NAME

rrdfetch – Fetch data from an RRD.

SYNOPSIS

rrdtool fetch *filename CF* [**––resolution|–r** *resolution*] [**––start|–s** *start*] [**––end|–e** *end*] [**––align–start|–a**] [**––daemon|–d** *address*]

DESCRIPTION

The **fetch** function is normally used internally by the graph function to get data from **RRD**s. **fetch** will analyze the **RRD** and try to retrieve the data in the resolution requested. The data fetched is printed to stdout. **UNKNOWN** data is often represented by the string "NaN" depending on your OS's printf function.

filename the name of the **RRD** you want to fetch the data from.

CF the consolidation function that is applied to the data you want to fetch (AVERAGE,MIN,MAX,LAST)

––resolution|–r *resolution* (default is the highest resolution)
the interval you want the values to have (seconds per value). An optional suffix may be used (e.g. `5m` instead of `300` seconds). **rrdfetch** will try to match your request, but it will return data even if no absolute match is possible. See "RESOLUTION INTERVAL".

––start|–s *start* (default end–1day)
start of the time series. A time in seconds since epoch (1970–01–01) is required. Negative numbers are relative to the current time. By default, one day worth of data will be fetched. See also "AT-STYLE TIME SPECIFICATION" for a detailed explanation on ways to specify the start time.

––end|–e *end* (default now)
the end of the time series in seconds since epoch. See also "AT-STYLE TIME SPECIFICATION" for a detailed explanation of how to specify the end time.

––align–start|–a
Automatically adjust the start time down to be aligned with the resolution. The end-time is adjusted by the same amount. This avoids the need for external calculations described in RESOLUTION INTERVAL, though if a specific RRA is desired this will not ensure the start and end fall within its bounds.

––daemon|–d *address*
Address of the rrdcached daemon. If specified, a `flush` command is sent to the server before reading the RRD files. This allows **rrdtool** to return fresh data even if the daemon is configured to cache values for a long time. For a list of accepted formats, see the **–l** option in the rrdcached manual.

```
rrdtool fetch --daemon unix:/var/run/rrdcached.sock /var/lib/rrd/foo.rrd AVE
```

Please note that due to thread-safety reasons, the time specified with **–s** and **–e** cannot use the complex forms described in "AT-STYLE TIME SPECIFICATION". The only accepted arguments are "simple integers". Positive values are interpreted as seconds since epoch, negative values (and zero) are interpreted as relative to *now*. So "1272535035" refers to "09:57:15 (UTC), April 29th 2010" and "–3600" means "one hour ago".

RESOLUTION INTERVAL

In order to get RRDtool to fetch anything other than the finest resolution RRA **both** the start and end time must be specified on boundaries that are multiples of the desired resolution. Consider the following example:

```
rrdtool create subdata.rrd -s 10 \
 DS:ds0:GAUGE:5m:0:U \
 RRA:AVERAGE:0.5:5m:300h \
 RRA:AVERAGE:0.5:15m:300h \
 RRA:AVERAGE:0.5:1h:50d \
 RRA:MAX:0.5:1h:50d \
 RRA:AVERAGE:0.5:1d:600d \
 RRA:MAX:0.5:1d:600d
```

This RRD collects data every 10 seconds and stores its averages over 5 minutes, 15 minutes, 1 hour, and 1 day, as well as the maxima for 1 hour and 1 day.

Consider now that you want to fetch the 15 minute average data for the last hour. You might try

```
rrdtool fetch subdata.rrd AVERAGE -r 15m -s -1h
```

However, this will almost always result in a time series that is **NOT** in the 15 minute RRA. Therefore, the highest resolution RRA, i.e. 5 minute averages, will be chosen which in this case is not what you want.

Hence, make sure that

1. both start and end time are a multiple of 900 (`15m`)
2. both start and end time are within the desired RRA

So, if time now is called "t", do

```
end time == int(t/900)*900,
start time == end time - 1hour,
resolution == 900.
```

Using the bash shell, this could look be:

```
TIME=$(date +%s)
RRDRES=900
rrdtool fetch subdata.rrd AVERAGE -r $RRDRES \
   -e $(($TIME/$RRDRES*$RRDRES)) -s e-1h
```

Or in Perl:

```
perl -e '$ctime = time; $rrdres = 900; \
         system "rrdtool fetch subdata.rrd AVERAGE \
                 -r $rrdres -e @{[int($ctime/$rrdres)*$rrdres]} -s e-1h"'
```

Or using the **--align-start** flag:

```
rrdtool fetch subdata.rrd AVERAGE -a -r 15m -s -1h
```

AT-STYLE TIME SPECIFICATION

Apart from the traditional *Seconds since epoch*, RRDtool does also understand at-style time specification. The specification is called "at-style" after the Unix command *at* (1) that has moderately complex ways to specify time to run your job at a certain date and time. The at-style specification consists of two parts: the **TIME REFERENCE** specification and the **TIME OFFSET** specification.

TIME REFERENCE SPECIFICATION

The time reference specification is used, well, to establish a reference moment in time (to which the time offset is then applied to). When present, it should come first, when omitted, it defaults to **now**. On its own part, time reference consists of a *time-of-day* reference (which should come first, if present) and a *day* reference.

The *time-of-day* can be specified as **HH:MM**, **HH.MM**, or just **HH**. You can suffix it with **am** or **pm** or use 24-hours clock. Some special times of day are understood as well, including **midnight** (00:00), **noon** (12:00) and British **teatime** (16:00).

