RRDTOOL mini HOWTO

Netherlabs BV (bert hubert <bert.hubert@netherlabs.nl>)
Remco van Mook <remco@virtu.nl>
v0.0.1 $Date: 2002/06/22 15:33:35 $

A very hands-on mini HOWTO for rrdtool

Contents

1 rrdtool

rrdtool is a very powerful tool for storing data and making pretty graphs. There is a lot of documentation, but no simple HOWTO. This is the simple HOWTO.

1.1 Vocabulary

Vocabulary:

```
RRD                     The Round Robin Database
rrd                     an RRD file, containing:
rrd variables           which each have a lot of:
rrd datapoints          which you can plot.
```

1.2 How to create halfway decent RRD files

When you use counters that are reset when you read them, use this script. Assumption here is that you read and reset the counters every 900 seconds (i.e. 15 minutes).

Let's create an rrd called 'datafile.rrd':

```
$ rrdtool create datafile.rrd \
        DS:packets:ABSOLUTE:900:0:10000000 \
        RRA:AVERAGE:0.5:1:9600 \
        RRA:AVERAGE:0.5:4:9600 \
        RRA:AVERAGE:0.5:24:6000
```

This rrd has one DataSource, called 'packets'. In theory you can have many DataSources within your rrd, but for now we'll stick to one.

ABSOLUTE refers to the data; when the figures you enter are not an ever-increasing counter, this is the way to go, other options like 'COUNTER' are available.

0 and 10000000 refer to the minimum and maximum value that can be entered as valid data.

There are 3 'Archives' inside this rrd, an average over 1 measurement (which is the measurement itself), average over 4 measurements and average over 24 measurements.

Given the 15-minute time for measurements, this gives you stored averages for: 15 minutes, 1 hour and 6 hours.

The final number is the number of values of that type that should be stored (i.e. how large the round-robin archive should be).

9600 values of 15-minute periods gives you 100 days in that accuracy; 9600 values of 1 hour gives you 400 days in 1-hour accuracy; 6000 values of 6 hour-averages gives you 1500 days of that accuracy. Trivial, no ? :)

Oh, and don't touch the 0.5 value. If you really want to know, read the documentation (man rrdcreate).

1.3 Update the information in a RRD file

Simple enough. Just run this command:

```
$ rrdtool update datafile.rrd time_t:value
```

You MUST always feed values in chronological order. Stuff breaks if you don't, so don't.

Time_t is the standard unix timestamp, the number of seconds since the first of January 1970 UTC. If you want to insert with a the timestamp of 'Now', just substitute 'N'. Example:

```
$ rrdtool update yourfile.rrd N:12322
```

This updates the first variable in the rrd file, and as we entered only one, that is clear enough.

1.4 Make cool graphs and amaze your friends

Lots of noise about preciously little, actually.

```
$ rrdtool graph graph.png DEF:pkt=datafile.rrd:packets:AVERAGE \
  LINE1:pkt#ff0000:Packets
```

By default, rrdtool wil produce PNG files, with a 1-day history.

with DEF you define a variable that should be plotted.

pkt

is the name of the variable, IN THE GRAPH

data.rrd:packets

is the name of the source rrd file and the name of the data, in the file, the one you specified in the create statement.

AVERAGE

is the consolidation method (choose from MIN,MAX,AVERAGE,LAST)

So it works like this: you take a variable from the rrd, which you then give a new name in the graph, so you can plot it. This is useful for when you are plotting data from multiple rrds, that might have identical names within those files. To plot these separately, in the graph they need to have different names of course.

LINE1 is a type of graph that can be drawn (other options are LINE2,LINE3,AREA,..) 'pkt' is the name of the variable IN THE GRAPH that should be plotted (as just defined by DEF) '#ff0000' is the color that should be used for this line (red, in this case) 'Packets' is what the line should be called in the legend.

Do not add a : between pkt and #ff0000!

adding a –start option allows you to change the timespan of the graph:

```
--start -1d gives a graph of the last day;
--start -1w gives a graph of the last week;
--start -1m gives a graph of the last month;
--start -1y gives a graph of the last year;
```

Basically, you can use all sorts of default 'at' notation here.

