PyMailq Documentation

Release 0.6.0

Denis 'jawa' Pompilio

Sep 01, 2017

Contents

The *pymailq* package makes it easy to view and control Postfix mails queue. It provide several classes to store, view and interact with mail queue using Postfix command line tools. This module is provided for automation and monitoring.

The *pymailq* package defines the following attribute:

pymailq.**DEBUG = False**
Boolean to control activation of the *debug()* decorator.

pymailq.**VERSION = '0.6.0'**
Current version of the package as `str()`.

The *pymailq* package defines the following decorators:

pymailq.**debug**(*function*)
Decorator to print some call informations and timing debug on stderr.

Function's name, passed args and kwargs are printed to stderr. Elapsed time is also print at the end of call. This decorator is based on the value of *DEBUG*. If `True`, the debug informations will be displayed.

The *pymailq* package provides the following submodules:

CHAPTER 1

pymailq.store – Mails queue storage objects

The `store` module provide several objects to convert mails queue content into python structures.

`PostqueueStore` Objects

class `pymailq.store.`**`PostqueueStore`**

Postfix mails queue informations storage.

The `PostqueueStore` provides methods to load Postfix queued mails informations into Python structures. Thoses structures are based on `Mail` and *`MailHeaders`* classes which can be processed by a *`MailSelector`* instance.

The `PostqueueStore` class defines the following attributes:

`mails`
Loaded *`MailClass`* objects `list()`.

`loaded_at`
`datetime.datetime` instance to store load date and time informations, useful for datas deprecation tracking. Updated on *`load()`* call with `datetime.datetime.now()` method.

`postqueue_cmd`
`list()` object to store Postfix command and arguments to view the mails queue content. Default is `["/usr/sbin/postqueue", "-p"]`.

`spool_path`
Postfix spool path string. Default is `"/var/spool/postfix"`.

`postqueue_mailstatus`
Postfix known queued mail status list. Default is `['active', 'deferred', 'hold']`.

`mail_id_re`
Python compiled regular expression object (`re.RegexObject`) provided by `re.compile()` method to match postfix IDs. Recognized IDs are hexadecimals, may

be 10 to 12 chars length and followed with * or !. Default used regular expression is: r"^[A-F0-9]{10,12}[*!]?$".

mail_addr_re
Python compiled regular expression object (re.RegexObject) provided by re.compile() method to match email addresses. Default used regular expression is: r"^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]+$"

MailClass
The class used to manipulate/parse mails individually. Default is Mail.

See also:

Python modules: datetime – Basic date and time types

re – Regular expression operations

Postfix manual: postqueue – Postfix queue control

RFC 3696 – Checking and Transformation of Names

The PostqueueStore instance provides the following methods:

PostqueueStore.**load**([*method*])
Load content from postfix mails queue.

Mails are loaded using postqueue command line tool or reading directly from spool. The optionnal argument, if present, is a method string and specifies the method used to gather mails informations. By default, method is set to "postqueue" and the standard Postfix queue control tool: postqueue is used.

Parameters
- **method** (*str*) – Method used to load mails from Postfix queue
- **filename** (*str*) – File to load mails from

Provided method str() name is directly used with getattr() to find a *self._load_from_<method>* method.

PostqueueStore.**_load_from_postqueue**()
Load content from postfix queue using postqueue command output.

Output lines from *_get_postqueue_output* are parsed to build Mail objects. Sample Postfix queue control tool (postqueue) output:

```
C0004979687     4769 Tue Apr 29 06:35:05  sender@domain.com
(error message from mx.remote1.org with parenthesis)
                                          first.rcpt@remote1.org
(error message from mx.remote2.org with parenthesis)
                                          second.rcpt@remote2.org
                                          third.rcpt@remote2.org
```

Parsing rules are pretty simple:
- Line starts with a valid *Mail.qid*: create new Mail object with *qid*, *size*, *date* and *sender* informations from line.

