



Periódico de la Comunidad Perl de Capital Federal

<http://cafe.pm.org/boletines/>

Cafe Perl v0.4

Table of Contents

Cafe Perl v0.4

CaFe Perl v0.4 - Periódico de la Comunidad Perl de Capital Federal	1
Editorial	1
PERlitas	1
SudorNews	3
System Administrator Appreciation Day	3
Búsqueda de DBA para PostgreSQL	3
Editor de gráficos vectoriales con interfaz Perl	3
Mejores prácticas en Perl	3
Learning Perl (4ta. Edición)	3
Proyecto opensource de BBC	3
Trabajador de Perl en juicio	4
Parrot 0.2.2 "Geeksunite" Released	4
Ponie Snapshot 4 Released	4
Pugs 6.2.8 y más ...	4
Es Perl aún relevante ??	4
III Barcelona Perl Festival	4
5ta. Jornada Regionales de Soft Libre	4
Maypole gana el premio Linux Journal Editor's Choice Award	4
Ten Roses busca cubrir posiciones	4
- Ingeniería de Sistemas	4
- Licenciatura en Sistemas	4
- Ciencias de la Computación.	4
Referencias (Mordiditas de aquí y de allá)	5
Introducción	5
El problema y la receta	5
La explicación	5
Una referencia es un escalar, pero un escalar no es una...	5
La caja que todo lo esconde	9
- vamos a hacer que las subs generadas sean anónimas, a...	12
- sólo vamos a exportar new, ya que el resto de las subs...	12
- las subs que implementan el código quedan sin...	12
Una mirada más allá	14
Infografía	14
- Perl builtin functions	14
- Perl references and nested data structures	14
- Mark's very short tutorial about references	14
- Object::MultiType - Perl Objects as Hash, Array, Scalar, Code...	14
POC (peace of code)	14
- no se puede utilizar bless ni tie ni puede ser un objeto	15
- no se pueden usar pseudohashes	15
- no puede haber más de un ; en la línea	15
- la solución debe tener 35 caracteres o menos	15
- no debe generar warnings ni errores, usando siempre "use..."	15
- no se puede agregar más código que el de esa...	15
- no se pueden usar módulos de ningún tipo	15

CaFe Perl v0.4 - Periódico de la Comunidad Perl de Capital Federal

Editorial

Hola colisteros. Llegó un nuevo mes, y con este el nuevo número de CaFe Perl.

Antes que nada quiero enviar un saludo especial a todos los sysadmins, ya que este pasado 29 de Julio fue el "System Administrator Appreciation Day" (hagan ustedes su propia traducción). Esta conmemoración se lleva a cabo el último día Viernes de Julio de cada año.

En otro orden este mes la novedad es que se agrandó el staff, porque tenemos un nuevo miembro (Andrés Kievsky) que se empieza a hacer cargo de la sección POC (Peace of Code) !!!

Sólo para fijar expectativas esta sección va a traer mensualmente un acertijo Perl para que, los que quieran resolverlo y mostrar sus habilidades, le envíen un e-mail a Andrés.

La más cálida de las bienvenidas para él, y estoy seguro que esta sección se va a convertir en un favorito de la publicación.

En el resto tenemos una entrevista al creador de un programa de automatización del hogar llamado MisterHouse y una serie de trucos con las referencias. Algo delicioso !!!

Espero que lo disfruten.

Hasta la próxima taza de CaFe Perl !!! ... eso sí, café del bueno ;-).

Víctor A. Rodríguez (Bit-Man)

PERlitas

Entrevista realizada y traducida al español por Víctor A. Rodríguez

MisterHouse (<http://misterhouse.sourceforge.net/>) es un programa para automatización del hogar escrito en Perl. Es divertido, free y completamente orientado a geeks. Escrito en Perl, genera eventos basados datos relacionados temporalmente, con web, provenientes de un socket o de una interfaz serial. Actualmente corre sobre Windows 95/98/NT/2k/XP y en la mayoría de las plataformas Unix, incluyendo Linux y Mac OS X.

