All of them are included in the output (unless limited with the `--duplicate=N` option). This may be particularly confusing for **folder:** or **path:** searches in a specified directory, as the messages may have duplicates in other directories that are included in the output, although these files alone would not match the search.

tags Output all tags that appear on any message matching the search terms, either one per line (`--format=text`), separated by null characters (`--format=text0`), as a JSON array (`--format=json`), or as an S-Expression list (`--format=sexp`).

`--sort=`(newest-first|oldest-first) This option can be used to present results in either chronological order (**oldest-first**) or reverse chronological order (**newest-first**).

Note: The thread order will be distinct between these two options (beyond being simply reversed). When sorting by **oldest-first** the threads will be sorted by the oldest message in each thread, but when sorting by **newest-first** the threads will be sorted by the newest message in each thread.

By default, results will be displayed in reverse chronological order, (that is, the newest results will be displayed first).

`--offset=[-]N` Skip displaying the first N results. With the leading '-', start at the Nth result from the end.

`--limit=N` Limit the number of displayed results to N.

`--exclude=(true|false|all|flag)` A message is called "excluded" if it matches at least one tag in search.exclude_tags that does not appear explicitly in the search terms. This option specifies whether to omit excluded messages in the search process.

true (default) Prevent excluded messages from matching the search terms.

all Additionally prevent excluded messages from appearing in displayed results, in effect behaving as though the excluded messages do not exist.

false Allow excluded messages to match search terms and appear in displayed results. Excluded messages are still marked in the relevant outputs.

flag Only has an effect when `--output=summary`. The output is almost identical to **false**, but the "match count" is the number of matching non-excluded messages in the thread, rather than the number of matching messages.

`--duplicate=N` For `--output=files`, output the Nth filename associated with each message matching the query (N is 1-based). If N is greater than the number of files associated with the message, don't print anything.

For `--output=messages`, only output message IDs of messages matching the search terms that have at least N filenames associated with them.

Note that this option is orthogonal with the **folder:** search prefix. The prefix matches messages based on filenames. This option filters filenames of the matching messages.

15.3 EXAMPLE

The following shows an example of the summary output format, with one message having multiple filenames.

```
% notmuch search date:today.. and tag:bad-news
thread:0000000000063c10 Today [1/1] Some Persun; To the bone (bad-news inbox unread)
thread:0000000000063c25 Today [1/1(2)] Ann Other; Bears (bad-news inbox unread)
thread:0000000000063c00 Today [1/1] A Thurd; Bites, stings, sad feelings (bad-news␣
↪unread)
```

15.4 EXIT STATUS

This command supports the following special exit status codes

`20` The requested format version is too old.

`21` The requested format version is too new.

15.5 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**, **notmuch-tag(1)** **notmuch-address(1)**

CHAPTER 16

notmuch-search-terms

16.1 SYNOPSIS

notmuch count [option . . .] <*search-term*> . . .

notmuch dump [–gzip] [–format=(batch-tag|sup)] [–output=<*file*>] [–] [<*search-term*> . . .]

notmuch reindex [option . . .] <*search-term*> . . .

notmuch search [option . . .] <*search-term*> . . .

notmuch show [option . . .] <*search-term*> . . .

notmuch tag +<*tag*> . . . -<*tag*> [–] <*search-term*> . . .

16.2 DESCRIPTION

Several notmuch commands accept a common syntax for search terms.

The search terms can consist of free-form text (and quoted phrases) which will match all messages that contain all of the given terms/phrases in the body, the subject, or any of the sender or recipient headers.

As a special case, a search string consisting of exactly a single asterisk ("*") will match all messages.

16.2.1 Search prefixes

In addition to free text, the following prefixes can be used to force terms to match against specific portions of an email, (where <brackets> indicate user-supplied values).

Some of the prefixes with <regex> forms can be also used to restrict the results to those whose value matches a regular expression (see **regex(7)**) delimited with //, for example:

```
notmuch search 'from:"/bob@.*[.]example[.]com/"'
```

body:<word-or-quoted-phrase> Match terms in the body of messages.

from:<name-or-address> or from:/<regex>/ The **from:** prefix is used to match the name or address of the sender of an email message.

to:<name-or-address> The **to:** prefix is used to match the names or addresses of any recipient of an email message, (whether To, Cc, or Bcc).

subject:<word-or-quoted-phrase> or subject:/<regex>/ Any term prefixed with **subject:** will match only text from the subject of an email. Searching for a phrase in the subject is supported by including quotation marks around the phrase, immediately following **subject:**.

attachment:<word> The **attachment:** prefix can be used to search for specific filenames (or extensions) of attachments to email messages.

mimetype:<word> The **mimetype:** prefix will be used to match text from the content-types of MIME parts within email messages (as specified by the sender).

tag:<tag> or tag:/<regex>/ or is:<tag> or is:/<regex>/ For **tag:** and **is:** valid tag values include **inbox** and **unread** by default for new messages added by **notmuch new** as well as any other tag values added manually with **notmuch tag**.

id:<message-id> or mid:<message-id> or mid:/<regex>/ For **id:** and **mid:**, message ID values are the literal contents of the Message-ID: header of email messages, but without the '<', '>' delimiters.

thread:<thread-id> The **thread:** prefix can be used with the thread ID values that are generated internally by notmuch (and do not appear in email messages). These thread ID values can be seen in the first column of output from **notmuch search**

thread:{<notmuch query>} Threads may be searched for indirectly by providing an arbitrary notmuch query in **{}**. For example, the following returns threads containing a message from mallory and one (not necessarily the same message) with Subject containing the word "crypto".

```
% notmuch search 'thread:"{from:mallory}" and thread:"{subject:crypto}"'
```

The performance of such queries can vary wildly. To understand this, the user should think of the query **thread:{<something>}** as expanding to all of the thread IDs which match **<something>**; notmuch then performs a second search using the expanded query.

path:<directory-path> or path:<directory-path>/ or path:/<regex>/** The **path:** prefix searches for email messages that are in particular directories within the mail store. The directory must be specified relative to the top-level maildir (and without the leading slash). By default, **path:** matches messages in the specified directory only. The "/**" suffix can be used to match messages in the specified directory and all its subdirectories recursively. **path:""** matches messages in the root of the mail store and, likewise, **path:**** matches all messages.

path: will find a message if *any* copy of that message is in the specific directory.

folder:<maildir-folder> or folder:/<regex>/ The **folder:** prefix searches for email messages by maildir or MH folder. For MH-style folders, this is equivalent to **path:**. For maildir, this includes messages in the "new" and "cur" subdirectories. The exact syntax for maildir folders depends on your mail configuration. For maildir++, **folder:""** matches the inbox folder (which is the root in maildir++), other folder names always start with ".", and nested folders are separated by "."s, such as **folder:.classes.topology**. For "file system" maildir, the inbox is typically **folder:INBOX** and nested folders are separated by slashes, such as **folder:classes/topology**.

folder: will find a message if *any* copy of that message is in the specific folder.

date:<since>..<until> or date:<date> The **date:** prefix can be used to restrict the results to only messages within a particular time range (based on the Date: header).

