imapautofiler Documentation

Release 1.6.0-2-g69f0a81

Doug Hellmann

Nov 19, 2017

Contents:

imapautofiler is a tool for managing messages on an IMAP server based on rules for matching properties such as the recipient or header content. The author uses it to move messages from his "Sent" folder to the appropriate archive folders.

CHAPTER 1

Installing

Install `imapautofiler` with pip under Python 3.5 or greater.

```
$ pip install imapautofiler
```

Note: Using a virtualenv is a good practice.

CHAPTER 2

Configuring

The application is configured through a YAML file. The default file, `~/.imapautofiler.yml`, is read if no other file is specified on the command line.

2.1 Server Connection

Each configuration file can hold one server specification.

```
server:
  hostname: example.com
  username: my-user@example.com
```

imapautofiler also supports using the keyring module to store and retrieve a password from your system keyring:

```
server:
  hostname: example.com
  username: my-user@example.com
  use_keyring: true
```

In this scenario, you will be asked for the password on first run. The password will be stored in your operating system's secure keyring and reused when running the app.

If you do not want to use the keyring, the connection section can optionally include a password.

```
server:
  hostname: example.com
  username: my-user@example.com
  password: super-secret
```

Warning: Because the password is kept in clear text, this mode of operation is only recommended when the configuration file is kept secure by other means.

If the password is not provided in the configuration file and `use_keyring` is not true, `imapautofiler` will prompt for a value when it tries to connect to the server.

You can also optionally provide the IMAP servers port and a custom CA file. This is helpful if your company uses custom ports and self issued certs.

```
server:
  hostname: example.com
  username: my-user@example.com
  port: 1234
  ca_file: path/to/ca_file.pem
```

2.2 Maildir Location

As an alternative to a server specification, the configuration file can refer to a local directory containing one or more Maildir folders. This is especially useful when combining imapautofiler with offlineimap.

```
maildir: ~/Mail
```

Note: The directory specified should not itself be a Maildir. It must be a regular directory with nested Maildir folders.

2.3 Trash Mailbox

The `trash` action, for discarding messages without deleting them immediately, requires a configuration setting to know the name of the trash mailbox. There is no default value.

```
trash-mailbox: INBOX.Trash
```

2.4 Mailboxes

The mailboxes that imapautofiler should process are listed under `mailboxes`. Each mailbox has a name and a list of rules.

```
mailboxes:
- name: INBOX
  rules: ...
- name: Sent
  rules: ...
```

2.5 Rules

The rules are organized by mailbox, and then listed in order. The first rule that matches a message triggers the associated action, and then processing for that message stops.

2.5.1 TimeLimit Rules

An Time Limit `time-limit` rule is added by specifying the 'age', number of days for the email to "live" in the specified mailbox. If age = 0, the rule is ignored.

```
- time-limit:
    age: 30
```

2.5.2 Header Rules

A `header` rule can match either a complete header value, a substring, or a regular expression against the contents of a specified message header. If a header does not exist, the content is treated as an empty string. The header text and pattern are both converted to lowercase before the comparison is performed.

This example rule matches messages with the string "[pyatl]" in the subject line.

```
- headers:
    - name: "subject"
      substring: "[pyatl]"
  action:
    name: "move"
    dest-mailbox: "INBOX.PyATL"
```

This example rule matches messages for which the "to" header matches the regular expression `notify-.*@disqus.net`.

```
- headers:
    - name: to
      regex: "notify-.*@disqus.net"
  action:
    name: trash
```

This example rule matches messages for which the "Message-Id" header is exactly `<4FF56508-357B-4E73-82DE-458D3EEB2753@example.com>`.

```
- headers:
    - name: to
      value: "<4FF56508-357B-4E73-82DE-458D3EEB2753@example.com>"
  action:
    name: trash
```

2.5.3 Combination Rules

It is frequently useful to be able to apply the same action to messages with different characteristics. For example, if a mailing list ID appears in the subject line or in the `list-id` header. The `or` rule allows nested rules. If any one matches, the combined rule matches and the associated action is triggered.

For example, this rule matches any message where the PyATL meetup mailing list address is in the `to` or `cc` headers.

```
- or:
    rules:
      - headers:
          - name: "to"
            substring: "pyatl-list@meetup.com"
      - headers:
```

```
          - name: "cc"
            substring: "pyatl-list@meetup.com"
  action:
    name: "move"
    dest-mailbox: "INBOX.PyATL"
```

For more complicated formulations, the `and` rule allows combining other rules so that they all must match the message before the action is taken.