Por favor Bruce, una introducción al grupo CaFe.pm

Soy un Ingeniero Electrónico de 46 años que vive en Rochester, Minnesota, USA y trabajo en la División de Servicios de Ingeniería de IBM. Nuestro grupo diseña circuitos integrados para una amplia variedad de empresas, como las CPUs para la supercomputadora BlueGene y las tres consolas de juegos de próxima generación.

Cuál fue su motivación para construir MisterHouse ??

Mi casa es de esas diseñadas para hacer uso eficiente de la energía y ser calefaccionadas con energía solar pasiva proveniente de las ventanas orientadas hacia el Sur [N. del T.: en el hemisferio Norte el Ecuador está hacia el Sur, de donde provienen los vientos cálidos, al contrario que en el hemisferio Norte]. Para administrar ese calor, instalé un sistema casero de cortinas controladas por motor, para poder abrirlas en esos días frescos pero soleados de invierno y cerrarlos de noche o cuando está nublado.

Necesitaba un software para hacer un monitoreo de la temperatura interior/exterior y el nivel de luz solar, y controlar las cortinas, la calefacción y los ventiladores en consecuencia. Una vez que tuve a MisterHouse haciendo estas tareas, lo extendí a otras tareas, menos prácticas pero divertidas, como el controlar o hablar a través de un par de robots.

Por qué se dice que está "completamente orientado a geeks" ?

Que tu casa te recuerde que tenés que regar las plantas o dónde está tu auto no es para todo el mundo. Se da mejor con los geeks. No es sólo que son el tipo de personas que podrían disfrutar esa clase de

interacción, es que también es necesario tener ciertas habilidades en computación para que funcione bien.

Y su familia, aceptó la idea fácilmente de lidiar con un sistema automatizado ?? Cómo lo manejó ??

Les gusta la mayormente. El bug ocasional que causa que suene Mozart a las 2 de la mañana no es muy popular, pero cosas como que se anuncie quien llama cuando el teléfono suena ayudan.

Qué habilidades (relacionadas con Perl o no) obtuvo mientras construyó MisterHouse ??

Uso algo de Perl en mi trabajo de diseño de chips en IBM. No uso nada de ese código en MisterHouse, pero uso en el trabajo muchos trucos de Perl que aprendí mientras escribía MisterHouse y viceversa.

Qué dispositivos agregó primero y por qué ?

Como con la mayoría de los programas de automatización del hogar, comenzamos soportando dispositivos X10, ya que son los más comunes, simple y económicos. Después de eso agregamos soporte para entradas y salidas digitales y entradas analógicas para elementos como monitores de puertas, control de hornos y monitoreo de temperatura.

Que le sugeriría a los futuros diseñadores y hobistas que se enfrentan a un nuevo proyecto ?

Háganlo open source. En el caso de automatización del hogar, no hay forma que una sola persona, o aún una pequeña compañía, puedan abarcar todas las posibles aplicaciones, así que es un lugar perfecto para el open source. Otros contribuyen escribiendo código para cubrir sus necesidades, que es compartida con otros.

En qué partes del código sugeriría a los codificadores novatos de Perl que miren, para poder tener una experiencia educativa placentera ?

Mucha gente aprende Perl para poder ejecutar MisterHouse. Hay algunos pocos que corren MisterHouse como una forma divertida de aprender Perl para otros usos.

Tenemos una variedad de código aquí : <http://misterhouse.net/examples.html>

Cómo se puede colaborar con MisterHouse ??

Tenemos más de 600 miembros en la lista de e-mail, con casi 300 de ellos listados como autores habiendo contribuido en alguna forma. Algunos simplemente envían sugerencias o encuentran bugs, otros envían patches para arreglar bugs o extender la funcionalidad, y algunos escriben nuevas aplicaciones. Yo tomo los cambios a través de la lista de e-mail, o me los envían directamente a mi, o a través del CVS de SourceForge.

Qué funcionalidades cree que faltan, y cuáles agregará pronto ??

El cáliz sagrado de la automatización del hogar es el reconocimiento de la voz en forma confiable y conveniente. Este con un buen micrófono de cabeza y en un cuarto silencioso trabaja bastante bien, pero no muchos de nosotros les gustaría andar con ellos por la casa.

Siendo un Ingeniero Electrónico, cómo cree que Perl entra en este campo. Y el open source ??

