imapautofiler Documentation

Release 1.1.1-18-gaccb741

Doug Hellmann

Aug 01, 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.

Server Connection

Each configuration file can hold one server specification.

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

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, `imapautofiler` will prompt for a value when it tries to connect to the server.

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.

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
```

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.

Header Rules

A `header` rule can match either a substring or 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
```

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"
```

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"
```

Actions

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

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 move 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.

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
```

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
```

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.

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.

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.

API Documentation

The `imapautofiler.actions` Module

The `imapautofiler.app` Module

The `imapautofiler.client` Module

The `imapautofiler.config` Module

The `imapautofiler.rules` Module

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