

---

# **alot Documentation**

*Release 0.7*

**Patrick Totzke**

**Oct 10, 2018**



---

# Contents

---

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Usage</b>	<b>5</b>
2.1	Command-Line Invocation . . . . .	5
2.2	UNIX Signals . . . . .	6
2.3	First Steps in the UI . . . . .	6
2.4	Commands . . . . .	6
2.5	Cryptography . . . . .	15
<b>3</b>	<b>Configuration</b>	<b>17</b>
3.1	Configuration Options . . . . .	17
3.2	Accounts . . . . .	27
3.3	Contacts Completion . . . . .	31
3.4	Key Bindings . . . . .	32
3.5	Hooks . . . . .	35
3.6	Theming . . . . .	38
<b>4</b>	<b>API and Development</b>	<b>43</b>
4.1	Overview . . . . .	43
4.2	Email Database . . . . .	43
4.3	User Interface . . . . .	44
4.4	User Settings . . . . .	45
4.5	Utils . . . . .	47
4.6	Commands . . . . .	47
4.7	Crypto . . . . .	47
<b>5</b>	<b>FAQ</b>	<b>49</b>
<b>6</b>	<b>Manpage</b>	<b>51</b>
6.1	Synopsis . . . . .	51
6.2	Description . . . . .	51
6.3	Options . . . . .	51
6.4	Commands . . . . .	52
6.5	Usage . . . . .	52
6.6	UNIX Signals . . . . .	52
6.7	See Also . . . . .	52



Alot is a terminal-based mail user agent for the notmuch mail system. It features a modular and command prompt driven interface to provide a full MUA experience as an alternative to the Emacs mode shipped with notmuch.



### dependencies

Alot depends on recent versions of notmuch and urwid. Note that due to restrictions on argparse and subprocess, you need to run `python 3.5` (see *faq*). A full list of dependencies is below:

- `libmagic` and `python bindings`, 5.04
- `configobj`, 4.7.0
- `libnotmuch` and its `python bindings`, 0.13
- `urwid toolkit`, 1.3.0
- `urwidtrees`, 1.0
- `gpg` and its `python bindings`, 1.9.0

---

**Note:** `urwidtrees` was only recently detached from `alot` and is not widely available as a separate package. You can install it e.g., via `pip` directly from github:

```
pip install --user https://github.com/pazz/urwidtrees/archive/master.zip
```

---

On debian/ubuntu the rest are packaged as:

```
python-setuptools python-magic python-configobj python-notmuch python-urwid python-gpg
```

On fedora/redhat these are packaged as:

```
python-setuptools python-magic python-configobj python-notmuch python-urwid python-gpg
```

Alot uses `mailcap` to look up mime-handler for inline rendering and opening of attachments. For a full description of the maicap protocol consider the manpage `mailcap(5)` or **RFC 1524**. To avoid surprises you should at least have an inline renderer (`copiousoutput`) set up for `text/html`, i.e. have something like this in your `~/.mailcap`:

```
text/html; w3m -dump -o document_charset={charset} '%s'; nametemplate=%s.html; ↵  
↪copiousoutput
```

### get and install alot

You can use *pip* to install directly from GitHub:

```
$ pip install --user https://github.com/pazz/alot/archive/master.zip
```

Don't have pip installed? Just download and extract, then run:

```
python setup.py install --user
```

Make sure `~/local/bin` is in your `PATH`. For system-wide installation omit the `-user` flag and call with the respective permissions.

### generate manual and manpage

To generate the documentation you need [sphinx](#), *1.07* installed. Go to `docs/` and do a:

```
make html  
make man
```

to generate the user manual and a man page. Both will end up in their respective subfolders in `docs/build`.



## 2.1 Command-Line Invocation

### Synopsis

alot [options ...] [subcommand]

### Options

- r, --read-only** open notmuch database in read-only mode
- c FILENAME, --config=FILENAME** configuration file (default: `~/config/alot/config`)
- n FILENAME, --notmuch-config=FILENAME** notmuch configuration file (default: `$NOTMUCH_CONFIG` or `~/notmuch-config`)
- C COLOURS, --colour-mode=COLOURS** number of colours to use on the terminal; must be 1, 16 or 256 (default: configuration option `colourmode` or 256)
- p PATH, --mailindex-path=PATH** path to notmuch index
- d LEVEL, --debug-level=LEVEL** debug level; must be one of debug, info, warning or error (default: info)
- l FILENAME, --logfile=FILENAME** log file (default: `/dev/null`)
- h, --help** display help and exit
- v, --version** output version information and exit

### Commands

alot can be invoked with an optional subcommand from the command line. Those have their own parameters (see e.g. `alot search -help`). The following commands are available.

**search** start in a search buffer using the query string provided as parameter (see *notmuch-search-terms(7)*)

**compose** compose a new message

**bufferlist** start with only a bufferlist buffer open

**taglist** start with only a taglist buffer open

**namedqueries** start with list of named queries

**pyshell** start the interactive python shell inside alot

## 2.2 UNIX Signals

**SIGUSR1** Refreshes the current buffer.

**SIGINT** Shuts down the user interface.

## 2.3 First Steps in the UI

The arrow keys, *page-up/down*, *j*, *k* and *Space* can be used to move the focus. *Escape* cancels prompts and *Enter* selects. Hit *:* at any time and type in commands to the prompt.

The interface shows one buffer at a time, you can use *Tab* and *Shift-Tab* to switch between them, close the current buffer with *d* and list them all with *;*.

The buffer type or *mode* (displayed at the bottom left) determines which prompt commands are available. Usage information on any command can be listed by typing *help YOURCOMMAND* to the prompt. The keybindings for the current mode are listed upon pressing *?*.

## 2.4 Commands

Alot interprets user input as command line strings given via its prompt or *bound to keys* in the config. Command lines are semi-colon separated command strings, each of which starts with a command name and possibly followed by arguments.

See the sections below for which commands are available in which (UI) mode. *global* commands are available independently of the mode.

*Global commands* globally available commands

*Commands in bufferlist mode* commands while listing active buffers

*Commands in envelope mode* commands during message composition

*Commands in namedqueries mode* commands while listing all named queries from the notmuch database

*Commands in search mode* commands available when showing thread search results

*Commands in taglist mode* commands while listing all tagstrings present in the notmuch database

*Commands in thread mode* commands available while displaying a thread

### 2.4.1 Global commands

The following commands are available globally:

**bclose**

close a buffer

**optional arguments**

—**redraw** redraw current buffer after command has finished

—**force** never ask for confirmation

**bnext**

focus next buffer

**bprevious**

focus previous buffer

**buffer**

focus buffer with given index

**argument** buffer index to focus

**bufferlist**

open a list of active buffers

**call**

execute python code

**argument** python command string to call

**compose**

compose a new email

**argument** None

**optional arguments**

—**sender** sender

—**template** path to a template message file

—**tags** comma-separated list of tags to apply to message

—**subject** subject line

—**to** recipients

—**cc** copy to

—**bcc** blind copy to

—**attach** attach files

—**omit\_signature** do not add signature

—**spawn** spawn editor in new terminal

**exit**

shut down cleanly

**flush**

flush write operations or retry until committed

**help**

display help for a command (use 'bindings' to display all keybindings interpreted in current mode)

**argument** command or ‘bindings’

**move**

move focus in current buffer

**argument** up, down, [half]page up, [half]page down, first, last

**namedqueries**

opens named queries buffer

**prompt**

prompts for commandline and interprets it upon select

**argument** initial content

**pyshell**

open an interactive python shell for introspection

**refresh**

refresh the current buffer

**reload**

reload all configuration files

**removequery**

removes a “named query” from the database

**argument** alias to remove

**optional arguments**

—**no-flush** postpone a writeout to the index (defaults to: ‘True’)

**repeat**

repeat the command executed last time

**savequery**

store query string as a “named query” in the database

**positional arguments** 0: alias to use for query string 1: query string to store

**optional arguments**

—**no-flush** postpone a writeout to the index (defaults to: ‘True’)

**search**

open a new search buffer. Search obeys the notmuch *search.exclude\_tags* setting.

