

Automated Report Generator With L^AT_EX & MetaPost

Tillman Hodgson

Initial Revision: March 4, 2003

Revision date: March 7, 2003

1 Background

1.1 The motivation

We've all been there.

Slaving away creating Yet Another Monthly Report¹. The particular process I had was particularly² torturous:

- Using a Java based tool to generate a report of raw numbers that are displayed in a web page
- Cut and paste the values into spreadsheet in order to generate a chart
- Cut and paste the chart into a word processor in order to generate a report
- Try and remember which group of people is supposed to receive the report and email it to them

This was getting ridiculous — I'm a Unix guru, after all!

1.2 The inspiration

I admit it, I love L^AT_EX. It's a manly tool with the finesse of a fencing blade and a depth that's astounding. It strikes me as being fundamentally *correct* in how it's put together, and it's the closest I've ever felt to having a printing press right at my fingertips³. It has a daunting learning cliff and doesn't forgive mistakes.

In other words, it's a perfect tool for Unix users.

L^AT_EX is fundamentally a Turing-complete programming language wrapped around the idea of desktop publishing⁴. It's a very flexible tool and can be easily automated at the Unix command line. Assuming that I could find a way to:

1. Generate performance stats
2. Generates graphs based on the stats
3. Wrap a report around the graphs

I'd be able to spend my month-end whittlin' away at a script like a happy hacker rather than digging the cut and paste trench. The challenge was on!

The performance stats were easy — these are well-understood⁵ tools of the trade. That left only the graph generation. Within short order I'd discovered that MetaPost would generate nice looking graphs⁶ and integrated nicely with L^AT_EX.

2 Tools of the trade

2.1 The usual suspects

On a BSD based Unix, the usual suspects for non-interactive tools would be `vmstat`, `iostat`, `uptime`⁷, `ps` and friends.

There are also some interactive looks like `sysstat` and `top` and a whole heap of specialized tools. These aren't interesting to us precisely because they're interactive, and thus more difficult to script. The right tool for the job and all that.

By the way, FreeBSD contains a great man page that covers the basics of tuning (`tuning(7)`)⁸. If you're new to tuning, it's definitely worth a read.

⁴In the traditional sense of typesetting, not the new-fangled Publishing Wizard sense

⁵Warning: the author is pulling wool over your eyes

⁶I've also used Perl::GD and gnuplot but now prefer MetaPost

⁷To obtain the load average

⁸Available on the web at <http://www.freebsd.org/cgi/man.cgi?query=tuning>

¹YAMR, as in "I was yammer'ing at the manager the other day ..."

²Or typically, depending on your experiences

³Warning: don't put your fingers too close to a printing press, they're called *presses* for a reason

2.2 But is there something better?

I'm a fan of `bsdsar`⁹ which is a `sar`-workalike. It gathers performance stats using the basic monitoring tools that any host should have, and then stores the result into a plain text database. You can then extract the details you're interested in later on. This is important because scripting the collection of performance stats is more complicated to get right than it seems.

2.3 Our contrived example

Our fictional environment is a single server. For keep our example simple, we'll also assume that you're only interested in CPU and disk space utilization.

Since it handles the data store aspect automatically (and thus saves us a step), `bsdsar` will be used in our contrived example.

Naturally, this approach is merely a contrived example. If you actually consider this to be adequate performance monitoring I fondly suspect that you have yet to discover uses for swap space, networking, or databases¹⁰. The reader is encouraged to implement something that might actually be useful for them.

3 Making the performance monitoring harness

3.1 Define the goal

As per our contrived example, we want to obtain CPU and disk utilization. Hourly "snapshots" should be fine since performance monitoring focuses on general trends.

3.2 Get the data

`bsdsar -d` gives us our disk operations per second and `bsdsar -u` gives us all sorts of CPU stats. This should be all we need for raw data.

Unfortunately, that was the easy part. Fortunately, the hard part is fun.

3.3 Massage the CPU data

We'll start with the CPU data since that seems to be a bit more complicated and we're manly Unix geeks. Or something like that. In any case, here's what the raw output of `bsdsar -u` looks like:

| Time | % User | % Sys | % Nice | % Intrpt | % Idle |
|-------|--------|-------|--------|----------|--------|
| 00:00 | 30 | 6 | 0 | 1 | 63 |
| 01:00 | 2 | 1 | 0 | 1 | 96 |
| 02:00 | 27 | 11 | 0 | 1 | 61 |
| 03:00 | 9 | 2 | 0 | 0 | 90 |
| 04:00 | 1 | 0 | 0 | 1 | 98 |
| 05:00 | 16 | 2 | 0 | 0 | 82 |
| 06:00 | 13 | 3 | 0 | 2 | 83 |
| 07:00 | 25 | 5 | 0 | 2 | 68 |
| 08:00 | 42 | 14 | 0 | 0 | 44 |
| 08:20 | 9 | 4 | 0 | 1 | 86 |
| 08:40 | 24 | 9 | 0 | 0 | 66 |
| 09:00 | 46 | 13 | 0 | 2 | 40 |
| 09:20 | 27 | 9 | 0 | 1 | 63 |
| 09:40 | 23 | 12 | 0 | 2 | 64 |
| 10:00 | 8 | 5 | 0 | 0 | 87 |

