afew Documentation

Release 0.1pre

Justus Winter

February 17, 2017

Contents

afew is an initial tagging script for notmuch mail:

- http://notmuchmail.org/
- http://notmuchmail.org/initial_tagging/

Its basic task is to provide automatic tagging each time new mail is registered with notmuch. In a classic setup, you might call it after *notmuch new* in an offlineimap post sync hook or in the notmuch *post-new* hook.

In addition to more elementary features such as adding tags based on email headers or maildir folders, handling killed threads and spam, it can do some heavy magic in order to /learn/ how to initially tag your mails based on their content.

fyi: afew plays nicely with *alot*, a GUI for notmuch mail ;)

- https://github.com/pazz/alot

Contents:

CHAPTER 1

Quick Start

The steps to get up and running are:

- install the afew package
- create the config files
- add a notmuch post-new hook that calls afew

Install

The following commands will get you going on Debian/Ubuntu systems:

```
$ sudo aptitude install notmuch python-notmuch dbacl
$ git clone git://github.com/teythoon/afew.git
$ cd afew
$ python setup.py install --prefix
```

Ensure that *~/.local/bin* is in your path. One way is to add the following to your *~/.bashrc*:

```
if [ -d ~/.local/bin ]; then
    PATH=$PATH:~/.local/bin
fi
```

See Installation for a more detailed guide.

Initial Config

Create the directories to hold the config files:

```
$ mkdir -p ~/.config/afew ~/.local/share/afew/categories
```

Make sure that *~/.notmuch-config* reads:

```
[new]
tags=new
```

Put a list of filters into *~/.config/afew/config*:

```
# This is the default filter chain
[SpamFilter]
[ClassifyingFilter]
[KillThreadsFilter]
[ListMailsFilter]
[ArchiveSentMailsFilter]
[InboxFilter]
```

And create a post-new hook for notmuch.

```
$ mkdir -p path/to/maildir/.notmuch/hooks
$ touch path/to/maildir/.notmuch/hooks/post-new
```

Then edit the *post-new* file to contain:

```
#!/bin/sh
$HOME/.local/bin/afew --tag --new
```

Next Steps

You can:

- add extra Filters for more custom filtering
- make use of the Move Mode to move your email between folders
- start using afew's automatic Classification system
- run afew against all your old mail by running *afew –tag –all*
- start Extending afew afew

CHAPTER 2

Installation

Requirements

afew works with python 2.7, 3.1 and 3.2

As well as notmuch and it's python bindings, you'll need dbacl for the text classification. On Debian/Ubuntu systems you can install these by doing:

```
$ sudo aptitude install notmuch python-notmuch dbacl python-dev python-setuptools
```

Unprivileged Install

And I'd like to suggest to install afew as your unprivileged user.

```
$ python setup.py install --prefix=~/.local
$ mkdir -p ~/.config/afew ~/.local/share/afew/categories
```

If you do, make sure *~/.local/bin* is in your path, say by putting the following in your *~/.bashrc*:

```
if [ -d ~/.local/bin ]; then
    PATH=$PATH:~/.local/bin
fi
```

If you want to do a system wide install you can leave off the *–prefix* option.

CHAPTER 3

Command Line Usage

Ultimately afew is a command line tool. You have to specify an action, and whether to act on all messages, or only on new messages. The actions you can choose from are:

tag run the tag filters. See Initial tagging.

watch continuously monitor the mailbox for new files

move-mails move mail files between maildir folders

learn=LEARN train the category with the messages matching the given query

update update the categories [requires no query]

update-reference update the reference category (takes quite some time) [requires no query]

classify classify each message matching the given query (to test the trained categories)

Initial tagging

Basic tagging stuff requires no configuration, just run

```
$ afew --tag --new
# or to tag *all* messages
$ afew --tag --all
```

To do this automatically you can add the following hook into your *~/.offlineimaprc*:

```
postsynchook = ionice -c 3 chrt --idle 0 /bin/sh -c "notmuch new && afew --tag --new"
```

There is a lot more to say about general filter Configuration and the different Filters provided by afew.

Simulation

Adding *–dry-run* to any *–tag* or *–sync-tags* action prevents modification of the notmuch db. Add some *-vv* goodness to see some action.