**argument** search string

**optional arguments**

—**sort** sort order; valid choices are: ‘oldest\_first’, ‘newest\_first’, ‘message\_id’, ‘unsorted’

**shellescape**

run external command

**argument** command line to execute

**optional arguments**

—**spawn** run in terminal window

—**thread** run in separate thread

—**refocus** refocus current buffer after command has finished

**taglist**

opens taglist buffer

**optional arguments**

—tags tags to display

## 2.4.2 Commands in *bufferlist* mode

The following commands are available in bufferlist mode:

**close**

close focussed buffer

**open**

focus selected buffer

## 2.4.3 Commands in *envelope* mode

The following commands are available in envelope mode:

**attach**

attach files to the mail

**argument** file(s) to attach (accepts wildcards)

**edit**

edit mail

**optional arguments**

—spawn spawn editor in new terminal

—refocus refocus envelope after editing (defaults to: 'True')

**encrypt**

request encryption of message before sendout

**argument** keyid of the key to encrypt with

**optional arguments**

—trusted only add trusted keys

**refine**

prompt to change the value of a header

**argument** header to refine

**retag**

set message tags

**argument** comma separated list of tags

**rmencrypt**

do not encrypt to given recipient key

**argument** keyid of the key to encrypt with

**save**

save draft

**send**

send mail

**set**

set header value

**positional arguments** 0: header to refine 1: value

**optional arguments**

—**append** keep previous values

**sign**

mark mail to be signed before sending

**argument** which key id to use

**tag**

add tags to message

**argument** comma separated list of tags

**toggleencrypt**

toggle if message should be encrypted before sendout

**argument** keyid of the key to encrypt with

**optional arguments**

—**trusted** only add trusted keys

**toggleheaders**

toggle display of all headers

**togglesign**

toggle sign status

**argument** which key id to use

**toggletags**

flip presence of tags on message

**argument** comma separated list of tags

**unattach**

remove attachments from current envelope

**argument** which attached file to remove

**unencrypt**

remove request to encrypt message before sending

**unset**

remove header field

**argument** header to refine

**unsign**

mark mail not to be signed before sending

**untag**

remove tags from message

**argument** comma separated list of tags

## 2.4.4 Commands in *namedqueries* mode

The following commands are available in *namedqueries* mode:

### **select**

search for messages with selected query

**argument** additional filter to apply to query

## 2.4.5 Commands in *search* mode

The following commands are available in *search* mode:

### **move**

move focus in search buffer

**argument** last

### **refine**

refine query

**argument** search string

### **optional arguments**

—**sort** sort order; valid choices are: ‘oldest\_first’, ‘newest\_first’, ‘message\_id’, ‘unsorted’

### **refineprompt**

prompt to change this buffers querystring

### **retag**

set tags of all messages in the thread that match the current query

**argument** comma separated list of tags

### **optional arguments**

—**no-flush** postpone a writeout to the index (defaults to: ‘True’)

—**all** retag all messages in search result

### **retagprompt**

prompt to retag selected thread’s or message’s tags

### **savequery**

store query string as a “named query” in the database. This falls back to the current search query in search buffers.

**positional arguments** 0: alias to use for query string 1: query string to store

### **optional arguments**

—**no-flush** postpone a writeout to the index (defaults to: ‘True’)

### **select**

open thread in a new buffer

### **sort**

set sort order

**argument** sort order; valid choices are: ‘oldest\_first’, ‘newest\_first’, ‘message\_id’, ‘unsorted’

### **tag**

add tags to all messages in the thread that match the current query

**argument** comma separated list of tags

**optional arguments**

—**no-flush** postpone a writeout to the index (defaults to: ‘True’)

—**all** retag all messages in search result

**toggletags**

flip presence of tags on this thread: a tag is considered present and will be removed if at least one message in this thread is tagged with it

**argument** comma separated list of tags

**optional arguments**

—**no-flush** postpone a writeout to the index (defaults to: ‘True’)

**untag**

remove tags from all messages in the thread that match the query

**argument** comma separated list of tags

**optional arguments**

—**no-flush** postpone a writeout to the index (defaults to: ‘True’)

—**all** retag all messages in search result

## 2.4.6 Commands in *taglist* mode

The following commands are available in taglist mode:

**select**

search for messages with selected tag

## 2.4.7 Commands in *thread* mode

The following commands are available in thread mode:

**bounce**

directly re-send selected message

**editnew**

edit message in as new

**optional arguments**

—**spawn** open editor in new window

**fold**

fold message(s)

**argument** query used to filter messages to affect

**forward**

forward message

**optional arguments**

—**attach** attach original mail

—**spawn** open editor in new window



**indent**

change message/reply indentation

**argument** None

**move**

move focus in current buffer

**argument** up, down, [half]page up, [half]page down, first, last, parent, first reply, last reply, next sibling, previous sibling, next, previous, next unfolded, previous unfolded, next NOTMUCH\_QUERY, previous NOTMUCH\_QUERY

**pipeto**

pipe message(s) to stdin of a shellcommand

**argument** shellcommand to pipe to

**optional arguments**

- all** pass all messages
- format** output format; valid choices are: ‘raw’, ‘decoded’, ‘id’, ‘filepath’ (defaults to: ‘raw’)
- separately** call command once for each message
- background** don’t stop the interface
- add\_tags** add ‘Tags’ header to the message
- shell** let the shell interpret the command
- notify\_stdout** display cmd’s stdout as notification
- field\_key** mailcap field key for decoding (defaults to: ‘copiousoutput’)

**print**

print message(s)

**optional arguments**

- all** print all messages
- raw** pass raw mail string
- separately** call print command once for each message
- add\_tags** add ‘Tags’ header to the message

**remove**

remove message(s) from the index

**optional arguments**

- all** remove whole thread

**reply**

reply to message

**optional arguments**

- all** reply to all
- list** reply to list
- spawn** open editor in new window

**retag**

set message(s) tags.

**argument** comma separated list of tags

**optional arguments**

—**all** tag all messages in thread

—**no-flush** postpone a writeout to the index (defaults to: ‘True’)

**retagprompt**

prompt to retag selected thread’s or message’s tags

**save**

save attachment(s)

**argument** path to save to

**optional arguments**

—**all** save all attachments

**select**

**select focussed element:**

- if it is a message summary, toggle visibility of the message;
- if it is an attachment line, open the attachment

**tag**

add tags to message(s)

**argument** comma separated list of tags

**optional arguments**

—**all** tag all messages in thread

—**no-flush** postpone a writeout to the index (defaults to: ‘True’)

**toggleheaders**

display all headers

**argument** query used to filter messages to affect

**togglesource**

display message source

**argument** query used to filter messages to affect

**toggletags**

flip presence of tags on message(s)

**argument** comma separated list of tags

**optional arguments**

—**all** tag all messages in thread

—**no-flush** postpone a writeout to the index (defaults to: ‘True’)

**unfold**

unfold message(s)

**argument** query used to filter messages to affect

**untag**

remove tags from message(s)

**argument** comma separated list of tags

**optional arguments**

- all** tag all messages in thread
- no-flush** postpone a writeout to the index (defaults to: ‘True’)

## 2.5 Cryptography

Alot has built in support for constructing signed and/or encrypted mails according to PGP/MIME ([RFC 3156](#), [RFC 3156](#)) via gnupg. It does however rely on a running *gpg-agent* to handle password entries.

---

**Note:** You need to have *gpg-agent* running to use GPG with alot!

*gpg-agent* will handle passphrase entry in a secure and configurable way, and it will cache your passphrase for some time so you don’t have to enter it over and over again. For details on how to set this up we refer to [gnupg’s manual](#).

---

### Signing outgoing emails

You can use the commands *sign*, *unsign* and *togglesign* in envelope mode to determine if you want this mail signed and if so, which key to use. To specify the key to use you may pass a hint string as argument to the *sign* or *togglesign* command. This hint would typically be a fingerprint or an email address associated (by gnupg) with a key.

Signing (and hence passwd entry) will be done at most once shortly before a mail is sent.

In case no key is specified, alot will leave the selection of a suitable key to gnupg so you can influence that by setting the *default-key* option in `~/ .gnupg/gpg.conf` accordingly.

You can set the default to-sign bit and the key to use for each *account* individually using the options *sign\_by\_default* and *gpg\_key*.

### Encrypt outgoing emails

You can use the commands *encrypt*, *unencrypt* and *toggleencrypt* and in envelope mode to ask alot to encrypt the mail before sending. The *encrypt* command accepts an optional hint string as argument to determine the key of the recipient.

You can set the default to-encrypt bit for each *account* individually using the option *encrypt\_by\_default*.

---

**Note:** If you want to access encrypt mail later it is useful to add yourself to the list of recipients when encrypting with gpg (not the recipients whom mail is actually send to). The simplest way to do this is to use the *encrypt-to* option in the `~/ .gnupg/gpg.conf`. But you might have to specify the correct encryption subkey otherwise gpg seems to throw an error.

---

The arrow keys, *page-up/down*, *j*, *k* and *Space* can be used to move the focus. *Escape* cancels prompts and *Enter* selects. Hit `:` at any time and type in commands to the prompt.

The interface shows one buffer at a time, you can use *Tab* and *Shift-Tab* to switch between them, close the current buffer with *d* and list them all with `;`.

The buffer type or *mode* (displayed at the bottom left) determines which prompt commands are available. Usage information on any command can be listed by typing *help YOURCOMMAND* to the prompt. The keybindings for the current mode are listed upon pressing `?`.

alot [options ...] [subcommand]

## 2.5.1 Cryptography

Alot has built in support for constructing signed and/or encrypted mails according to PGP/MIME ([RFC 3156](#), [RFC 3156](#)) via gnupg. It does however rely on a running *gpg-agent* to handle password entries.

---

**Note:** You need to have *gpg-agent* running to use GPG with alot!

*gpg-agent* will handle passphrase entry in a secure and configurable way, and it will cache your passphrase for some time so you don't have to enter it over and over again. For details on how to set this up we refer to [gnupg's manual](#).

