POD Translation by *pod2pdf*

ajf@afco.demon.co.uk

Issue02.pod

Table of Contents
Issue02.pod

CaFe Perl v0.2 - Periódico de la Comunidad Perl de Capital Federal

Editorial

Hola gente del mundo Perl. Obligados, por gusto, por amor al arte o por el arte del amor estás acá frente a esta nueva publicación de CaFe Perl. Ã menlo o ódienlo, pero tomen una postura, hagan de esa su bandera y pónganla a blandir. Eso es el movimiento Open Source : corazón, tripa y convicción; eso y todo lo que hace a la libertad. Libertad de elegir hacer o no hacer, criticar o no, avanzar ... o avanzar.

Este número se viene un tanto renovado :

- con el agregado de algún que otro punto que fue sugerido en la encuesta del pasado mes de Mayo, más el poder llegar a quienes tienen conocimiento más avanzados. Resultado : una nueva sección llamada Mordiditas de aquí y de allá.
- el resumen de la lista sinceramente no tiene sentido, sobre todo porque no hay un tráfico como para perderse en la nebulosa de los mensajes y necesitar un mapa ... o un resumen
- Si bien CaFe Perl va a seguir existiendo primariamente en su formato POD, también va a estar disponible en HTML y PDF (por supuesto, generados a partir desde POD)

Espero que lo disfruten. Hasta la próxima taza de CaFe Perl !!!

Víctor A. Rodríguez (Bit-Man)

PERlitas

MKDoc es un manejador de contenidos (Content Managent System, o CMS) que, por supuesto es Open Source, y que nos permite :

- Gestionar y distribuir fácilmente contenidos en Internet
- Crear y gestionar comunidades online en su web site
- Publicar materiales en varios idiomas
- Cumplir con la sección 508 (*http://www.section508.gov/*) de standards de accesibilidad

Esta entrevista no explora como Perl está involucrado en MKDoc, como suele ocurrir en estas entrevistas, sino que aporta un nuevo ángulo desde el cual la problemática es la continuidad de un proyecto y cómo la comunidad Open Source, y Perl como parte de él, nos ayudan a no quedar atrapadas en las arena.

Pero basta de charla, disfrutemos los jugosos comentarios de Chris Croome, nuestro entrevistado de este número de CaFe Perl.

Chris, por favor una pequeña introducción para la gente de CaFe.pm

No soy un programador, me dedico a arquitectura de la información, diseño de interfaces y a los templates de XML para MKDoc, y estuve involucrado con MKDoc desde su concepción.

A veces el proyecto se queda sin combustible, pero "el show debe continuar". Cómo estaba compuesto MKDoc y que circunstancias llevaron a un paro total al proyecto

No se paró exactamente, pero si se desaceleró bastante desde que el desarrollador principal se fue el año pasado. MKDoc fue desarrollado cuando había muy pocos sistemas de manejo de contenidos por ahí, y fue muy innovador en su momento (pocos CMS por entonces estaban compenetrados con la accesibilidad y la adhesión a standards). Fue desarrollado principalmente porque nosotros (yo y unos amigos establecimos una pequeña compañía de diseño web en 1996) nos aburrimos de actualizar sites web para los clientes y buscamos que ellos mismos pudieran actualizarlos.

Nos juntamos con un joven y talentoso programador Perl y comenzamos a desarrollar un CMS que evolucionó en MKDoc. Siempre quise que MKDoc fuera con licencia GPL y aunque muchas de sus partes (como el sistema de plantillas Petal) eventualmente terminaron en CPAN este comenzó y siguió

siendo non-free hasta el año pasado. Esto resultó en que nunca tuvo una gran base de usuarios o una comunidad de usuarios y desarrolladores alrededor.

El desarrollador principal de MKDoc se fue el año pasado, y no quiero entrar en detalles pero las cosas fueron bastante poco placenteras en el suceso de los acontecimientos.

Así que inicialmente comenzaron el proyecto para "evitar el aburrimiento". Qué experiencias (positivas o no tanto) les dejó el proyecto ??

Bueno, no estaba hablando 100% en serio cuando dije que la motivación era evitar hacer tareas aburridas, pero hay algo de verdad — MKDoc fue diseñado para que la gente que produce contenidos para sites web pueda agregarlos directamente, y no dependa de terceros.

No tengo una buena respuesta acerca de qué experiencias gané de trabajar con MKDoc, preguntame en algunos años !

Y cuál fue el ambiente después del alto y cuál la reacción inmediata ?