Hmmm. That only goes until 10:00 ... this must be *today's* data. Since we'll end up running this from `cron`, we should use yesterday's data and get a full 24 hours worth. `bsdsar` has an `-n` option that takes a date in MMDD format. Some trickery with the `date` command should be able to automate this:

```
bsdsar -n `date -j -v -1d "+%m%d"` -u
```

Another problem is that we only want the hourly data and `bsdsar` is giving us data for 20 minutes after and 40 minutes after the hour for the hours between 8:00 and 17:00. A quick bit of `grep` work ought to get rid of the extra lines:

```
grep -v ^[0-9][0-9]:[24]0
```

While we've got the `grep` tool out, we may as well filter out the column headings:

```
grep ^[0-9]
```

MetaPost doesn't understand time values very well, unfortunately, so we'll have to convert those hours to simple integer format by dropping the redundant ":00" information:

```
sed -e 's/:00/'
```

⁹You can find it the FreeBSD ports collection, naturally

¹⁰Among many other potential performance bottlenecks

One last thing and I think we've got it. We're only interested in how busy the CPU is, which really means the inverse of how idle the CPU is. So the only column of data that we need is the % idle column. Besides, managers are happier when higher values mean that things are in better shape and this matches that interpretation better.

```
awk '{print $1, " ", $6}'
```

Oops! We need to capture this in a file. Let's add this to the end:

```
> cpu.data
```

So when we put it all together it looks something like this:

```
bsdsar -n `date -j -v-1d "+%m%d"` -u | \
grep -v ^[0-9][0-9]:[24]0 | grep ^[0-9] | \
sed -e 's/:00//' | \
awk '{print $1, " ", $6}' > cpu.data
```

The output file `cpu.data` looks like this:

```
00 73
01 77
02 77
03 72
04 97
05 78
06 84
07 67
08 71
09 37
10 87
11 77
12 34
13 91
14 63
15 86
16 89
17 86
18 69
19 81
20 71
21 78
22 80
23 64
```

Perfect!

3.4 Massage the disk data

Following a similar process for disk utilization, we should be able to obtain the disk operations per second with something like:

```
bsdsar -n `date -j -v-1d "+%m%d"` -d | \
grep -v ^[0-9][0-9]:[24]0 | grep ^[0-9] | \
sed -e 's/:00//' | \
awk '{print $1, " ", $2}' > disk.data
```

4 Making the MetaPost skeleton

So now we have our performance monitoring data in some nice and simple plain text two-column files. This is handy because MetaPost can use these files directly with its `gdraw` command.

To make a graph, MetaPost needs to be told to use its graphing module. You then need to provide the details of the graph (such as its size and what to draw) inside of a "fig" statement. Lastly, you need to say `bye` so that MetaPost knows when to stop processing the file. Here's what our initial skeleton to graph our CPU utilization graph looks like:

```
input graph;
beginfig(1);
  draw begingraph(3.5in,2.5in);
  gdraw "cpu.data";
endgraph;
endfig;
bye;
```

We're going to need to add some labels to the graph or it'll be hard to understand. We can use the `glabel` command to do that. Note that you need to wrap your text in `btex` and `etex` statements so that it knows where your text begins and ends:

```
input graph;
beginfig(1);
  draw begingraph(3.5in,2.5in);
  glabel.bot(btex Hour etex, OUT);
  glabel.lft(btex CPU Percent Idle etex, OUT);
  gdraw "cpu.data";
endgraph;
endfig;
bye;
```

Lastly, we need to create our second graph for the disk utilization. Rotating the left-hand label would also look more professional:

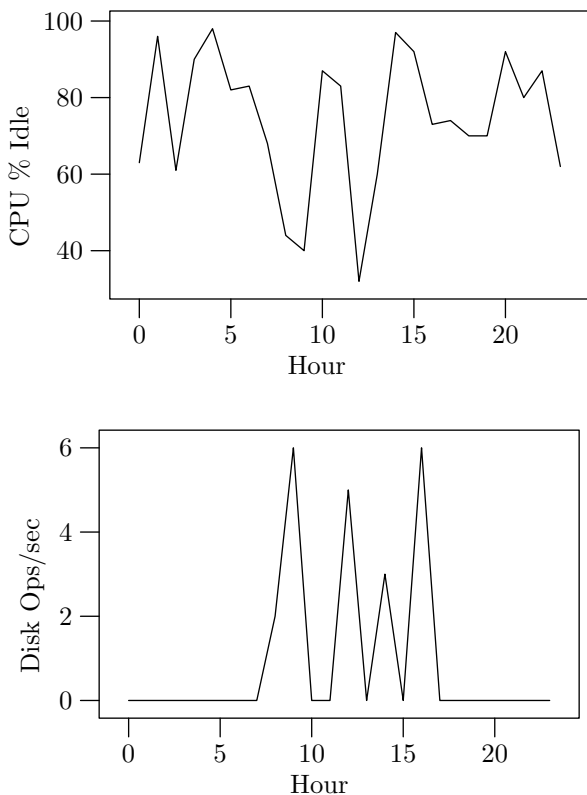
```
input graph;
beginfig(1);
  draw begingraph(3.5in,2.5in);
  glabel.bot(btex Hour etex, OUT);
  glabel.lft(btex CPU \% Idle etex rotated 90, OUT);
  gdraw "cpu.data";
endgraph;
endfig;
beginfig(2);
  draw begingraph(3.5in,2.5in);
```

```

    glabel.bot(btex Hour etex, OUT);
    glabel.lft(btex Disk Ops/sec etex rotated 90, OUT);
    gdraw "disk.data";
endgraph;
endfig;
bye;

