R for SysAdmins

Working with sysstat Data

Author: Kwan Lowe
Date: 2015.09.08

The previous introduction on R for SysAdmins received a good response. Several asked about using R for graphing sysstat as I mentioned in the conclusion. This tutorial walks through using R to produce graphs of that data.

Keep in mind that R is not traditionally considered as a language for system administration. Normally we turn to bash, awk, Python or Ruby for many tasks. However, R is particularly well-suited for working with data and is arguably the best tool for the job.

This tutorial will walk through graphing the output from the sysstat/sar utility. There are other tools such as kSar, Cacti, etc. that do similar functions. R, however, can automate some of the analysis functions that are missing in most (all?) the freely-available tools. For example, we can easily run Bayesian analyses on our data to alert on resource trends or run an anomaly detection function to see hotspots in activity. Creating interactive applications for end-users is, though not trivial, facilitated with the many web packages available for R.

First, we generate some data using the *sar* utility.

```
sar > sar.out
```

This generates a file containing this output:

```
Linux 3.10.0-229.11.1.el7.x86_64 (cerberus.digitalhermit.com) 09/08/2015 _x86_64_
(2 CPU)

12:00:01 AM     CPU     %user     %nice   %system   %iowait    %steal     %idle
12:10:01 AM     all      0.11      0.00      0.17      0.02      0.00     99.70
12:20:01 AM     all      0.08      0.00      0.14      0.01      0.00     99.77
[...]
02:10:01 PM     all      1.17      0.00      0.30      0.06      0.00     98.48
02:20:01 PM     all      0.86      0.00      0.21      0.02      0.00     98.92
Average:        all      0.16      0.00      0.12      0.03      0.00     99.70
```

There are many options to sar but these are beyond scope of this tutorial.
Now we can start constructing our R script. First is to load in a few libraries to handle dates and graphics. Check the resources section below for more information on these packages.

```
library(lubridate)
library(lattice)
```

Next, we wead the input into a raw file. This will facilitate dealing with the non-data lines later. The sar output from my machine contains three lines of header and a final line with averages.

```
sar_raw <- readLines("sar.out")
```

The first line of our input is useful to keep. So we pull into its own variable for later use. We use the strsplit() and unlist() functions to parse the character vector (a "string" in normal Linux parlance). We also use the %>% operator to chain inputs, much like the pipe (|) char does on the command line.

```
sar_title <- sar_raw[1] %>% strsplit("\t") %>% unlist()
```

From the sar_title header, we can extract some useful bits such as the date of the report. This is important because the date is only kept in the header. To extract the date of the report we use the lubridate::mdy (month-day-year) function which extracts the time from a character string:

```
report_date <- mdy(sar_title[2])
```

This allows us to pull the year, month, date, etc. with

```
year(report_date)
```

Next, we actually grab the data into a proper data table (the above was just a raw read into a temporary table). This allows some easier manipulation such as dropping the last few lines.

```
sar_dat <- read.table(text=sar_raw, skip=2, nrows=(length(sar_raw) - 4), header=TRUE )
```

At this point we can delete the temporary sar_raw table:

```
rm("sar_raw")
```

Now we rename the tables to make it look prettier when graphed. There are other ways of labeling but this works in a pinch.

```
names(sar_dat) <- c("Time", "MM", "CPU", "User", "Nice", "System", "IOWait", "Steal",
"Idle")
```

The main issue with using the names() function is that the input data order may change. Not a big worry here, but keep this in mind if you are reading in other types of data.

To avoid this, you can explicitly rename the columns using the setnames() function.

```
#   data.table package.
#   library(data.table)
#   setnames(mydata, "newname", "oldname")
```

Then, add a new column pTime that's a combination of the Time and MM columns. Though the character vector for holding the date/time is perfectly readable for humans, to use it properly in R we need to convert it to a Posix date format. A couple notes:

- We use the %I to indicate a 12-hour format. If %H were used, it would assume 24hr format and drop the %p specifier.
- By default, the posixct time stamps gets the current date and time. We clean this up by using the date info we saved earlier in report_date.

We use dplyr::mutate to add another column to our data frame that contains the Posix date. We then add another column, pDate, that parses the pTime column into a Posix date and corrects the timestamp to that of the report date.

```
sar_dat <- within(sar_dat, pTime <- paste(Time, MM))
sar_dat <- mutate(sar_dat, pDate = parse_date_time(pTime, "%I%M%S %p",
tz="America/New_York"))
year(sar_dat[,"pDate"])  <- year(report_date)
month(sar_dat[,"pDate"]) <- month(report_date)
day(sar_dat[,"pDate"])   <- day(report_date)
```

Now, say we're only interested in the CPU information. We can create a new dataframe containing just that information:

```
sar_cpu <- select(sar_dat, pDate, User, System)
```