---

### Signing outgoing emails

You can use the commands *sign*, *unsign* and *togglesign* in envelope mode to determine if you want this mail signed and if so, which key to use. To specify the key to use you may pass a hint string as argument to the *sign* or *togglesign* command. This hint would typically be a fingerprint or an email address associated (by gnupg) with a key.

Signing (and hence passwd entry) will be done at most once shortly before a mail is sent.

In case no key is specified, alot will leave the selection of a suitable key to gnupg so you can influence that by setting the *default-key* option in `~/ .gnupg/gpg.conf` accordingly.

You can set the default to-sign bit and the key to use for each *account* individually using the options *sign\_by\_default* and *gpg\_key*.

### Encrypt outgoing emails

You can use the commands *encrypt*, *unencrypt* and *toggleencrypt* and in envelope mode to ask alot to encrypt the mail before sending. The *encrypt* command accepts an optional hint string as argument to determine the key of the recipient.

You can set the default to-encrypt bit for each *account* individually using the option *encrypt\_by\_default*.

---

**Note:** If you want to access encrypt mail later it is useful to add yourself to the list of recipients when encrypting with gpg (not the recipients whom mail is actually send to). The simplest way to do this is to use the *encrypt-to* option in the `~/ .gnupg/gpg.conf`. But you might have to specify the correct encryption subkey otherwise gpg seems to throw an error.

---

Alot reads a config file in “INI” syntax: It consists of key-value pairs that use “=” as separator and “#” is comment-prefixes. Sections and subsections are defined using square brackets.

The default location for the config file is `~/ .config/alot/config`.

All configs are optional, but if you want to send mails you need to specify at least one *account* in your config.

### 3.1 Configuration Options

The following lists all available config options with their type and default values. The type of an option is used to validate a given value. For instance, if the type says “boolean” you may only provide “True” or “False” as values in your config file, otherwise alot will complain on startup. Strings *may* be quoted but do not need to be.

#### **ask\_subject**

**Type** boolean

**Default** True

#### **attachment\_prefix**

directory prefix for downloading attachments

**Type** string

**Default** “~”

#### **auto\_remove\_unread**

automatically remove ‘unread’ tag when focussing messages in thread mode

**Type** boolean

**Default** True

**auto\_replyto\_mailinglist**

Automatically switch to list reply mode if appropriate

**Type** boolean

**Default** False

**bounce\_force\_address**

Always use the accounts main address when constructing “Resent-From” headers for bounces. Set this to False to use the address string as received in the original message.

**Type** boolean

**Default** False

**bounce\_force\_realname**

Always use the proper realname when constructing “Resent-From” headers for bounces. Set this to False to use the realname string as received in the original message.

**Type** boolean

**Default** True

**bufferclose\_focus\_offset**

offset of next focused buffer if the current one gets closed

**Type** integer

**Default** -1

**bufferlist\_statusbar**

Format of the status-bar in bufferlist mode. This is a pair of strings to be left and right aligned in the status-bar that may contain variables:

- *{buffer\_no}*: index of this buffer in the global buffer list
- *{total\_messages}*: total number of messages indexed by notmuch
- *{pending\_writes}*: number of pending write operations to the index

**Type** mixed\_list

**Default** [{buffer\_no}: bufferlist], {input\_queue} total messages: {total\_messages}

**bug\_on\_exit**

confirm exit

**Type** boolean

**Default** False

**colourmode**

number of colours to use on the terminal

**Type** option, one of ['1', '16', '256']

**Default** 256

**complete\_matching\_abook\_only**

in case more than one account has an address book: Set this to True to make tab completion for recipients during compose only look in the abook of the account matching the sender address

**Type** boolean

**Default** False

**compose\_ask\_tags**

prompt for initial tags when compose

**Type** boolean

**Default** False

**displayed\_headers**

headers that get displayed by default

**Type** string list

**Default** From, To, Cc, Bcc, Subject

**edit\_headers\_blacklist**

see *edit\_headers\_whitelist*

**Type** string list

**Default** Content-Type, MIME-Version, References, In-Reply-To

**edit\_headers\_whitelist**

Which header fields should be editable in your editor used are those that match the whitelist and don't match the blacklist. in both cases '\*' may be used to indicate all fields.

**Type** string list

**Default** \*,

**editor\_cmd**

editor command if unset, alot will first try the EDITOR env variable, then /usr/bin/editor

**Type** string

**Default** None

**editor\_in\_thread**

call editor in separate thread. In case your editor doesn't run in the same window as alot, setting true here will make alot non-blocking during edits

**Type** boolean

**Default** False

### **editor\_spawn**

use *terminal\_cmd* to spawn a new terminal for the editor? equivalent to always providing the *-spawn=yes* parameter to compose/edit commands

**Type** boolean

**Default** False

### **editor\_writes\_encoding**

file encoding used by your editor

**Type** string

**Default** "UTF-8"

### **envelope\_headers\_blacklist**

headers that are hidden in envelope buffers by default

**Type** string list

**Default** In-Reply-To, References

### **envelope\_statusbar**

Format of the status-bar in envelope mode. This is a pair of strings to be left and right aligned in the status-bar. Apart from the global variables listed at *bufferlist\_statusbar* these strings may contain variables:

- *{to}*: To-header of the envelope

**Type** mixed\_list

**Default** [{buffer\_no}: envelope], {input\_queue} total messages: {total\_messages}

### **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. .. note:: this config setting is equivalent to, but independent of, the 'search.exclude\_tags' in the notmuch config.

**Type** string list

**Default** ,

### **flush\_retry\_timeout**

timeout in seconds after a failed attempt to writeout the database is repeated. Set to 0 for no retry.

**Type** integer

**Default** 5

### **followup\_to**

When one of the recipients of an email is a subscribed mailing list, set the "Mail-Followup-To" header to the list of recipients without yourself

**Type** boolean



**Default** False

**forward\_force\_address**

Always use the accounts main address when constructing “From” headers for forwards. Set this to False to use the address string as received in the original message.

**Type** boolean

**Default** False

**forward\_force\_realname**

Always use the proper realname when constructing “From” headers for forwards. Set this to False to use the realname string as received in the original message.

**Type** boolean

**Default** True

**forward\_subject\_prefix**

String prepended to subject header on forward only if original subject doesn’t start with ‘Fwd:’ or this prefix

**Type** string

**Default** “Fwd: “

**handle\_mouse**

enable mouse support - mouse tracking will be handled by urwid

---

**Note:** If this is set to True mouse events are passed from the terminal to urwid/alot. This means that normal text selection in alot will not be possible. Most terminal emulators will still allow you to select text when shift is pressed.

---

**Type** boolean

**Default** False

**history\_size**

The number of command line history entries to save

---

**Note:** You can set this to -1 to save *all* entries to disk but the history file might get *very* long.

---

**Type** integer

**Default** 50

**honor\_followup\_to**

When group-reply-ing to an email that has the “Mail-Followup-To” header set, use the content of this header as the new “To” header and leave the “Cc” header empty

**Type** boolean

**Default** False

#### **hooksfile**

where to look up hooks

**Type** string

**Default** “~/config/alot/hooks.py”

#### **initial\_command**

initial command when none is given as argument:

**Type** string

**Default** “search tag:inbox AND NOT tag:killed”

#### **input\_timeout**

timeout in (floating point) seconds until partial input is cleared

**Type** float

**Default** 1.0

#### **mailinglists**

The list of addresses associated to the mailinglists you are subscribed to

**Type** string list

**Default** ,

#### **msg\_summary\_hides\_threadwide\_tags**

In a thread buffer, hide from messages summaries tags that are common to all messages in that thread.

**Type** boolean

**Default** True

#### **namedqueries\_statusbar**

Format of the status-bar in named query list mode. This is a pair of strings to be left and right aligned in the status-bar. These strings may contain variables listed at *bufferlist\_statusbar* that will be substituted accordingly.

**Type** mixed\_list

**Default** [{buffer\_no}: namedqueries], {query\_count} named queries

#### **notify\_timeout**

time in secs to display status messages

**Type** integer

**Default** 2

**periodic\_hook\_frequency**

The number of seconds to wait between calls to the loop\_hook

**Type** integer

**Default** 300

**prefer\_plaintext**

prefer plaintext alternatives over html content in multipart/alternative

**Type** boolean

**Default** False

**print\_cmd**

how to print messages: this specifies a shell command used for printing. threads/messages are piped to this command as plain text. muttprint/a2ps works nicely

**Type** string

**Default** None

**prompt\_suffix**

Suffix of the prompt used when waiting for user input

**Type** string

**Default** “.”

**quit\_on\_last\_bclose**

shut down when the last buffer gets closed

**Type** boolean

**Default** False

**quote\_prefix**

String prepended to line when quoting

**Type** string

**Default** “> “

**reply\_account\_header\_priority**

The list of headers to match to determine sending account for a reply. Headers are searched in the order in which they are specified here, and the first header containing a match is used. If multiple accounts match in that header, the one defined first in the account block is used.

