NAME

notmuch-sexp-queries − s-expression syntax for notmuch queries

SYNOPSIS

notmuch *subcommand* **−−query=sexp** [option ...] **−−** '(and (to santa) (date december))'

DESCRIPTION

Notmuch supports an alternative query syntax based on *S−expressions* . It can be selected with the command line **−−query=sexp** or with the appropriate option to the library function **notmuch_query_create_with_syntax()**. Support for this syntax is currently optional, you can test if your build of notmuch supports it with

```
$ notmuch config get built_with.sexp_queries
```

S−EXPRESSIONS

An *s−expression* is either an atom, or list of whitespace delimited s−expressions inside parentheses. Atoms are either

basic value

A basic value is an unquoted string containing no whitespace, double quotes, or parentheses.

quoted string

Double quotes (") delimit strings possibly containing whitespace or parentheses. These can contain double quote characters by escaping with backslash. E.g. **"this is a quote \""**.

S−EXPRESSION QUERIES

An s−expression query is either an atom, the empty list, or a *compound query* consisting of a prefix atom (first element) defining a *field*, *logical operation*, or *modifier*, and 0 or more subqueries.

*****

"*" matches any non−empty string in the current field.

()

The empty list matches all messages

term

Match all messages containing *term*, possibly after stemming or phrase splitting. For discussion of stemming in notmuch see **notmuch−search−terms(7)**. Stemming only applies to unquoted terms (basic values) in s−expression queries. For information on phrase splitting see *FIELDS*.

(*field* q_1 q_2 ... q_n **)**

Restrict the queries q_1 to q_n to *field*, and combine with *and* (for most fields) or *or*. See *FIELDS* for more information.

(*operator* q_1 q_2 ... q_n **)**

Combine queries q_1 to q_n. Currently supported operators are **and**, **or**, and **not**. **(not** q_1 ... q_n **)** is equivalent to **(and (not** q_1 **) ... (not** q_n **))**.

(*modifier* q_1 q_2 ... q_n **)**

Combine queries q_1 to q_n, and reinterpret the result (e.g. as a regular expression). See*MODIFIERS* for more information.

(macro (p_1 ... p_n **) body)**

Define saved query with parameter substitution. The syntax is recognized only in saved s−expression queries (see **squery.*** in **notmuch−config(1)**). Parameter names in **body** must be prefixed with **,** to be expanded (see *MACRO EXAMPLES*). Macros may refer to other macros, but only to their own parameters [1].

FIELDS

Fields [2] correspond to attributes of mail messages. Some are inherent (and immutable) like **subject**, while others **tag** and **property** are settable by the user. Each concrete field in*the table below* is discussed further under "Search prefixes" in **notmuch−search−terms(7)**. The row *user* refers to user defined fields, described in **notmuch−config(1)**.

Most fields are either *phrase fields* [3] (which match sequences of words), or *term fields* [4] (which match exact strings). *Phrase splitting* breaks the term (basic value or quoted string) into words, ignore punctuation. Phrase splitting is applied to terms in phrase (probabilistic) fields. Both phrase splitting and stemming apply only in phrase fields.

Each term or phrase field has an associated combining operator (**and** or **or**) used to combine the queries from each element of the tail of the list. This is generally **or** for those fields where a message has one such attribute, and **and** otherwise.

Term or phrase fields can contain arbitrarily complex queries made up from terms, operators, and modifiers, but not other fields.

Fields with supported modifiers

field	combine	type	expand	wildcard	regex
none	and		no	yes	no
user	and	phrase	no	yes	no
attachment	and	phrase	yes	yes	no
body	and	phrase	no	no	no
date		range	no	no	no
folder	or	phrase	yes	yes	yes
from	and	phrase	yes	yes	yes
id	or	term	no	yes	yes
is	and	term	yes	yes	yes
lastmod		range	no	no	no
mid	or	term	no	yes	yes
mimetype	or	phrase	yes	yes	no
path	or	term	no	yes	yes
property	and	term	yes	yes	yes
subject	and	phrase	yes	yes	yes
tag	and	term	yes	yes	yes
thread	or	term	yes	yes	yes
to	and	phrase	yes	yes	no

MODIFIERS

Modifiers refer to any prefixes (first elements of compound queries) that are neither operators nor fields.

(infix *atom* **)**

Interpret *atom* as an infix notmuch query (see **notmuch−search−terms(7)**). Not supported inside fields.

