The mairix program

This manual describes how to use
the mairix program for indexing and
searching email messages stored in maildir folders.

Richard P. Curnow

Table of Contents

1 Introduction

1.1 Background

The *mairix* program arose from a need to index and search 100's or 1000's of email messages in an efficient way. It began life supporting just Maildir format folder, but now MH and mbox formats are also supported.

I use the *mutt* email client. *mutt* has a feature called *limit*, where the display of messages in the current folder can be filtered based on matching regular expressions in particular parts of the messages. I find this really useful. But there is a snag - it only works on the current folder. If you have messages spread across many folders, you're out of luck with limit. OK - so why not keep all messages in a single folder? The problem is that the performance drops badly. This is true regardless of folder format - mbox, maildir etc, though probably worse for some formats than others depending on the sizes of messages in the folders.

So on the one hand, we want small folders to keep the performance high. But on the other hand, we want useful searching.

I use the maildir format for my incoming folders. This scheme has one file per message. On my inboxes[1], I like this for 2 reasons :

- Fast deletion of messages I don't want to keep (spam, circulars, mailing list threads I'm not interested in etc). (Compare mbox, where the whole file would need to be rewritten.)
- No locking issues whatever. Maybe I'm over cautious, but I don't really trust all that locking stuff to protect a single mbox file in all cases, and a single file seems just too vulnerable to corruption.) Also, I sometimes read the mail over NFS mounted filesystems, where locking tends to be a real disaster area.

Since I'm using maildir for inboxes, I've traditionally used it for all my folders, for uniformity.

So, I hear you ask, if you use a one-file-per-message format, why not just use find + egrep to search for messages? I saw the following problems with this:

- What if I want to find all messages to/cc me, from Homer Simpson, dated between 1 and 2 months ago, with the word "wubble" in the body? This would involve a pretty nasty set of regexps in a pipeline of separate egreps (and bear in mind, headers could be split over line boundaries...)
- What if the message body has quoted-printable (or worse, base64) transfer encoding? The egrep for "wubble" could come very unstuck.
- How would the matching messages be conveniently arranged into a new folder to allow browsing with mutt?
- What if I wanted to see all messages in the same threads as those matching the above condition?
- If I had 1000's of messages, this wasn't going to be quick, especially if I wanted to keep tuning the search condition.[2].

So find + egrep was a non-starter. I looked around for other technology. I found *grepmail*, but this only works for mbox format folders, and involved scanning each message every time (so lost on the speed issue).

I decided that this was going to be my next project, and mairix was born. By the way, the name originally came from abbreviating *MAildIR IndeX*, but this is now an anachronism since MH and mbox are supported too.

---

[1] of which I have many, because I (naturally) use *procmail* to split my incoming mail

[2] This may be a non-issue for people with the lastest technology under their desk, but at the time I started writing mairix, I had a 1996 model 486 at home

2 Installation

There is not much to this. In the simplest case you can just do

```
./configure
make
make install
```

You need to be root to run the final step unless you're installing under your own home directory somewhere.

However, you might want to tune the options further. The '`configure`' script shares its common options with the usual autoconf-generated scripts, even though it's not autoconf-generated itself. For example, a fuller build could use

```
CC=gcc CFLAGS="-O2 -Wall" ./configure \
    --prefix=/opt/mairix \
    --infodir=/usr/share/info
make
make install
make docs
make install_docs
```

The final step is to create a '`~/.mairixrc`' file. An example is included in the file '`dotmairixrc.eg`'. Just copy that to '`~/.mairixrc`' and edit it.

3 Use

3.1 Overview of use

mairix has two modes of use : index building and searching. The searching mode runs whenever the command line contains any expressions to search for. Otherwise, the indexing mode is run.

To begin with, an indexing run must be performed before searching will work at all. Otherwise your search will be operating on an empty database and won't produce any output.

The output of the search mode is usually placed in a *match folder*. You can select the type of folder that is used. For Maildir, it is just a normal maildir directory (i.e. containing '`new`', '`tmp`' and '`cur`') subdirectories. If you select MH it is a directory containing entries with numerical filenames, so you can open it as a normal MH folder in your mail program. If you select mbox, it is a single file in mbox format.