See **DATE AND TIME SEARCH** below for details on the range expression, and supported syntax for <since> and <until> date and time expressions.

The time range can also be specified using timestamps without including the date prefix using a syntax of:

<initial-timestamp>..<final-timestamp>

Each timestamp is a number representing the number of seconds since 1970-01-01 00:00:00 UTC. Specifying a time range this way is considered legacy and predates the date prefix.

lastmod:<initial-revision>..<final-revision> The **lastmod:** prefix can be used to restrict the result by the database revision number of when messages were last modified (tags were added/removed or filenames changed). This is usually used in conjunction with the `--uuid` argument to **notmuch search** to find messages that have changed since an earlier query.

query:<name> The **query:** prefix allows queries to refer to previously saved queries added with **notmuch-config(1)**.

property:<key>=<value> The **property:** prefix searches for messages with a particular <key>=<value> property pair. Properties are used internally by notmuch (and extensions) to add metadata to messages. A given key can be present on a given message with several different values. See **notmuch-properties(7)** for more details.

User defined prefixes are also supported, see **notmuch-config(1)** for details.

16.2.2 Operators

In addition to individual terms, multiple terms can be combined with Boolean operators (**and**, **or**, **not**, and **xor**). Each term in the query will be implicitly connected by a logical AND if no explicit operator is provided (except that terms with a common prefix will be implicitly combined with OR). The shorthand '-<term>' can be used for 'not <term>' but unfortunately this does not work at the start of an expression. Parentheses can also be used to control the combination of the Boolean operators, but will have to be protected from interpretation by the shell, (such as by putting quotation marks around any parenthesized expression).

In addition to the standard boolean operators, Xapian provides several operators specific to text searching.

```
notmuch search term1 NEAR term2
```

will return results where term1 is within 10 words of term2. The threshold can be set like this:

```
notmuch search term1 NEAR/2 term2
```

The search

```
notmuch search term1 ADJ term2
```

will return results where term1 is within 10 words of term2, but in the same order as in the query. The threshold can be set the same as with NEAR:

```
notmuch search term1 ADJ/7 term2
```

16.2.3 Stemming

Stemming in notmuch means that these searches

```
notmuch search detailed
notmuch search details
notmuch search detail
```

will all return identical results, because Xapian first "reduces" the term to the common stem (here 'detail') and then performs the search.

There are two ways to turn this off: a search for a capitalized word will be performed unstemmed, so that one can search for "John" and not get results for "Johnson"; phrase searches are also unstemmed (see below for details). Stemming is currently only supported for English. Searches for words in other languages will be performed unstemmed.

16.2.4 Wildcards

It is possible to use a trailing '*' as a wildcard. A search for 'wildc*' will match 'wildcard', 'wildcat', etc.

16.2.5 Boolean and Probabilistic Prefixes

Xapian (and hence notmuch) prefixes are either **boolean**, supporting exact matches like "tag:inbox" or **probabilistic**, supporting a more flexible **term** based searching. Certain **special** prefixes are processed by notmuch in a way not strictly fitting either of Xapian's built in styles. The prefixes currently supported by notmuch are as follows.

Boolean **tag:**, **id:**, **thread:**, **folder:**, **path:**, **property:**

Probabilistic **body:**, **to:**, **attachment:**, **mimetype:**

Special **from:**, **query:**, **subject:**

16.2.6 Terms and phrases

In general Xapian distinguishes between lists of terms and **phrases**. Phrases are indicated by double quotes (but beware you probably need to protect those from your shell) and insist that those unstemmed words occur in that order. One useful, but initially surprising feature is that the following are equivalent ways to write the same phrase.

- "a list of words"
- a-list-of-words
- a/list/of/words
- a.list.of.words

Both parenthesised lists of terms and quoted phrases are ok with probabilistic prefixes such as **to:**, **from:**, and **subject:**. In particular

```
subject:(pizza free)
```

is equivalent to

```
subject:pizza and subject:free
```

Both of these will match a subject "Free Delicious Pizza" while

```
subject:"pizza free"
```

will not.

16.2.7 Quoting

Double quotes are also used by the notmuch query parser to protect boolean terms, regular expressions, or subqueries containing spaces or other special characters, e.g.

```
tag:"a tag"
```

```
folder:"/^.*/(Junk|Spam)$/"
```

```
thread:"{from:mallory and date:2009}"
```

As with phrases, you need to protect the double quotes from the shell e.g.

```
% notmuch search 'folder:"/^.*/(Junk|Spam)$/"'
% notmuch search 'thread:"{from:mallory and date:2009}" and thread:{to:mallory}'
```

16.3 DATE AND TIME SEARCH

notmuch understands a variety of standard and natural ways of expressing dates and times, both in absolute terms ("2012-10-24") and in relative terms ("yesterday"). Any number of relative terms can be combined ("1 hour 25 minutes") and an absolute date/time can be combined with relative terms to further adjust it. A non-exhaustive description of the syntax supported for absolute and relative terms is given below.

16.3.1 The range expression

date:<since>..<until>

The above expression restricts the results to only messages from <since> to <until>, based on the Date: header.

<since> and <until> can describe imprecise times, such as "yesterday". In this case, <since> is taken as the earliest time it could describe (the beginning of yesterday) and <until> is taken as the latest time it could describe (the end of yesterday). Similarly, date:january..february matches from the beginning of January to the end of February.

If specifying a time range using timestamps in conjunction with the date prefix, each timestamp must be preceded by @ (ASCII hex 40). As above, each timestamp is a number representing the number of seconds since 1970-01-01 00:00:00 UTC. For example:

> date:@<initial-timestamp>..@<final-timestamp>

Currently, spaces in range expressions are not supported. You can replace the spaces with '_', or (in most cases) '-', or (in some cases) leave the spaces out altogether. Examples in this man page use spaces for clarity.

Open-ended ranges are supported. I.e. it's possible to specify date:..<until> or date:<since>.. to not limit the start or end time, respectively.

16.3.2 Single expression

date:<expr> works as a shorthand for date:<expr>..<expr>. For example, date:monday matches from the beginning of Monday until the end of Monday.