1.5 What is plotted?

If you enter 'COUNTER', you indicate that you are feeding the rrd data which is ever-increasing. This would lead to boring graphs, so rrdtool then plots the *difference* betweet a measurement and the previous one, divided by the elapsed time.

If you enter 'ABSOLUTE', rrdtool will assume that the counter resets upon reading, and plots the value divided by the time since the previous measurement.

For measuring pre-calculated data that should simply be plotted, use 'GAUGE'. Useful for measuring temperature, or diskspace used.

1.6 Calculations within your graph

Sometimes you need to perform calculations on your data, for example, to turn 'lines per second' into 'lines per hour'. The rrd *always* stores data in 'per second' format if you have specified COUNTER or ABSOLUTE.

You can do calculations on a DEFinition with a CDEF. For reasons which are unclear, the rrdtool author has decided to use the very arcane reverse polish notation, which is slightly easier to parse for programmers, but lots harder to write for people.

Let's say you have a DEF called 'lines', which stands for 'lines per second', and you want to plot 'lines per hour'. Then add this:

CDEF:lph=lines,3600,*

Now you can replace this:

LINE1:lines#ff0000:"Lines per second"

with:

LINE1:lph#ff0000:"Lines per hour"

But you can't delete the original DEF! It is needed to retrieve data from the rrd.

2 How to do this in a sane way:

I suggest that you make a '/var/log/rrd' and make a hierarchy from there. Then, you can easily clone this hierarchy somewhere else to store the pictures.

2.1 Script for cloning the hierarchy:

```
#! /bin/sh

cd /var/log/rrd
for i in `find . -type d | cut -c3-`;
do
  mkdir -p /var/www/rrd/$i
done
cd -
```

2.2 Script for updating the image files in a cloned hierarchy

```
#! /bin/sh

cd /var/log/rrd
rrds=`find . -type f -name '*.rrd' | cut -c3-`

for i in $rrds;
  do
  j=`echo $i | sed 's/.rrd//'`
  rrdtool graph /var/www/rrd/$j-day.png --start -1d DEF:pkt=$i:packet:AVERAGE LINE1:pkt#ff0000:Packets/sec
  rrdtool graph /var/www/rrd/$j-week.png --start -1w DEF:pkt=$i:packet:AVERAGE LINE1:pkt#ff0000:Packets/sec
  rrdtool graph /var/www/rrd/$j-month.png --start -1m DEF:pkt=$i:packet:AVERAGE LINE1:pkt#ff0000:Packets/sec
  rrdtool graph /var/www/rrd/$j-year.png --start -1y DEF:pkt=$i:packet:AVERAGE LINE1:pkt#ff0000:Packets/sec
done
cd -
```

2.3 Script that builds plain-jane HTML around this structure:

```
#! /bin/sh

cd /var/log/rrd
dirs=`find . -type d | grep -v images`
for i in $dirs;
  do
  html=/var/www/rrd/$i/index.html

  echo "<html><body>" > $html

  j=`echo $i | cut -c3-`
  echo "Processing $html "
```

```
    rrds=`find $i -type f -name '*.rrd' -maxdepth 1 | cut -c2-`

    for k in $rrds;
      do
      l=`echo $k | sed 's/.rrd//' | cut -c2-`
      m=`basename $k .rrd`
      if [ -d ./$l ];
        then

        echo "<a HREF=$m/>$m<br>" >> $html
        echo "<img src=$m-day.png border=0>" >> $html
        echo "<img src=$m-week.png border=0>" >> $html
        echo "<img src=$m-month.png border=0>" >> $html
        echo "<img src=$m-year.png border=0>" >> $html
        echo "</a><br>" >> $html

      else
        echo "$m<br>" >> $html
        echo "<img src=$m-day.png border=0>" >> $html
        echo "<img src=$m-week.png border=0>" >> $html
        echo "<img src=$m-month.png border=0>" >> $html
        echo "<img src=$m-year.png border=0>" >> $html
        echo "<br>" >> $html
      fi
    done
    cat "</body></html>" >> $html
  done
  cd -
```