Move Mode

To invoke afew in move mode, provide the *–move-mails* option on the command line. Move mode will respect *–dry-run*, so throw in *–verbose* and watch what effects a real run would have.

In move mode, afew will check all mails (or only recent ones) in the configured maildir folders, deciding whether they should be moved to another folder.

The decision is based on rules defined in your config file. A rule is bound to a source folder and specifies a target folder into which a mail will be moved that is matched by an associated query.

This way you will be able to transfer your sorting principles roughly to the classic folder based maildir structure understood by your traditional mail server. Tag your mails with notmuch, call afew *–move-mails* in an offlineimap presynchook and enjoy a clean inbox in your webinterface/GUI-client at work.

For information on how to configure rules for move mode, what you can do with it and what you can't, please refer to Move Mode.

Classify

The *–learn*, *–update*, *–update-references* and *–classify* actions all relate to learning how to filter your email. See the Classification page for details.

Commandline help

The full set of options is:

```
$ afew --help
Usage: afew [options] [--] [query]

Options:
  -h, --help            show this help message and exit

  Actions:
    Please specify exactly one action (both update actions can be
    specified simultaniously).

    -t, --tag           run the tag filters
    -w, --watch         continuously monitor the mailbox for new files
    -l LEARN, --learn=LEARN
                        train the category with the messages matching the
                        given query
    -u, --update        update the categories [requires no query]
    -U, --update-reference
                        update the reference category (takes quite some time)
                        [requires no query]
    -c, --classify      classify each message matching the given query (to
                        test the trained categories)
    -m, --move-mails    move mail files between maildir folders

  Query modifiers:
    Please specify either --all or --new or a query string. The default
    query for the update actions is a random selection of
    REFERENCE_SET_SIZE mails from the last REFERENCE_SET_TIMEFRAME days.

    -a, --all           operate on all messages
    -n, --new           operate on all new messages

  General options:
    -C NOTMUCH_CONFIG, --notmuch-config=NOTMUCH_CONFIG
```

```
                      path to the notmuch configuration file [default:
                      $NOTMUCH_CONFIG or ~/.notmuch-config]
  -e ENABLE_FILTERS, --enable-filters=ENABLE_FILTERS
                      filter classes to use, separated by ',' [default:
                      filters specified in afew's config]
  -d, --dry-run       don't change the db [default: False]
  -R REFERENCE_SET_SIZE, --reference-set-size=REFERENCE_SET_SIZE
                      size of the reference set [default: 1000]
  -T DAYS, --reference-set-timeframe=DAYS
                      do not use mails older than DAYS days [default: 30]
  -v, --verbose       be more verbose, can be given multiple times
```

CHAPTER 4

Configuration

Configuration File

Customization of tag filters takes place in afew's config file in *~/.config/afew/config*.

NotMuch Config

afew tries to adapt to the new tag that notmuch sets on new email, but has mostly been developed and used against the **new** tag. To use that, make sure that *~/.notmuch-config* contains:

```
[new]
tags=new
```

Filter Configuration

You can modify filters, and define your own versions of the base Filter that allow you to tag messages in a similar way to the *notmuch tag* command, using the config file. The default config file is:

```
[SpamFilter]
[ClassifyingFilter]
[KillThreadsFilter]
[ListMailsFilter]
[ArchiveSentMailsFilter]
[InboxFilter]
```

See the Filters page for the details of those filters and the custom arguments they accept.

You can add filters based on the base filter as well. These can be customised by specifying settings beneath them. The standard settings, which apply to all filters, are:

message text that will be displayed while running this filter if the verbosity is high enough.

query the query to use against the messages, specified in standard notmuch format. Note that you don't need to specify the **new** tag - afew will add that when run with the *–new* flag.

tags the tags to add or remove for messages that match the query. Tags to add are preceded by a **+** and tags to remove are preceded by a **-**. Multiple tags are separated by semicolons.

tags_blacklist if the message has one of these tags, don't add *tags* to it. Tags are separated by semicolons.