16.3.3 Relative date and time

[N|number] (years|months|weeks|days|hours|hrs|minutes|mins|seconds|secs) [. . .]

All refer to past, can be repeated and will be accumulated.

Units can be abbreviated to any length, with the otherwise ambiguous single m being m for minutes and M for months.

Number can also be written out one, two, . . . , ten, dozen, hundred. Additionally, the unit may be preceded by "last" or "this" (e.g., "last week" or "this month").

When combined with absolute date and time, the relative date and time specification will be relative from the specified absolute date and time.

Examples: 5M2d, two weeks

16.3.4 Supported absolute time formats

- H[H]:MM[:SS] [(am|a.m.|pm|p.m.)]
- H[H] (am|a.m.|pm|p.m.)
- HHMMSS
- now
- noon
- midnight
- Examples: 17:05, 5pm

16.3.5 Supported absolute date formats

- YYYY-MM[-DD]
- DD-MM[-[YY]YY]
- MM-YYYY
- M[M]/D[D][/[YY]YY]
- M[M]/YYYY
- D[D].M[M][.[YY]YY]
- D[D][(st|nd|rd|th)] Mon[thname] [YYYY]
- Mon[thname] D[D][(st|nd|rd|th)] [YYYY]
- Wee[kday]

Month names can be abbreviated at three or more characters.

Weekday names can be abbreviated at three or more characters.

Examples: 2012-07-31, 31-07-2012, 7/31/2012, August 3

16.3.6 Time zones

- (+|-)HH:MM
- (+|-)HH[MM]

Some time zone codes, e.g. UTC, EET.

16.4 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-reindex(1)**, **notmuch-properties(1)**, ***notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, ***notmuch-show(1)**, **notmuch-tag(1)**

CHAPTER 17

notmuch-show

17.1 SYNOPSIS

notmuch show [*option* ...] <*search-term*> ...

17.2 DESCRIPTION

Shows all messages matching the search terms.

See **notmuch-search-terms(7)** for details of the supported syntax for <search-terms>.

The messages will be grouped and sorted based on the threading (all replies to a particular message will appear immediately after that message in date order). The output is not indented by default, but depth tags are printed so that proper indentation can be performed by a post-processor (such as the emacs interface to notmuch).

Supported options for **show** include

`--entire-thread=(true|false)` If true, **notmuch show** outputs all messages in the thread of any message matching the search terms; if false, it outputs only the matching messages. For `--format=json` and `--format=sexp` this defaults to true. For other formats, this defaults to false.

`--format=(text|json|sexp|mbox|raw)`

text (default for messages) The default plain-text format has all text-content MIME parts decoded. Various components in the output, (**message**, **header**, **body**, **attachment**, and MIME **part**), will be delimited by easily-parsed markers. Each marker consists of a Control-L character (ASCII decimal 12), the name of the marker, and then either an opening or closing brace, ('{' or '}'), to either open or close the component. For a multipart MIME message, these parts will be nested.

json The output is formatted with Javascript Object Notation (JSON). This format is more robust than the text format for automated processing. The nested structure of multipart MIME messages is reflected in nested JSON output. By default JSON output includes all messages in a matching thread; that is, by default, `--format=json` sets `--entire-thread`. The caller can disable this behaviour by setting

`--entire-thread=false`. The JSON output is always encoded as UTF-8 and any message content included in the output will be charset-converted to UTF-8.

sexp The output is formatted as the Lisp s-expression (sexp) equivalent of the JSON format above. Objects are formatted as property lists whose keys are keywords (symbols preceded by a colon). True is formatted as `t` and both false and null are formatted as `nil`. As for JSON, the s-expression output is always encoded as UTF-8.

mbox All matching messages are output in the traditional, Unix mbox format with each message being prefixed by a line beginning with "From " and a blank line separating each message. Lines in the message content beginning with "From " (preceded by zero or more '>' characters) have an additional '>' character added. This reversible escaping is termed "mboxrd" format and described in detail here:

http://homepage.ntlworld.com/jonathan.deboynepollard/FGA/mail-mbox-formats.html

raw (default if `--part` is given) Write the raw bytes of the given MIME part of a message to standard out. For this format, it is an error to specify a query that matches more than one message.

If the specified part is a leaf part, this outputs the body of the part after performing content transfer decoding (but no charset conversion). This is suitable for saving attachments, for example.

For a multipart or message part, the output includes the part headers as well as the body (including all child parts). No decoding is performed because multipart and message parts cannot have non-trivial content transfer encoding. Consumers of this may need to implement MIME decoding and similar functions.

`--format-version=N` Use the specified structured output format version. This is intended for programs that invoke **notmuch(1)** internally. If omitted, the latest supported version will be used.

`--part=N` Output the single decoded MIME part N of a single message. The search terms must match only a single message. Message parts are numbered in a depth-first walk of the message MIME structure, and are identified in the 'json', 'sexp' or 'text' output formats.

Note that even a message with no MIME structure or a single body part still has two MIME parts: part 0 is the whole message (headers and body) and part 1 is just the body.

`--verify` Compute and report the validity of any MIME cryptographic signatures found in the selected content (e.g., "multipart/signed" parts). Status of the signature will be reported (currently only supported with `--format=json` and `--format=sexp`), and the multipart/signed part will be replaced by the signed data.

`--decrypt=(false|auto|true|stash)` If `true`, decrypt any MIME encrypted parts found in the selected content (e.g., "multipart/encrypted" parts). Status of the decryption will be reported (currently only supported with `--format=json` and `--format=sexp`) and on successful decryption the multipart/encrypted part will be replaced by the decrypted content.

`stash` behaves like `true`, but upon successful decryption it will also stash the message's session key in the database, and index the cleartext of the message, enabling automatic decryption in the future.

If `auto`, and a session key is already known for the message, then it will be decrypted, but notmuch will not try to access the user's keys.

Use `false` to avoid even automatic decryption.

Non-automatic decryption (`stash` or `true`, in the absence of a stashed session key) expects a functioning **gpg-agent(1)** to provide any needed credentials. Without one, the decryption will fail.

Note: setting either `true` or `stash` here implies `--verify`.

Here is a table that summarizes each of these policies:

	false	auto	true	stash
Show cleartext if session key is already known		X	X	X
Use secret keys to show cleartext			X	X
Stash any newly recovered session keys, reindexing message if found				X

Note: `--decrypt=stash` requires write access to the database. Otherwise, `notmuch show` operates entirely in read-only mode.

Default: `auto`

`--exclude=(true|false)` Specify whether to omit threads only matching search.exclude_tags from the search results (the default) or not. In either case the excluded message will be marked with the exclude flag (except when output=mbox when there is nowhere to put the flag).