El ambiente fue de liberación, lo primero que hice fue poner el código como GPL, el día después que se fue.

Qué le advertiría a los que están interesados en comenzar un nuevo proyecto ?? Y en convertir uno a Open Source ??

Ponerse de acuerdo en la licencia desde el principio porque esto puede ahorrar muchos problemas y también hacer mucha investigación sobre lo que está disponible — no tiene sentido comenzar un proyecto de CMS hora ya que hay muchos ahí fuera...

Ayudó el poner algunos módulos disponibles en CPAN ??

El módulo de plantillas (Petal, the Perl Template Attribute Language) fue un gran éxito, hay una comunidad en torno a él y se usa para muchas cosas más que MKDoc. Los fixes y reportes de errores de la comunidad para este código han sido muy útiles. Los otro módulos no han sido tomados de la misma forma, pero es entendible ya que no son tan claramente aplicables a otros programas.

Qué otras herramientas les dio la conversión de MKDoc al Open Source ??

Bueno sin GNU/Linux, Perl y Apache y todo el código de CPAN nunca hubiéramos podido hacer MKDoc con los recursos que tenemos, así que fue totalmente indispensable.

Y que ocurrió finalmente, y cuál es el estado actual del proyecto ??

Desde que el desarrollo decreció no hubo mucho interés en el código — los CMS hecho en PHP son más fáciles de instalar y ahora hay también existe Plone. El desarrollo en los últimos seis meses fue hecho exclusivamente para los clientes — cuando tuvimos los fondos como para agregar funcionalidad entonces contratamos el trabajo para hacerlo.

Todo el trabajo fue hecho sobre la versión estable (1.6) y la versión siguiente (1.8) no estuvo progresando muy rápido.

Qué será agregado a Perl/MKDoc para ser instalado tan fácil como los CMSs en PHP ??

mod_perl 2 está casi saliendo y me gustaría tener MKDoc trabajando con Apache 2/mod_perl 2 y disponible en paquetes nativos para las distros, rpm para Fedora, SuSE y Mandriva, deb para debian y Ubuntu para que la gente simplemente agregue un ‘repo’ a su ‘config’ de ‘apt’ o ‘yum’ y hagan un ‘apt-get install mkdoc’

Así y todo no sería tan fácil como en los CMSs PHP ya que sería necesario el usuario root pero no creo que haya una forma de no hacerlo así y es común para aplicaciones basadas en mod_perl,

Java y Zope, en contraste muchas aplicaciones PHP son diseñadas para ser usadas en server sin acceso a root.

Alguna experiencia, divertida o no, que tuvieron mientras este proceso se llevaba a cabo, y que quiera compartir ??

No recuerdo ninguna anécdota divertida, de lo que tengo más memoria es de cambiar la licencia — liberar el código da una gran sensación :-)

Algo más que nos quiera decir y que no le preguntamos ??

Bueno lo que es más interesante sobre la forma de producción del software libre es el potencial de aplicar este método de producción a la producción de otras cosas — creo que el mundo será un lugar mucho mejor si el modo de producción del Software Libre fuera generalizado y aplicado a la producción de todo.

Conclusiones (por Víctor A. Rodríguez)

Simplemente tengo que decir que sin el movimiento Open Source el proyecto no podría llevarse a cabo para una compañía del tamaño de MKDoc Ltd. donde los recursos disponibles son la primer señal de stop y el pensamiento creativo es la herramienta de cambio a mano para uso diario. Y sin liberar el código, la adopción y contribución se hubiera detenido.

El infierno no está en llamas, simplemente no es Open Source.

Infografía

- MKDoc : *http://www.mkdoc.org/*
- MKDoc Ltd. : *http://mkdoc.com*

SudorNews

- Perl Medic

Un nuevo libro de Addison Wesley dedicado a las mejores prácticas en Perl para que la próxima persona que tenga que acceder al código que estamos escribiendo, y mantener la aplicación, no tenga que ir al médico por perderse entre las líneas de código y no poder volver a la realidad.

http://books.slashdot.org/article.pl?sid=05/05/02/214235

- MySQL in a Nutshell

Esta vez es Russell Dyer, de la mano de O'Reilly, el que viene a rescatarnos, principalmente porque parte del libro cubre las API de MySQL para ser accedidas desde Perl. Ah, si también trata de otros lenguajes y otras tópicos de MySQL, pero no creo que eso sea importante ;-)
http://www.oreilly.com/catalog/mysqlian/

- 5.8.7 RC1 está disponible

Adivinaste, se trata del último release de Perl, un Release Candidate en este caso, que provee bug fixes y actualizacion de módulos ya integrados anteriormente pero actualizados a sus últimas versiones disponibles desde CPAN.
ftp://ftp.cpan.org/pub/CPAN/authors/id/N/NW/NWCLARK/perl-5.8.7-RC1.tar.bz2
http://search.cpan.org/~nwclark/perl-5.8.7-RC1/pod/perl587delta.pod

- mod_perl 2.0.0 disponible !!!