**Type** string list

**Default** From, To, Cc, Envelope-To, X-Envelope-To, Delivered-To

### **reply\_force\_address**

Always use the accounts main address when constructing “From” headers for replies. Set this to False to use the address string as received in the original message.

**Type** boolean

**Default** False

### **reply\_force\_realname**

Always use the proper realname when constructing “From” headers for replies. Set this to False to use the realname string as received in the original message.

**Type** boolean

**Default** True

### **reply\_subject\_prefix**

String prepended to subject header on reply only if original subject doesn’t start with ‘Re:’ or this prefix

**Type** string

**Default** “Re: “

### **search\_statusbar**

Format of the status-bar in search mode. This is a pair of strings to be left and right aligned in the status-bar. Apart from the global variables listed at [bufferlist\\_statusbar](#) these strings may contain variables:

- *{querystring}*: search string
- *{result\_count}*: number of matching messages
- *{result\_count\_positive}*: ‘s’ if result count is greater than 0.

**Type** mixed\_list

**Default** [{buffer\_no}: search] for “{querystring}”, {input\_queue} {result\_count} of {total\_messages} messages

### **search\_threads\_sort\_order**

default sort order of results in a search

**Type** option, one of [‘oldest\_first’, ‘newest\_first’, ‘message\_id’, ‘unsorted’]

**Default** newest\_first

### **show\_statusbar**

display status-bar at the bottom of the screen?

**Type** boolean

**Default** True

### **tabwidth**

number of spaces used to replace tab characters

**Type** integer

**Default** 8

#### **taglist\_statusbar**

Format of the status-bar in taglist mode. This is a pair of strings to be left and right aligned in the status-bar. These strings may contain variables listed at *bufferlist\_statusbar* that will be substituted accordingly.

**Type** mixed\_list

**Default** [{buffer\_no}: taglist], {input\_queue} total messages: {total\_messages}

#### **template\_dir**

templates directory that contains your message templates. It will be used if you give *compose -template* a filename without a path prefix.

**Type** string

**Default** "\$XDG\_CONFIG\_HOME/alot/templates"

#### **terminal\_cmd**

set terminal command used for spawning shell commands

**Type** string

**Default** "x-terminal-emulator -e"

#### **theme**

name of the theme to use

**Type** string

**Default** None

#### **themes\_dir**

directory containing theme files.

**Type** string

**Default** "\$XDG\_CONFIG\_HOME/alot/themes"

#### **thread\_authors\_me**

Word to replace own addresses with. Works in combination with *thread\_authors\_replace\_me*

**Type** string

**Default** "Me"

#### **thread\_authors\_order\_by**

When constructing the unique list of thread authors, order by date of author's first or latest message in thread

**Type** option, one of ['first\_message', 'latest\_message']

**Default** first\_message

#### **thread\_authors\_replace\_me**

Replace own email addresses with “me” in author lists Uses own addresses and aliases in all configured accounts.

**Type** boolean

**Default** True

#### **thread\_focus\_linewise**

Split message body linewise and allows to (move) the focus to each individual line. Setting this to False will result in one potentially big text widget for the whole message body.

**Type** boolean

**Default** True

#### **thread\_indent\_replies**

number of characters used to indent replies relative to original messages in thread mode

**Type** integer

**Default** 2

#### **thread\_statusbar**

Format of the status-bar in thread mode. This is a pair of strings to be left and right aligned in the status-bar. Apart from the global variables listed at *bufferlist\_statusbar* these strings may contain variables:

- *{tid}*: thread id
- *{subject}*: subject line of the thread
- *{authors}*: abbreviated authors string for this thread
- *{message\_count}*: number of contained messages

**Type** mixed\_list

**Default** [{buffer\_no}: thread] {subject}, {input\_queue} total messages: {total\_messages}

#### **thread\_subject**

What should be considered to be “the thread subject”. Valid values are:

- ‘notmuch’ (the default), will use the thread subject from notmuch, which depends on the selected sorting method
- ‘oldest’ will always use the subject of the oldest message in the thread as the thread subject

**Type** option, one of ['oldest', 'notmuch']

**Default** notmuch

#### **timestamp\_format**

timestamp format in `strftime` format syntax

**Type** string

**Default** None

### `user_agent`

value of the User-Agent header used for outgoing mails. setting this to the empty string will cause alot to omit the header all together. The string ‘{version}’ will be replaced by the version string of the running instance.

**Type** string

**Default** “alot/{version}”

## 3.1.1 Notmuch options

The following lists the notmuch options that alot reads.

### `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.

**Type** semicolon separated list

**Default** empty list

## 3.2 Accounts

In order to be able to send mails, you have to define at least one account subsection in your config: There needs to be a section “accounts”, and each subsection, indicated by double square brackets defines an account.

Here is an example configuration

```
[accounts]
  [[work]]
    realname = Bruce Wayne
    address = b.wayne@wayneenterprises.com
    alias_regexp = b.wayne\+.\+@wayneenterprises.com
    gpg_key = D7D6C5AA
    sendmail_command = msmtplib --account=wayne -t
    sent_box = maildir:///home/bruce/mail/work/Sent
    draft_box = maildir:///home/bruce/mail/work/Drafts

  [[secret]]
    realname = Batman
    address = batman@batcave.org
    aliases = batman@batmobile.org,
    sendmail_command = msmtplib --account=batman -t
    signature = ~/.batman.vcf
    signature_as_attachment = True
```

**Warning:** Sending mails is only supported via a sendmail shell command for now. If you want to use a sendmail command different from *sendmail -t*, specify it as *sendmail\_command*.

The following entries are interpreted at the moment:

**address**

your main email address

**Type** string

**alias\_regexp**

a regex for catching further aliases (like + extensions).

**Type** string

**Default** None

**aliases**

used to clear your addresses/ match account when formatting replies

**Type** string list

**Default** ,

**case\_sensitive\_username**

Whether the server treats the address as case-sensitive or case-insensitive (True for the former, False for the latter)

---

**Note:** The vast majority (if not all) SMTP servers in modern use treat usernames as case insensitive, you should only set this if you know that you need it.

---

**Type** boolean

**Default** False

**draft\_box**

where to store draft mails, e.g. *maildir:///home/you/mail/Drafts*. You can use mbox, maildir, mh, babyl and mmdf in the protocol part of the URL.

---

**Note:** You will most likely want drafts indexed by notmuch to be able to later access them within alot. This currently only works for maildir containers in a path below your notmuch database path.

---

**Type** mail\_container

**Default** None

**draft\_tags**

list of tags to automatically add to draft messages



**Type** string list

**Default** draft

### **encrypt\_by\_default**

Alot will try to GPG encrypt outgoing messages by default when this is set to *all* or *trusted*. If set to *all* the message will be encrypted for all recipients for who a key is available in the key ring. If set to *trusted* it will be encrypted to all recipients if a trusted key is available for all recipients (one where the user id for the key is signed with a trusted signature).

---

**Note:** If the message will not be encrypted by default you can still use the *toggleencrypt*, *encrypt* and *unencrypt* commands to encrypt it.

---

Deprecated since version 0.4: The values *True* and *False* are interpreted as *all* and *none* respectively. *0*, *1*, *true*, *True*, *false*, *False*, *yes*, *Yes*, *no*, *No*, will be removed before 1.0, please move to *all*, *none*, or *trusted*.

**Type** option, one of ['all', 'none', 'trusted', 'True', 'False', 'true', 'false', 'Yes', 'No', 'yes', 'no', '1', '0']

**Default** none

### **encrypt\_to\_self**

If this is true when encrypting a message it will also be encrypted with the key defined for this account.

**Warning:** Before 0.6 this was controlled via `gpg.conf`.

**Type** boolean

**Default** True

### **gpg\_key**

The GPG key ID you want to use with this account.

**Type** string

**Default** None

### **passed\_tags**

list of tags to automatically add to passed messages

**Type** string list

**Default** passed

### **realname**

used to format the (proposed) From-header in outgoing mails

**Type** string

**replied\_tags**

list of tags to automatically add to replied messages

**Type** string list

**Default** replied

**sendmail\_command**

sendmail command. This is the shell command used to send out mails via the sendmail protocol

**Type** string

**Default** “sendmail -t”

**sent\_box**

where to store outgoing mails, e.g. *maildir:///home/you/mail/Sent*. You can use mbox, maildir, mh, babyl and mmdf in the protocol part of the URL.

---

**Note:** If you want to add outgoing mails automatically to the notmuch index you must use maildir in a path within your notmuch database path.

---

**Type** mail\_container

**Default** None

**sent\_tags**

list of tags to automatically add to outgoing messages

**Type** string list

**Default** sent

**sign\_by\_default**

Outgoing messages will be GPG signed by default if this is set to True.

**Type** boolean

**Default** False

**signature**

path to signature file that gets attached to all outgoing mails from this account, optionally renamed to *signature\_filename*.

**Type** string

**Default** None

**signature\_as\_attachment**

attach signature file if set to True, append its content (mimetype text) to the body text if set to False.