Queue ID	Size	Reception date and time				Sender
C0004979687	4769	Tue	Apr	29	06:35:05	user@domain.com

- Line starts with a parenthesis: store error messages to last created Mail object's *errors* attribute.
- Any other matches: add new recipient to the *recipients* attribute of the last created Mail object.

Optionnal argument `filename` can be set with a file containing output of the postqueue command. In this case, output lines of postqueue command are directly read from `filename` and parsed, the postqueue command is never used.

`PostqueueStore.`**`_load_from_spool`**`()`
Load content from postfix queue using files from spool.

Mails are loaded using the command defined in *`postqueue_cmd`* attribute. Some informations may be missing using the *`_load_from_spool()`* method, including at least *`Mail.status`* field.

Loaded mails are stored as `Mail` objects in *`mails`* attribute.

> **Warning:** Be aware that parsing mails on disk is slow and can lead to high load usage on system with large mails queue.

`PostqueueStore.`**`_get_postqueue_output`**`()`
Get Postfix postqueue command output.

This method used the postfix command defined in *`postqueue_cmd`* attribute to view the mails queue content.

Command defined in *`postqueue_cmd`* attribute is runned using a `subprocess.Popen` instance.

Returns Command's output lines.
Return type `list()`
See also:

Python module: `subprocess` – Subprocess management

`PostqueueStore.`**`_is_mail_id`**`(`*`mail_id`*`)`
Check mail_id for a valid postfix queued mail ID.

Validation is made using a `re.RegexObject` stored in the *`mail_id_re`* attribute of the `PostqueueStore` instance.

Parameters **`mail_id`** (*`str`*) – Mail Postfix queue ID string
Returns True or false
Return type `bool()`

`Mail` Objects

class `pymailq.store.`**`Mail`**`(`*`mail_id`*`[, `*`size`*`[, `*`date`*`[, `*`sender`*`]]])`
Simple object to manipulate email messages.

This class provides the necessary methods to load and inspect mails content. This object functionnalities are mainly based on `email` module's provided class and methods. However, `email.message.Message` instance's stored informations are extracted to extend `Mail` instances attributes.

Initialization of `Mail` instances are made the following way:

Parameters

- **`mail_id`** (*`str`*) – Mail's queue ID string
- **`size`** (*`int`*) – Mail size in Bytes (Default: `0`)

- **date** (*datetime.datetime*) – Acceptance date and time in mails queue. (Default: None)
- **sender** (*str*) – Mail sender string as seen in mails queue. (Default: empty str())

The Mail class defines the following attributes:

qid
Mail Postfix queue ID string, validated by *_is_mail_id()* method.

size
Mail size in bytes. Expected type is int().

parsed
bool() value to track if mail's content has been loaded from corresponding spool file.

parse_error
Last encountered parse error message str().

date
datetime object of acceptance date and time in mails queue.

status
Mail's queue status str().

sender
Mail's sender str() as seen in mails queue.

recipients
Recipients list() as seen in mails queue.

errors
Mail deliver errors list() as seen in mails queue.

head
Mail's headers *MailHeaders* structure.

postcat_cmd
Postfix command and arguments list() for mails content parsing. Default command and arguments list is build at initialization with: ["postcat", "-qv", self.qid]

The Mail instance provides the following methods:

Mail.**parse**()
Parse message content.

This method use Postfix mails content parsing command defined in *postcat_cmd* attribute. This command is runned using subprocess.Popen instance.

Parsed headers become attributes and are retrieved with help of message_from_string() function provided by the email module.

See also:

Postfix manual: postcat – Show Postfix queue file contents

Mail.**dump**()
Dump mail's gathered informations to a dict object.

Mails informations are splitted in two parts in dictionnary. `postqueue` key regroups every informations directly gathered from Postfix queue, while `headers` regroups *`MailHeaders`* attributes converted from mail content with the *`parse()`* method.

If mail has not been parsed with the *`parse()`* method, informations under the `headers` key will be empty.