If `--entire-thread` is specified then complete threads are returned regardless (with the excluded flag being set when appropriate) but threads that only match in an excluded message are not returned when `--exclude=true`.

The default is `--exclude=true`.

`--body=(true|false)` If true (the default) **notmuch show** includes the bodies of the messages in the output; if false, bodies are omitted. `--body=false` is only implemented for the text, json and sexp formats and it is incompatible with `--part > 0`.

This is useful if the caller only needs the headers as body-less output is much faster and substantially smaller.

`--include-html` Include "text/html" parts as part of the output (currently only supported with `--format=text`, `--format=json` and `--format=sexp`). By default, unless `--part=N` is used to select a specific part or `--include-html` is used to include all "text/html" parts, no part with content type "text/html" is included in the output.

A common use of **notmuch show** is to display a single thread of email messages. For this, use a search term of "thread:<thread-id>" as can be seen in the first column of output from the **notmuch search** command.

17.3 EXIT STATUS

This command supports the following special exit status codes

`20` The requested format version is too old.

`21` The requested format version is too new.

17.4 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-tag(1)**

CHAPTER 18

notmuch-tag

18.1 SYNOPSIS

notmuch tag [options ...] +<*tag*>|-<*tag*> [–] <*search-term*> ...

notmuch tag –batch [–input=<*filename*>]

18.2 DESCRIPTION

Add/remove tags for all messages matching the search terms.

See **notmuch-search-terms(7)** for details of the supported syntax for <*search-term*>.

Tags prefixed by '+' are added while those prefixed by '-' are removed. For each message, tag changes are applied in the order they appear on the command line.

The beginning of the search terms is recognized by the first argument that begins with neither '+' nor '-'. Support for an initial search term beginning with '+' or '-' is provided by allowing the user to specify a "–" argument to separate the tags from the search terms.

notmuch tag updates the maildir flags according to tag changes if the **maildir.synchronize_flags** configuration option is enabled. See **notmuch-config(1)** for details.

Supported options for **tag** include

`--remove-all` Remove all tags from each message matching the search terms before applying the tag changes appearing on the command line. This means setting the tags of each message to the tags to be added. If there are no tags to be added, the messages will have no tags.

`--batch` Read batch tagging operations from a file (stdin by default). This is more efficient than repeated **notmuch tag** invocations. See *TAG FILE FORMAT* below for the input format. This option is not compatible with specifying tagging on the command line.

`--input=<filename>` Read input from given file, instead of from stdin. Implies `--batch`.

18.3 TAG FILE FORMAT

The input must consist of lines of the format:

+<*tag*>|-<*tag*> [. . .] [–] <*query*>

Each line is interpreted similarly to **notmuch tag** command line arguments. The delimiter is one or more spaces ' '. Any characters in <*tag*> **may** be hex-encoded with %NN where NN is the hexadecimal value of the character. To hex-encode a character with a multi-byte UTF-8 encoding, hex-encode each byte. Any spaces in <tag> **must** be hex-encoded as %20. Any characters that are not part of <*tag*> **must not** be hex-encoded.

In the future tag:"tag with spaces" style quoting may be supported for <*tag*> as well; for this reason all double quote characters in <*tag*> **should** be hex-encoded.

The <*query*> should be quoted using Xapian boolean term quoting rules: if a term contains whitespace or a close paren or starts with a double quote, it must be enclosed in double quotes (not including any prefix) and double quotes inside the term must be doubled (see below for examples).

Leading and trailing space ' ' is ignored. Empty lines and lines beginning with '#' are ignored.

18.3.1 EXAMPLE

The following shows a valid input to batch tagging. Note that only the isolated '*' acts as a wildcard. Also note the two different quotings of the tag **space in tags**

```
+winner *
+foo::bar%25 -- (One and Two) or (One and tag:winner)
+found::it -- tag:foo::bar%
# ignore this line and the next

+space%20in%20tags -- Two
# add tag '(tags)', among other stunts.
+crazy{ +(tags) +&are +#possible\ -- tag:"space in tags"
+match*crazy -- tag:crazy{
+some_tag -- id:"this is ""nauty)"""
```

18.4 SEE ALSO

notmuch(1), **notmuch-config(1)**, **notmuch-count(1)**, **notmuch-dump(1)**, **notmuch-hooks(5)**, **notmuch-insert(1)**, **notmuch-new(1)**, **notmuch-reply(1)**, **notmuch-restore(1)**, **notmuch-search(1)**, **notmuch-search-terms(7)**, **notmuch-show(1)**,

CHAPTER 19

Python Bindings

Pythonic API to the notmuch database.

19.1 Creating Objects

Only the *Database* object is meant to be created by the user. All other objects should be created from this initial object. Users should consider their signatures implementation details.

19.2 Errors

All errors occuring due to errors from the underlying notmuch database are subclasses of the *NotmuchError*. Due to memory management it is possible to try and use an object after it has been freed. In this case a *ObjectDestroyedError* will be raised.

19.3 Memory Management

Libnotmuch uses a hierarchical memory allocator, this means all objects have a strict parent-child relationship and when the parent is freed all the children are freed as well. This has some implications for these Python bindings as parent objects need to be kept alive. This is normally schielded entirely from the user however and the Python objects automatically make sure the right references are kept alive. It is however the reason the `BaseObject` exists as it defines the API all Python objects need to implement to work correctly.

19.4 Collections and Containers

Libnotmuch exposes nearly all collections of things as iterators only. In these python bindings they have sometimes been exposed as `collections.abc.Container` instances or subclasses of this like `collections.abc.Set` or `collections.abc.Mapping` etc. This gives a more natural API to work with, e.g. being able to treat tags as sets. However it does mean that the `__contains__()`, `__len__()` and frieds methods on these are usually more and essentially O(n) rather than O(1) as you might usually expect from Python containers.

class notmuch2.**AtomicContext**(*db*, *ptr_name*)

Context manager for atomic support.

This supports the notmuch_database_begin_atomic and notmuch_database_end_atomic API calls. The object can not be directly instantiated by the user, only via `Database.atomic`. It does keep a reference to the `Database` instance to keep the C memory alive.

Raises

- **XapianError** – When this is raised at enter time the atomic section is not active. When it is raised at exit time the atomic section is still active and you may need to try using `force_end()`.
- **ObjectDestroyedError** – if used after destroyed.

abort(*self*)

Abort the transaction.

Aborting a transaction will not commit any of the changes, but will also implicitly close the database.

force_end(*self*)

Force ending the atomic section.