**Type** boolean

**Default** False**signature\_filename**signature file's name as it appears in outgoing mails if *signature\_as\_attachment* is set to True**Type** string**Default** None

### 3.3 Contacts Completion

For each *account* you can define an address book by providing a subsection named *abook*. Crucially, this section needs an option *type* that specifies the type of the address book. The only types supported at the moment are “shellcommand” and “abook”. Both respect the *ignorecase* option which defaults to *True* and results in case insensitive lookups.

**shellcommand**

Address books of this type use a shell command in combination with a regular expression to look up contacts.

The value of *command* will be called with the search prefix as only argument for lookups. Its output is searched for email-name pairs using the regular expression given as *regexp*, which must include named groups “email” and “name” to match the email address and realname parts respectively. See below for an example that uses *abook*

```
[accounts]
  [[youraccount]]
    # ...
    [[abook]]
      type = shellcommand
      command = abook --mutt-query
      regexp = '^(?P<email>[^@]+@[^\t]+\t+(?P<name>[^\t]+)'
```

See [here](#) for alternative lookup commands. The few others I have tested so far are:

**goobook** for cached google contacts lookups. Works with the above default regexp

```
command = goobook query
regexp = '^(?P<email>[^@]+@[^\t]+\t+(?P<name>[^\t]+)'
```

**nottoomuch-addresses** completes contacts found in the notmuch index:

```
command = nottoomuch-addresses.sh
regexp = \" (?P<name>.)\" \\s*(?P<email>.*+?@.+?)>
```

**notmuch-abook** completes contacts found in database of notmuch-abook:

```
command = notmuch_abook.py lookup
regexp = ^((?P<name>[^\s\<]*)\s+)?(?P<email>[^@]+?@[^>]+)?>?$
```

**notmuch address** Since version *0.19*, notmuch itself offers a subcommand *address*, that returns email addresses found in the notmuch index. Combined with the *date:* syntax to query for mails within a certain timeframe, this allows to search contacts that you've sent emails to (output all addresses from the *To*, *Cc* and *Bcc* headers):

```
command = 'notmuch address --format=json --output=recipients date:1Y.. AND_
↳from:my@address.org'
regexp = '\[?{"name": "(?P<name>.*)", "address": "(?P<email>.+)", "name-addr
↳": ".*"}[,\ ]?]'
shellcommand_external_filtering = False
```

If you want to search for senders in the *From* header (which should be must faster according to *notmuch address docs*), then use the following command:

```
command = 'notmuch address --format=json date:1Y..'
```

Don't hesitate to send me your custom *regexp* values to list them here.

### abook

Address books of this type directly parse *abooks* contact files. You may specify a path using the “*abook\_contacts\_file*” option, which defaults to `~/ .abook/addressbook`. To use the default path, simply do this:

```
[accounts]
[[youraccount]]
# ...
[[[abook]]]
    type = abook
```

## 3.4 Key Bindings

If you want to bind a command to a key you can do so by adding the pair to the *[bindings]* section. This will introduce a *global* binding, that works in all modes. To make a binding specific to a mode you have to add the pair under the subsection named like the mode. For instance, if you want to bind *T* to open a new search for threads tagged with ‘todo’, and be able to toggle this tag in search mode, you'd add this to your config

```
[bindings]
T = search tag:todo

[[search]]
t = toggletags todo
```

Known modes are:

- bufferlist
- envelope
- namedqueries
- search
- taglist
- thread

Have a look at the *urwid User Input* documentation on how key strings are formatted.

### 3.4.1 Default bindings

User-defined bindings are combined with the default bindings listed below.

```

up = move up
down = move down
page up = move page up
page down = move page down
mouse press 4 = move up
mouse press 5 = move down
j = move down
k = move up
'g g' = move first
G = move last
' ' = move page down
'ctrl d' = move halfpage down
'ctrl u' = move halfpage up
@ = refresh
? = help bindings
I = search tag:inbox AND NOT tag:killed
'#' = taglist
shift tab = bprevious
U = search tag:unread
tab = bnext
\ = prompt 'search '
d = bclose
$ = flush
m = compose
o = prompt 'search '
q = exit
';' = bufferlist
':' = prompt
. = repeat

```

**[bufferlist]**

```

x = close
enter = open

```

**[search]**

```

enter = select
a = toggletags inbox
& = toggletags killed
! = toggletags flagged
s = toggletags unread
l = retagprompt
O = refineprompt
| = refineprompt

```

**[envelope]**

```

a = prompt 'attach ~/ '
y = send
P = save
s = 'refine Subject'
f = prompt 'set From '
t = 'refine To'
b = 'refine Bcc'
c = 'refine Cc'
S = togglesign
enter = edit
'g f' = togglesource

```

(continues on next page)

```

[taglist]
    enter = select

[namedqueries]
    enter = select

[thread]
    enter = select
    C = fold *
    E = unfold *
    c = fold
    e = unfold
    < = fold
    > = unfold
    [ = indent -
    ] = indent +
    'g f' = togglesource
    H = toggleheaders
    P = print --all --separately --add_tags
    S = save --all
    g = reply --all
    f = forward
    p = print --add_tags
    n = editnew
    b= bounce
    s = save
    r = reply
    | = prompt 'pipeto '

    'g j' = move next sibling
    'g k' = move previous sibling
    'g h' = move parent
    'g l' = move first reply
    ' ' = move next

```

In prompts the following hardcoded bindings are available.

Key	Function
Ctrl-f/b	Moves the cursor one character to the right/left
Alt-f/b Shift-right/left	Moves the cursor one word to the right/left
Ctrl-a/e	Moves the cursor to the beginning/end of the line
Ctrl-d	Deletes the character under the cursor
Alt-d	Deletes everything from the cursor to the end of the current or next word
Alt-Delete/Backspace Ctrl-w	Deletes everything from the cursor to the beginning of the current or previous word
Ctrl-k	Deletes everything from the cursor to the end of the line
Ctrl-u	Deletes everything from the cursor to the beginning of the line

### 3.4.2 Overwriting defaults

To disable a global binding you can redefine it in your config to point to an empty command string. For example, to add a new global binding for key *a*, which is bound to *toggletags inbox* in search mode by default, you can remap it as follows.

```
[bindings]
a = NEW GLOBAL COMMAND

[[search]]
a =
```

If you omit the last two lines, *a* will still be bound to the default binding in search mode.

## 3.5 Hooks

Hooks are python callables that live in a module specified by *hooksfile* in the config. Per default this points to `~/ .config/alot/hooks.py`.

### 3.5.1 Pre/Post Command Hooks

For every *COMMAND* in mode *MODE*, the callables `pre_MODE_COMMAND()` and `post_MODE_COMMAND()` – if defined – will be called before and after the command is applied respectively. In addition callables `pre_global_COMMAND()` and `post_global_COMMAND()` can be used. They will be called if no specific hook function for a mode is defined. The signature for the pre-send hook in envelope mode for example looks like this:

```
pre_envelope_send(ui=None, dbm=None, cmd=None)
```

#### Parameters

- **ui** (`alot.ui.UI`) – the main user interface
- **dbm** (`alot.db.manager.DBManager`) – a database manager
- **cmd** (`alot.commands.Command`) – the Command instance that is being called

Consider this pre-hook for the exit command, that logs a personalized goodbye message:

```
import logging
from alot.settings.const import settings
def pre_global_exit(**kwargs):
    accounts = settings.get_accounts()
    if accounts:
        logging.info('goodbye, %s!' % accounts[0].realname)
    else:
        logging.info('goodbye!')
```

### 3.5.2 Other Hooks

Apart from command pre- and posthooks, the following hooks will be interpreted:

```
reply_prefix(realname, address, timestamp[, ui=None, dbm=None])
```

Is used to reformat the first indented line in a reply message. This defaults to ‘Quoting %s (%s)n’ % (realname, timestamp)’ unless this hook is defined

#### Parameters

- **realname** (*str*) – name or the original sender
- **address** (*str*) – address of the sender

- **timestamp** (`datetime.datetime`) – value of the Date header of the replied message

**Return type** string

**forward\_prefix** (`realname`, `address`, `timestamp`[, `ui=None`, `dbm=None` ])

Is used to reformat the first indented line in a inline forwarded message. This defaults to ‘Forwarded message from %s (%s)n’ % (realname, timestamp)’ if this hook is undefined

**Parameters**

- **realname** (`str`) – name or the original sender
- **address** (`str`) – address of the sender
- **timestamp** (`datetime.datetime`) – value of the Date header of the replied message

**Return type** string

**pre\_edit\_translate** (`text`[, `ui=None`, `dbm=None` ])

Used to manipulate a message’s text *before* the editor is called. The text might also contain some header lines, depending on the settings `edit_headers_whitelist` and `edit_header_blacklist`.