Otro gran avance de la ciencia que ha quedado eclipsado. Ya está disponible en CPAN, y para que lo tengan en cuenta, sin dudarlo. Los nuevos aportes de mod_perl son soporte de threads, mayor independencia de los cambios producidos en Apache y, por supuesto, una mayor estabilidad debido a este último punto.
http://search.cpan.org/~gozer/mod_perl-2.0.0/
http://search.cpan.org/~gozer/mod_perl-2.0.0/docs/user/intro/overview.pod

- Nueva lista relacionada con Perl : gentoo-perl

Dedicada a la implementación de Perl en la distribución Linux Gentoo. Aún sin mucho movimiento, pero bastante prometedora.
http://www.gentoo.org/main/en/lists.xml

- Gateway wireless hecho con Perl

Un proyecto casero sobre cómo administrar un gateway wireless desde una vieja PC usando Linux y Perl
http://www.perl.com/pub/a/2005/05/19/wireless_gw.html

- Parrot 0.2.0

Nueva versión de la máquina virtual que soportará a Perl 6 (entre otros). En esta versión se agrega soporte para el dispatch multimethod (declaración de la misma subrutina con varios puntos de entrada según los parámetros con los que se llama). Algo divertido de ver.

http://www.parrotcode.org

- Pugs 6.2.5

Una implementación de Perl 6 antes que la implementación sobre Parrot, como para ir comprobando ciertas funcionalidades. Un soporte mejorado de objetos y la posibilidad de usar módulos de Perl5 desde Perl 6.
http://search.cpan.org/dist/Perl6-Pugs/

- Manipular documentos Word con Perl

Ni una palabra que agregar, sólo satisfacción.
http://www.perl.com/pub/a/2005/05/26/word_control.html

- Ofertas laborales varias de Ten Roses

Unas observaciones importantes para estas ofertas :
Nombrar como referente a Walter Lamagna.
En todos los casos, el lugar de trabajo es en la zona de Tribunales, Capital Federal.
Se ofrece trabajo con continuidad y posibilidades ciertas de crecimiento.
Se valora la capacidad de investigar e incorporar nuevas tecnologías.
Enviar currículum vitae a hr@tenroses.com.ar

- Analista Funcional Senior (Ten Roses)

Estudiantes o graduados recientemente de las carreras de Ingeniería de Sistemas, Licenciatura en Sistemas o Ciencias de la Computación.
Excluyente : Experiencia mínima 3 años como analista funcional de aplicaciones web. Muy buen manejo de Inglés Técnico e interacción con clientes del exterior.
Preferente : Conocimientos de tecnologías web: Java, HTML, XML, etc., Conocimiento de Base de Datos, Experiencia en metodologías de desarrollo RUP y Agile, Conocimiento de herramientas de seguimiento de proyectos

- Java Developer Junior (Ten Roses)

Excluyente : Conocimientos de Java u otro lenguaje orientado a objetos, Conocimientos básicos en SQL
Preferente : Conocimientos de idioma inglés técnico, Conocimientos de HTML

- Java Developer (Ten Roses)

Excluyente : Experiencia de 2 años al menos en desarrollo de aplicaciones Web, en lenguaje Java, Conocimientos de SQL
Preferente : Conocimientos de XML/XSL, Conocimientos de idioma inglés técnico, Conocimientos de GUI en Java (AWT/SWING/Applets)

- Front End Developer (Ten Roses)

Excluyente : Experiencia en HTML, CSS, Javascript.