Muchos de los problemas que enfrentamos en el diseño de chips son nuevos o únicos a un chip específico, así que no siempre tenemos una solución existente que podamos usar. Normalmente codificamos scripts Perl para descartar soluciones rápidamente. Estos programas son muy específicos a nuestro proceso de diseño o chip, así que no son buenos candidatos para compartir como open source.

Algún módulo de CPAN favorito ??

Ese sería el módulo SerialPort de Bill Birthisel's. Es el que nos permite hablar con los dispositivos seriales, en Unix y Windows.

Qué usos de MisterHouse le sorprendieron ??

Algunos ejemplos de uso: MisterCar, MisterBus, MisterAquarium, MisterDoorRoom. Un grupo en el laboratorio Fermi lo usa como MisterLab, controlando algunos de sus displays del cuarto de control y dando alertas cuando sus monitores de anti-materia muestran problemas. Algo así como un Scotty computarizado de Star Trek !

Recibió algún feedback de la industria (comentarios, propuestas de negocio, etc.) ??

No demasiado. Lo disfruto como un hobby, así que no tengo un interés en esto cómo un negocio.

Tiene alguna experiencia, divertida o no, que tuvo mientras construía MisterHouse y que quiera compartir ??

Un momento de orgullo para los autores open source es cuando hacen su primer anuncio público anunciando que su software está disponible. Cuando lo envié a la lista comp.home.automation de UseNet, no revisé muy cuidadosamente los resultados de mi chequeador de sintaxis, así que no noté que cambió MisterHouse por mysterious. Mysterious House [N. del T.: "la casa misteriosa"] no es lo que tenía en mente !

Algo más que quiera decirnos y no le preguntamos ??

No se necesita una casa para usar MisterHouse. Tenemos personas usándolo en departamentos, dormitorios, peceras, autos, y en laboratorios nacionales de ciencia !

SudorNews

System Administrator Appreciation Day

Esta conmemoración se lleva a cabo el último día Viernes de Julio de cada año (<http://www.sysadminday.com/>), y tiene por objeto el recordar que quienes mantienen funcionando nuestros sistemas al más mínimo detalle. A todos gracias !!!

Búsqueda de DBA para PostgreSQL

Trabajo part time para una empresa reconocida de informática se necesita un DBA PostgreSQL. Enviar currículum a José Dragone (josedragone@hotmail.com)

Editor de gráficos vectoriales con interfaz Perl

Inkscape (<http://www.inkscape.org/>) es un editor que como novedad, y al estilo de The Gimp, posee interfaz para ser accedido desde Perl, Ruby y Python. Todo un ejemplo.

Mejores prácticas en Perl

Un libro que promete (<http://www.oreilly.com/catalog/perlbp/>) escrito por el ya conocido Damian Conway (ooh, un libro de Perl editado por O'Reilly que no tiene un camello en la tapa !!!). A juzgar por el artículo "Ten Essential Development Practices" (<http://www.perl.com/pub/a/2005/07/14/bestpractices.html>) pretende disponer una guía para mejorar nuestra codificación. Apto y aplicable no sólo para Perl.

Learning Perl (4ta. Edición)

Salió la nueva edición del ya conocido "Llama book" (<http://www.oreilly.com/catalog/learnperl4/>). Una joya renovada para reflejar las versiones 5.8 de Perl. Para los que no pueden esperar, hay un capítulo de ejemplo (<http://www.oreilly.com/catalog/learnperl4/chapter/ch11.pdf>)

Proyecto opensource de BBC

La BBC (British Broadcasting Corporation) está haciendo disponible una cantidad de código como open source (<http://www.bbc.co.uk/opensource/>) bajo una serie de conocidas licencias (GPL, LGPL, BSD, etc.). Como no podía estar ausente nuestro amigo CPAN también hay disponibles módulos de Perl

(<http://www.bbc.co.uk/opensource/projects/cpan/>).

Trabajador de Perl en juicio

Chip Salzenberg, pumpking para perl 5.004, tuvo problemas con su ex- empleador y su casa fue cateada y todo su equipo informático incautado en Abril. Para que puedan seguir toda esta problemática esta persona creo un site, y de paso aprender los qué hacer y que no (<http://geeksunite.net/>).