This can only be called once __exit__ has been called. It will attept to close the atomic section (again). This is useful if the original exit raised an exception and the atomic section is still open. But things are pretty ugly by now.

Raises

- **XapianError** – If exiting fails, the atomic section is not ended.
- **UnbalancedAtomicError** – If the database was currently not in an atomic section.
- **ObjectDestroyedError** – if used after destroyed.

class notmuch2.**BinString**

A str subclass with binary data.

Most data in libnotmuch should be valid ASCII or valid UTF-8. However since it is a C library these are represented as bytestrings intead which means on an API level we can not guarantee that decoding this to UTF-8 will both succeed and be lossless. This string type converts bytes to unicode in a lossy way, but also makes the raw bytes available.

This object is a normal unicode string for most intents and purposes, but you can get the original bytestring back by calling `bytes()` on it.

classmethod from_cffi(*cls*, *cdata*)

Create a new string from a CFFI cdata pointer.

class notmuch2.**Database**(*path=None*, *mode=MODE.READ_ONLY*)

Toplevel access to notmuch.

A `Database` can be opened read-only or read-write. Modifications are not atomic by default, use `begin_atomic()` for atomic updates. If the underlying database has been modified outside of this class a `XapianError` will be raised and the instance must be closed and a new one created.

You can use an instance of this class as a context-manager.

Variables

- **MODE** – The mode a database can be opened with, an enumeration of `READ_ONLY` and `READ_WRITE`
- **SORT** – The sort order for search results, `OLDEST_FIRST`, `NEWEST_FIRST`, `MESSAGE_ID` or `UNSORTED`.
- **EXCLUDE** – Which messages to exclude from queries, `TRUE`, `FLAG`, `FALSE` or `ALL`. See the query documentation for details.
- **AddedMessage** – A namedtuple `(msg, dup)` used by `add()` as return value.
- **STR_MODE_MAP** – A map mapping strings to `MODE` items. This is used to implement the `ro` and `rw` string variants.
- **closed** – Boolean indicating if the database is closed or still open.

Parameters

- **path** (*str, bytes, os.PathLike or pathlib.Path*) – The directory of where the database is stored. If `None` the location will be read from the user's configuration file, respecting the `NOTMUCH_CONFIG` environment variable if set.
- **mode** (`MODE` or str.) – The mode to open the database in. One of `MODE.READ_ONLY` OR `MODE.READ_WRITE`. For convenience you can also use the strings `ro` for `MODE.READ_ONLY` and `rw` for `MODE.READ_WRITE`.

Raises

- **KeyError** – if an unknown mode string is used.
- **OSError** – or subclasses if the configuration file can not be opened.
- **configparser.Error** – or subclasses if the configuration file can not be parsed.
- **NotmuchError** – or subclasses for other failures.

add(*self*, *filename*, ***, *sync_flags=False*, *indexopts=None*)

Add a message to the database.

Add a new message to the notmuch database. The message is referred to by the pathname of the maildir file. If the message ID of the new message already exists in the database, this adds `pathname` to the list of list of files for the existing message.

Parameters

- **filename** (*str, bytes, os.PathLike or pathlib.Path.*) – The path of the file containing the message.
- **sync_flags** (*bool*) – Whether to sync the known maildir flags to notmuch tags. See `Message.flags_to_tags()` for details.
- **indexopts** (`IndexOptions` or *None*) – The indexing options, see `default_indexopts()`. Leave as *None* to use the default options configured in the database.

Returns A tuple where the first item is the newly inserted messages as a `Message` instance, and the second item is a boolean indicating if the message inserted was a duplicate. This is the namedtuple `AddedMessage(msg, dup)`.

Return type Database.AddedMessage

If an exception is raised, no message was added.

Raises

- **XapianError** – A Xapian exception occurred.

- **FileError** – The file referred to by `pathname` could not be opened.
- **FileNotEmailError** – The file referreed to by `pathname` is not recognised as an email message.
- **ReadOnlyDatabaseError** – The database is opened in READ_ONLY mode.
- **UpgradeRequiredError** – The database must be upgraded first.
- **ObjectDestroyedError** – if used after destroyed.

atomic(*self*)

Return a context manager to perform atomic operations.

The returned context manager can be used to perform atomic operations on the database.

Note: Unlinke a traditional RDBMS transaction this does not imply durability, it only ensures the changes are performed atomically.

Raises **ObjectDestroyedError** – if used after destroyed.

close(*self*)

Close the notmuch database.

Once closed most operations will fail. This can still be useful however to explicitly close a database which is opened read-write as this would otherwise stop other processes from reading the database while it is open.

Raises **ObjectDestroyedError** – if used after destroyed.

config

Return a mutable mapping with the settings stored in this database.

This returns an mutable dict-like object implementing the collections.abc.MutableMapping Abstract Base Class.

Return type Config

Raises **ObjectDestroyedError** – if used after destroyed.

count_messages(*self*, *query*, ***, *omit_excluded=EXCLUDE.TRUE*, *sort=SORT.UNSORTED*, *exclude_tags=None*)

Search the database for messages.

Returns An iterator over the messages found.

Return type MessageIter

Raises **ObjectDestroyedError** – if used after destroyed.

classmethod create(*cls*, *path=None*)

Create and open database in READ_WRITE mode.

This is creates a new notmuch database and returns an opened instance in `MODE.READ_WRITE` mode.

Parameters **path** (*str, bytes or os.PathLike*) – The directory of where the database is stored. If `None` the location will be read from the user's configuration file, respecting the `NOTMUCH_CONFIG` environment variable if set.

Raises

- **OSError** – or subclasses if the configuration file can not be opened.
- **configparser.Error** – or subclasses if the configuration file can not be parsed.

- **`NotmuchError`** – if the config file does not have the database.path setting.
- **`FileError`** – if the database already exists.

Returns The newly created instance.

`default_indexopts` (*self*)

Returns default index options for the database.

Raises **`ObjectDestroyedError`** – if used after destroyed.

Returns `IndexOptions`.

`static default_path` (*cfg_path=None*)

Return the path of the user's default database.

This reads the user's configuration file and returns the default path of the database.

Parameters **`cfg_path`** (*`str, bytes, os.PathLike or pathlib.Path.`*) – The pathname of the notmuch configuration file. If not specified tries to use the pathname provided in the **:env:'NOTMUCH_CONFIG'** environment variable and falls back to :file:**'**~/.notmuch-config.

Returns The path of the database, which does not necessarily exists.

Return type pathlib.Path

Raises

- **`OSError`** – or subclasses if the configuration file can not be opened.
- **`configparser.Error`** – or subclasses if the configuration file can not be parsed.