(matching q_1 q_2 ... q_n **) (of** q_1 q_2 ... q_n **)**

Match all messages have the same values of the current field as those matching all of q_1 ... q_n. Supported in most term [7] or phrase fields. Most commonly used in the **thread** field.

(query *atom* **)**

Expand to the saved query named by *atom*. See **notmuch−config(1)** for more. Note that the saved

query must be in infix syntax (**notmuch−search−terms(7)**). Not supported inside fields.

(regex *atom* **) (rx** *atom* **)**
Interpret *atom* as a POSIX.2 regular expression (see **regex(7)**). This applies in term fields and a subset [5] of phrase fields (see *Fields with supported modifiers*).

(starts−with *subword* **)**
Matches any term starting with *subword*. This applies in either phrase or term*fields*, or outside of fields [6]. Note that a **starts−with** query cannot be part of a phrase. The atom ***** is a synonym for **(starts−with "")**.

EXAMPLES

Wizard
Match all messages containing the word "wizard", ignoring case.

added
Match all messages containing "added", but also those containing "add", "additional", "Additional", "adds", etc... via stemming.

(and Bob Marley)
Match messages containing words "Bob" and "Marley", or their stems The words need not be adjacent.

(not Bob Marley)
Match messages containing neither "Bob" nor "Marley", nor their stems,

"quick fox" quick−fox quick@fox
Match the *phrase* "quick" followed by "fox" in phrase fields (or outside a field). Match the literal string in a term field.

(folder (of (id 1234@invalid)))
Match any message in the same folder as the one with Message−Id "*1234@invalid*"

(id 1234@invalid blah@test)
Matches Message−Id "*1234@invalid*" *or* Message−Id "*blah@test*"

(and (infix "date:2009−11−18..2009−11−18") (tag unread))
Match messages in the given date range with tag unread.

(starts−with prelim)
Match any words starting with "prelim".

(subject quick "brown fox")
Match messages whose subject contains "quick" (anywhere, stemmed) and the phrase "brown fox".

(subject (starts−with prelim))
Matches any word starting with "prelim", inside a message subject.

(subject (starts−wih quick) "brown fox")
Match messages whose subject contains "quick brown fox", but also "brown fox quicksand".

(thread (of (id 1234@invalid)))
Match any message in the same thread as the one with Message−Id "*1234@invalid*"

(thread (matching (from bob@example.com) (to bob@example.com)))
Match any (messages in) a thread containing a message from "*bob@example.com*" and a (possibly

distinct) message to "bob at example.com")

(to (or bob@example.com mallory@example.org)) (or (to bob@example.com) (to mallory@example.org))
Match in the "To" or "Cc" headers, "*bob@example.com*", "*mallory@example.org*", and also "*-bob@example.com.au*" since it contains the adjacent triple "bob", "example", "com".

(not (to *))
Match messages with an empty or invalid 'To' and 'Cc' field.

(List *)
Match messages with a non−empty List−Id header, assuming configuration **index.header.List=List−Id**

MACRO EXAMPLES

A macro that takes two parameters and applies different fields to them.

```
$ notmuch config set squery.TagSubject '(macro (tagname subj) (and (tag ,tagnam
$ notmuch search −−query=sexp '(TagSubject inbox maildir)'
```

Nested macros are allowed.

```
$ notmuch config set squery.Inner '(macro (x) (subject ,x))'
$ notmuch config set squery.Outer  '(macro (x y) (and (tag ,x) (Inner ,y)))'
$ notmuch search −−query=sexp '(Outer inbox maildir)'
```

Parameters can be re−used to reduce boilerplate. Any field, including user defined fields is permitted within a macro.

```
$ notmuch config set squery.About '(macro (name) (or (subject ,name) (List ,nam
$ notmuch search −−query=sexp '(About notmuch)'
```

NOTES

[1] Technically macros impliment lazy evaluation and lexical scope. There is one top level scope containing all macro definitions, but all parameter definitions are local to a given macro.

[2] a.k.a. prefixes

[3] a.k.a. probabilistic prefixes

[4] a.k.a. boolean prefixes

[5] Due to the implemention of phrase fields in Xapian, regex queries could only match individual words.

[6] Due the the way **body** is implemented in notmuch, this modifier is not supported in the **body** field.

[7] Due to the way recursive **path** queries are implemented in notmuch, this modifier is not supported in the **path** field.

AUTHOR

Carl Worth and many others

COPYRIGHT