Sieve Tutorial

Index

The Sieve mail filtering language (RFC 5228) is a simple language that is optimised to allow efficient server-side filtering of mails. This tutorial introduces some of the most commonly used features in Sieve scripts.

The format of a Sieve filter

A Sieve filter file has no complicated structure. It contains a list of commands, such as "`discard`", "`if`", "`fileinto`" etc. An example script is:

```
require ["fileinto", "reject"];

# Daffy Duck is a good friend of mine.
if address :is "from" "daffy.duck@example.com"
{
    fileinto "friends";
}

# Reject mails from the hunting enthusiasts at example.com.
if header :contains "list-id" "<duck-hunting.example.com>"
{
    reject "No violence, please";
}

# The command "keep" is executed automatically, if no other action is taken.
```

The first line of the script (the `require` command) tells the Sieve interpreter that the optional command `fileinto` will be used. Then two filter rules follow. The first filter rule stores all mails from "`daffy.duck@example.com`" into the mailbox named "`friends`". The second rule rejects mails with a List-Id field in the header containing the string "`<duck-hunting.example.com>`".

If no condition in a script matches then the default action is applied which is an implicit "`keep`" command. That command stores the mail in the default mailbox, usually `INBOX`.

What happens if one gets a mail from `daffy.duck@example.com` that also contains the List-Id `<duck-hunting.example.com>`? In that case both commands, "`fileinto`" and "`discard`" are executed. Since a "`discard`" only cancels the implicit "`keep`" command, the net result of the script is to "`fileinto`" the mail to the folder "`friends`". It is good practise to avoid ambiguities like this by using "`elsif`" blocks.

Comments can appear at any point of a Sieve filter file. There are two types of comments, a one-line comment and a C-style comment:

```
# The hash character starts a one-line comment.
# Everything after a # character until the end of line is ignored.

/* this is a bracketed (C-style) comment. This type of comment can stretch
 * over many lines. A bracketed comment begins with a forward slash, followed
 * by an asterisk and ends with the inverse sequence: an asterisk followed
 * by a forward slash. */
```

Comparison against addresses

The "`address`" test selects the address fields, such as the `From:`, `To:` or `Sender:` fields. The advantage over a "`header`" test is that we can test for the local part or the domain of an address.

Three optional arguments can be used to check against email-addresses: "`:localpart`", "`:domain`", and "`:all`". The local part of an address is the string before the @ sign, the domain is the string after the @ sign. The argument "`:all`" compares the whole address.

```
# The two test below are equivalent;
# The first variant is clearer and probably also more efficient.
if address :is :domain "to" "example.com"
{
    fileinto "examplecom";
}

```

```
if address :matches :all "to" "*@example.com"
{
    fileinto "examplecom";
}
```

Often, addresses are in the form `"Firstname Lastname" <localpart@domain>`. The "`address`" test conveniently matches only the email address itself, not a person's name and not the < and > characters.

Comparing non-address header fields

Any header field can be used with the "`header`" test. This can be useful to filter messages from mailing lists in separate folders.

```
# File mails with a Spamassassin score of 4.0 or more
# into the "junk" folder.
if header :contains "x-spam-level" "****"
{
    fileinto "junk";
}
```

Match Type

Sieve provides three ways to compare strings. These types are called match types are: "`:is`", "`:contains`" and "`:matches`".

The "`:is`" type compares two whole strings with each other. If they are the same from the first to the last character then the test evaluates to true. The "`:contains`" is a sub-string match, useful if we know a part of a string, but not the whole one. The third type, "`:matches`", is used to, well, match strings. We can use the wildcard "`?`" for one unknown character and the asterisk "`*`" for zero or more unknown characters.

```
# Reject all messages that contain the string "viagra"in the Subject.
if header :contains "subject" "viagra"
{
    reject "go away!";
}
# Silently discard all messages sent from the tax man
elsif address :matches :domain "from" "*hmrc.gov.uk"
{
    discard;
}
```

Lists of Strings

Often we want to match an address filed (such as "to") with many addresses. We use lists of strings for this. A list of strings is a sequence of strings, separated by commas and enclosed in square brackets. For example:

```
# A mail to any of the recipients in the list of strings is filed to the folder
"friends".
if address :is "from" ["daffy.duck@example.com", "porky.pig@example.com",
"speedy.gonzales@example.com"]
{
    fileinto "friends";
}
```

This works also the other way round. We can check if either the "from" header or the "sender" header is a particular address:

```
# Check if either the "from" or the "sender" header is from Porky.
if address :is ["from", "sender"] "porky.pig@example.com"
{
    fileinto "friends";
}
```

Now the question is: can we combine the examples above and have two lists compared with each other? The answer is yes we can. In this case the test is true if any field from the former list matches a string from the latter. We can write for example:

```
# Match "from" or the "sender" file with any of Daffy, Porky or Speedy.
if address :is ["from", "sender"] ["daffy.duck@example.com",
"porky.pig@example.com", "speedy.gonzales@example.com"]
{
    fileinto "friends";
}
```

We have already met a list of strings, in the `require ["fileinto", "reject"];` command.

Test Lists (allof, anyof)

Sometimes we want to combine more tests into one expression. The commands "`allof`" and "`anyof`" allow to combine more tests. As their name suggests, the "`allof`" test list returns true if every single test in parenthesis are true (it is a logical "and"), while the "`anyof`" test list returns true if at least one of the single tests in parenthesis is true (it is a logical "or").

The syntax is "`allof (test 1, test2, ...)`" and "`anyof (test 1, test2, ...)`".

```
# This test checks against Spamassassin's header fields:
# If the spam level ls 4 or more and the Subject contains too
# many illegal characters, then silently discard the mail.
if allof (header :contains "X-Spam-Level" "****",
          header :contains "X-Spam-Report" "FROM_ILLEGAL_CHARS")
{
    discard;
}
# Discard mails that do not have a Date: or From: header field
# or mails that are sent from the marketing department at example.com.
elsif anyof (not exists ["from", "date"],
        header :contains "from" "marketing@example.com") {
    discard;
}
```

Filter on message size

Mails can be tested for their size with the "`size`" test. The tagged arguments "`:over`" and "`:under`" are used to specify if the following number is the upper limit or the lower limit of the test.

```
# Delete messages greater than half a MB
if size :over 500K
{
    discard;
}
# Also delete small mails, under 1k
if size :under 1k
{
    discard;
}
```

Last modified: 05 Dec 2016