Returns Mail gathered informations
Return type `dict`

`MailHeaders` Objects

class `store.`**`MailHeaders`**

Simple object to store mail headers.

Object's attributes are dynamicly created while parent `Mail` object's method *`parse()`* is called. Those attributes are retrieved with help of `message_from_string()` method provided by the `email` module.

Standard RFC *822-style* mail headers becomes attributes including but not limited to:

- *`Received`*
- *`From`*
- *`To`*
- *`Cc`*
- *`Bcc`*
- *`Sender`*
- *`Reply-To`*
- *`Subject`*

Case is kept while creating attribute and access will be made with `Mail.From` or `Mail.Received` for example. All those attributes will return *list* of values.

See also:

Python modules: `email` – An email and MIME handling package

`email.message.Message` – Representing an email message

RFC 822 – Standard for ARPA Internet Text Messages

CHAPTER 2

pymailq.selector – Mails queue filtering

The `selector` module mainly provide a selector class to interact with structures from the `store` module.

`MailSelector` Objects

class `selector.`**`MailSelector`**(*store*)

Mail selector class to request mails from store matching criterias.

The *`MailSelector`* instance provides the following attributes:

`mails`
Currently selected `Mail` objects `list()`

`store`
Linked `PostqueueStore` at the *`MailSelector`* instance initialization.

`filters`
Applied filters `list()` on current selection. Filters list entries are tuples containing `(function.__name__, args, kwargs)` for each applied filters. This list is filled by the *`filter_registration()`* decorator while calling filtering methods. It is possible to replay registered filter using *`replay_filters()`* method.

The *`MailSelector`* instance provides the following methods:

`MailSelector.`**`filter_registration`**(*function*)

Decorator to register applied filter.

This decorated is used to wrap selection methods `lookup_*`. It registers a `(function.__name__, args, kwargs)` `tuple()` in the *`filters`* attribute.

`MailSelector.`**`reset`**()

Reset mail selector with initial store mails list.

Selected `Mail` objects are deleted and the *`mails`* attribute is removed for memory releasing purpose (with help of `gc.collect()`). Attribute *`mails`* is then reinitialized a copy of *`store`*'s *`mails`* attribute.

Registered *filters* are also emptied.

MailSelector.**replay_filters**()
Reset selection with store content and replay registered filters.

Like with the *reset()* method, selected Mail objects are deleted and reinitialized with a copy of *store*'s *mails* attribute.

However, registered *filters* are kept and replayed on resetted selection. Use this method to refresh your store content while keeping your filters.

MailSelector.**lookup_status**(*status*)
Lookup mails with specified postqueue status.
Parameters **status** (*list*) – List of matching status to filter on.
Returns List of newly selected Mail objects
Return type list()

MailSelector.**lookup_sender**(*sender*[, *partial*])
Lookup mails send from a specific sender.

Optionnal parameter partial allow lookup of partial sender like @domain.com or sender@. By default, partial is False and selection is made on exact sender.

Note: Matches are made against *Mail.sender* attribute instead of real mail header *Sender*.

Parameters
- **sender** (*str*) – Sender address to lookup in Mail objects selection.
- **exact** (*bool*) – Allow lookup with partial or exact match

Returns List of newly selected Mail objects
Return type list()

MailSelector.**lookup_error**(*error_msg*)
Lookup mails with specific error message (message may be partial).
Parameters **error_msg** (*str*) – Error message to filter on
Returns List of newly selected Mail objects'
Return type list()

MailSelector.**lookup_date**([*start*[, *stop*]])
Lookup mails send on specific date range(s).
Parameters
- **start** (*datetime.date*) – Start date (Default: None)
- **stop** (*datetime.date*) – Stop date (Default: None)

Returns List of newly selected Mail objects
Return type list()

MailSelector.**lookup_size**([*smin*[, *smax*]])
Lookup mails send with specific size.