The *day* can be specified as *month-name day-of-the-month* and optional a 2- or 4-digit *year* number (e.g. March 8 1999). Alternatively, you can use *day-of-week-name* (e.g. Monday), or one of the words:

yesterday, **today**, **tomorrow**. You can also specify the *day* as a full date in several numerical formats, including **MM/DD/[YY]YY**, **DD.MM.[YY]YY**, or **YYYYMMDD**.

NOTE1: this is different from the original *at*(1) behavior, where a single-number date is interpreted as MMDD[YY]YY.

NOTE2: if you specify the *day* in this way, the *time-of-day* is REQUIRED as well.

Finally, you can use the words **now**, **start**, **end** or **epoch** as your time reference. **Now** refers to the current moment (and is also the default time reference). **Start** (**end**) can be used to specify a time relative to the start (end) time for those tools that use these categories (**rrdfetch**, rrdgraph) and **epoch** indicates the *IX epoch (*IX timestamp 0 = 1970−01−01 00:00:00 UTC). **epoch** is useful to disambiguate between a timestamp value and some forms of abbreviated date/time specifications, because it allows one to use time offset specifications using units, eg. **epoch**+19711205s unambiguously denotes timestamp 19711205 and not 1971−12−05 00:00:00 UTC.

Month and day of the week names can be used in their naturally abbreviated form (e.g., Dec for December, Sun for Sunday, etc.). The words **now**, **start**, **end** can be abbreviated as **n**, **s**, **e**.

TIME OFFSET SPECIFICATION

The time offset specification is used to add/subtract certain time intervals to/from the time reference moment. It consists of a *sign* (**+** or **−**) and an *amount*. The following time units can be used to specify the *amount*: **years**, **months**, **weeks**, **days**, **hours**, **minutes**, or **seconds**. These units can be used in singular or plural form, and abbreviated naturally or to a single letter (e.g. +3days, −1wk, −3y). Several time units can be combined (e.g., −5mon1w2d) or concatenated (e.g., −5h45min = −5h−45min = −6h+15min = −7h+1h30m−15min, etc.)

NOTE3: If you specify time offset in days, weeks, months, or years, you will end with the time offset that may vary depending on your time reference, because all those time units have no single well defined time interval value (1 year contains either 365 or 366 days, 1 month is 28 to 31 days long, and even 1 day may be not equal to 24 hours twice a year, when DST-related clock adjustments take place). To cope with this, when you use days, weeks, months, or years as your time offset units your time reference date is adjusted accordingly without too much further effort to ensure anything about it (in the hope that *mktime*(3) will take care of this later). This may lead to some surprising (or even invalid!) results, e.g. 'May 31 −1month' = 'Apr 31' (meaningless) = 'May 1' (after *mktime*(3) normalization); in the EET timezone '3:30am Mar 29 1999 −1 day' yields '3:30am Mar 28 1999' (Sunday) which is an invalid time/date combination (because of 3am −> 4am DST forward clock adjustment, see the below example).

In contrast, hours, minutes, and seconds are well defined time intervals, and these are guaranteed to always produce time offsets exactly as specified (e.g. for EET timezone, '8:00 Mar 27 1999 +2 days' = '8:00 Mar 29 1999', but since there is 1−hour DST forward clock adjustment that occurs around 3:00 Mar 28 1999, the actual time interval between 8:00 Mar 27 1999 and 8:00 Mar 29 1999 equals 47 hours; on the other hand, '8:00 Mar 27 1999 +48 hours' = '9:00 Mar 29 1999', as expected)

NOTE4: The single-letter abbreviation for both **months** and **minutes** is **m**. To disambiguate them, the parser tries to read your mind :) by applying the following two heuristics:

1. If **m** is used in context of (i.e. right after the) years, months, weeks, or days it is assumed to mean **months**, while in the context of hours, minutes, and seconds it means minutes. (e.g., in −1y6m or +3w1m **m** is interpreted as **months**, while in −3h20m or +5s2m **m** the parser decides for **minutes**).
2. Out of context (i.e. right after the **+** or **−** sign) the meaning of **m** is guessed from the number it directly follows. Currently, if the number's absolute value is below 6 it is assumed that **m** means **months**, otherwise it is treated as **minutes**. (e.g., −6m == −6m minutes, while +5m == +5 months)

Final NOTES: Time specification is case-insensitive. Whitespace can be inserted freely or omitted altogether. There are, however, cases when whitespace is required (e.g., 'midnight Thu'). In this case you should either quote the whole phrase to prevent it from being taken apart by your shell or use '_' (underscore) or ',' (comma) which also count as whitespace (e.g., midnight_Thu or midnight,Thu).

TIME SPECIFICATION EXAMPLES

Oct 12 — October 12 this year

−1month or *−1m* — current time of day, only a month before (may yield surprises, see NOTE3 above).

noon yesterday −3hours — yesterday morning; can also be specified as *9am−1day*.

23:59 31.12.1999 — 1 minute to the year 2000.

12/31/99 11:59pm — 1 minute to the year 2000 for imperialists.

12am 01/01/01 — start of the new millennium

end−3weeks or *e−3w* — 3 weeks before end time (may be used as start time specification).

start+6hours or *s+6h* — 6 hours after start time (may be used as end time specification).

931200300 — 18:45 (UTC), July 5th, 1999 (yes, seconds since 1970 are valid as well).

19970703 12:45 — 12:45 July 3th, 1997 (my favorite, and it has even got an ISO number (8601)).

ENVIRONMENT VARIABLES

The following environment variables may be used to change the behavior of `rrdtool fetch`:

RRDCACHED_ADDRESS

If this environment variable is set it will have the same effect as specifying the `--daemon` option on the command line. If both are present, the command line argument takes precedence.

AUTHOR

Tobias Oetiker <tobi@oetiker.ch>