For example, this rule matches any message sent to the PyATL meetup mailing list address with a subject including the text `"meeting update"`.

```
- and:
    rules:
      - headers:
          - name: "to"
            substring: "pyatl-list@meetup.com"
      - headers:
          - name: "subject"
            substring: "meeting update"
  action:
    name: "move"
    dest-mailbox: "INBOX.PyATL"
```

2.5.4 Recipient Rules

The example presented for `or` rules is a common enough case that it is supported directly using the `recipient` rule. If any header listing a recipient of the message matches the substring or regular expression, the action is triggered.

This example is equivalent to the example for `or`.

```
- recipient:
    substring: "pyatl-list@meetup.com"
  action:
    name: "move"
    dest-mailbox: "INBOX.PyATL"
```

2.6 Actions

Each rule is associated with an *action* to be triggered when the rule matches a message.

2.6.1 Move Action

The `move` action copies the message to a new mailbox and then deletes the version in the source mailbox. This action can be used to automatically file messages.

The example below moves any message sent to the PyATL meetup group mailing list into the mailbox `INBOX.PyATL`.

```
- recipient:
    substring: "pyatl-list@meetup.com"
  action:
    name: "move"
    dest-mailbox: "INBOX.PyATL"
```

Different IMAP servers may use different naming conventions for mailbox hierarchies. Use the `--list-mailboxes` option to the command line program to print a list of all of the mailboxes known to the account.

2.6.2 Sort Action

The `sort` action uses data in a message header to determine the destination mailbox for the message. This action can be used to automatically file messages from mailing lists or other common sources if the corresponding mailbox hierarchy is established. A `sort` action is equivalent to `move` except that the destination is determined dynamically.

The action settings may contain a `header` entry to specify the name of the mail header to examine to find the destination. The default is to use the `to` header.

The action data may contain a `dest-mailbox-regex` entry for parsing the header value to obtain the destination mailbox name. If the regex has one match group, that substring will be used. If the regex has more than one match group, the `dest-mailbox-regex-group` option must specify which group to use (0-based numerical index). The default pattern is `([\w-+]+)@` to match the first part of an email address.

The action data must contain a `dest-mailbox-base` entry with the base name of the destination mailbox. The actual mailbox name will be constructed by appending the value extracted via `dest-mailbox-regex` to the `dest-mailbox-base` value. The `dest-mailbox-base` value should contain the mailbox separator character (usually `.`) if the desired mailbox is a sub-folder of the name given.

The example below sorts messages associated with two mailing lists into separate mailboxes under a parent mailbox `INBOX.ML`. It uses the default regular expression to extract the prefix of the `to` header for each message. Messages to the `python-committers@python.org` mailing list are sorted into `INBOX.ML.python-committers` and messages to the `sphinx-dev@googlegroups.com` list are sorted into `INBOX.ML.sphinx-dev`.

```
- or:
    rules:
      - recipient:
          substring: python-committers@python.org
      - recipient:
          substring: sphinx-dev@googlegroups.com
  action:
    name: sort
    dest-mailbox-base: "INBOX.ML."
```

2.6.3 Sort Mailing List Action

The `sort-mailing-list` action works like `sort` configured to read the `list-id` header and extract the portion of the ID between < and >. if they are present. If there are no angle brackets in the ID, the entire value is used. As with `sort` the `dest-mailbox-regex` can be specified in the rule to change this behavior.

The example below sorts messages to any mailing list into separate folders under `INBOX.ML`.

```
- is-mailing-list: {}
  action:
    name: sort-mailing-list
    dest-mailbox-base: "INBOX.ML."
```

2.6.4 Trash Action

Moving messages to the "trash can" is a less immediate way of deleting them. Messages in the trash can can typically be recovered until they expire, or until the trash is emptied explicitly.

Using this action requires setting the global `trash-mailbox` option (see *Trash Mailbox*). If the action is triggered and the option is not set, the action reports an error and processing stops.

This example moves messages for which the "to" header matches the regular expression `notify-.*@disqus.net` to the trash mailbox.

```
- headers:
    - name: to
      regex: "notify-.*@disqus.net"
  action:
    name: trash
```

2.6.5 Delete Action

The `delete` action is more immediately destructive. Messages are permanently removed from the mailbox as soon as the mailbox is closed.