**Parameters** **text** (`str`) – text representation of mail as displayed in the interface and as sent to the editor

**Return type** `str`

**post\_edit\_translate** (`text`[, `ui=None`, `dbm=None` ])

used to manipulate a message’s text *after* the editor is called, also see `pre_edit_translate`

**Parameters** **text** (`str`) – text representation of mail as displayed in the interface and as sent to the editor

**Return type** `str`

**text\_quote** (`message`)

used to transform a message into a quoted one

**Parameters** **message** (`str`) – message to be quoted

**Return type** `str`

**timestamp\_format** (`timestamp`)

represents given timestamp as string

**Parameters** **timestamp** (`datetime`) – timestamp to represent

**Return type** `str`

**touch\_external\_cmdlist** (`cmd`, `shell=shell`, `spawn=spawn`, `thread=thread`)

used to change external commands according to given flags shortly before they are called.

**Parameters**

- **cmd** (`list of str`) – command to be called
- **shell** (`bool`) – is this to be interpreted by the shell?
- **spawn** (`bool`) – should be spawned in new terminal/environment
- **threads** – should be called in new thread

**Returns** triple of amended command list, shell and thread flags

**Return type** list of `str`, `bool`, `bool`

**reply\_subject** (`subject`)

used to reformat the subject header on reply



**Parameters** **subject** (*str*) – subject to reformat

**Return type** *str*

**forward\_subject** (*subject*)

used to reformat the subject header on forward

**Parameters** **subject** (*str*) – subject to reformat

**Return type** *str*

**pre\_buffer\_open** (*ui=None, dbm=None, buf=buf*)

run before a new buffer is opened

**Parameters** **buf** (*alot.buffer.Buffer*) – buffer to open

**post\_buffer\_open** (*ui=None, dbm=None, buf=buf*)

run after a new buffer is opened

**Parameters** **buf** (*alot.buffer.Buffer*) – buffer to open

**pre\_buffer\_close** (*ui=None, dbm=None, buf=buf*)

run before a buffer is closed

**Parameters** **buf** (*alot.buffer.Buffer*) – buffer to open

**post\_buffer\_close** (*ui=None, dbm=None, buf=buf, success=success*)

run after a buffer is closed

**Parameters**

- **buf** (*alot.buffer.Buffer*) – buffer to open
- **success** (*boolean*) – true if successfully closed buffer

**pre\_buffer\_focus** (*ui=None, dbm=None, buf=buf*)

run before a buffer is focused

**Parameters** **buf** (*alot.buffer.Buffer*) – buffer to open

**post\_buffer\_focus** (*ui=None, dbm=None, buf=buf, success=success*)

run after a buffer is focused

**Parameters**

- **buf** (*alot.buffer.Buffer*) – buffer to open
- **success** (*boolean*) – true if successfully focused buffer

**exit** ()

run just before the program exits

**sanitize\_attachment\_filename** (*filename=None, prefix="", suffix=""*)

returns *prefix* and *suffix* for a sanitized filename to use while opening an attachment. The *prefix* and *suffix* are used to open a file named *prefix* + *XXXXXX* + *suffix* in a temporary directory.

**Parameters**

- **filename** (*str or None*) – filename provided in the email (can be None)
- **prefix** (*str*) – prefix string as found on mailcap
- **suffix** (*str*) – suffix string as found on mailcap

**Returns** tuple of *prefix* and *suffix*

**Return type** (*str, str*)

`loop_hook` (*ui=None*)

Run on a period controlled by `_periodic_hook_frequency`

**Parameters** `ui` (`alot.ui.UI`) – the main user interface

## 3.6 Theming

Alot can be run in 1, 16 or 256 colour mode. The requested mode is determined by the command-line parameter `-C` or read from option `colourmode` config value. The default is 256, which scales down depending on how many colours your terminal supports.

Most parts of the user interface can be individually coloured to your liking. To make it easier to switch between or share different such themes, they are defined in separate files (see below for the exact format). To specify the theme to use, set the `theme` config option to the name of a theme-file. A file by that name will be looked up in the path given by the `themes_dir` config setting which defaults to `$XDG_CONFIG_HOME/alot/themes`, and `~/.config/alot/themes/`, if `XDG_CONFIG_HOME` is empty or not set. If the `themes_dir` is not present then the contents of `$XDG_DATA_DIRS/alot/themes` will be tried in order. This defaults to `/usr/local/share/alot/themes` and `/usr/share/alot/themes`, in that order. These locations are meant to be used by distro packages to put themes in.

### 3.6.1 Theme Files

contain a section for each `MODE` plus “help” for the bindings-help overlay and “global” for globally used themables like footer, prompt etc. Each such section defines colour *attributes* for the parts that can be themed. The names of the themables should be self-explanatory. Have a look at the default theme file at `alot/defaults/default.theme` and the config spec `alot/defaults/default.theme` for the exact format.

### 3.6.2 Colour Attributes

Attributes are *sextuples* of `urwid Attribute strings` that specify foreground and background for mono, 16 and 256-colour modes respectively. For mono-mode only the flags `blink`, `standup`, `underline` and `bold` are available, 16c mode supports these in combination with the colour names:

brown	dark red	dark magenta	dark blue	dark cyan	dark green
yellow	light red	light magenta	light blue	light cyan	light green
black	dark gray	light gray	white		

In high-colour mode, you may use the above plus grayscales `g0` to `g100` and colour codes given as `#` followed by three hex values. See [here](#) and [here](#) for more details on the interpreted values. A colour picker that makes choosing colours easy can be found in `alot/extra/colour_picker.py`.

As an example, check the setting below that makes the footer line appear as underlined bold red text on a bright green background:

```
[[global]]
#name      mono fg      mono bg      16c fg      16c bg      256c fg
↪          256c bg
#          /          /          /          /          /
↪          /          /          /          /          /
#          v          v          v          v          v
↪          v          v          v          v          v
footer = 'bold,underline', '', 'light red, bold, underline', 'light green', 'light
↪red, bold, underline', '#8f6'
```

### 3.6.3 Search mode thread lines

The subsection ‘[[threadline]]’ of the ‘[search]’ section in *Theme Files* determines how to present a thread: here, *attributes* ‘normal’ and ‘focus’ provide fallback/spacer themes and ‘parts’ is a (string) list of displayed subwidgets. Possible part strings are:

- authors
- content
- date
- mailcount
- subject
- tags

For every listed part there must be a subsection with the same name, defining

**normal** *attribute* used for this part if unfocussed

**focus** *attribute* used for this part if focussed

**width** tuple indicating the width of the part. This is either (*‘fit’*, *min*, *max*) to force the widget to be at least *min* and at most *max* characters wide, or (*‘weight’*, *n*) which makes it share remaining space with other ‘weight’ parts.

**alignment** how to place the content string if the widget space is larger. This must be one of ‘right’, ‘left’ or ‘center’.

#### Dynamic theming of thread lines based on query matching

To highlight some thread lines (use different attributes than the defaults found in the ‘[[threadline]]’ section), one can define sections with prefix ‘threadline’. Each one of those can redefine any part of the structure outlined above, the rest defaults to values defined in ‘[[threadline]]’.

The section used to theme a particular thread is the first one (in file-order) that matches the criteria defined by its ‘query’ and ‘tagged\_with’ values:

- If ‘query’ is defined, the thread must match that querystring.
- If ‘tagged\_with’ is defined, its value (string list) must be a subset of the accumulated tags of all messages in the thread.

---

**Note:** that ‘tagged\_with = A,B’ is different from ‘query = “is:A AND is:B”’: the latter will match only if the thread contains a single message that is both tagged with A and B.

Moreover, note that if both query and tagged\_with is undefined, this section will always match and thus overwrite the defaults.

---

The example below shows how to highlight unread threads: The date-part will be bold red if the thread has unread messages and flagged messages and just bold if the thread has unread but no flagged messages:

```
[search]
# default threadline
[[threadline]]
  normal = 'default', 'default', 'default', 'default', '#6d6', 'default'
  focus = 'standout', 'default', 'light gray', 'dark gray', 'white', '#68a'
  parts = date, mailcount, tags, authors, subject
```

(continues on next page)

(continued from previous page)

```

[[[date]]]
    normal = 'default','default','light gray','default','g58','default'
    focus = 'standout','default','light gray','dark gray','g89','#68a'
    width = 'fit',10,10
# ...

# highlight threads containing unread and flagged messages
[[[threadline-flagged-unread]]]
    tagged_with = 'unread','flagged'
[[[date]]]
    normal = 'default','default','light red,bold','default','light red,bold',
↪'default'

# highlight threads containing unread messages
[[[threadline-unread]]]
    query = 'is:unread'
[[[date]]]
    normal = 'default','default','light gray,bold','default','g58,bold',
↪'default'

```

### 3.6.4 Tagstring Formatting

One can specify how a particular tagstring is displayed throughout the interface. To use this feature, add a section *[tags]* to you alot config (not the theme file) and for each tag you want to customize, add a subsection named after the tag. Such a subsection may define

**normal** *attribute* used if unfocussed

**focus** *attribute* used if focussed

**translated** fixed string representation for this tag. The tag can be hidden from view, if the key *translated* is set to ‘’, the empty string.