You configure the path for the match folder in your '`~/.mairixrc`' file. When writing to a mfolder in maildir or MH format, mairix will populate it with symbolic links pointing to the paths of the real messages that were matched by the search expression.[1] If a message in a mbox folder matches, mairix will copy the message contents to a single file in the mfolder directory.

If the mfolder is in mbox format, mairix will copy the message contents of each matching message into the mfolder file. (There is no way of exploiting symlinks to avoid the copying in this case.)

If desired, mairix can produce just a list of files that match the search expression and omit the building of the match folder (the so-called 'raw' output mode). This mode of operation may be useful in communicating the results of the search to other programs.

3.2 Indexing strategy and search capabilities

mairix works exclusively in terms of *words*. The index that's built in non-search mode contains a table of which words occur in which messages. Hence, the search capability is based on finding messages that contain particular words. *mairix* defines a word as any string of alphanumeric characters + underscore. Any whitespace, punctuation, hyphens etc are treated as word boundaries.

mairix has special handling for the `To:`, `Cc:` and `From:` headers. Besides the normal word scan, these headers are scanned a second time, where the characters '`@`', '`-`' and '`.`' are also treated as word characters. This allows most (if not all) email addresses to appear in the database as single words. So if you have a mail from `wibble@foobar.zzz`, it will match on both these searches

```
mairix f:foobar
mairix f:wibble@foobar.zzz
```

It should be clear by now that the searching cannot be used to find messages matching general regular expressions. Personally, I don't find that much use anyway for locating old messages - I'm far more likely to remember particular keywords that were in the messages, or details of the recipients, or the approximate date.

It's also worth pointing out that there is no 'locality' information stored, so you can't search for messages that have one words 'close' to some other word. For every message and every word, there is a simple yes/no condition stored - whether the message contains the word in a particular header or in the body. So far this has proved to be adequate. mairix has a similar feel to using an Internet search engine.

There are three further searching criteria that are supported (besides word searching):

- Searching for messages whose `Date:` header is in a particular range
- Searching for messages whose size is in a particular range. (I see this being used mainly for finding 'huge' messages, as you're most likely to want to cull these to recover disc space.)

[1] Although symlinks use up more inodes than hard links, I decided they were more useful because it makes it possible to see the filenames of the original messages via `ls -l`.

- Searching for messages with a particular substring in their paths. You can use this feature to limit the search to particular folders in your mail hierarchy, for example.

3.3 The '~/.mairixrc' file

3.3.1 Overview

This file contains information about where you keep your mail folders, where you want the index file to be stored and where you want the match folder to be, into which the search mode places the symlinks.

mairix searches for this file at '`~/.mairixrc`' unless you specify the '`-f`' command line option.

If a # character appears in the file, the rest of that line is ignored. This allows you to specify comments.

There are 3 entries ('`base`', '`mfolder`' and '`database`') that must appear in the file. Also at least one of '`maildir`', '`mh`' and '`mbox`' must appear. Optionally, the '`mformat`' entry may appear. An example illustrates:

```
base=/home/richard/mail
maildir=new-mail:new-chrony
maildir=recent...:ancient...
mh=an_mh_folder
mbox=archive1:archive2
mfolder=mfolder
mformat=maildir
database=/home/richard/.mairix_database
```

3.3.2 mairixrc file keys

The keys are as follows:

base — This is the path to the common parent directory of all your maildir folders.

maildir — This is a colon-separated list of the Maildir folders (relative to '`base`') that you want indexed. Any entry that ends '`...`' is recursively scanned to find any Maildir folders underneath it.

More than one line starting with '`maildir`' can be included. In this case, mairix joins the lines together with colons as though a single list of folders had been given on a single very long line.