```

Save this final version to a file called `perfgraphs.mp`. If you were to run the command `mpost perfgraphs11`, MetaPost would create two files (`perfgraphs.1` and `perfgraphs.2`). These files are your graphs, in encapsulated PostScript (EPS) format. They look like this:



5 Making the L^AT_EX skeleton

This isn't a L^AT_EX tutorial¹², so a simple bare-bones L^AT_EX skeleton is given here with no explanations as to how or why it works. At the very least you should add your own wording - after all, the graphs are pretty but a report should contain some witty analysis as well.

¹¹The `.mp` file extension is optional with the `mpost` command

¹²I recommend *The Not So Short Introduction to L^AT_EX2 ϵ* for that, available from CTAN

```

\documentclass[11pt]{article}
\usepackage{times}
\usepackage{textcomp}
\usepackage{lastpage}
\usepackage{latexsym}
\usepackage[dvips]{graphicx}
\usepackage[colorlinks, bookmarks, dvips]{hyperref}
\usepackage{fancyvrb}

\title{Daily Report for Yesterday}
\author{Tillman Hodgson}
\date{\today}

\begin{document}

\maketitle
\tableofcontents

\section{Executive Summary}

Put your summary here.

\section{CPU Utilization}

\begin{figure}
\centering
\includegraphics{perfgraphs.1}
\caption{Yesterdays CPU idle time}
\end{figure}

Put your witty analysis here.

\section{Disk Utilization}

\begin{figure}
\centering
\includegraphics{perfgraphs.2}
\caption{Yesterdays disk utilization}
\end{figure}

Put your witty analysis here.

\section{Recommendations}

Put your witty recommendations here.

\end{document}

```

6 Automating the works

6.1 Making the makefile

L^AT_EX can be time-consuming to work with by hand because the document must be “compiled” before it can be viewed. Automating this with a `make` can turn this into a single command. To do that, let's create a Makefile¹³:

```

#####
# Makefile for automated report generator #
#####

pdf:
    mpost perfgraphs
    latex report.ltx
    latex report.ltx

```

¹³I assume that you named your L^AT_EX file `report.ltx`

```

dvips -Ppdf -G0 report.dvi
ps2pdf report.ps

clean:
rm -f *.aux
rm -f *.log
rm -f *.dvi
rm -f *.toc
rm -f *.out
rm -f *.lof
rm -f *.lot
rm -f *.ps
rm -f *.pdf

```

To make your report (in PDF format) just type `make pdf` and a file called `report.pdf` will be created. To get rid of all the temporary files that it creates, type `make clean`¹⁴.

Note that we run `latex` twice in the Makefile. That's due to the particularities of \LaTeX and how it creates things like a table of contents. We also use regular `latex` and then convert the resulting PostScript file to PDF rather than using `pdflatex` directly because `pdflatex` can't handle the file format of our graphs (encapsulated PostScript). Normally this would result in bitmapped fonts being used (which look *horrible* in PDF format), but clever use of the `times` PostScript font package in our `report.ltx` file and some options given to `dvips` in our Makefile take care of that.

6.2 Enter the cron job

We want this to run each day, sometime after midnight, using yesterday's `bsdsar` data. After the report has been generated, we'd like to email to our manager. Something like this added to `/etc/crontab` ought to work:

```

22 6 * * * root /path/to/report_Makefile/make pdf && \
mail -s 'Yesterdays report' boss@example.com \
< /path/to/report/report.pdf

```

That's about it for our contrived example.

7 Where to next?

Aside from the obvious “monitor more/better performance metrics”, there are lots of areas where this little piece of automation could be further refined:

- What if you have more than one server? Centralized performance monitoring is a big topic and your choice of software can drastically affect your ability to automate report generation.

- The graphs can be made much fancier. The documents *Drawing Graphs with MetaPost* and *Some Experiences in Running MetaFont and MetaPost* (both available on CTAN) should be enough to get you started.
- The one part that's not automated is the entering of your textual analysis — building some kind of system to facilitate this would be neat. Perhaps a web front-end so the non- \LaTeX crowd can use it as a “black box” solution or extracting the information directly from a daily operations log¹⁵ would work.

¹⁴It's not necessary to do that though, \LaTeX will overwrite them if necessary

¹⁵You are keeping one of these, right?