:raises NotmuchError if the config file does not have the database.path setting.

`find` (*self, msgid*)

Return the message matching the given message ID.

If a message with the given message ID is found a *`Message`* instance is returned. Otherwise a `LookupError` is raised.

Parameters **`msgid`** (*`str`*) – The message ID to look for.

Returns The message instance.

Return type Message

Raises

- **`LookupError`** – If no message was found.
- **`OutOfMemoryError`** – When there is no memory to allocate the message instance.
- **`XapianError`** – A Xapian exception ocurred.
- **`ObjectDestroyedError`** – if used after destroyed.

`get` (*self, filename*)

Return the *`Message`* given a pathname.

If a message with the given pathname exists in the database return the *`Message`* instance for the message. Otherwise raise a `LookupError` exception.

Parameters **`filename`** (*`str, bytes, os.PathLike or pathlib.Path`*) – The pathname of the message.

Returns The message instance.

Return type Message

Raises

- **LookupError** – If no message was found. This is also a subclass of `KeyError`.
- **OutOfMemoryError** – When there is no memory to allocate the message instance.
- **XapianError** – A Xapian exception ocurred.
- **ObjectDestroyedError** – if used after destroyed.

messages(*self*, *query*, ***, *omit_excluded=EXCLUDE.TRUE*, *sort=SORT.UNSORTED*, *exclude_tags=None*)

Search the database for messages.

Returns An iterator over the messages found.

Return type MessageIter

Raises

- **OutOfMemoryError** – if no memory is available to allocate the query.
- **ObjectDestroyedError** – if used after destroyed.

needs_upgrade

Whether the database should be upgraded.

If *True* the database can be upgraded using `upgrade()`. Not doing so may result in some operations raising `UpgradeRequiredError`.

A read-only database will never be upgradable.

Raises ObjectDestroyedError – if used after destroyed.

path

The pathname of the notmuch database.

This is returned as a `pathlib.Path` instance.

Raises ObjectDestroyedError – if used after destroyed.

remove(*self*, *filename*)

Remove a message from the notmuch database.

Removing a message which is not in the database is just a silent nop-operation.

Parameters filename (*str, bytes, os.PathLike or pathlib.Path.*) – The pathname of the file containing the message to be removed.

Returns True if the message is still in the database. This can happen when multiple files contain the same message ID. The true/false distinction is fairly arbitrary, but think of it as `dup = db.remove_message(name); if dup: ...`.

Return type bool

Raises

- **XapianError** – A Xapian exception ocurred.
- **ReadOnlyDatabaseError** – The database is opened in READ_ONLY mode.
- **UpgradeRequiredError** – The database must be upgraded first.
- **ObjectDestroyedError** – if used after destroyed.

revision (*self*)
The currently committed revision in the database.

Returned as a `(revision, uuid)` namedtuple.

Raises **ObjectDestroyedError** – if used after destroyed.

tags
Return an immutable set with all tags used in this database.

This returns an immutable set-like object implementing the collections.abc.Set Abstract Base Class. Due to the underlying libnotmuch implementation some operations have different performance characteristics then plain set objects. Mainly any lookup operation is O(n) rather then O(1).

Normal usage treats tags as UTF-8 encoded unicode strings so they are exposed to Python as normal unicode string objects. If you need to handle tags stored in libnotmuch which are not valid unicode do check the *ImmutableTagSet* docs for how to handle this.

Return type ImmutableTagSet

Raises **ObjectDestroyedError** – if used after destroyed.

upgrade (*self*, *progress_cb=None*)
Upgrade the database to the latest version.

Upgrade the database, optionally with a progress callback which should be a callable which will be called with a floating point number in the range of [0.0 .. 1.0].

version
The database format version.

This is a positive integer.

Raises **ObjectDestroyedError** – if used after destroyed.

class notmuch2.**DbRevision** (*rev*, *uuid*)
A database revision.

The database revision number increases monotonically with each commit to the database. Which means user-visible changes can be ordered. This object is sortable with other revisions. It carries the UUID of the database to ensure it is only ever compared with revisions from the same database.

rev
The revision number, a positive integer.

uuid
The UUID of the database, consider this opaque.

exception notmuch2.**DuplicateMessageIdError** (*status=None*, *message=None*)
Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

exception notmuch2.**FileError** (*status=None*, *message=None*)
Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

exception `notmuch2.`**`FileNotEmailError`**(*status=None*, *message=None*)
Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

exception `notmuch2.`**`IllegalArgumentError`**(*status=None*, *message=None*)
Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

class `notmuch2.`**`ImmutableTagSet`**(*parent*, *ptr_name*, *cffi_fn*)
The tags associated with a message thread or whole database.

Both a thread as well as the database expose the union of all tags in messages associated with them. This exposes these as a `collections.abc.Set` object.

Note that due to the underlying notmuch API the performance of the implementation is not the same as you would expect from normal sets. E.g. the `__contains__()` and `__len__()` are O(n) rather then O(1).

Tags are internally stored as bytestrings but normally exposed as unicode strings using the UTF-8 encoding and the *ignore* decoder error handler. However the *`iter()`* method can be used to return tags as bytestrings or using a different error handler.

Note that when doing arithmetic operations on tags, this class will return a plain normal set as it is no longer associated with the message.

Parameters

- **`parent`** – the parent object
- **`ptr_name`** – the name of the attribute on the parent which will return the memory pointer. This allows this object to access the pointer via the parent's descriptor and thus trigger `MemoryPointer`'s memory safety.
- **`cffi_fn`** – the callable CFFI wrapper to retrieve the tags iter. This can be one of notmuch_database_get_all_tags, notmuch_thread_get_tags or notmuch_message_get_tags.

`iter`(*self*, ***, *encoding=None*, *errors='strict'*)
Aternate iterator constructor controlling string decoding.

Tags are stored as bytes in the notmuch database, in Python it's easier to work with unicode strings and thus is what the normal iterator returns. However this method allows you to specify how you would like to get the tags, defaulting to the bytestring representation instead of unicode strings.

Parameters

- **encoding** – Which codec to use. The default *None* does not decode at all and will return the unmodified bytes. Otherwise this is passed on to `str.decode()`.
- **errors** – If using a codec, this is the error handler. See `str.decode()` to which this is passed on.

Raises **NullPointerError** – When things do not go as planned.

class notmuch2.**Message**(*parent*, *msg_p*, ***, *db*)

An email message stored in the notmuch database retrieved via a query.

This should not be directly created, instead it will be returned by calling methods on *Database*. A message keeps a reference to the database object since the database object can not be released while the message is in use.