Each colon-separated entry may be a wildcard. See the discussion under mbox (below) for the wildcard syntax. For example

```
maildir=zzz/foo*...
```

will match maildir folders like these (relative to the folder_base)

```
zzz/foobar/xyz
zzz/fooquux
zzz/foo
zzz/fooabc/u/v/w
```

and

```
maildir=zzz/foo[abc]*
```

will match maildir folders like these (relative to the folder_base)

```
zzz/fooa
zzz/fooaaaxyz
zzz/foobcd
zzz/foocccccccc
```

If a folder name contains a colon, you can write this by using the sequence '`\:`' to escape the colon. Otherwise, the backslash character is treated normally. (If the folder name actually contains the sequence '`\:`', you're out of luck.)

mh
: This is a colon-separated list of the MH folders (relative to '`base`') that you want indexed. Any entry that ends '`...`' is recursively scanned to find any MH folders underneath it.

  More than one line starting with '`mh`' can be included. In this case, mairix joins the lines together with colons as though a single list of folders had been given on a single very long line.

  Each colon-separated entry may be a wildcard, see the discussion under maildir (above) and mbox (below) for the syntax and semantics of specifying wildcards.

mbox
: This is a colon-separated list of the mbox folders (relative to '`base`') that you want indexed.

  Each colon-separated item in the list can be suffixed by '`...`'. If the item matches a regular file, that file is treated as a mbox folder and the '`...`' suffix is ignored. If the item matches a directory, a recursive scan of everything inside that directory is made, and all regular files are initially considered as mbox folders. (Any directories found in this scan are themselves scanned, since the scan is recursive.)

  Each colon-separated item may contain wildcard operators, but only in its final path component. The wildcard operators currently supported are

  *
  : Match zero or more characters (each character matched is arbitrary)

  ?
  : Match exactly one arbitrary character

  [abcs-z]
  : Character class : match a single character from the set a, b, c, s, t, u, v, w, x, y and z.

    To include a literal '`]`' in the class, place it immediately after the opening '`[`'. To include a literal '`-`' in the class, place it immediately before the closing '`]`'.

  If these metacharacters are included in non-final path components, they have no special meaning.

  Here are some examples

  mbox=foo/bar*
  : matches '`foo/bar`', '`foo/bar1`', '`foo/barrrr`' etc

  mbox=foo*/bar*
  : matches '`foo*/bar`', '`foo*/bar1`', '`foo*/barrrr`' etc

  mbox=foo/*
  : matches '`foo/bar`', '`foo/bar1`', '`foo/barrrr`', '`foo/foo`', '`foo/x`' etc

  mbox=foo...
  : matches any regular file in the tree rooted at '`foo`'

  mbox=foo/*...
  : same as before

  mbox=foo/[a-z]*...
  : matches '`foo/a`', '`foo/aardvark/xxx`', '`foo/zzz/foobar`', '`foo/w/x/y/zzz`', but **not** '`foo/A/foobar`'

  Regular files that are mbox folder candidates are examined internally. Only files containing standard mbox '`From `' separator lines will be scanned for messages.

More than one line starting with '`mbox`' can be included. In this case, mairix joins the lines together with colons as though a single list of folders had been given on a single very long line.

mairix performs **no** locking of mbox folders when it is accessing them. If a mail delivery program is modifying the mbox at the same time, it is likely that one or messages in the mbox will never get indexed by mairix (until the database is removed and recreated from scratch, anyway.) The assumption is that mairix will be used to index archive folders rather than incoming ones, so this is unlikely to be much of a problem in reality.

mairix can support a maximum of 65536 separate mboxes, and a maximum of 65536 messages within any one mbox.

omit This is a colon-separated list of glob patterns for folders to be omitted from the indexing. This allows wide wildcards to be used in the *maildir*, *mh* and *mbox* arguments, with the *omit* option used to selectively remove unwanted folders from the folder lists. Within the glob patterns, a single '`*`' matches any sequence of characters other than '`/`'. However '`**`' matches any sequence of characters including '`/`'. This allows glob patterns to be constructed which have a wildcard for just one directory component, or for any number of directory components.

The *omit* option can be specified as many times as required so that the list of patterns doesn't all have to fit on one line.