Preferente : Conocimientos de XML/XSL, Conocimientos de GUI en Java (AWT/SWING/Applets) (preferente, no excluyente), Conocimientos de idioma ingles técnico (lecto escritura)

- Analista de QA (Sr./Ssr.) (Ten Roses)

Profesional o estudiante avanzado de Sistemas
Experiencia en posición similar. Armado de casos de prueba, validación y
verificación de requerimientos.
Conocimientos de SQL y bases de datos relacionales.
Preferentemente con experiencia en desarrollo de aplicaciones web (Java,JSP, JavaScript, JDBC, HTML, XML/XSL)
Nivel intermedio o superior de inglés (oral y escrito)

- Business Systems Developer (Ten Roses)

We‘re looking for a motivated self-starter to manage the design, administration, and deployment of accounting and business systems. This position will be responsible for day-to-day development and management of existing systems and for assisting in coming up with solutions that enhance the overall effectiveness of the internal processes using internally developed solutions and best-of-breed software. Areas of opportunity include systems that manage accounting, reporting, shipping, order tracking, CRM, SFA, and BPM.
An ideal candidate will have at least 5 years experience in systems integration and will possess exceptional problem solving skills. More specifically, the applicant must be familiar with Great Plains or equivalent accounting software, be fluent in programming and managing MS Access, MS SQL, and PostgreSQL databases and be able to combine the information from disparate databases into a unified reporting methodology.
Experience in the following areas would also be beneficial:
Implementation of SFA and CRM solutions
Implementing BPM solutions
Great Plains integration manager
XML development
Writing functional requirements documentation.

Mordiditas de aquí y de allá

Introducción
En esta primera entrega de Mordiditas de aquí y de allá vamos a empezar por algo simple, básico y más de una vez encarado : cómo ordenar un array. No te dejes engañar puede ser trillado, puede implementarse con una llamada a una función, pero vas a ver cómo las cosas pueden complicarse a tal punto que vas a cambiar tu opinión, y en el camino vas a aprender un par de cosas interesantes.

La metodología de este artículo es la siguiente. Primero se presenta el problema junto con la solución cuestión que puedas usarlo como si fuera una receta, después viene la parte de la explicación de cómo se llega a esta receta final, y es la parte más jugosa de todo el asunto. Sin más, ahí vamos ...

El problema y la receta
Cómo ordenar un array : simplemente usando la función sort(). Por default esta compara cada elemento del array como strings, entonces para hacerla un poco más flexible se permite especificar qué comparación usar , a través de una referencia a una sub anónima que tenga el código para poder comparar los mismos :

```
@ordered = sort { $a <=> $b } @unordered;
```

En este caso estamos suponiendo que el array tiene elementos numéricos, y hacemos la comparación basados en este aspecto (tener en cuenta que los elementos a comparar son pasados a la sub anónima como $a y $b).

Ahora bien, si el array tiene muuuuuuuuchos elementos y la comparación compleja, entonces podemos hacer que todo vaya un poco más rápido de la siguiente forma :

```
@ordered =  map { $_->[1] }
        sort { $a->[0] <=> $b->[0] }
        map { [compute(), $_] } @unordered;
```

Donde se hace una división de tareas : compute() hace el cálculo de los valores complejos (por ejemplo, substrings, longitudes, etc) para ser calculados sólo una vez (y que en definitiva es lo que enlentece el proceso) y en la comparación sólo se compara usando los valores pre-calculados. Si tomamos como ejemplo que necesitamos ordenar un array basándonos en la longitud de sus elementos, que son strings, entonces la ordenación sin optimizar sería :

```
@ordered = sort { length $a <=> length $b } @unordered;
```

Si queremos hacer una optimización, entonces usamos la segunda forma pero usando la siguiente función compute() :

```
@ordered =  map { $_->[1] }
            sort { $a->[0] <=> $b->[0] }
            map { [length $_, $_] } @unordered;
```

Si usás esta receta, entonces todo va a funcionar, vos vas a estar contento, te van a felicitar porque todo anda rápido, todos te van a querer tener de amigo ... pero vos no vas a saber por qué. Si querés enterarte seguí leyendo, que no te vas a arrepentir.

La explicación

Como suele ser costumbre en cada programa que encaramos generalmente se nos presenta la oportunidad de reusar algún algoritmo, método o cosa que se le parezca. Básicamente como somos muuuuuuy vagos no vamos a ponernos a pensar de nuevo el mismo problema que ya resolvimos, pero como nos gustan los desafíos también tenemos la oportunidad de hacer que cada vez sea más fácil poder usar nuestros trabajos anteriores.

Una de esas oportunidades es, sin duda, el ordenamiento de datos y particularmente cuando se encuentran en un array (sí, acertaste, nuestros queridos @array).

Primero repasemos algunas características de los arrays :

- son una estructura de datos que, en otras palabras, es una forma de organizar la información
- constan de una cantidad de elementos que 1) son todos del mismo tipo y 2) están puestos bajo un mismo contenedor (el array en si)
- cada elemento se accede a través de un índice, que es un número entero que va entre dos valores máximo y mínimo.