Both arguments smin and smax are optionnal and default is set to 0. Maximum size is ignored if setted to 0. If both smin and smax are setted to 0, no filtering is done and the entire Mail objects selection is returned.
Parameters
- **smin** (*int*) – Minimum size (Default: 0)
- **smax** (*int*) – Maximum size (Default: 0)

Returns List of newly selected Mail objects
Return type list()

CHAPTER 3

pymailq.control – Mails queue adminitrative operations

The `control` module define a basic python class to simplify administrative operations against the mails queue. This module is mainly based on the postsuper administrative tool functionnalities.

`QueueControl` Objects

class `control.`**`QueueControl`**
Postfix queue control using postsuper command.

The *`QueueControl`* instance defines the following attributes:

`use_sudo`
Boolean to control the use of *sudo* to invoke Postfix command. Default is `False`

`postsuper_cmd`
Postfix command and arguments `list()` for mails queue administrative operations. Default is `["postsuper"]`

`known_operations`
Known Postfix administrative operations `dict` to associate operations to command arguments. Known associations are:

```
 delete: -d
   hold: -h
release: -H
requeue: -r
```

> **Warning:** Default known associations are provided for the default mails queue administrative command postsuper.

See also:

Postfix manual: postsuper – Postfix superintendent

The *QueueControl* instance provides the following methods:

QueueControl.**_operate**(*operation*, *messages*)
Generic method to lead operations messages from postfix mail queue.

Operations can be one of Postfix known operations stored in *known_operations* attribute. Operation argument is directly converted and passed to the *postsuper_cmd* command.

Parameters
- **operation** (*str*) – Known operation from *known_operations*.
- **messages** (*list*) – List of Mail objects targetted for operation.

Returns Command's *stderr* output lines

Return type list()

QueueControl.**delete_messages**(*messages*)
Delete several messages from postfix mail queue.

This method is a partial() wrapper on *_operate()*. Passed operation is delete

QueueControl.**hold_messages**(*messages*)
Hold several messages from postfix mail queue.

This method is a partial() wrapper on *_operate()*. Passed operation is hold

QueueControl.**release_messages**(*messages*)
Release several messages from postfix mail queue.

This method is a partial() wrapper on *_operate()*. Passed operation is release

QueueControl.**requeue_messages**(*messages*)
Requeue several messages from postfix mail queue.

This method is a partial() wrapper on *_operate()*. Passed operation is requeue

CHAPTER 4

pymailq.shell – Mails queue management shell

The *pymailq.shell* module provide a shell to view and control Postfix mails queue. More documentation will soon be available.

CHAPTER 5

pqshell – A shell-like to interact with a Postfix mails queue

DESCRIPTION

pqshell is a shell-like to interact with Postfix mails queue. It provide simple means to view the queue content, filter mails on criterias like *Sender* or *delivery errors* and lead administrative operations.

SYNOPSIS

pqshell [options]

FEATURES

- Asynchronous interactions with Postfix mails queue.
- Mails filtering on various criterias.
- Administrative operations on mails queue
- History and autocomplete via readline, if installed.

OPTIONS

No options supported.

SHELL COMMANDS

An inside help is available with the help command. Each provided command takes subcommands and command's help can be obtained while running it without argument.

store

Control of Postfix queue content storage

Subcommands:

status Show store status.

load Load Postfix queue content.

Example:

```
PyMailq (sel:0)> store status
store is not loaded
PyMailq (sel:0)> store load
590 mails loaded from queue
PyMailq (sel:590)> store status
store loaded with 590 mails at 2014-05-05 13:43:22.592767
```

select

Select mails from Postfix queue content. Filters are cumulatives and designed to simply implement advanced filtering with simple syntax. The default prompt will show how many mails are currently selected by all applied filters. Order of filters application is also important.

Subcommands:

date Select mails by date.

Usage: `select date <DATESPEC>`

Where <*DATESPEC*> can be:

```
YYYY-MM-DD (exact date)
YYYY-MM-DD..YYYY-MM-DD (within a date range (included))
+YYYY-MM-DD (after a date (included))
-YYYY-MM-DD (before a date (included))
```

error Select mails by error message. Specified error message can be partial and will be check against the whole error message.