As an example,

```
mbox=bulk...
omit=bulk/spam*
```

will index all mbox folders at any level under the '`bulk`' subdirectory of the base folder, except for those folders whose names start '`bulk/spam`', e.g. '`bulk/spam`', '`bulk/spam2005`' etc. In constrast,

```
mbox=bulk...
omit=bulk/spam**
```

will index all mbox folders at any level under the '`bulk`' subdirectory of the base folder, except for those folders whose names start '`bulk/spam`', e.g. '`bulk/spam`', '`bulk/spam2005`', '`bulk/spam/2005`', '`bulk/spam/2005/jan`' etc.

nochecks This takes no arguments. If a line starting with '`nochecks`' is present, it is the equivalent of specifying the '`-Q`' flag to every indexing run.

mfolder This defines the name of the *match* folder (within the directory specified by '`base`') into which the search mode writes its output. (If the mformat used is '`raw`', then this setting is not used and may be excluded.)

If the first character of the **mfolder** value is '`/`' or '`.`', it is taken as a pathname in its own right. This allows you to specify absolute paths and paths relative to the current directory where the mfolder should be written. Otherwise, the value of **mfolder** is appended to the value of **base**, in the same way as for the source folders.

mformat This defines the type of folder used for the *match folder* where the search results go. There are four valid settings for this '`mh`', '`maildir`', '`mbox`' or '`raw`'. If the '`raw`' setting is used then mairix will just print out the path names of the files that match and no match folder will be created. '`maildir`' is the default if this option is not defined. The setting is case-insensitive.

database This defines the path where mairix's index database is kept. You can keep this file anywhere you like.

It is illegal to have a folder listed twice. Once mairix has built a list of all the messages currently in your folders, it will search for duplicates before proceeding. If any duplicates are found (arising

from the same folder being specified twice), it will give an error message and exit. This is to prevent corrupting the index database file.

3.3.3 mairixrc expansions

The part of each line in '`.mairixrc`' following the equals sign can contain the following types of expansion:

Home directory expansion
: If the sequence '`~/`' appears at the start of the text after the equals sign, it is expanded to the user's home directory. Example:

```
database=~/Mail/mairix_database
```

Environment expansion
: If a '`$`' is followed by a sequence of alpha-numeric characters (or '`_`'), the whole string is replaced by looking up the corresponding environment variable. Similarly, if '`$`' is followed by an open brace ('`{`'), everything up to the next close brace is looked up as an environment variable and the result replaces the entire sequence.

  Suppose in the shell we do

```
export FOO=bar
```

  and the '`.mairixrc`' file contains

```
maildir=xxx/$FOO
mbox=yyy/a${FOO}b
```

  this is equivalent to

```
maildir=xxx/bar
mbox=yyy/abarb
```

  If the specified environment variable is not set, the replacement is the empty string.

3.4 Setting up the match folder

If the match folder does not exist when running in search mode, it is automatically created. For '`mformat=maildir`' (the default), this should be all you need to do. If you use '`mformat=mh`', you may have to run some commands before your mailer will recognize the folder. e.g. for mutt, you could do

```
mkdir -p /home/richard/Mail/mfolder
touch /home/richard/Mail/mfolder/.mh_sequences
```

which seems to work. Alternatively, within mutt, you could set *mbox_type* to '`mh`' and save a message to '`+mfolder`' to have mutt set up the structure for you in advance.

If you use Sylpheed, the best way seems to be to create the new folder from within Sylpheed before letting mairix write into it. This seems to be all you need to do.

3.5 Command line options

The command line syntax is

For indexing mode:

```
mairix [-f path] [-p] [-v] [-Q]
```

For search mode

```
mairix [-f path] [-t] [-v] [-a] [-r] [-o mfolder] expr1 [expr2] ... [exprn]
```

For database dump mode

```
mairix [-f path] -d
```

The '`-f`' or '`--rcfile`' flag allows a different path to the '`mairixrc`' file to be given, replacing the default of '`~/.mairixrc`'.

The '`-p`' or '`--purge`' flag is used in indexing mode. Indexing works incrementally. When new messages are found, they are scanned and information about the words they contain is appended onto the existing information. When messages are deleted, holes are normally left in the message sequence. These holes take up space in the database file. This flag will compress the deleted paths out of the database to save space. Additionally, where '`mbox`' folders are in use, information in the database about folders that no longer exist, or which are no longer referenced in the rc-file, will be purged also.