So to add the **deer** tag to any message to or from *antelope@deer.com* you could do:

```
[Filter.1]
query = 'antelope@deer.com'
tags = +deer
message = Wild animals ahoy
```

You can also (in combination with the InboxFilter) have email skip the Inbox by removing the new tag before you get to the InboxFilter:

```
[Filter.2]
query = from'pointyheaded@boss.com'
tags = -new;+boss
message = Message from above
```

Full Sample Config

Showing some sample configs is the easiest way to understand. The notmuch initial tagging page shows a sample config:

```
# immediately archive all messages from "me"
notmuch tag -new -- tag:new and from:me@example.com

# delete all messages from a spammer:
notmuch tag +deleted -- tag:new and from:spam@spam.com

# tag all message from notmuch mailing list
notmuch tag +notmuch -- tag:new and to:notmuch@notmuchmail.org

# finally, retag all "new" messages "inbox" and "unread"
notmuch tag +inbox +unread -new -- tag:new
```

The (roughly) equivalent set up in afew would be:

```
[ArchiveSentMailsFilter]

[Filter.spamcom]
message = Delete all messages from spammer
query = from:spam@spam.com
tags = +deleted;-new

[Filter.notmuch]
message = Tag all messages from the notmuch mailing list
query = to:notmuch@notmuchmail.org
tags = +notmuch

[InboxFilter]
```

Not that the queries do not generally include *tag:new* because this is implied when afew is run with the *–new* flag.

The differences between them is that

- the ArchiveSentMailsFilter will add the **sent** tag, as well as archiving the email. And it will not archive email that has been sent to one of your own addresses.
- the InboxFilter does not add the **unread** tag. But most mail clients will manage the unread status directly in maildir.

More Filter Examples

Here are a few more example filters from github dotfiles:

```
[Filter.1]
query = 'sicsa-students@sicsa.ac.uk'
tags = +sicsa
message = sicsa

[Filter.2]
query = 'from:foosoc.ed@gmail.com OR from:GT Silber OR from:lizzie.brough@eusa.ed.ac.uk'
tags = +soc;+foo
message = foosoc

[Filter.3]
query = 'folder:gmail/G+'
tags = +G+
message = gmail spam

# skip inbox
[Filter.6]
query = 'to:notmuch@notmuchmail.org AND (subject:emacs OR subject:elisp OR "(defun" OR "(setq" OR PA
tags = -new
message = notmuch emacs stuff
```

CHAPTER 5

Filters

The default filter set (if you don't specify anything in the config) is:

```
[SpamFilter]
[ClassifyingFilter]
[KillThreadsFilter]
[ListMailsFilter]
[ArchiveSentMailsFilter]
[InboxFilter]
```

The standard filter Configuration can be applied to these filters as well. Though note that most of the filters below set their own value for message, query and/or tags, and some ignore some of the standard settings.

SpamFilter

The settings you can use are:

- spam_tag = <tag>
- Add <tag> to all mails recognized as spam.
- The default is 'spam'.
- You may use it to tag your spam as 'junk', 'scum' or whatever suits your mood. Note that only a single tag is supported here.

Email will be considered spam if the header *X-Spam-Flag* is present.

ClassifyingFilter

This filter will tag messages based on what it has learnt from seeing how you've tagged messages in the past. See Classification for more details.

KillThreadsFilter

If the new message has been added to a thread that has already been tagged **killed** then add the **killed** tag to this message. This allows for ignoring all replies to a particular thread.

ListMailsFilter

This filter looks for the *List-Id* header, and if it finds it, adds a tag **lists** and a tag named **lists/<list-id>**.

SentMailsFilter

The settings you can use are:

- sent_tag = <tag>
- Add <tag> to all mails sent from one of your configured mail addresses.
- The default is to add no tag, so you need to specify something.
- You may e.g. use it to tag all mails sent by you as 'sent'. This may make special sense in conjunction with a mail client that is able to not only search for threads but individual mails as well.

  More accurately, it looks for emails that are from one of your addresses *and not* to any of your addresses.
- to_transforms = <transformation rules>
- Transform *To*/*Cc*/*Bcc* e-mail addresses to tags according to the specified rules. <transformation rules> is a space separated list consisting of 'user_part@domain_part:tags' style pairs. The colon separates the e-mail address to be transformed from tags it is to be transformed into. ':tags' is optional and if empty, 'user_part' is used as tag. 'tags' can be a single tag or semi-colon separated list of tags.
- It can be used for example to easily tag posts sent to mailing lists which at this stage don't have *List-Id* field.