Usage: `select error <error_msg>`

replay Reset content of selector with store content and replay filters.

reset Reset content of selector with store content, remove filters.

rmfilter Remove filter previously applied. Filters ids are used to specify filter to remove.

Usage: `select rmfilter <filterid>`

sender Select mails from sender.

Usage: `select sender <sender> [exact]`

size Select mails by size in Bytes. Signs - and + are supported, if not specified, search for exact size. Size range is allowed by using − (lesser than) and + (greater than).

Usage: `select size <-n|n|+n> [-n]`

status Select mails with specific postfix status.

Usage: `select status <status>`

Filtering Example:

```
PyMailq (sel:608)> select size -5000
PyMailq (sel:437)> select sender MAILER-DAEMON
PyMailq (sel:316)> select status active
PyMailq (sel:0)>
```

Filters management:

```
PyMailq (sel:608)> select size -5000
PyMailq (sel:437)> select sender MAILER-DAEMON
PyMailq (sel:316)> show filters
0: select size:
    smax: 5000
    smin: 0
1: select sender:
    partial: True
    sender: MAILER-DAEMON
PyMailq (sel:316)> select rmfilter 1
PyMailq (sel:437)> select sender greedy-sender@domain.com
PyMailq (sel:25)> select reset
Selector resetted with store content (608 mails)
PyMailq (sel:608)>
```

show

Display the content of current mails selection or specific mail IDs. Modifiers have been implemented to allow quick output manipulation. These allow you to sort, limit or even output a ranking by specific field. By default, output is sorted by **date of acceptance** in queue.

Optionnal modifiers can be provided to alter output:

`limit <n>` Display the first n entries.

`sortby <field> [asc|desc]` Sort output by field asc or desc. Default sorting is made descending.

`rankby <field>` Produce mails ranking by field.

Known fields:

- `qid` – Postqueue mail ID.
- `date` – Mail date.
- `sender` – Mail sender.
- `recipients` – Mail recipients (list, no sort).
- `size` – Mail size.
- `errors` – Postqueue deferred error messages (list, no sort).

Subcommands:

filters Show filters applied on current mails selection.

Usage: `show filters`

selected Show selected mails.

Usage: `show selected [modifiers]`

Example:

```
PyMailq (sel:608)> show selected limit 5
2014-05-05 20:54:24 699C11831669 [active] jjj@dom1.com (14375B)
2014-05-05 20:43:39 8D60C13C14C6 [deferred] bbb@dom9.com (39549B)
2014-05-05 20:35:08 B0077198BC31 [deferred] rrr@dom2.com (4809B)
2014-05-05 20:30:09 014E21AB4B78 [deferred] aaa@dom7.com (2450B)
2014-05-05 20:25:04 CF1BE127A8D3 [deferred] xxx@dom2.com (4778B)
...Preview of first 5 (603 more)...
PyMailq (sel:608)> show selected sortby sender limit 5 asc
2014-05-02 11:36:16 40AA9149A9D7 [deferred] aaa@dom1.com (8262B)
2014-05-01 05:30:23 5E0B2162BE63 [deferred] bbb@dom4.com (3052B)
2014-05-02 05:30:20 653471AC5F76 [deferred] ccc@dom5.com (3052B)
2014-05-02 09:49:01 A00D3159AEE [deferred] ddd@dom1.com (3837B)
2014-05-05 18:18:59 98E9A790749 [deferred] ddd@dom2.com (1551B)
...Preview of first 5 (603 more)...
PyMailq (sel:608)> show selected rankby sender limit 5
sender                                      count
==============================================
jjj@dom8.com                                334
xxx@dom4.com                                43
nnn@dom1.com                                32
ccc@dom3.com                                14
sss@dom5.com                                13
...Preview of first 5 (64 more)...
```

Python Module Index

p

Index

Symbols

D

E

F

H

K

L

M

P

Q

R

S

U

V