The '`-v`' or '`--verbose`' flag is used in indexing mode. It causes more information to be shown during the indexing process. In search mode, it causes debug information to be shown if there are problems creating the symlinks. (Normally this would be an annoyance. If a message matches multiple queries when using '`-a`', mairix will try to create the same symlink multiple times. This prevents the same message being shown multiple times in the match folder.)

The '`-Q`' or '`--no-integrity-checks`' flag is used in indexing mode. Normally, mairix will do various integrity checks on the database after loading it in, and before writing the modified database out again. The checking helps to detect mairix bugs much earlier, but it has a performance penalty. This flag skips the checks, at the cost of some loss in robustness. See also the '`nochecks`' directive in Section 3.3 [mairixrc], page 4.

The '`--unlock`' flag is used in any mode. mairix dot-locks the database file to prevent corruption due to concurrent accesses. If the process holding the lock exits prematurely for any reason, the lockfile will be left behind. By using the '`--unlock`' option, an unwanted lockfile can be conveniently removed.

The '`-t`' or '`--threads`' option applies to search mode. Normally, only the messages matching all the specified expressions are included in the *match folder* that is built. With the '`-t`' flag, any message in the same thread as one of the matched messages will be included too. Note, the threading is based on processing the `Message-ID`, `In-Reply-To` and `References` headers in the messages. Some mailers don't generate these headers in a co-operative way and will cause problems with this threading support. (Outlook seems to be one culprit.) If you are plagued by this problem, the 'edit threads' patch to mutt may be useful to you.

The '`-d`' or '`--dump`' option causes mairix to dump the database contents in human-readable form to stdout. It is mainly for use in debugging. If this option is specified, neither indexing nor searching are performed.

The '`-a`' or '`--augment`' option also applies to search mode. Normally, the first action of the search mode is to clear any existing message links from the match folder. With the '`-a`' flag, this step is suppressed. It allows the folder contents to be built up by matching with 2 or more diverse sets of match expressions. If this mode is used, and a message matches multiple queries, only a single symlink will be created for it.

The '`-r`' or '`--raw-output`' option is used to force the raw output mode for a particular search, in preference to the output format defined by the '`mformat`' line in the '`mairixrc`' file. This may be useful for identifying which mbox contains a particular match, since there is way to see this when the matching messages are placed in the mfolder in this case. (Note for matches in maildir and MH folders when '`mformat`' is maildir or MH, the symbolic links in the mfolder will show the path to the matching message.)

The '`-o`' or '`--mfolder`' option is used in search mode to specify a match folder different to the one specified in the '`mairixrc`' to be used. The path given by the '`mfolder`' argument after this flag is relative to the folder base directory given in the '`mairixrc`' file, in the same way as the directory in the mfolder specification in that file is. So if your '`mairixrc`' file contains

```
base=/home/foobar/Mail
```

and you run mairix like this

```
mairix -o mfolder2 make,money,fast
```

mairix will find all of your saved junk emails containing these three words and put the results into '`/home/foobar/Mail/mfolder2`'.

The '`-o`' argument obeys the same conventions regarding initial '`/`' and '`.`' characters as the **mfolder** line in the '`.mairixrc`' file does.

Mairix will refuse to output search results (whether specified by the '`-o`' or in the '`.mairixrc`' file) into one of the folders that are indexed; it figures out that list by looking in the '`.mairixrc`' file, or in the file you specify using the '`-f`' option. This sanity check prevents you inadvertantly destroying one of your important folders (but won't catch all such cases, sadly).