**translation** a pair of strings that define a regular substitution to compute the string representation on the fly using *re.sub*. This only really makes sense if one uses a regular expression to match more than one tagstring (see below).

The following will make alot display the “todo” tag as “TODO” in white on red.

```

[tags]
  [[todo]]
    normal = '',', 'white','light red', 'white','#d66'
    translated = TODO

```

Utf-8 symbols are welcome here, see e.g. <http://panmental.de/symbols/info.htm> for some fancy symbols. I personally display my maildir flags like this:

```

[tags]

[[flagged]]
    translated =
    normal = '',', 'light red',',', 'light red',',',
    focus = '',', 'light red',',', 'light red',',',

[[unread]]
    translated =

```

(continues on next page)

(continued from previous page)

```
[[replied]]
  translated =

[[encrypted]]
  translated =
```

You may use regular expressions in the tagstring subsections to theme multiple tagstrings at once (first match wins). If you do so, you can use the *translation* option to specify a string substitution that will rename a matching tagstring. *translation* takes a comma separated *pair* of strings that will be fed to `re.sub()`. For instance, to theme all your `nmbug` tagstrings and especially colour tag `notmuch::bug` red, do the following:

```
[[notmuch::bug]]
  translated = 'nm:bug'
  normal = "", "", "light red, bold", "light blue", "light red, bold", "#88d"

[[notmuch::.*]]
  translation = 'notmuch:.(*)','nm:\1'
  normal = "", "", "white", "light blue", "#fff", "#88d"
```



## 4.1 Overview

The main component is `alot.ui.UI`, which provides methods for user input and notifications, sets up the widget tree and maintains the list of active buffers. When you start up `alot`, `init.py` initializes logging, parses settings and commandline args and instantiates the UI instance of that gets passed around later. From its constructor this instance starts the `urwidmainloop` that takes over.

Apart from the central UI, there are two other “managers” responsible for core functionalities, also set up in `init.py`:

- `ui.dbman`: a `DBManager` to access the email database and
- `alot.settings.settings`: a `SettingsManager` to access user settings

Every user action, triggered either by key bindings or via the command prompt, is given as commandline string that gets translated to a `Command` object which is then applied. Different actions are defined as subclasses of `Command`, which live in `alot/commands/MODE.py`, where `MODE` is the name of the mode (`Buffer` type) they are used in.

## 4.2 Email Database

The python bindings to `libnotmuch` define `notmuch.Thread` and `notmuch.Message`, which unfortunately are very fragile. `Alot` defines the wrapper classes `alot.db.Thread` and `alot.db.Message` that use an `manager.DBManager` instance to transparently provide persistent objects.

`alot.db.Message` moreover contains convenience methods to extract information about the message like reformatted header values, a summary, decoded and interpreted body text and a list of `Attachments`.

The central UI instance carries around a `DBManager` object that is used for any lookups or modifications of the email base. `DBManager` can directly look up `Thread` and `Message` objects and is able to postpone/cache/retry writing operations in case the Xapian index is locked by another process.

### 4.2.1 Database Manager

### 4.2.2 Errors

### 4.2.3 Wrapper

### 4.2.4 Other Structures

### 4.2.5 Utilities

## 4.3 User Interface

Alot sets up a widget tree and a `mainloop` in the constructor of `alot.ui.UI`. The visible area is a `urwid.Frame`, where the footer is used as a status line and the body part displays the currently active `alot.buffer.Buffer`.

To be able to bind keystrokes and translate them to `Commands`, keypresses are *not* propagated down the widget tree as is customary in `urwid`. Instead, the root widget given to `urwid.mainloop` is a custom wrapper (`alot.ui.Inputwrap`) that interprets key presses. A dedicated `SendKeypressCommand` can be used to trigger key presses to the wrapped root widget and thereby accessing standard `urwid` behaviour.

In order to keep the interface non-blocking and react to events like terminal size changes, `alot` makes use of `asyncio` - which allows asynchronous calls without the use of callbacks. `Alot` makes use of the python 3.5 `async/await` syntax

```
async def greet(ui): # ui is instance of alot.ui.UI
    name = await ui.prompt('pls enter your name')
    ui.notify('your name is: ' + name)
```

### 4.3.1 UI - the main component

### 4.3.2 Buffers

A buffer defines a view to your data. It knows how to render itself, to interpret keypresses and is visible in the “body” part of the widget frame. Different modes are defined by subclasses of the following base class.

Available modes are:

Mode	Buffer Subclass
search	SearchBuffer
thread	ThreadBuffer
bufferlist	BufferlistBuffer
taglist	TagListBuffer
namedqueries	NamedQueriesBuffer
envelope	EnvelopeBuffer

### 4.3.3 Widgets

What follows is a list of the non-standard `urwid` widgets used in `alot`. Some of them respect *user settings*, themes in particular.



## utils

Utility Widgets not specific to alot

**class** `alot.widgets.utils.AttrFlipWidget` (*w, maps, init\_map='normal'*)  
An AttrMap that can remember attributes to set

## globals

### bufferlist

Widgets specific to Bufferlist mode

**class** `alot.widgets.bufferlist.BufferlineWidget` (*buffer*)  
selectable text widget that represents a Buffer in the BufferlistBuffer.

## search

## thread

### 4.3.4 Completion

`alot.ui.UI.prompt()` allows tab completion using a `Completer` object handed as ‘completer’ parameter. `alot.completion` defines several subclasses for different occasions like completing email addresses from an `AddressBook`, `notmuch` tagstrings. Some of these actually build on top of each other; the `QueryCompleter` for example uses a `TagsCompleter` internally to allow tagstring completion after “is:” or “tag:” keywords when typing a `notmuch` querystring.

All these classes override the method `complete()`, which for a given string and cursor position in that string returns a list of tuples (*completed\_string, new\_cursor\_position*) that are taken to be the completed values. Note that *completed\_string* does not need to have the original string as prefix. `complete()` may rise `alot.errors.CompletionError` exceptions.

## 4.4 User Settings

Alot sets up a `SettingsManager` to access user settings defined in different places uniformly. There are four types of user settings:

what?	location	accessible via
alot config	<code>~/.config/alot/config</code> or given by command option <code>-c</code> .	<code>SettingsManager.get()</code>
hooks – user provided python code	<code>~/.config/alot/hooks.py</code> or as given by the <i>hooksfile</i> config value	<code>SettingsManager.get_hook()</code>
notmuch config	<code>~/.notmuch-config</code> or given by <i>\$NOTMUCH_CONFIG</i> or given by command option <code>-n</code>	<code>SettingsManager.get_notmuch_setting()</code>
mailcap – defines shellcommands to handle mime types	<code>~/.mailcap (/etc/mailcap)</code>	<code>SettingsManager.mailcap_find_match()</code>

## 4.4.1 Settings Manager

## 4.4.2 Errors

**exception** `alot.settings.errors.ConfigError`  
could not parse user config

**exception** `alot.settings.errors.NoMatchingAccount`  
No account matching requirements found.

## 4.4.3 Utils

`alot.settings.utils.read_config` (*configpath=None, specpath=None, checks=None, report\_extra=False*)  
get a (validated) config object for given config file path.

### Parameters

- **configpath** (*str* or *list(str)*) – path to config-file or a list of lines as its content
- **specpath** (*str*) – path to spec-file
- **checks** (*dict str->callable,*) – custom checks to use for validator. see [validate docs](#)
- **report\_extra** (*boolean*) – log if a setting is not present in the spec file

**Raises** `ConfigError`

**Return type** `configobj.ConfigObj`

`alot.settings.utils.resolve_att` (*a, fallback*)  
replace “ and ‘default’ by fallback values

## 4.4.4 Themes

## 4.4.5 Accounts

## 4.4.6 Addressbooks

**class** `alot.addressbook.AddressBook` (*ignorecase=True*)  
can look up email addresses and realnames for contacts.

---

**Note:** This is an abstract class that leaves `get_contacts()` unspecified. See `AbookAddressBook` and `ExternalAddressbook` for implementations.

---

**get\_contacts** ()  
list all contacts tuples in this abook as (name, email) tuples

**lookup** (*query=""*)  
looks up all contacts where name or address match query

**class** `alot.addressbook.abook.AbookAddressBook` (*path='~/abook/addressbook', \*\*kwargs*)  
AddressBook that parses abook’s config/database files

**Parameters** **path** (*str*) – path to abook addressbook file

```
get_contacts ()
    list all contacts tuples in this abook as (name, email) tuples
```

## 4.5 Utils

## 4.6 Commands

User actions are represented by `Command` objects that can then be triggered by `alot.ui.UI.apply_command()`. Command-line strings given by the user via the prompt or key bindings can be translated to `Command` objects using `alot.commands.commandfactory()`. Specific actions are defined as subclasses of `Command` and can be registered to a global command pool using the `registerCommand` decorator.

---

**Note:** that the return value of `commandfactory()` depends on the current *mode* the user interface is in. The mode identifier is a string that is uniquely defined by the currently focuses `Buffer`.

