

Generamos diagramas Google con un script Perl

DIBUJANDO DATOS

Un módulo de CPAN pasa instrucciones de dibujo en Perl orientado a objetos a Google Chart, que dibuja diagramas muy atractivos visualmente. **POR MICHAEL SCHILLI**

Hoy día se ve a muy poca gente con portátiles Windows en conferencias de software libre, a menos que quieran realmente ser el centro de atención, como lo sería un hombre de las cavernas. Durante un tiempo estuve buscando un sustituto a mi viejo portátil, hasta que una oferta de Dell atrajo mi atención: un precioso netbook Mini 9 Ubuntu, al imbatible precio de 230 \$ (véase la Figura 1). Finalmente me decidí. Leif, un chico del trabajo, incluso le dio a mi gadget un gracioso apodo: “Mini-Yo”, en referencia al pequeño clon del Dr. Maligno de la segunda película de Austin Powers.

Mi primera impresión fue excitante. A pesar de algunos extraños problemas con *ssh* y el driver inalámbrico, que pude resolver en línea, ¡funcionaba de verdad! A continuación me dispuse a reemplazar los escasos 512 MB de RAM por los 2 GB de un suministrador sin

marca de apenas 9,95 \$. Pero poco después tuve una sospecha: ¿consumiría ahora más batería el Netbook en modo suspendido y la descargaría prematuramente? Siendo ingeniero de profesión, tenía que investigar.

De Notas a Diagramas

En primer lugar, reinstalé el viejo módulo de memoria, suspendí el ordenador e hice lecturas del estado de la batería de la máquina reanimada a intervalos regulares en el curso de las siguientes 36 horas. La Figura 2 muestra mis notas manuscritas: una lista de porcentajes de descargas y horas.

Un día y medio después repetí el procedimiento con el chip de 2GB reinstalado. Los dos conjuntos de datos utilizan intervalos de medida irregulares y diferentes debido al método poco ortodoxo que utilicé. Para yuxtaponer la información gráficamente, como muestra la Figura 3, en primer lugar tenía

que ejecutar el script del Listado 1 para normalizar los datos antes de ejecutar el script *graph-discharge* del Listado 2.

Los resultados muestran que las baterías inicialmente se descargan aproximadamente a la misma velocidad con ambos chips de memoria. Conforme se aproximan al punto medio de descarga, el módulo de memoria mayor hace que la batería se descargue más rápidamente, lo cual no es preocupante, pero siempre es bueno visualizar los hechos con un atractivo diagrama.

Deslocalización del Gráfico

El gráfico no se generó con un programa ejecutado en mi máquina local, sino por un ordenador en un cluster, cortesía de Google.

El script Perl simplemente crea una URL, como muestra la Figura 4, y la envía al servicio Google Chart, el cual devuelve una imagen en formato PNG como resultado. Google tiene una res-



Figura 1: El pequeño netbook Dell con Ubuntu.

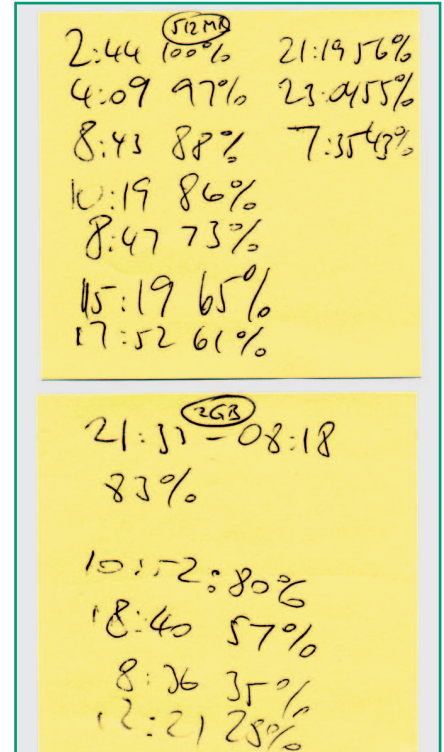


Figura 2: Mis notas con los estados de la batería.

tricción de 50.000 accesos por día, de sobra para nuestro ejemplo.

Orientación a Objetos vs. la Jungla URL

Para construir la URL de la Figura 4, el generador de gráficos tiene que seguir la Google Chart Developer's Guide [2] cuidadosamente y codificar diversas reglas, no sin cierto dolor, en abreviaturas difíciles de leer. Para facilitar las cosas, CPAN ofrece el módulo *Google::Chart*, que proporciona una interfaz orientada a objetos para definir el gráfico. Construye la URL paso a paso con el uso de llamadas a métodos fácilmente comprensibles.