The search mode runs when there is at least one search expression. Search expressions can take forms such as (in increasing order of complexity):

- A date expression. The format for specifying the date is described in section Section 3.6 [date_syntax], page 11.
- A size expression. This matches all messages whose size in bytes is in a particular range. For example, to match all messages bigger than 1 Megabyte the following command can be used

  ```
  mairix z:1m-
  ```

  To match all messages between 10kbytes and 20kbytes in size, the following command can be used:

  ```
  mairix z:10k-20k
  ```
- A word, e.g. '`pointer`'. This matches any message with the word '`pointer`' in the `To`, `Cc`, `From` or `Subject` headers, or in the message body.[2]
- A word in a particular part of the message, e.g. '`s:pointer`'. This matches any message with the word '`pointer`' in the subject. The qualifiers for this are :

  `t:pointer`
  : to match '`pointer`' in the `To:` header,

  `c:pointer`
  : to match '`pointer`' in the `Cc:` header,

  `a:pointer`
  : to match '`pointer`' in the `To:`, `Cc:` or `From:` headers ('`a`' meaning '`address`'),

  `f:pointer`
  : to match '`pointer`' in the `From:` header,

  `s:pointer`
  : to match '`pointer`' in the `Subject:` header,

  `b:pointer`
  : to match '`pointer`' in the message body.

  `m:pointer`
  : to match messages having a Message-ID header of '`pointer`'.

  Multiple fields may be specified, e.g. `sb:pointer` to match in the `Subject:` header or the body.
- A negated word, e.g. '`s:~pointer`'. This matches all messages that don't have the word '`pointer`' in the subject line.
- A substring match, e.g. '`s:point=`'. This matches all messages containing a word in their subject line where the word has '`point`' as a substring, e.g. '`pointer`', '`disappoint`'.
- An approximate match, e.g. '`s:point=1`'. This matches all messages containing a word in their subject line where the word has '`point`' as a substring with at most one error, e.g. '`jointed`' contains '`joint`' which can be got from '`point`' with one letter changed. An error can be a single letter changed, inserted or deleted.

[2] Message body is taken to mean any body part of type text/plain or text/html. For text/html, text within meta tags is ignored. In particular, the URLs inside `<A HREF="...">` tags are not currently indexed. Non-text attachments are ignored. If there's an attachment of type message/rfc822, this is parsed and the match is performed on this sub-message too. If a hit occurs, the enclosing message is treated as having a hit.

- A left-anchored substring match, e.g. '`s:^point=`'. This matches all messages containing a word in their subject line where the word begins with the string '`point`'. (This feature is intended to be useful for inflected languages where the substring search is used to avoid the grammatical ending on the word.) This left-anchored facility can be combined with the approximate match facility, e.g. '`s:^point=1`'.

  Note, if the '`^`' prefix is used without the '`=`' suffix, it is ignored. For example, '`s:^point`' means the same thing as '`s:point`'.

- A disjunction, e.g. '`s:pointer/dereference`'. This matches all messages with one or both of the words '`pointer`' and '`dereference`' in their subject lines.

- Each disjunction may be a conjunction, e.g. '`s:null,pointer/dereference=2`' matches all messages whose subject lines either contain both the words '`null`' and '`pointer`', or contain the word '`dereference`' with up to 2 errors (or both).

- A path expression. This matches all messages with a particular substring in their path. The syntax is very similar to that for words within the message (above), and all the rules for '`+`', '`,`', approximate matching etc are the same. The word prefix used for a path expression is '`p:`'. Examples:

  ```
  mairix p:/archive/
  ```

  matches all messages with '`/archive/`' in their path, and

  ```
  mairix p:wibble=1 s:wibble=1
  ```

  matches all messages with '`wibble`' in their path and in their subject line, allowing up to 1 error in each case (the errors may be different for a particular message.)

  Path expressions always use substring matches and never exact matches (it's very unlikely you want to type in the whole of a message path as a search expression!) The matches are always **case-sensitive**. (All matches on words within messages are case-insensitive.) There is a limit of 32 characters on the match expression.

The binding order of the constructions is:

1. Individual command line arguments define separate conditions which are AND-ed together
2. Within a single argument, the letters before the colon define which message parts the expression applies to. If there is no colon, the expression applies to all the headers listed earlier and the body.
3. After the colon, commas delineate separate disjuncts, which are OR-ed together.
4. Each disjunct may contain separate conjuncts, which are separated by plus signs. These conditions are AND-ed together.
5. Each conjunct may start with a tilde to negate it, and may be followed by a slash to indicate a substring match, optionally followed by an integer to define the maximum number of errors allowed.