Pero en Perl tienen ciertas características que los hacen versátiles :

- En Perl los arrays comienzan con el elemento cero ($array[0] es el primer elemento del array @array)
- Cuando se los define no se especifica el tamaño de la dimensión (cantidad de elementos que va a contener) sino que a medida que se van agregando o quitando elementos al mismo y este automágicamente va ajustando su tamaño (arrays dinámicos)

Ahora bien toda magia necesita tener un mago para lucirse (o era al revés ?) y en este caso un array por si mismo puede quedar muy bonito pero si no podemos operar con él entonces estamos en un problema (usamos memoria para almacenar algo que después no sabemos usar). Una de las bellezas de los arrays es que una vez que sabemos realizar una operación sobre uno de los elementos, entonces podemos actuar sobre cualquiera de los elementos en de la misma forma.

Por ejemplo, supongamos tenemos un array con 12 elementos y que cada elemento es nuestro gasto en cursos de Perl de los últimos doce meses. De repente recordamos que estos valores que almacenamos son sin impuestos, y para que no tengamos problemas de contabilidad es que necesitamos agregarle a cada mes un 21% de impuestos. Básicamente lo que hacemos es, al primer elemento multiplicarlo por 1.21, al segundo por 1.21 ... y así hasta el último :

```
$array[0] .= 1.21;
$array[1] .= 1.21;
...
$array[10] .= 1.21;
$array[11] .= 1.21;
```

Esto es fácil, gracias al copy & paste, si sabemos que la cantidad de elementos del array es definida y conocida de antemano. Pero también observando y generalizando podemos decir que :

```
# Para todo elemento $i del array :
$array[$i] .= 1.21;
```

Ajhá, entonces ... por qué no usar un loop (for, while o algo por el estilo) :

```
for $i ( 0 .. 11 ) {
    $array[$i] .= 1.21;
};
```

Supongamos que ahora queremos ordenar nuestro array de mayor a menor, y así saber que valores de dinero (máximo y mínimo) hemos gastado en nuestra educación en Perl. Una vez ordenado el array de mayor a menor, $array[0] contendrá el mayor gasto y $array[11] el menor (ahora con los impuestos incluidos ;-)). Para hacerlo, no hay más que llamar a nuestra querida sort(), indicándole que vamos a comparar números, y todo está listo !!!

```
@array2 = sort { $a <=> $b } @array;
```

Hasta acá todo muy bonito, porque cuando tenemos unos pocos elementos se usa poca memoria, hacer operaciones no se tarda mucho y ya. Pero si tenemos un array que tiene muchos elementos y nuestra función de comparación es un poco complicada (más tarde vamos a cuantificar que significa ?muchos elementos? y ?un poco complicada?, entonces todo se aclara un poco).

Básicamente cada vez que sort() tiene que comparar dos elementos llama a nuestra sub, lo que hace unas 46 veces para ordenar un array de 10 elementos. El número se eleva a 14.000 cuando son 1.000 elementos, así que imagínense como sigue creciendo esto para números más altos. Entonces si la sub de comparación comienza a complicarse más allá de una comparación, tenemos que ver cómo hacer para que este impacto sea lo menos posible.

El por qué de esto parece un poco en el aire, como traído de los pelos. Para esto vamos a analizar un algoritmo de ordenación. Uno de los más simples es el de selección, el que consiste en tomar el primer elemento de un array, compararlo con todos los demás elementos hasta que se encuentre con uno menor, entonces se intercambian ambos de lugar (swap) y se sigue así hasta el último elemento. De esta forma, luego de esta primera pasada, el primer elemento será el menor de todo el array. En una segunda pasada empezaríamos con el segundo elemento y lo comparamos con todos los restantes, quedando el primer elemento como el menor de todo el array y el segundo como el segundo menor. Si seguimos así, al hacer la pasada N-1 (donde N es la cantidad de elementos del array) tendremos todo el array ordenado de menor a mayor. Un ejemplo :

Array desordenado :
[40,21,4,9,10,35]
Primera pasada:
[21,40,4,9,10,35] <— se comparan el 40 y el 21 (se intercambian por ser 21 el menor)
[4,40,21,9,10,35] <— se comparan el 21 y el 4 (se intercambian por ser 4 el menor)
[4,40,21,9,10,35] <— se comparan el 4 y el 9
[4,40,21,9,10,35] <— se comparan el 4 y el 10
[4,40,21,9,10,35] <— se comparan el 4 y el 35