Pero antes de comenzar a definir el gráfico, previamente habrá que consolidar las mediciones.

Curvando el Tiempo

Durante mis mediciones, medí los valores de descarga de la batería a intervalos regulares que no eran los mismos para los dos chips de memoria. Para yuxtaponer las dos curvas de descargas en una comparación directa, el script *data-normalize* (Listado 1) sitúa los tiempos en un eje temporal común. Ambos conjuntos de mediciones comienzan ahora en la hora 0 virtual.

Empezando con puntos temporales absolutos, el script calcula los tiempos relativos sustrayendo la hora virtual de comienzo.

Por ejemplo, si el primer conjunto incluye medidas a las 8:00 y 9:00 y el segundo conjunto tiene valores para las 11:00 y 11:30, el eje temporal virtual compartido comenzaría a las 0:00 e incluiría un segundo punto, para el conjunto 2 a las 0:30 y para el conjunto 1 a la 1:00.

Para hacer esto, el script *data-normalize* usa la estructura *\$data*, que contiene una referencia a un hash con las claves "2gb" y "0.5gb", referenciando cada una

de ellas a un array que contiene los puntos con los pares de hora y capacidad de la batería.

Normalizar entre 2 y 98

El objetivo de este método es comenzar los dos gráficos en un punto común y normalizar tanto los valores de hora en el eje X como las medidas de capacidad de la batería en el eje Y en un entero en el rango de 0 a 100, como aguarda el servicio Chart.

El script usa el módulo *DateTime* de CPAN para estos cálculos y fija la hora de la primera serie de medidas, seleccio-

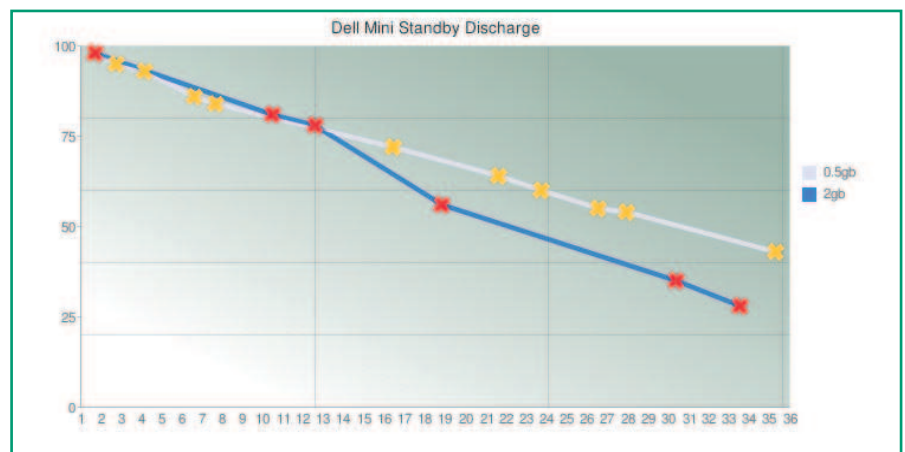


Figura 3: Google Chart representa el rendimiento de la batería del netbook con diferentes chips de memoria.

nada aleatoriamente, como el punto de partida del eje temporal común como `$day_start`. Guarda la hora de la medida actual como `$time_current`, mientras que el día en el cual se efectuó la última medida se guarda en `$day_current`.

Si el script encuentra que ha pasado un día entre dos medidas (por ejemplo si a una medida a las 23:00 le sigue una medida a las 8:00), las líneas 44 y 45 añaden un día a los contadores. Las líneas 50 a 51 calculan el número de minutos desde la última medida y guardan el valor en la variable `$x` (de valores del eje X), de donde se coge en la línea 53 y se lleva al array `@results`, junto con el nombre del conjunto de mediciones y el valor actual de la batería.

Las líneas 56 y 61 mantienen los valores más altos de X e Y y los guardan en

las entradas de hash `$max->{x}` y `$max->{y}`, respectivamente. Se usan más tarde en el bucle for de las líneas 68 y siguientes para normalizar todos los valores X/Y en un rango de 0 a 100. Además de esto, la variable `$margin` crea un margen con un ancho de 2 a derecha e izquierda antes de que `data-normalize` normalice finalmente los valores X/Y dentro del rango de 2 a 98. La razón de esto es evidente viendo posteriormente el gráfico; queda más bonito si el gráfico no toca los ejes horizontal y vertical. La sentencia `print` de la línea 70 envía los resultados a STDOUT (véase la Figura 5).