Now some examples. Suppose my email address is `richard@doesnt.exist`.

The following will match all messages newer than 3 months from me with the word '`chrony`' in the subject line:

```
mairix d:3m- f:richard+doesnt+exist s:chrony
```

Suppose I don't mind a few spurious matches on the address, I want a wider date range, and I suspect that some messages I replied to might have had the subject keyword spelt wrongly (let's allow up to 2 errors):

```
mairix d:6m- f:richard s:chrony=2
```

3.6 Syntax used for specifying dates

This section describes the syntax used for specifying dates when searching using the '`d:`' option.

Dates are specified as a range. The start and end of the range can both be specified. Alternatively, if the start is omitted, it is treated as being the beginning of time. If the end is omitted, it is treated as the current time.

There are 4 basic formats:

'`d:start-end`'
: Specify both start and end explicitly

'`d:start-`'
: Specify start, end is the current time

'`d:-end`' Specify end, start is 'a long time ago' (i.e. early enough to include any message).

'`d:period`'
: Specify start and end implicitly, as the start and end of the period given.

The start and end can be specified either absolute or relative. A relative endpoint is given as a number followed by a single letter defining the scaling:

letter	**meaning**	**example**	**meaning**
d	days	3d	3 days
w	weeks	2w	2 weeks (14 days)
m	months	5m	5 months (150 days)
y	years	4y	4 years (4*365 days)

Months are always treated as 30 days, and years as 365 days, for this purpose.

Absolute times can be specified in a lot of forms. Some forms have different meanings when they define a start date from that when they define an end date. Where a single expression specifies both the start and end (i.e. where the argument to d: doesn't contain a '`-`'), it will usually have different interpretations in the two cases.

In the examples below, suppose the current date is Sunday May 18th, 2003 (when I started to write this material.)

Example	**Start date**	**End date**	**Notes**
d:20030301−20030425	March 1st, 2003	25th April, 2003	
d:030301−030425	March 1st, 2003	April 25th, 2003	century assumed
d:mar1−apr25	March 1st, 2003	April 25th, 2003	
d:Mar1−Apr25	March 1st, 2003	April 25th, 2003	case insensitive
d:MAR1−APR25	March 1st, 2003	April 25th, 2003	case insensitive
d:1mar−25apr	March 1st, 2003	April 25th, 2003	date and month in either order
d:2002	January 1st, 2002	December 31st, 2002	whole year
d:mar	March 1st, 2003	March 31st, 2003	most recent March
d:oct	October 1st, 2002	October 31st, 2002	most recent October
d:21oct−mar	October 21st, 2002	March 31st, 2003	start before end
d:21apr−mar	April 21st, 2002	March 31st, 2003	start before end
d:21apr−	April 21st, 2003	May 18th, 2003	end omitted
d:−21apr	January 1st, 1900	April 21st, 2003	start omitted
d:6w−2w	April 6th, 2003	May 4th, 2003	both dates relative
d:21apr−1w	April 21st, 2003	May 11th, 2003	one date relative
d:21apr−2y	April 21st, 2001	May 11th, 2001	start before end
d:99−11	January 1st, 1999	May 11th, 2003	2 digits are a day of the month if possible, otherwise a year

d:99oct−1oct	October 1st, 1999	October 1st, 2002	end before now, single digit is a day of the month
d:99oct−01oct	October 1st, 1999	October 31st, 2001	2 digits starting with zero treated as a year
d:oct99−oct1	October 1st, 1999	October 1st, 2002	day and month in either order
d:oct99−oct01	October 1st, 1999	October 31st, 2001	year and month in either order

The principles in the table work as follows.

- When the expression defines a period of more than a day (i.e. if a month or year is specified), the earliest day in the period is taken when the start date is defined, and the last day in the period if the end of the range is being defined.
- The end date is always taken to be on or before the current date.
- The start date is always taken to be on or before the end date.