ArchiveSentMailsFilter

It extends *SentMailsFilter* with the following feature:

- Emails filtered by this filter have the **new** tag removed, so will not have the **inbox** tag added by the InboxFilter.

InboxFilter

This removes the **new** tag, and adds the **inbox** tag, to any message that isn't killed or spam. (The new tags are set in your notmuch config, and default to just **new**.)

HeaderMatchingFilter

This filter adds tags to a message if the named header matches the regular expression given. The tags can be set, or based on the match. The settings you can use are:

- header = <header_name>
- pattern = <regex_pattern>
- tags = <tag_list>

If you surround a tag with *{}* then it will be replaced with the named match.

Some examples are:

```
[HeaderMatchingFilter.1]
header = X-Spam-Flag
pattern = YES
tags = +spam

[HeaderMatchingFilter.2]
header = List-Id
pattern = <(?P<list_id>.*)>
tags = +lists;+{list_id}

[HeaderMatchingFilter.3]
header = X-Redmine-Project
pattern = (?P<project>.*)
tags = +redmine;+{project}
```

SpamFilter and ListMailsFilter are implemented using HeaderMatchingFilter, and are only slightly more complicated than the above examples.

FolderNameFilter

This looks at which folder each email is in and uses that name as a tag for the email. So if you have a procmail or sieve set up that puts emails in folders for you, this might be useful.

- folder_explicit_list = <folder list>
- Tag mails with tag in <folder list> only. <folder list> is a space separated list, not enclosed in quotes or any other way.
- Empty list means all folders (of course blacklist still applies).
- The default is empty list.
- You may use it e.g. to set tags only for specific folders like 'Sent'.
- folder_blacklist = <folder list>
- Never tag mails with tag in <folder list>. <folder list> is a space separated list, not enclosed in quotes or any other way.
- The default is to blacklist no folders.
- You may use it e.g. to avoid mails being tagged as 'INBOX' when there is the more standard 'inbox' tag.
- folder_transforms = <transformation rules>
- Transform folder names according to the specified rules before tagging mails. <transformation rules> is a space separated list consisting of 'folder:tag' style pairs. The colon separates the name of the folder to be transformed from the tag it is to be transformed into.
- The default is to transform to folder names.
- You may use the rules e.g. to transform the name of your 'Junk' folder into your 'spam' tag or fix capitalization of your draft and sent folder:

```
folder transforms = Junk:spam Drafts:draft Sent:sent
```

- maildir_separator = <sep>
- Use <sep> to split your maildir hierarchy into individual tags.
- The default is to split on '.'

- If your maildir hierarchy is represented in the filesystem as collapsed dirs, <sep> is used to split it again before applying tags. If your maildir looks like this:

```
[...]
/path/to/maildir/devel.afew/[cur|new|tmp]/...
/path/to/maildir/devel.alot/[cur|new|tmp]/...
/path/to/maildir/devel.notmuch/[cur|new|tmp]/...
[...]
```

the mails in your afew folder will be tagged with 'devel' and 'afew'.

If instead your hierarchy is split by a more conventional '/' or any other divider

```
[...]
/path/to/maildir/devel/afew/[cur|new|tmp]/...
/path/to/maildir/devel/alot/[cur|new|tmp]/...
/path/to/maildir/devel/notmuch/[cur|new|tmp]/...
[...]
```

you need to configure that divider to have your mails properly tagged:

```
maildir_separator = /
```

Customizing filters

To customize these filters, there are basically two different possibilities:

Let's say you like the SpamFilter, but it is way too polite

1. Create an filter object and customize it

```
[SpamFilter.0] # note the index
message = meh
```

The index is required if you want to create a new SpamFilter *in addition to* the default one. If you need just one customized SpamFilter, you can drop the index and customize the default instance.

2. Create a new type...

```
[ShitFilter(SpamFilter)]
message = I hatez teh spam!
```

and create an object or two

```
[ShitFilter.0]
[ShitFilter.1]
message = Me hatez it too.
```

You can provide your own filter implementations too. You have to register your filters via entry points. See the afew setup.py for examples on how to register your filters. To add your filters, you just need to install your package in the context of the afew application.