Tipo XY

Mientras buscaba un formato de gráfico adecuado, encontré el tipo "lxy" [2], el cual aguarda un conjunto de coordena-

das X e Y para una línea dada. Esto la hace fundamentalmente diferente de otros formatos, que suponen que los valores medidos están en la misma coordenada X para cualquier valor que visualizan. La línea 19 de `graph-discharge` por tanto define 'XY' para la opción `type` en la llamada al constructor del nuevo objeto de tipo `Google::Chart`.

El Listado 2 lee en primer lugar la salida del script `data-normalize` (línea 08) y ordena los valores X e Y en la estructura `$data`. Al final del bucle `while` de la línea 15, `$data->{"0.5gb"}->{x}` contiene todas las marcas de tiempo normalizadas para la configuración de 512 MB, y `$data->{"0.5gb"}->{y}` es un array con los estados correspondientes de la batería. Esto facilita pasarle un puntero al array de los registros X/Y al

Listado 1: data-normalize

```
01 #!/usr/local/bin/perl -w
02 use strict;
03 use DateTime;
04
05 my @result = ();
06 my $max = {};
07
08 my $data = {
09     "2gb" => [qw(
10         21:33 100 08:18 83 10:52 80
11         18:40 57 08:36 35 12:21 28
12     )],
13     "0.5gb" => [qw(
14         14:44 100 16:09 97 18:08 95
15         20:43 88 22:19 86 08:47 73
16         15:19 65 17:52 61 21:19 56
17         23:04 55 07:35 43
18     )]];
19
20 for my $conf (keys %$data) {
21     my $points = $data->{ $conf };
22     my $day_start;
23     my $day_current;
24
25     while( my($time, $charge) =
26         splice( @$points, 0, 2 ) ) {
27
28         my($hour, $minute) = split /:/, $time;
29
30         if(!defined $day_start) {
31             $day_start = DateTime->today();
32             $day_start->set_hour( $hour );
33             $day_start->set_minute( $minute );
34             $day_current = $day_start->clone();
35         }
36
37         my $time_current =
38             $day_current->clone();
39
40         $time_current->set_hour( $hour );
41         $time_current->set_minute( $minute );
42
43         if($time_current < $day_current) {
44             $time_current->add( days => 1 );
45             $day_current->add( days => 1 );
46         }
47
48         $day_current = $time_current->clone();
49
50         my $x = (($time_current->epoch() -
51             $day_start->epoch()) / 60);
52
53         push @result, [ $conf, $x, $charge ];
54
55         if(!exists $max->{x} or
56             $max->{x} < $x) {
57             $max->{x} = $x;
58         }
59         if(!exists $max->{y} or
60             $max->{y} < $charge) {
61             $max->{y} = $charge;
62         }
63     }
64 }
65
66 my $margin = 2;
67
68 for my $result (@result) {
69     my($symbol, $x, $y) = @$result;
70     print "$symbol ",
71         int($x*(100-2*$margin)/
72             $max->{x})+$margin,
73         " ",
74         int($y*(100-2*$margin)/
75             $max->{y})+$margin,
76         "\n";
77 }
```

parámetro *data* del objeto *Google::Chart* en la línea 21, tal y como lo espera el gráfico XY.

El parámetro *size* (línea 27) configura las dimensiones del gráfico: 750 por 400 es más o menos el máximo. Si intentamos dimensiones mayores, Google protestará y devolverá un mensaje “Bad Request”.

El título del gráfico, que se muestra en la parte superior, se configura mediante la opción *title* de la línea 29.

Cajón de Sastre Profesional

Para reemplazar el fondo blanco del gráfico por otro con un gradiente de blanco

a verde oliva de aspecto más profesional, la opción *fill* en la línea 33 configura el fondo a “*Linear-Gradient*”. Como se puede leer en la sección de colores [2], necesitamos un valor “c” para rellenar el área del gráfico. *color1* y *color2* aguardan dos colores RGB codificados en hexadecimal. Si el valor de *offset* es 0, el correspondiente color es puro a la izquierda del gráfico y se difumina a la derecha. Un valor de 1 hace que el gradiente vaya de derecha a izquierda. Los valores de las líneas 36 hasta 41 definen por tanto un fondo del gráfico con un gradiente blanco a la izquierda a verde oliva a la derecha. Un ángulo de 45 grados define

un gradiente diagonal desde la parte inferior izquierda hasta la parte superior derecha.