This example deletes messages for which the "to" header matches the regular expression `notify-.*@disqus.net`.

```
- headers:
    - name: to
      regex: "notify-.*@disqus.net"
  action:
    name: delete
```

2.7 Complete example configuration file

Here's an example of a configuration file with all the possible parts.

```
server:
  hostname: imap.gmail.com
  username: user@example.com
  password: xxxxxxxxxxxxx

trash-mailbox: "[Gmail]/Trash"

mailboxes:
- name: INBOX
  rules:
  - headers:
    - name: "from"
      substring: user1@example.com
    action:
      name: "move"
      dest-mailbox: "User1 correspondence"
  - headers:
    - name: recipient
      substring: dev-team
    - name: subject
      substring: "[Django] ERROR"
    action:
      name: "move"
      dest-mailbox: "Django Errors"
```

CHAPTER 3

Running

Run `imapautofiler` on the command line.

```
$ imapautofiler -h
usage: imapautofiler [-h] [-v] [--debug] [-c CONFIG_FILE] [--list-mailboxes]

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         report more details about what is happening
  --debug               turn on imaplib debugging output
  -c CONFIG_FILE, --config-file CONFIG_FILE
  --list-mailboxes      instead of processing rules, print a list of mailboxes
```

When run with no arguments, it reads the default configuration file and processes the rules.

```
$ imapautofiler
Password for my-user@example.com
Trash: 13767 (Re: spam message from disqus comment) to INBOX.Trash
Move: 13771 (Re: [Openstack-operators] [deployment] [oslo] [ansible] [tripleo]␣
↪[kolla] [helm] Configuration management with etcd / confd) to INBOX.OpenStack.Misc␣
↪Lists
imapautofiler: encountered 10 messages, processed 2
```

Different IMAP servers may use different naming conventions for mailbox hierarchies. Use the `--list-mailboxes` option to the command line program to print a list of all of the mailboxes known to the account.

```
$ imapautofiler --list-mailboxes
Password for my-user@example.com:
INBOX
INBOX.Archive
INBOX.Conferences.PyCon-Organizers
INBOX.ML.TIP
INBOX.ML.python-announce-list
INBOX.OpenStack.Dev List
INBOX.PSF
```

```
INBOX.Personal
INBOX.PyATL
INBOX.Sent Items
INBOX.Sent Messages
INBOX.Trash
INBOX.Work
```

CHAPTER 4

Contributing

The source code and bug tracker for imapautofiler are hosted on github.

https://github.com/dhellmann/imapautofiler

The source code is released under the Apache 2.0 license. All patches should use the same license.

When reporting a bug, please specify the version of imapautofiler you are using.

4.1 Message handling rules

imapautofiler operation is driven by *rules* and *actions* defined in the *configuration file*.

Each rule is evaluated by a class derived from *Rule* and implemented in `imapautofiler/rules.py`.

A rule must implement the `check()` method to support the interface defined by the abstract base class. The method receives an email.message.Message instance containing the message being processed. `check()` must return `True` if the rule should be applied to the message, or `False` if it should not.

Each new rule must be handled in the *factory()* function so that when the name of the rule is encountered the correct rule class is instantiated and returned.

4.2 Message handling actions

Each action triggered by a rule is evaluated by a class derived from *Action* and implemented in `imapautofiler/actions.py`.

An action must implement the `invoke()` method to support the interface defined by the abstract base class. The method receives an `IMAPClient` instance (from the imapclient package) connected to the IMAP server being scanned, a string message ID, and an email.message.Message instance containing the message being processed. `invoke()` must perform the relevant operation on the message.

Each new action must be handled in the *factory()* function so that when the name of the action is encountered the correct action class is instantiated and returned.

4.3 API Documentation

4.3.1 The imapautofiler.actions Module

class imapautofiler.actions.**Action**(*action_data*, *cfg*)
Bases: object

Base class

NAME = None

invoke(*conn*, *mailbox_name*, *message_id*, *message*)
Run the action on the message.

Parameters

- **conn** (imapautofiler.client.Client) – connection to mail server
- **mailbox_name** (*str*) – name of the mailbox holding the message
- **message_id** (*str*) – ID of the message to process
- **message** (*email.message.Message*) – the message object to process

class imapautofiler.actions.**Delete**(*action_data*, *cfg*)
Bases: *imapautofiler.actions.Action*

Delete the message immediately.

The action is indicated with the name delete.

NAME = 'delete'

invoke(*conn*, *mailbox_name*, *message_id*, *message*)

class imapautofiler.actions.**Move**(*action_data*, *cfg*)
Bases: *imapautofiler.actions.Action*

Move the message to a different folder.

The action is indicated with the name move.

The action data must contain a dest-mailbox entry with the name of the destination mailbox.