Note that this represents a message in the notmuch database. For full email functionality you may want to use the `email` package from Python's standard library. You could e.g. create this as such:

```
notmuch_msg = db.get_message(msgid)  # or from a query
parser = email.parser.BytesParser(policy=email.policy.default)
with notmuch_msg.path.open('rb) as fp:
    email_msg = parser.parse(fp)
```

Most commonly the functionality provided by notmuch is sufficient to read email however.

Messages are considered equal when they have the same message ID. This is how libnotmuch treats messages as well, the `pathnames()` function returns multiple results for duplicates.

Parameters

- **parent** (*NotmuchObject*) – The parent object. This is probably one off a *Database*, `Thread` or `Query`.
- **db** (*Database*) – The database instance this message is associated with. This could be the same as the parent.
- **msg_p** (*<cdata>*) – The C pointer to the `notmuch_message_t`.
- **dup** (*None or bool*) – Whether the message was a duplicate on insertion.

date

The message date as an integer.

The time the message was sent as an integer number of seconds since the *epoch*, 1 Jan 1970. This is derived from the message's header, you can get the original header value with *header()*.

Raises **ObjectDestroyedError** – if used after destroyed.

excluded

Indicates whether this message was excluded from the query.

When a message is created from a search, sometimes messages that where excluded by the search query could still be returned by it, e.g. because they are part of a thread matching the query. the `Database.query()` method allows these messages to be flagged, which results in this property being set to *True*.

Raises **ObjectDestroyedError** – if used after destroyed.

filenames(*self*)

Return an iterator of all files for this message.

If multiple files contained the same message ID they will all be returned here. The files are returned as intances of `pathlib.Path`.

Returns Iterator yielding `pathlib.Path` instances.

Return type iter

Raises **`ObjectDestroyedError`** – if used after destroyed.

`filenamesb`(*self*)

Return an iterator of all files for this message.

This is like `pathnames()` but the files are returned as byte objects instead.

Returns Iterator yielding `bytes` instances.

Return type iter

Raises **`ObjectDestroyedError`** – if used after destroyed.

`frozen`(*self*)

Context manager to freeze the message state.

This allows you to perform atomic tag updates:

```
with msg.frozen():
    msg.tags.clear()
    msg.tags.add('foo')
```

Using This would ensure the message never ends up with no tags applied at all.

It is safe to nest calls to this context manager.

Raises

- **`ReadOnlyDatabaseError`** – if the database is opened in read-only mode.
- **`UnbalancedFreezeThawError`** – if you somehow managed to call __exit__ of this context manager more than once. Why did you do that?
- **`ObjectDestroyedError`** – if used after destroyed.

`ghost`

Indicates whether this message is a ghost message.

A ghost message if a message which we know exists, but it has no files or content associated with it. This can happen if it was referenced by some other message. Only the *`messageid`* and *`threadid`* attributes are valid for it.

Raises **`ObjectDestroyedError`** – if used after destroyed.

`header`(*self*, *name*)

Return the value of the named header.

Returns the header from notmuch, some common headers are stored in the database, others are read from the file. Headers are returned with their newlines stripped and collapsed concatenated together if they occur multiple times. You may be better off using the standard library email package's `email.message_from_file(msg.path.open())` if that is not sufficient for you.

Parameters **`header`** (*`str or bytes`*) – Case-insensitive header name to retrieve.

Returns The header value, an empty string if the header is not present.

Return type str

Raises

- **`LookupError`** – if the header is not present.
- **`NullPointerError`** – For unexpected notmuch errors.

- **ObjectDestroyedError** – if used after destroyed.

messageid

The message ID as a string.

The message ID is decoded with the ignore error handler. This is fine as long as the message ID is well formed. If it is not valid ASCII then this will be lossy. So if you need to be able to write the exact same message ID back you should use `messageidb`.

Note that notmuch will decode the message ID value and thus strip off the surrounding < and > characters. This is different from Python's `email` package behaviour which leaves these characters in place.

Returns The message ID.

Return type *BinString*, this is a normal str but calling bytes() on it will return the original bytes used to create it.

Raises **ObjectDestroyedError** – if used after destroyed.

path

A pathname of the message as a pathlib.Path instance.

If multiple files in the database contain the same message ID this will be just one of the files, chosen at random.

Raises **ObjectDestroyedError** – if used after destroyed.

pathb

A pathname of the message as a bytes object.

See *path* for details, this is the same but does return the path as a bytes object which is faster but less convenient.

Raises **ObjectDestroyedError** – if used after destroyed.

properties

A map of arbitrary key-value pairs associated with the message.

Be aware that properties may be used by other extensions to store state in. So delete or modify with care.

The properties map is somewhat special. It is essentially a multimap-like structure where each key can have multiple values. Therefore accessing a single item using `PropertiesMap.get()` or `PropertiesMap.__getitem__()` will only return you the *first* item if there are multiple and thus are only recommended if you know there to be only one value.

Instead the map has an additional `PropertiesMap.all()` method which can be used to retrieve all properties of a given key. This method also allows iterating of a a subset of the keys starting with a given prefix.

replies(*self*)

Return an iterator of all replies to this message.

This method will only work if the message was created from a thread. Otherwise it will yield no results.

Returns An iterator yielding *Message* instances.

Return type MessageIter

tags

The tags associated with the message.

This behaves as a set. But removing and adding items to the set removes and adds them to the message in the database.

Raises

- **`ReadOnlyDatabaseError`** – When manipulating tags on a database opened in read-only mode.
- **`ObjectDestroyedError`** – if used after destroyed.

`threadid`

The thread ID.

The thread ID is decoded with the surrogateescape error handler so that it is possible to reconstruct the original thread ID if it is not valid UTF-8.

Returns The thread ID.

Return type *`BinString`*, this is a normal str but calling bytes() on it will return the original bytes used to create it.

Raises **`ObjectDestroyedError`** – if used after destroyed.

`class` `notmuch2.`**`MutableTagSet`**(*parent*, *ptr_name*, *cffi_fn*)

The tags associated with a message.

This is a `collections.abc.MutableSet` object which can be used to manipulate the tags of a message.

Note that due to the underlying notmuch API the performance of the implementation is not the same as you would expect from normal sets. E.g. the `in` operator and variants are O(n) rather then O(1).

Tags are bytestrings and calling `iter()` will return an iterator yielding bytestrings. However the `iter()` method can be used to return tags as unicode strings, while all other operations accept either byestrings or unicode strings. In case unicode strings are used they will be encoded using utf-8 before being passed to notmuch.

`add`(*self*, *tag*)

Add a tag to the message.