Segunda pasada:

[4,21,40,9,10,35] <— se comparan el 40 y el 21 (se intercambian por ser 21 el menor)
[4,9,40,21,10,35] <— se comparan el 21 y el 9 (se intercambian por ser 9 el menor)
[4,9,40,21,10,35] <— se comparan el 9 y el 10
[4,9,40,21,10,35] <— se comparan el 9 y el 35

Como vemos, luego de la segunda pasada tenemos que los dos primeros elementos (4 y 9) son los menores del array. Al finalizar la quinta pasada, el array quedará ordenado : [4,9,10,21,35,40].

En este punto ya podemos mostrar que en la primer pasada se hicieron 5 comparaciones, en la segunda 4, en la tercera 3, en la cuarta 2 y en la quinta 1 (quedan sólo dos elementos) con lo cual se han hecho 15 comparaciones en un array de 6 elementos. Ahora que queda un poco más claro el por qué se necesitan muchas más comparaciones que elementos tiene el array, también hay que aclarar que esto depende del método de ordenación que se utilice.

Volviendo a nuestro análisis del uso de sort() optimizado para velocidad, nuestra sub de comparación consta del cómputo de los distintos valores (para $a y $b y su comparación). Por ejemplo, si tenemos que comparar un array que tiene referencias a un hash y queremos ordenar por la longitud del apellido, entonces esto se convierte en :

```
sort { length ($a->{apellido}) <=> length ($b->{apellido}) ) } @data
```

y si además queremos agregar que si tienen igual longitud ordenar por el nro. de documento :

```
sort { (length ($a->{apellido}) <=> length ($b->{apellido}) )
        || ($a->{documento} <=> $b->{documento}) ) ) } @data
```

vemos que siempre se presenta una parte da calculo que puede ser tan simple como extraer un valor ($a-{documento}, $b-{documento}) o realizar un cálculo (length $a-{apellido}, length $b-{apellido}) y posteriormente se hace la comparación de estos valores. Lo cierto es que para cada comparación se hacen los cálculos cada vez que se necesitan, y este es uno de los puntos que nos consume tiempo en forma innecesaria, ya que una vez calculados estos valores podrían almacenarse y reutilizarse (veamos que si para ordenar 1.000 elementos se hacen 14.000 comparaciones, entonces eso significa que para todos lo elementos la porción del cálculo se hace al menos una vez durante la comparación). En nuestro caso lo que causa estos problemas es el cálculo de la longitud del apellido, con lo que podemos hacer un pre-cálculo y almacenarlo en un array para su posterior uso.

Para esta tarea nos valemos de una pequeña ayuda, que es la función map(). Esta nos permite hacer una iteración de todos los elementos de una lista, accediéndolos a través de la variable $_, operando sobre los mismos y devolviéndolos en un contexto de lista. Para obtener la longitud de todos los elementos de un array, y almacenarlos en otro array, tenemos :

```
@longitud = map { length $_ } @strings;
```

Ahora bien, volviendo a la etapa del pre-cálculo usando map(), esta nos queda :

```
@temp = map { [ length $_->{apellido}, $_] } @data;
```

Listo, ya tenemos en @temp no solo los elementos sino también los cálculos de las longitudes correspondientes, almacenados como una referencia a un array de dos elementos, de la siguiente forma :

```
...

$temp[$i]->[0] = length $array[$i]->{apellido};
                 ## valor pre-calculado en base al elemento $i
$temp[$i]->[1] = $array[$i];
                 ## valor del elemento $i
```

```
...
```

Ahora que llegamos a este punto simplemente nos queda hacer el sort() pero solamente haciendo la comparación de los valores pre-calculados (si calculamos de nuevo perdemos la ventaja y el tiempo usado en generar @temp) :

```
@temp2 = sort { ($a->[0] <=> $b->[0] )
                || ($a->[1]->{documento} <=> $b->[1]->{documento}) } @temp;
```

A no desesperar, falta un último paso, y es que ahora tenemos el array ordenado pero los elementos originales están en $temp[$i]-[1], con lo cual debemos desreferenciarlos :

```
@ordenados = map { $_->[1] } @temp2;
```

Pero bien, en vez de usar esta sucesión de arrays podemos utilizarlos en cascada (o pipe, como más te guste ;-)), aprovechando directamente la operación de una parte de la optimización como entrada de la otra :

```
@ordenados = map { $_->[1] }
             sort { ($a->[0] <=> $b->[0] )
                    || ($a->[1]->{documento} <=> $b->[1]->{documento}) }
             map { [ length $_->{apellido}, $_] } @data;
```

Ahora si, ya sabemos cómo deslumbrar y por qué lo hicimos pero ... todavía no cuantificamos cuándo es muchos elementos en un array y cuándo una evaluación es muy pesada como para utilizar este método en lugar del sort simple (vamos, que tenemos que justificar que todo este esfuerzo no fue en vano).