La opción *grid* (línea 45) dibuja una rejilla en el gráfico, dividiendo el tiempo transcurrido total, aproximadamente 36 horas, en la dirección *X* en tres secciones de 33 partes de 100. En la dirección *Y*, las líneas horizontales están espaciadas 20 puntos. El etiquetado de ejes del gráfico se define por el parámetro *axis* en la línea 50. El eje *X* está etiquetado con los valores de horas desde 1 a 36 a distancias equidistantes. El eje *Y* está etiquetado con los valores de 0 a 100 en pasos de 25. Tenga en cuenta que este etique-

Listado 2: graph-discharge

```

001 #!/usr/local/bin/perl -w           031 },
002 use strict;                         032
003 use Google::Chart;                  033 fill => {
004 use Google::Chart::Marker;         034     module =>
005                                     "LinearGradient",
006 my $data = {};                      035     args => {
007                                     036         target => "c",
008 open PIPE, ".data-normalize        037         angle  => 45,
009 |" or die;                           038         color1 => "abbaab",
010 while(<PIPE>) {                      039         offset1 => 1,
011     my($symbol, $x, $y) =           040         color2 => "FFFFFF",
012     split ' ', $_;                  041         offset2 => 0,
013     push @($data->{ $symbol          042     }
014     }->{x} }, $x;                    043 },
015     push @($data->{ $symbol          044
016     }->{y} }, $y;                    045 grid => {
017                                     046     x_step_size => 33,
018 my $graph =                          047     y_step_size => 20,
019     Google::Chart->new(              048 },
020     type => 'XY',                    049
021     data =>                            050 axis => [
022     [$data->{"0.5gb"}->{x},           051     { location => 'x',
023     $data->{"0.5gb"}->{y},           052     labels => [1..36],
024     $data->{"2gb"}->{x},             053     },
025     $data->{"2gb"}->{y},             054     { location => 'y',
026     ],                                055     labels =>
027     size => '750x400',               056     [0,25,50,75,100],
028     title => {                        057     },
029     text => "Dell Mini                058 ],
030     Standby Discharge"              059     color => ['E6E9FD',
031                                     '4D89F9'],
032                                     060
033                                     061 legend => ['0.5gb', '2gb'],
034                                     062
035                                     063 margin => [50, 50, 50, 50,
036                                     064     100, 100],
037                                     065
038                                     066     marker =>
039                                     Google::Chart::Marker->new(
040                                     067     markerset => [
041                                     068     { marker_type => 'x',
042                                     069     color => 'FFCC33',
043                                     070     dataset => 0,
044                                     071     datapoint => -1,
045                                     072     size => 15,
046                                     073     priority => 1,
047                                     074     },
048                                     075     { marker_type => 'x',
049                                     076     color => 'FF0000',
050                                     077     dataset => 1,
051                                     078     datapoint => -1,
052                                     079     size => 15,
053                                     080     priority => 1,
054                                     081     },
055                                     082     { marker_type => 'D',
056                                     083     color => 'E6E9FD', #
057                                     084     light blue
058                                     085     dataset => 0,
059                                     086     datapoint => -1,
060                                     087     size => 4,
061                                     088     priority => -1,
062                                     089     },
063                                     090     { marker_type => 'D',
064                                     091     color => '4D89F9', #
065                                     092     blue
066                                     093     dataset => 1,
067                                     094     datapoint => -1,
068                                     095     size => 4,
069                                     096     priority => -1,
070                                     097     },
071                                     098     ]),
072                                     099 );
073                                     100 $graph->render_to_file(filena
074                                     me =>
075                                     "chart.png");
076                                     101 system("xv chart.png");

```



Figura 4: Cuando se le presenta esta URL, Google Chart contesta con una imagen PNG como el gráfico de la Figura 3.

tado es completamente independiente de los registros que le pasemos: la información está normalizada en el rango de 1 a 100 para ambos ejes.

Los colores de línea para ambos conjuntos de registros se definen por la opción *color* en la línea 59 en el orden de los datos de entrada. *E6E9FD* es un azul claro, mientras que *4D89F9* es un azul cielo.

Para permitir al observador interpretar el gráfico, *legend* en la línea 61 dibuja una leyenda en la parte derecha del gráfico, dibujando cuadrados con los colores de línea junto con el nombre de los conjuntos de datos correspondientes.

Para evitar que los ejes del gráfico choquen con el borde de la imagen, la opción *margin* de la línea 63 fija un borde de 50 píxeles en todas las direcciones. Los últimos dos valores, que son

100 en ambos casos, configuran la separación de la leyenda en la dirección X desde el margen derecho y la separación entre el borde superior del gráfico y la ubicación de la leyenda en la dirección Y.