NAME = 'move'

invoke(*conn*, *src_mailbox*, *message_id*, *message*)

class imapautofiler.actions.**Sort**(*action_data*, *cfg*)
Bases: *imapautofiler.actions.Action*

Move the message based on parsing a destination from a header.

The action is indicated with the name sort.

The action may contain a header entry to specify the name of the mail header to examine to find the destination. The default is to use the to header.

The action data may contain a dest-mailbox-regex entry for parsing the header value to obtain the destination mailbox name. If the regex has one match group, that substring will be used. If the regex has more than one match group, the dest-mailbox-regex-group option must specify which group to use (0-based numerical index). The default pattern is ([\w-+]+)@ to match the first part of an email address.

The action data must contain a dest-mailbox-base entry with the base name of the destination mailbox. The actual mailbox name will be constructed by appending the value extracted via dest-mailbox-regex to

the `dest-mailbox-base` value. The `dest-mailbox-base` value should contain the mailbox separator character (usually `.`) if the desired mailbox is a sub-folder of the name given.

NAME = 'sort'

invoke(*conn*, *src_mailbox*, *message_id*, *message*)

class imapautofiler.actions.**SortMailingList**(*action_data*, *cfg*)
Bases: *imapautofiler.actions.Sort*

Move the message based on the mailing list id.

The action is indicated with the name `sort-mailing-list`.

This action is equivalent to the `sort` action with header set to `list-id` and `dest-mailbox-regex` set to `<?([^.]+)\..*>?`.

NAME = 'sort-mailing-list'

class imapautofiler.actions.**Trash**(*action_data*, *cfg*)
Bases: *imapautofiler.actions.Move*

Move the message to the trashcan.

The action is indicated with the name `trash`.

The action expects the global configuration setting `trash-mailbox`.

NAME = 'trash'

imapautofiler.actions.**factory**(*action_data*, *cfg*)
Create an Action instance.

Parameters

- **action_data** (*dict*) – portion of configuration describing the action
- **cfg** (*dict*) – full configuration data

Using the action type, instantiate an action object that can process a message.

4.3.2 The imapautofiler.app Module

imapautofiler.app.**list_mailboxes**(*cfg*, *debug*, *conn*)
Print a list of the mailboxes.

Parameters

- **cfg** (*dict*) – full configuration
- **debug** (*bool*) – flag to control debug output
- **conn** (*imapautofiler.client.Client*) – IMAP server onnection

Used by the `--list-mailboxes` switch.

imapautofiler.app.**main**(*args=None*)

imapautofiler.app.**process_rules**(*cfg*, *debug*, *conn*)
Run the rules from the configuration file.

Parameters

- **cfg** (*dict*) – full configuration
- **debug** (*bool*) – flag to control debug output

- **conn** (imapautofiler.client.Client) – IMAP server onnection

4.3.3 The imapautofiler.client Module

Mail client API.

class imapautofiler.client.**Client**(*cfg*)
Bases: object

close()
Close the connection, flushing any pending changes.

copy_message(*src_mailbox*, *dest_mailbox*, *message_id*, *message*)
Create a copy of the message in the destination mailbox.

Parameters

- **src_mailbox** (*str*) – name of the source mailbox
- **dest_mailbox** (*src*) – name of the destination mailbox
- **message_id** (*str*) – ID of the message to copy
- **message** (*email.message.Message*) – message instance

delete_message(*src_mailbox*, *message_id*, *message*)
Remove the message.

Parameters

- **src_mailbox** (*str*) – name of the source mailbox
- **message_id** (*str*) – ID of the message to copy
- **message** (*email.message.Message*) – message instance

expunge()
Flush any pending changes.

list_mailboxes()
Return a list of mailbox names.

mailbox_iterate(*mailbox_name*)
Iterate over messages from the mailbox.

Produces tuples of (message_id, message).

move_message(*src_mailbox*, *dest_mailbox*, *message_id*, *message*)
Move the message from the source to the destination mailbox.

Parameters

- **src_mailbox** (*str*) – name of the source mailbox
- **dest_mailbox** (*src*) – name of the destination mailbox
- **message_id** (*str*) – ID of the message to copy
- **message** (*email.message.Message*) – message instance

class imapautofiler.client.**IMAPClient**(*cfg*)
Bases: *imapautofiler.client.Client*

close()

copy_message(*src_mailbox*, *dest_mailbox*, *message_id*, *message*)

delete_message(*src_mailbox*, *message_id*, *message*)

expunge()

list_mailboxes()
Return a list of folder names.

mailbox_iterate(*mailbox_name*)

class imapautofiler.client.**MaildirClient**(*cfg*)
Bases: *imapautofiler.client.Client*

close()

copy_message(*src_mailbox*, *dest_mailbox*, *message_id*, *message*)

delete_message(*src_mailbox*, *message_id*, *message*)

expunge()

list_mailboxes()

mailbox_iterate(*mailbox_name*)

imapautofiler.client.**open_connection**(*cfg*)
Open a connection to the mail server.