Parrot 0.2.2 "Geeksunite" Released

Parrot es la máquina virtual sobre la que correrá Perl 6 (<http://www.parrotcode.org/>), y también servirá de host a otros lenguajes como Ruby y Python. Se encuentra disponible la versión 0.2.2 "Geeksunite" (<ftp://ftp.cpan.org/pub/CPAN/authors/id/L/LT/LTOETSCH/parrot-0.2.2.tar.gz>).

Ponie Snapshot 4 Released

Ponie, destinado a ejecutar Perl 5 sobre Parrot, posee ya su cuarto snapshot disponible (<http://opensource.fotango.com/~nclark/ponie-4.tar.bz2/> / <http://opensource.fotango.com/~nclark/ponie-4.tar.gz>).

Pugs 6.2.8 y más ...

Para los que quieren experimentar con Perl 6 ya salió la nueva versión de la implementación de Perl 6 sobre Haskell (<http://pugscode.org/dist/Perl6-Pugs-6.2.8.tar.gz>). También puede encontrarse un live CD para los que no quieren instalarlo y sólo ver de qué se trata (<http://linide.sf.net/pugs-livecd-6.2.8.iso>). Principalmente esta versión es un punto de inflexión ya que en lugar de ser una prueba conceptual del mismo está mudando su foco, tendiendo a ser una suite de compilación para Perl 6.

Es Perl aún relevante ??

Esta vez es Tim O'Reilly quien toma la posta (http://www.oreillynet.com/pub/a/oreilly/ask_tim/2004/perl_0707.html) y si bien Perl ha perdido parte de su popularidad hay dudas sobre si Perl 6 traerá nueva vida o no. Lo cierto es que nunca puede darse por descontado, básicamente por el apoyo de la comunidad y que siempre Perl se caracterizó por acercar ciertas implementaciones que en otros lenguajes parecían lejanas o casi imposibles.

III Barcelona Perl Festival

El 14 de Julio se realizó este festival (<http://barcelona.pm.org/events/festival3/index.html>) dedicado, esta vez, a mod_perl 2.0. Stas Bekman fue el invitado de lujo.

5ta. Jornada Regionales de Soft Libre

A realizarse en Rosario entre los días 20 y 23 de Noviembre (<http://www.ant.org.ar/jornadas/html>). De entrada libre y gratuita, se hará en las instalaciones del centro ferial más importante de la ciudad, y se espera una gran afluencia de público.

Maypole gana el premio Linux Journal Editor's Choice Award

Maypole (<http://maypole.perl.org/>) es un framework hecho en Perl para aplicaciones web orientadas al modelo MVC (Model-View-Controller). Diseñado para minimizar los requerimientos de codificación, ha ganado este premio de la conocida Linux Journal (<http://www.linuxjournal.com/>).

Ten Roses busca cubrir posiciones

Estudiantes o graduados recientemente de las siguientes carreras:

- - Ingeniería de Sistemas
- - Licenciatura en Sistemas
- - Ciencias de la Computación.

Lugar de trabajo : 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.

Interesados enviar currículum vitae a la brevedad a hr@tenroses.com.ar , indicando como referente a Walter Lamagna y la referencia del puesto a cubrir :

Ref: Java Developer Junior

Excluyente:

Conocimientos de Java u otro lenguaje orientado a objetos

Conocimientos básicos en SQL

Preferente:

Conocimientos de idioma inglés técnico

Conocimientos de HTML

Ref: Java Developer

Excluyente:

Experiencia de 2 años al menos en desarrollo de aplicaciones Web, en lenguaje Java o C++.

Conocimientos de SQL

Preferente:

Conocimientos de XML/XSL

Conocimientos de idioma inglés técnico

Conocimientos de GUI en Java (AWT/SWING/Applets)

Ref: Web Developer

Excluyente:

Experiencia en HTML, CSS, Javascript.