Si quisiéramos ver los puntos de datos individuales como cruces, podemos configurar la opción *marker* como se muestra en la línea 65. Los primeros dos elementos en *markerset* definen la apariencia de las cruces para los dos conjuntos de datos, que podemos especificar como *dataset = > 0* y *dataset = > 1*. Si *datapoint* está fijado a *-1*, el servicio de gráficos dibujará cada uno de los puntos de datos, aunque podemos usar una selección en su lugar. Los marcadores con una prioridad de *1* se dibujan en último lugar por *graph-discharge*, asegurando de esta manera que las líneas del gráfico no cubren las cruces.

Si las líneas del gráfico no alcanzan nuestros estándares estéticos, debido a que preferimos líneas más gruesas, por ejemplo, podemos usar una configuración de *marker* que es ligeramente contraintuitiva. En lugar de usar 'x' como el *marker_type*, una entrada como 'D' configura las propiedades de la línea, como el grosor y el color. En otras palabras, si preferimos líneas más gruesas para el gráfico, simplemente debemos especificar el mismo color que se usó previamente con la opción *color* de la línea 59 y configurar un grosor del marcador a 4. La prioridad *-1* evita que las líneas tapen ninguna cruz, asegurándose de que se dibujan en primer lugar.

Finalmente, el método *render_to_file* llamado en la línea 98 crea una URL como la de la Figura 4 y la envía a Google, que devuelve el resultado en aproximadamente un segundo en forma de un archivo con formato PNG, que podemos

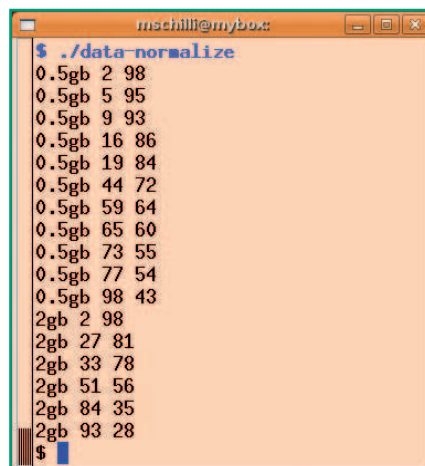


Figura 5: Los resultados se mandan a STDOUT.

guardar en nuestro disco con el nombre de archivo especificado.

Instalación

El módulo *Google::Chart* está disponible desde CPAN y requiere el sistema de objetos posmodernos *Moose* (véase el cuadro “Programación Posmoderna”), el cual necesita toda una retahíla de dependencias. Por tanto, es una buena idea utilizar una consola CPAN o un administrador de paquetes. En el momento de escribir este artículo, *Google::Chart* aún carecía de un par de funcionalidades, pero tras una breve conversación con los desarrolladores, tomaron rápidamente mis parches de la página colaborativa GitHub.com [3] y los aplicaron a la versión de desarrollo 0.05014_01, que junto con la versión estable, está disponible para su descarga [4]. ¡Larga vida GitHub: el nuevo amanecer de la colaboración del software libre!

Computación Posmoderna

Larry Wall, creador de Perl, lo presentó inicialmente como un lenguaje posmoderno en su charla del Linux World de 1999 [5]. Al contrario que el modernismo, en el que según su opinión parece que todas las cosas están aisladas, el posmodernismo se centra en la visión global. Los lenguajes de computación moderna toman un concepto (objetos, pila de código, paréntesis) y los llevan a efecto. Los lenguajes posmodernos integran muchos conceptos, permitiendo a cada uno de ellos funcionar a su manera. “El Modernista cree más en OR que en AND. Los Posmodernistas creen más en AND que en OR”.

Wall señala, en resumen, que los posmodernistas eligen, sin necesidad de justificar sus elecciones, “lo que aman y lo que odian”. Por extensión, Linux y el movimiento del software libre son posmodernos.

RECURSOS

- [1] Listados de este artículo: <http://www.linux-magazine.es/Magazine/Downloads/57>
- [2] Google Chart API Developer's Guide: <http://code.google.com/apis/chart>
- [3] Proyecto GitHub para el módulo *Google::Chart*: <http://github.com/lestrat/google-chart/tree/master>
- [4] Versión CPAN del módulo *Google::Chart* (incluye la versión de prueba 0.05014_01): <http://search.cpan.org/dist/Google-Chart>
- [5] Perl, el primer lenguaje de computación posmoderno: <http://www.perl.com/lpt/a/109>