CHAPTER 6

Move Mode

Configuration Section

Here is a full sample configuration for move mode:

```
[MailMover]
folders = INBOX Junk
max_age = 15

# rules
INBOX = 'tag:spam':Junk 'NOT tag:inbox':Archive
Junk = 'NOT tag:spam AND tag:inbox':INBOX 'NOT tag:spam':Archive
```

Below we explain what each bit of this means.

Rules

First you need to specify which folders should be checked for mails that are to be moved (as a whitespace separated list):

```
folders = INBOX Junk
```

Then you have to specify rules that define move actions of the form

```
<src> = ['<qry>':<dst>]+
```

Every mail in the <*src*> folder that matches a <*qry*> will be moved into the <*dst*> folder associated with that query. A message that matches multiple queries will be copied to multiple destinations.

You can bind as many rules to a maildir folder as you deem necessary. Just add them as elements of a (whitespace separated) list.

Please note, though, that you need to specify at least one rule for every folder given by the *folders* option and at least one folder to check in order to use the move mode.

```
INBOX = 'tag:spam':Junk
```

will bind one rule to the maildir folder *INBOX* that states that all mails in said folder that carry (potentially among others) the tag **spam** are to be moved into the folder *Junk*.

With <*qry*> being an arbitrary notmuch query, you have the power to construct arbitrarily flexible rules. You can check for the absence of tags and look out for combinations of attributes:

```
Junk = 'NOT tag:spam AND tag:inbox':INBOX 'NOT tag:spam':Archive
```

The above rules will move all mails in *Junk* that don't have the **spam** tag but do have an **inbox** tag into the directory *INBOX*. All other mails not tagged with **spam** will be moved into *Archive*.

Max Age

You can limit the age of mails you want to move by setting the *max_age* option in the configuration section. By providing

```
max_age = 15
```

afew will only check mails at most 15 days old.

Limitations

(1) Rules don't manipulate tags.

```
INBOX = 'NOT tag:inbox':Archive
Junk = 'NOT tag:spam':INBOX
```

The above combination of rules might prove tricky, since you might expect de-spammed mails to end up in *INBOX*. But since the *Junk* rule will *not* add an **inbox** tag, the next run in move mode might very well move the matching mails into *Archive*.

Then again, if you remove the **spam** tag and do not set an **inbox** tag, how would you come to expect the mail would end up in your INBOX folder after moving it? ;)

(2) There is no 1:1 mapping between folders and tags. And that's a feature. If you tag a mail with two tags and there is a rule for each of them, both rules will apply. Your mail will be copied into two destination folders, then removed from its original location.

CHAPTER 7

Classification

In Action

Let's train on an existing tag *spam*:

```
$ afew --learn spam -- tag:spam
```

Let's build the reference category. This is important to reduce the false positive rate. This may take a while...

```
$ afew --update-reference
```

And now let's create a new tag from an arbitrary query result:

```
$ afew -vv --learn sourceforge -- sourceforge
```

Let's see how good the classification is:

```
$ afew --classify -- tag:inbox and not tag:killed
Sergio López <slpml@sinrega.org> (2011-10-08) (bug-hurd inbox lists unread) --> no match
Patrick Totzke <reply+i-1840934-9a702d09342dca2b120126b26b008d0deea1731e@reply.github.com> (2011-10-0
[...]
```

As soon as you trained some categories, afew will automatically tag your new mails using the classifier. If you want to disable this feature, either use the *–enable-filters* option to override the default set of filters or remove the files in your afew state dir:

```
$ ls ~/.local/share/afew/categories
alot juggling  reference_category  sourceforge  spam
```

You need to update the category files periodically. I'd suggest to run

```
$ afew --update
```

on a weekly and

```
$ afew --update-reference
```

on a monthly basis.

CHAPTER 8

Extending afew

You can put python files in *~/.config/afew/* and they will be imported by afew. If you use that python file to define a *Filter* class and use the *register_filter* decorator then you can refer to it in your filter configuration.

So an example small filter you could add might be:

```
from afew.Filter import Filter, register_filter

PROJECT_MAPPING = {
    'fabric': 'deployment',
    'oldname': 'new-name',
}

@register_filter
class RedmineFilter(Filter):
    message = 'Create tag based on redmine project'
    query = 'NOT tag:redmine'

    def handle_message(self, message):
        project = message.get_header('X-Redmine-Project')
        if project in PROJECT_MAPPING:
            project = PROJECT_MAPPING[project]
        self.add_tags(message, 'redmine', project)
```

We have defined the *message* and *query* class variables that are used by the parent class *Filter*. The *message* is printed when running with verbose flags. The *query* is used to select messages to run against - here we ensure we don't bother looking at messages we've already looked at.