Antes que nada una aclaración para poner en contextos las pruebas de performance. Todos los scripts se ejecutaron en el siguiente equipo :

- Apple PowerBook con procesador PowerPC G3 de 400 MHz, 768 MB de RAM
- Sistema operativo Mac OS X 10.3.9
- Perl v5.8.1-RC3 built for darwin-thread-multi-2level
- módulo Benchmark v1.0501

Como siempre, y para estos menesteres, viene en nuestra ayuda un módulo de CPAN llamado Benchmark (). En principio nos permite ejecutar trozos de código en forma repetida (función timethese()) y su posterior análisis (función cmpthese()). Veamos el script básico usado para tratar de cuantificar este punto :

```
#!/usr/bin/perl

use strict;
use warnings;
use Benchmark qw(:all) ;

sub getArrayRandom($);

my $count = -4;     ## ejecuta el test durante 4 segundos para cada sub()
my $items = 10000;  ## No. of array elements

my (@strings, @sorted);

getArrayRandom( \@strings );

print "Ejecutando para ’$items’ items y durante ’". (-$count)."’segundos\n\n\n
```

```
my $results = timethese($count, {
                ’zeroOrder’ => sub {
                    @sorted = sort computeAndCompareStrSimple @strings;
                    },

            ’optimal’ => sub {
                    @sorted =       map { $_->[1] }
                                sort { $a->[0] <=> $b->[0] }
                                map { [ computeOptSimple($_) , $_ ] } @strings
                            },
                        });

cmpthese( $results ) ;

sub computeOptSimple($) {
    return length $_[0];
};

sub computeAndCompareStrSimple($$) {
    return ( (length $_[0]) <=> (length $_[1]));
};

sub getArrayRandom($) {
    my $ref = shift;

    for( my $i=0; $i < $items; $i++ ) {
        my ($string, $len) = ( "", int( rand(100) ) );
        $string .= chr( int( rand(254) ) )
        while ( (length $string) < $len );
            push @$ref, $string;
    };
 };
```

En este caso vamos a comparar dos códigos llamados ‘zeroOrder’ y ‘optimal’ correspondientes a el sort simple/tradicional y la optimizada (descripta en este artículo) respectivamente. Al ejecutarlo (observar que se hace una comparación simple de con 4 segundos de CPU cada uno y un array de 10.000 items, obtenemos los siguientes números :

```
Ejecutando para ’10000’ items y durante ’4’ segundos

Benchmark: running optimal, zeroOrder for at least 4 CPU seconds...

    optimal: 13 wallclock secs ( 4.23 usr + 0.07 sys = 4.30 CPU) @ 1.16/s (n=5
    zeroOrder : 12 wallclock secs ( 4.17 usr + 0.04 sys = 4.21 CPU) @ 2.38/s (

                        Rate    optimal zeroOrder
    optimal         1.16/s    --      -51%
    zeroOrder       2.38/s   104%      --
```

Mmmmmm ... malo, malo ... el código sin optimizar corre más rápido que el optimizado, con lo cual seguramente una de las cosas que puede estar pasando es que estamos gastando demasiado en usar un sistema más indirecto y que consume más recursos en generar un ahorro que el ahorro mismo (hay que notar que si bien ahorro tiempo pre-calculando también se consume tiempo en realizar dos map() extras, y estos últimos podrían llegar a consumir demasiado) .