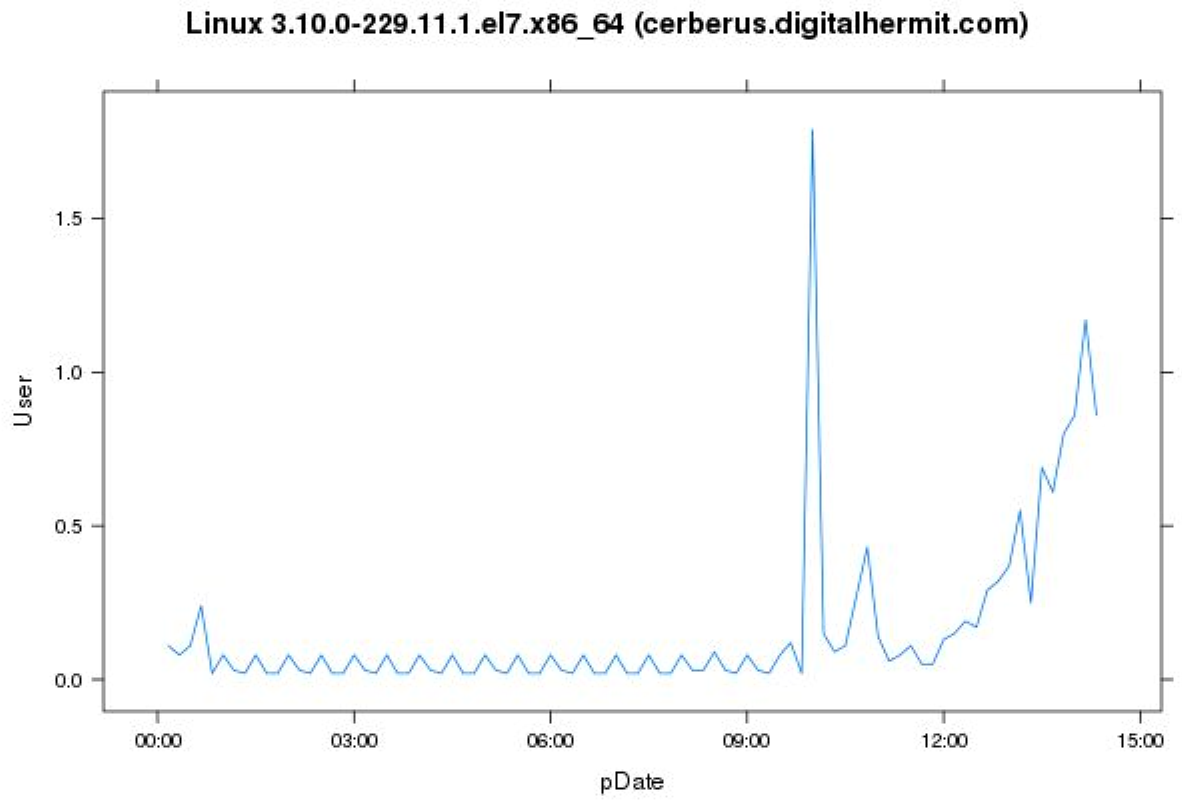
Finally, graph it using the lattice graphics package. Here we plot the User vs pDate with data found in sar_cpu.

```
xyplot(User ~ pDate,sar_cpu, type=c("l"))
```

To make it complete, we add a title using the main= parameter, substituting the header from the sar_title we saved earlier.

```
sar_plot <- xyplot(User ~ pDate,sar_cpu, type=c("l"), main=sar_title[1])
print(sar_plot)
```

This produces the following graph:

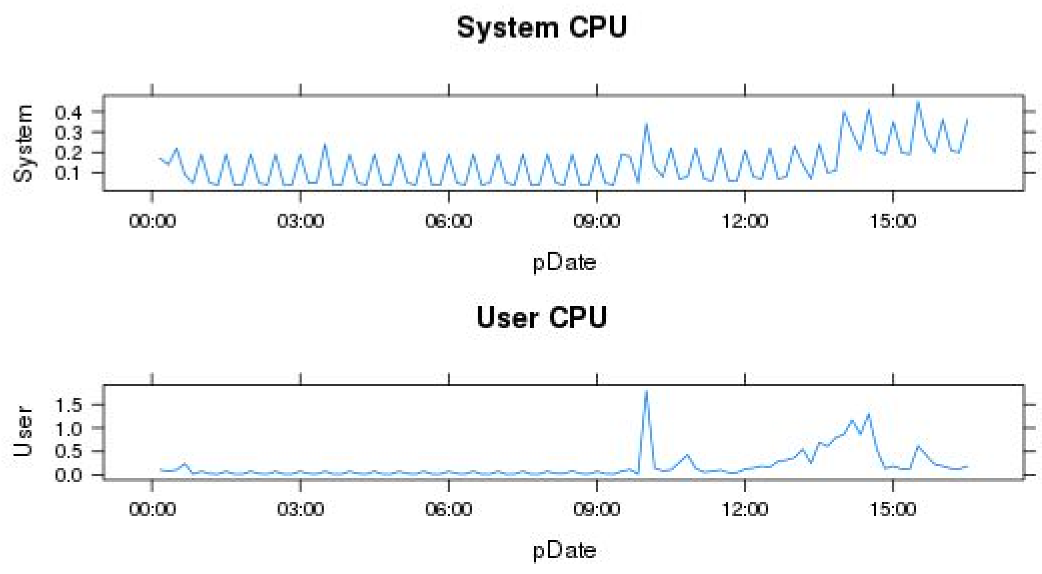


For these examples I deliberately added a few extra steps for clarity. Even so, this is less than 20 lines of code. My actual R-script is under ten lines. It would be difficult to do it as easily in any other language. Even using Excel, which is notoriously difficult to automate, it takes multiple lines of code just to properly format the data.

Anyhow, I hope this whets your appetite for learning more about R.

More examples:

```
        library(gridExtra)
        plot1 <- xyplot(System ~ pDate,sar_cpu, type=c("l"), main="System CPU")
        plot2 <- xyplot(User ~ pDate,sar_cpu, type=c("l"), main="User CPU")
        grid.arrange(plot1, plot2)
```



Resources:

https://www.r-project.org/
https://cran.r-project.org/web/packages/lubridate/lubridate.pdf
https://cran.r-project.org/web/packages/lattice/lattice.pdf
https://cran.r-project.org/web/packages/dplyr/dplyr.pdf