4.3.4 The imapautofiler.config Module

imapautofiler.config.**get_config**(*filename*)
Return the configuration data.

Parameters **filename** (*str*) – name of configuration file to read

Read `filename` and parse it as a YAML file, then return the results.

4.3.5 The imapautofiler.i18n Module

imapautofiler.i18n.**get_header_value**(*msg*, *name*)
Handle header decoding and return a string we examine.

4.3.6 The imapautofiler.lookup Module

imapautofiler.lookup.**make_lookup_table**(*cls*, *attr_name*)

4.3.7 The imapautofiler.rules Module

class imapautofiler.rules.**And**(*rule_data*, *cfg*)
Bases: *imapautofiler.rules.Rule*

True if all of the sub-rules are true.

The rule data must contain a `rules` list with other rules specifications.

Actions on the sub-rules are ignored.

NAME = 'and'

check(*message*)

class imapautofiler.rules.**HeaderExactValue**(*rule_data*, *cfg*)
Bases: imapautofiler.rules._HeaderMatcher

class imapautofiler.rules.**HeaderExists**(*rule_data*, *cfg*)
Bases: *imapautofiler.rules.Rule*

Looks for a message to have a given header.

NAME = 'header-exists'

check(*message*)

class imapautofiler.rules.**HeaderRegex**(*rule_data*, *cfg*)
Bases: imapautofiler.rules._HeaderMatcher

Implements regular expression matching for headers.

class imapautofiler.rules.**HeaderSubString**(*rule_data*, *cfg*)
Bases: imapautofiler.rules._HeaderMatcher

Implements substring matching for headers.

class imapautofiler.rules.**Headers**(*rule_data*, *cfg*)
Bases: *imapautofiler.rules.Rule*

True if all of the headers match.

The rule data must contain a `headers` list of mappings containing a `name` for the header itself and either a `substring` key mapped to a simple string or a `regex` key mapped to a regular expression to be matched against the value of the header.

NAME = 'headers'

check(*message*)

class imapautofiler.rules.**IsMailingList**(*rule_data*, *cfg*)
Bases: *imapautofiler.rules.HeaderExists*

Looks for a message to have a given header.

NAME = 'is-mailing-list'

class imapautofiler.rules.**Or**(*rule_data*, *cfg*)
Bases: *imapautofiler.rules.Rule*

True if any one of the sub-rules is true.

The rule data must contain a `rules` list with other rules specifications.

Actions on the sub-rules are ignored.

NAME = 'or'

check(*message*)

class imapautofiler.rules.**Recipient**(*rule_data*, *cfg*)
Bases: *imapautofiler.rules.Or*

True if any recipient sub-rule matches.

The rule data must contain a `recipient` mapping containing either a `substring` key mapped to a simple string or a `regex` key mapped to a regular expression.

NAME = 'recipient'

class imapautofiler.rules.**Rule**(*rule_data*, *cfg*)
Bases: object

Base class

NAME = None

check(*message*)
Test the rule on the message.

Parameters

- **conn** (*imapclient.IMAPClient*) – connection to IMAP server
- **message** (*email.message.Message*) – the message object to process

get_action()

class imapautofiler.rules.**TimeLimit**(*rule_data*, *cfg*)
Bases: *imapautofiler.rules.Rule*

True if message is older than the specified 'age' measured in number of days.

NAME = 'time-limit'

check(*message*)

imapautofiler.rules.**factory**(*rule_data*, *cfg*)
Create a rule processor.

Parameters

- **rule_data** (*dict*) – portion of configuration describing the rule
- **cfg** (*dict*) – full configuration data

Using the rule type, instantiate a rule processor that can check the rule against a message.

4.4 Local Test Maildir

Use `tools/maildir_test_data.py` to create a local test Maildir with a few sample messages. The script requires several dependencies, so for convenience there is a tox environment pre-configured to run it in a virtualenv.

The script requires one argument to indicate the parent directory where the Maildirs should be created.

```
$ tox -e testdata -- /tmp/testdata
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

N

O

P

R

S

T