The *handle_message()* method is the key one to implement. This will be called for each message that matches the query. The argument is a notmuch message object and the key methods used by the afew filters are *get_header()*, *get_filename()* and *get_thread()*.

Of the methods inherited from the *Filter* class the key ones are *add_tags()* and *remove_tags()*, but read about the Implementation or just read the source code to get your own ideas.

Once you've defined your filter, you can add it to your config like any other filter:

```
[RedmineFilter]
```

CHAPTER 9

Implementation

Database Manager

The design of the database manager was inspired by alots database manager `alot.db.DBManager`.

class `afew.Database.`**Database**

Convenience wrapper around *notmuch*.

add_message(*path*, *sync_maildir_flags=False*, *new_mail_handler=None*)

Adds the given message to the notmuch index.

Parameters

- **path** (*str*) – path to the message
- **sync_maildir_flags** (*bool*) – if *True* notmuch converts the standard maildir flags to tags
- **new_mail_handler** (a function that is called with a `notmuch.Message` object as its only argument) – callback for new messages

Raises `notmuch.NotmuchError` if adding the message fails

Returns a `notmuch.Message` object

close()

Closes the notmuch database if it has been opened.

do_query(*query*)

Executes a notmuch query.

Parameters **query** (*str*) – the query to execute

Returns the query result

Return type `notmuch.Query`

get_messages(*query*, *full_thread=False*)

Get all messages mathing the given query.

Parameters

- **query** (*str*) – the query to execute using *Database.do_query()*
- **full_thread** (*bool*) – return all messages from mathing threads

Returns an iterator over `notmuch.Message` objects

mail_bodies_matching(**args*, ***kwargs*)
Filters each message yielded from *Database.get_messages()* through *afew.utils.extract_mail_body()*.

This functions accepts the same arguments as *Database.get_messages()*.

Returns an iterator over `list` of `str`

remove_message(*path*)
Remove the given message from the notmuch index.

Parameters **path** (*str*) – path to the message

walk_replies(*message*)
Returns all replies to the given message.

Parameters **message** (`notmuch.Message`) – the message to start from

Returns an iterator over `notmuch.Message` objects

walk_thread(*thread*)
Returns all messages in the given thread.

Parameters **message** (`notmuch.Thread`) – the tread you are interested in

Returns an iterator over `notmuch.Message` objects

Filter

Mail classification

class `afew.DBACL.`**Classifier**(*categories*, *database_directory=u'/home/docs/.local/share/afew/categories'*)

class `afew.DBACL.`**DBACL**(*database_directory=u'/home/docs/.local/share/afew/categories'*)

Configuration management

Miscellanious utility functions

`afew.utils.`**extract_mail_body**(*message*)
Extract the plain text body of the message with signatures stripped off.

Parameters **message** (`notmuch.Message`) – the message to extract the body from

Returns the extracted text body

Return type `list` of `str`

`afew.utils.`**filter_compat**(**args*)
Compatibility wrapper for filter builtin.

The semantic of the filter builtin has been changed in python3.x. This is a temporary workaround to support both python versions in one code base.

`afew.utils.`**strip_signatures**(*lines*, *max_signature_size=10*)
Strip signatures from a mail. Used to filter mails before classifying mails.

Parameters

- **lines** (list of str) – a mail split at newlines
- **max_signature_size** (*int*) – consider message parts up to this size as signatures

Returns the mail with signatures stripped off

Return type list of str

```
>>> strip_signatures([
...     'Huhu',
...     '--',
...     'Ikke',
... ])
['Huhu']
>>> strip_signatures([
...     'Huhu',
...     '--',
...     'Ikke',
...     '**',
...     "Sponsored by PowerDoh\'",
...     "Sponsored by PowerDoh\'",
...     "Sponsored by PowerDoh\'",
...     "Sponsored by PowerDoh\'",
...     "Sponsored by PowerDoh\'",
... ], 5)
['Huhu']
```

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

a

A

C

D

E

F

G

M

R

S

W