Bueno, vayamos un poco más lejos y subamos la complejidad de la comparación, usando una función que consuma un poco más de recursos, extrayendo una cifra del elemento del string (ver que los elementos están generados al azar) y comparándolos. Si analizamos la implementación de

computeAndCompareStr() y computeOpt() vamos a ver que está hecha con expresiones regulares que en principio aceptemos son un punto importante de discusión al momento de la performance (en general consumidoras de CPU) :

```
#!/usr/bin/perl

use strict;
use warnings;
use Benchmark qw(:all) ;

sub getArrayRandom($);

my $count = -8;     ## ejecuta el test durante 8 segundos para cada sub()
my $items = 10000;  ## No. of array elements

my (@strings, @sorted);

getArrayRandom( \@strings );

print "Ejecutando para '$items' items y durante '". (-$count)."' segundos\n\n\

my $results = timethese($count, {
            'zeroOrder' => sub {
            @sorted = sort computeAndCompareStr @strings;
        },

        'optimal' => sub {
            @sorted = map { $_->[1] }
                      sort { $a->[0] <=> $b->[0] }
                      map { [ computeOpt($_) , $_ ] } @strings;
        },
        });

        cmpthese( $results ) ;

sub computeOpt($) {
    $_[0] =~ /(\d+)/;
    return $1 || 0;
};

sub computeAndCompareStr($$) {
    $_[0] =~ /(\d+)/;
    my $x = $1 || 0;
    $_[1] =~ /(\d+)/;
    return ($x <=> ($1 || 0));
    };

sub getArrayRandom($) {
    my $ref = shift;

    for( my $i=0; $i < $items; $i++ ) {
        my ($string, $len) = ( "", int( rand(100) ) );
        $string .= chr( int( rand(254) ) )
           while ( (length $string) < $len );
        push @$ref, $string;
    };
};
```

Y al ejecutarlo obtenemos :

Ejecutando para ‘10000’ items y durante ‘8’ segundos

Benchmark: running optimal, zeroOrder for at least 8 CPU seconds...

```
        optimal: 26 wallclock secs ( 8.34 usr + 0.17 sys = 8.51 CPU) @ 1.18/s (n=1
        zeroOrder: 26 wallclock secs ( 8.60 usr + 0.10 sys = 8.70 CPU) @ 0.57/s (n

                    s/iter  zeroOrder optimal
        zeroOrder     1.74        --      -51%
        optimal      0.851      104%       --
```

Ahora si, esto va tomando color no es cierto ?? Finalmente no pude sucumbir y decidí darle una vuelta de rosca adicional para ver cuál es el límite para decidir qué son muchos elementos, y empezar a disminuir la cantidad. La verdad es que aún con 10 elementos y la última búsqueda (compleja) el método mejorado resultó un 20% mejor que el método directo.

Conclusiones

Si en la ordenación de una array, la comparación de los elementos conlleva operaciones típicamente consumidoras de recursos (en su mayoría CPU) es conveniente usar el método indirecto aún con pocos elementos (aún con 10 podría ser efectivo). En cambio si se trata de comparaciones simples podría bastar con la forma simple, aún para 10.000 elementos, según las comparaciones. Si se trata de comparaciones simples con pocos elementos, olvídalo, no hay problemas a resolver.

Infografía

- Métodos de ordenamiento : *http://algoritmia.net/articles.php?id=31*
- Sorting a list by a computable field (Perl CookBook) : *http://www.unix.org.ua/orelly/perl/cookbook/ch04_16.htm*
- Benchmark - benchmark running times of Perl code : *http://search.cpan.org/~nwclark/perl-5.8.6/lib/Benchmark.pm*
- Perl builtin functions : *http://search.cpan.org/~nwclark/perl/pod/perlfunc.pod*

CaFe Perl en Perl

Y sigue el avance ... aunque se ha convertido casi en una cuestión puramente personal, de a poco todo va tomando su color. En este mes el enfoque fue hacia la facilidad de edición. Para esto la meta era poder usar un editor de texto con capacidades avanzadas (al estilo de Koffice, OpenOffice.org o MS Word) y a partir de ahí generar el formato POD, en lugar de trabajar sobre POD directamente que, convengamos, se empieza a hacer tedioso si se usa más allá de los comentarios en un programa.

La meta fue cumplida parcialmente, hoy en día esta es la primer edición editada con OpenOffice.org, y convertida a formato POD usando el programa myoo2pod.pl. Este último no es más ni menos que una mejora del ejemplo oo2pod que viene incluido en el módulo OpenOffice::OODoc, pero que aún le falta implementar, básicamente, la traducción de los formatting codes (bold, italic, links, etc.) que son agregados en una tarea de post-edición manual sobre el formato POD.

Como de costumbre si quieren saber un poco más de qué se trata, colaborar con esta mejora en la edición a través de la programación, o lo que sea pueden contactarme a victor {at} bit-man {dot} com {dot} ar o acceder a la página web de esta mini-cruzada :
http://www.bit-man.com.ar/es/CaFePMenPerl