---



---

**Note:** The names of the commands available to the user in any given mode do not correspond one-to-one to these subclasses. You can register a `Command` multiple times under different names, with different forced constructor parameters and so on. See for instance the definition of `BufferFocusCommand` in ‘`commands/globals.py`’:

```
@registerCommand(MODE, 'bprevious', forced={'offset': -1},
                 help='focus previous buffer')
@registerCommand(MODE, 'bnext', forced={'offset': +1},
                 help='focus next buffer')
class BufferFocusCommand(Command):
    def __init__(self, buffer=None, offset=0, **kwargs):
        ...
```

### 4.6.1 Globals

### 4.6.2 Envelope

### 4.6.3 Bufferlist

### 4.6.4 Search

### 4.6.5 Taglist

### 4.6.6 Namedqueries

### 4.6.7 Thread

## 4.7 Crypto



1. Why reinvent the wheel? Why not extend an existing MUA to work nicely with notmuch?

alot makes use of existing solutions where possible: It does not fetch, send or edit mails; it lets `notmuch` handle your mailindex and uses a `toolkit` to render its display. You are responsible for `automatic initial tagging`.

This said, there are few CLI MUAs that could be easily and naturally adapted to using notmuch. Rebuilding an interface from scratch using `friendly and extensible tools` seemed easier and more promising.

Update: see `mutt-kz` for a fork of mutt..

2. What's with the snotty name?

It's not meant to be presumptuous. I like the dichotomy; I like to picture the look on someone's face who reads the `User-Agent` header "notmuch/alot"; I like cookies; I like [this comic strip](#).

3. I want feature X!

Me too! Feel free to file a new or comment on existing `issues` if you don't want/have the time/know how to implement it yourself. Be verbose as to how it should look or work when it's finished and give it some thought how you think we should implement it. We'll discuss it from there.

4. Why are the default key bindings so counter-intuitive?

Be aware that the bindings for all modes are `fully configurable`. That said, I choose the bindings to be natural for me. I use `vim` and `pentadactyl` a lot. However, I'd be interested in discussing the defaults. If you think your bindings are more intuitive or better suited as defaults for some reason, don't hesitate to send me your config. The same holds for the theme settings you use. Tell me. Let's improve the defaults.

5. Help! I don't see `text/html` content!

better: How do I properly set up an inline renderer for `text/html`? Try `w3m` and put the following into your `~/ .mailcap`:

```
text/html; w3m -dump -o document_charset=%{charset} '%s'; nametemplate=
↳ %s.html; copiousoutput
```

Most text based browsers have a dump mode that can be used here.

### 6. Why are you doing \$THIS not \$THAT way?

Lazy and Ignorance: In most cases I simply did not or still don't know a better solution. I try to outsource as much as I can to well established libraries and be it only to avoid having to read rfc's. But there are lots of tasks I implemented myself, possibly overlooking a ready made and available solution. Twisted is such a feature-rich but gray area in my mind for example. If you think you know how to improve the current implementation let me know!

The few exceptions to above stated rule are the following:

- The modules `cmd` and `cmd2`, that handle all sorts of convenience around command objects hate `urwid`: They are painfully strongly coupled to user in/output via `stdin` and `out`.
- *notmuch reply* is not used to format reply messages because 1. it is not offered by `notmuch`'s library but is a feature of the CLI. This means we would have to call the `notmuch` binary, something that is avoided where possible. 2. As there is no *notmuch forward* equivalent, this (very similar) functionality would have to be re-implemented anyway.

### 7. I thought alot ran on Python 2?

It used to. When we made the transition to Python 3 we didn't maintain Python 2 support. If you still need Python 2 support the 0.7 release is your best bet.

### 8. I thought alot used twisted?

It used to. After we switched to python 3 we decided to switch to `asyncio`, which reduced the number of dependencies we have. Twisted is an especially heavy dependency, when we only used their `async` mechanisms, and not any of the other goodness that twisted has to offer.

## 6.1 Synopsis

alot [options ...] [subcommand]

## 6.2 Description

Alot is a terminal-based mail user agent for the notmuch mail system. It features a modular and command prompt driven interface to provide a full MUA experience as an alternative to the Emacs mode shipped with notmuch.

## 6.3 Options

- r, --read-only** open notmuch database in read-only mode
- c FILENAME, --config=FILENAME** configuration file (default: `~/config/alot/config`)
- n FILENAME, --notmuch-config=FILENAME** notmuch configuration file (default: `$NOTMUCH_CONFIG` or `~/notmuch-config`)
- C COLOURS, --colour-mode=COLOURS** number of colours to use on the terminal; must be 1, 16 or 256 (default: configuration option *colourmode* or 256)
- p PATH, --mailindex-path=PATH** path to notmuch index
- d LEVEL, --debug-level=LEVEL** debug level; must be one of debug, info, warning or error (default: info)
- l FILENAME, --logfile=FILENAME** log file (default: `/dev/null`)
- h, --help** display help and exit
- v, --version** output version information and exit

## 6.4 Commands

**search** start in a search buffer using the query string provided as parameter (see *notmuch-search-terms* (7))

**compose** compose a new message

**bufferlist** start with only a bufferlist buffer open

**taglist** start with only a taglist buffer open

**namedqueries** start with list of named queries

**pyshell** start the interactive python shell inside alot

## 6.5 Usage

The arrow keys, *page-up/down*, *j*, *k* and *Space* can be used to move the focus. *Escape* cancels prompts and *Enter* selects. Hit `:` at any time and type in commands to the prompt.

The interface shows one buffer at a time, you can use *Tab* and *Shift-Tab* to switch between them, close the current buffer with *d* and list them all with `;`.

The buffer type or *mode* (displayed at the bottom left) determines which prompt commands are available. Usage information on any command can be listed by typing *help YOURCOMMAND* to the prompt. The keybindings for the current mode are listed upon pressing `?`.

## 6.6 UNIX Signals

**SIGUSR1** Refreshes the current buffer.

**SIGINT** Shuts down the user interface.

## 6.7 See Also

*notmuch* (1)

Alot is a terminal-based mail user agent for the notmuch mail system. It features a modular and command prompt driven interface to provide a full MUA experience as an alternative to the Emacs mode shipped with notmuch.



### a

- alot, 43
- alot.account, 46
- alot.addressbook, 46
- alot.addressbook.abook, 46
- alot.addressbook.external, 47
- alot.commands, 47
- alot.db, 43
- alot.db.errors, 44
- alot.settings.errors, 46
- alot.settings.manager, 45
- alot.settings.utils, 46
- alot.ui, 44
- alot.utils, 47
- alot.widgets.bufferlist, 45
- alot.widgets.utils, 45



**A**

AbookAddressBook (class in alot.addressbook.abook), 46  
 AddressBook (class in alot.addressbook), 46  
 alot (module), 43  
 alot.account (module), 46  
 alot.addressbook (module), 46  
 alot.addressbook.abook (module), 46  
 alot.addressbook.external (module), 47  
 alot.commands (module), 47  
 alot.db (module), 43  
 alot.db.errors (module), 44  
 alot.settings.errors (module), 46  
 alot.settings.manager (module), 45  
 alot.settings.utils (module), 46  
 alot.ui (module), 44  
 alot.utils (module), 47  
 alot.widgets.bufferlist (module), 45  
 alot.widgets.utils (module), 45  
 AttrFlipWidget (class in alot.widgets.utils), 45

**B**

BufferlineWidget (class in alot.widgets.bufferlist), 45

**C**

ConfigError, 46

**E**

EDITOR, 19  
 environment variable  
   EDITOR, 19  
   PATH, 4  
 exit() (built-in function), 37

**F**

forward\_prefix() (built-in function), 36  
 forward\_subject() (built-in function), 37

**G**

get\_contacts() (alot.addressbook.abook.AbookAddressBook method), 46  
 get\_contacts() (alot.addressbook.AddressBook method), 46

**L**

lookup() (alot.addressbook.AddressBook method), 46  
 loop\_hook() (built-in function), 37

**N**

NoMatchingAccount, 46

**P**

PATH, 4  
 post\_buffer\_close() (built-in function), 37  
 post\_buffer\_focus() (built-in function), 37  
 post\_buffer\_open() (built-in function), 37  
 post\_edit\_translate() (built-in function), 36  
 pre\_buffer\_close() (built-in function), 37  
 pre\_buffer\_focus() (built-in function), 37  
 pre\_buffer\_open() (built-in function), 37  
 pre\_edit\_translate() (built-in function), 36  
 pre\_envelope\_send() (built-in function), 35

**R**

read\_config() (in module alot.settings.utils), 46  
 reply\_prefix() (built-in function), 35  
 reply\_subject() (built-in function), 36  
 resolve\_att() (in module alot.settings.utils), 46  
 RFC  
   RFC 1524, 3  
   RFC 3156, 15, 16

**S**

sanitize\_attachment\_filename() (built-in function), 37  
 SIGINT, 6, 52  
 SIGUSR1, 6, 52

## T

`text_quote()` (built-in function), 36

`timestamp_format()` (built-in function), 36

`touch_external_cmdlist()` (built-in function), 36