Parameters

- **`tag`** (*`str or bytes. A str will be encoded using UTF-8.`*) – The tag to add.
- **`sync_flags`** – Whether to sync the maildir flags with the new set of tags. Leaving this as *None* respects the configuration set in the database, while *True* will always sync and *False* will never sync.
- **`sync_flags`** – NoneType or bool

Raises

- **`TypeError`** – If the tag is not a valid type.
- **`TagTooLongError`** – If the added tag exceeds the maximum lenght, see `notmuch_cffi.NOTMUCH_TAG_MAX`.
- **`ReadOnlyDatabaseError`** – If the database is opened in read-only mode.

`clear`(*self*)

Remove all tags from the message.

Raises **`ReadOnlyDatabaseError`** – If the database is opened in read-only mode.

`discard`(*self*, *tag*)

Remove a tag from the message.

Parameters

- **`tag`** (*`str of bytes. A str will be encoded using UTF-8.`*) – The tag to remove.

- **sync_flags** – Whether to sync the maildir flags with the new set of tags. Leaving this as *None* respects the configuration set in the database, while *True* will always sync and *False* will never sync.
- **sync_flags** – NoneType or bool

Raises

- **TypeError** – If the tag is not a valid type.
- **TagTooLongError** – If the tag exceeds the maximum lenght, see `notmuch_cffi.NOTMUCH_TAG_MAX`.
- **ReadOnlyDatabaseError** – If the database is opened in read-only mode.

from_maildir_flags (*self*)

Update the tags based on the state in the message's maildir flags.

This function examines the filenames of 'message' for maildir flags, and adds or removes tags on 'message' as follows when these flags are present:

Flag Action if present —- ————— 'D' Adds the "draft" tag to the message 'F' Adds the "flagged" tag to the message 'P' Adds the "passed" tag to the message 'R' Adds the "replied" tag to the message 'S' Removes the "unread" tag from the message

For each flag that is not present, the opposite action (add/remove) is performed for the corresponding tags.

Flags are identified as trailing components of the filename after a sequence of ":2,".

If there are multiple filenames associated with this message, the flag is considered present if it appears in one or more filenames. (That is, the flags from the multiple filenames are combined with the logical OR operator.)

to_maildir_flags (*self*)

Update the message's maildir flags based on the notmuch tags.

If the message's filename is in a maildir directory, that is a directory named `new` or `cur`, and has a valid maildir filename then the flags will be added as such:

'D' if the message has the "draft" tag 'F' if the message has the "flagged" tag 'P' if the message has the "passed" tag 'R' if the message has the "replied" tag 'S' if the message does not have the "unread" tag

Any existing flags unmentioned in the list above will be preserved in the renaming.

Also, if this filename is in a directory named "new", rename it to be within the neighboring directory named "cur".

In case there are multiple files associated with the message all filenames will get the same logic applied.

exception notmuch2.**NotmuchError** (*status=None*, *message=None*)

Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

classmethod exc_type (*cls*, *status*)

Return correct exception type for notmuch status.

class notmuch2.**NotmuchObject**(*parent*, **args*, ***kwargs*)

Base notmuch object syntax.

This base class exists to define the memory management handling required to use the notmuch library. It is meant as an interface definition rather than a base class, though you can use it as a base class to ensure you don't forget part of the interface. It only concerns you if you are implementing this package itself rather then using it.

libnotmuch uses a hierarchical memory allocator, where freeing the memory of a parent object also frees the memory of all child objects. To make this work seamlessly in Python this package keeps references to parent objects which makes them stay alive correctly under normal circumstances. When an object finally gets deleted the `__del__()` method will be called to free the memory.

However during some peculiar situations, e.g. interpreter shutdown, it is possible for the `__del__()` method to have been called, whele there are still references to an object. This could result in child objects asking their memeory to be freed after the parent has already freed the memory, making things rather unhappy as double frees are not taken lightly in C. To handle this case all objects need to follow the same protocol to destroy themselves, see `destroy()`.

Once an object has been destroyed trying to use it should raise the *`ObjectDestroyedError`* exception. For this see also the convenience `MemoryPointer` descriptor in this module which can be used as a pointer to libnotmuch memory.

alive

Whether the object is still alive.

This indicates whether the object is still alive. The first thing this needs to check is whether the parent object is still alive, if it is not then this object can not be alive either. If the parent is alive then it depends on whether the memory for this object has been freed yet or not.

exception notmuch2.**NullPointerError**(*status=None*, *message=None*)

Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

exception notmuch2.**ObjectDestroyedError**(*status=None*, *message=None*)

The object has already been destroyed and it's memory freed.

This occurs when `destroy()` has been called on the object but you still happen to have access to the object. This should not normally occur since you should never call `destroy()` by hand.

exception notmuch2.**OutOfMemoryError**(*status=None*, *message=None*)

Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

exception notmuch2.**PathError**(*status=None*, *message=None*)

Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

exception `notmuch2.`**`ReadOnlyDatabaseError`**(*status=None*, *message=None*)
Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

exception `notmuch2.`**`TagTooLongError`**(*status=None*, *message=None*)
Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

class `notmuch2.`**`TagsIter`**(*parent*, *tags_p*, ***, *encoding=None*, *errors='strict'*)
Iterator over tags.

This is only an interator, not a container so calling `__iter__()` does not return a new, replenished iterator but only itself.

Parameters

- **`parent`** – The parent object to keep alive.
- **`tags_p`** – The CFFI pointer to the C-level tags iterator.
- **`encoding`** – Which codec to use. The default *None* does not decode at all and will return the unmodified bytes. Otherwise this is passed on to `str.decode()`.
- **`errors`** – If using a codec, this is the error handler. See `str.decode()` to which this is passed on.

Raises **`ObjectDestroyedError`** – if used after destroyed.

exception `notmuch2.`**`UnbalancedAtomicError`**(*status=None*, *message=None*)
Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

exception `notmuch2.`**`UnbalancedFreezeThawError`**(*status=None*, *message=None*)

Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

exception `notmuch2.`**`UnsupportedOperationError`**(*status=None*, *message=None*)

Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

exception `notmuch2.`**`UpgradeRequiredError`**(*status=None*, *message=None*)

Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

exception `notmuch2.`**`XapianError`**(*status=None*, *message=None*)

Base exception for errors originating from the notmuch library.

Usually this will have two attributes:

Status This is a numeric status code corresponding to the error code in the notmuch library. This is normally fairly meaningless, it can also often be `None`. This exists mostly to easily create new errors from notmuch status codes and should not normally be used by users.

Message A user-facing message for the error. This can occasionally also be `None`. Usually you'll want to call `str()` on the error object instead to get a sensible message.

CHAPTER 20

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index