Conocimientos de lenguajes de HTML embebido: JSP, PHP, ASP

Preferente:

Conocimientos de XML/XSL

Conocimientos de GUI en Java (AWT/SWING/Applets)

Conocimientos de idioma inglés técnico (lecto escritura)

Referencias (Mordiditas de aquí y de allá)

realizado por Víctor A. Rodríguez

Introducción

Una vez más frente a la hoja en blanco me invade esa rara sensación, ese miedo de no saber dónde poner el pie para dar el próximo paso y eso me recuerda a cuando descubrí las referencias, y por primera vez sentí esa sensación de no saber que había detrás de todo ese manejo que, en cierta forma, simulaba a los punteros. La verdad es que no la sentí cuando vi y usé punteros en C por primera vez, porque en realidad está más ligado al hecho físico de la ubicación de una variable en una porción de memoria, y en Perl no es sólo eso (vamos a verlo en la medida que avancemos e investiguemos a las referencias).

Para este artículo es necesario conocer qué son las referencias y leer los párrafos de perldoc perlref y perldoc perlreftut citados en cada una de las partes de esta sección.

El problema y la receta

Esta vez no hay problema y receta, simplemente curiosidad. O mejor digámoslo así el problema era tratar de descubrir aspectos distintos de las referencias, y la receta era leer documentación pertinente :) y dejarse llevar por la imaginación. Lo que van a leer es el resultado de esta experiencia. Juzguen por ustedes mismos.

La explicación

Que explicación dar a la vida o al arte, simplemente son manifestaciones y las disfrutamos como tales. Se pueden llegar a comprender, pero lo importante es poder disfrutarlas. Lo mismo va con esta parte de la sección, porque algunas de las cosas simplemente se me ocurrieron y, en general, no funcionaron pero lo bueno es que sirvieron para conocer un poco más y perderle un poco de miedo a las referencias.

Una referencia es un escalár, pero un escalár no es una referencia

Según reza la biblia de todas nuestras desdichas ("El nuevo testamento" a.k.a. perldoc perlreftut) :

"A reference is a scalar value that refers to an entire array or an entire hash (or to just about anything else)".

Bravo !! entonces podemos manejar una referencia como un escalar y convertirlo en una referencia a lo que se nos de la gana. Puntualizando vamos a tratar de convertir una referencia a un hash en una referencia a un array (o al revés).

Si generamos una referencia, entonces en realidad estamos generando un escalar (de hecho lo almacenamos en uno) entonces podemos manipularlo como tal. Veamos el siguiente código :

```
use warnings;
use strict;

my @data = qw/Nombre Victor Edad 39/;
my $refArray = \@data;

print "Segundo elemento : " . $refArray->[1] . "\n";
print $refArray . "\n";]
```

Esto nos da la siguiente salida :

```
Segundo elemento : Victor
ARRAY(0x809f18)
```

Lo que nos muestra lo que ya sabíamos, pero que además (esto está hecho a propósito) el array tiene una cierta forma de hash (las claves podrían ser 'Nombre' y 'Edad' y sus contenidos Victor y 39), con lo que podemos copiar el array sobre un hash y usarlo como tal agregando el siguiente código al anterior :

```
my %hash = @data;
my $refHash = \%hash;

print "Nombre : " . $refHash->{'Nombre'} . "\n";
print $refHash . "\n";
```

nos da a la salida :

```
Segundo elemento : Victor
ARRAY(0x809f18)
Nombre : Victor
HASH(0x80f3d4)
```

El problema con este método de convertir un array en un hash es que hay que copiar todo el contenido de uno en el otro, y en caso de estructuras grandes esto se convierte en un problema de performance, ya sea por la memoria duplicada o por el tiempo necesario para hacer la copia (o por ambos en el peor de los casos), así que trataremos de acceder el array directamente como si fuera un hash :

```
use warnings;
use strict;

my @data = qw/Nombre Victor Edad 39/;
my $refArray = \@data;

print "Nombre (como array): " . $refArray->{'Nombre'} . "\n";
```

lo que nos da como salida :

Can't coerce array into hash at test01.pl line 7.

Lo cual suena lógico, ya que si el tipo de referencia es a un array y lo queremos acceder como hash, el mecanismo de acceso nos lo hace notar. Y ahora el cuarto paso (y definitivo) es que si miramos a ambas referencias vemos que están compuestas por un indicativo del tipo de referencia destino (HASH o ARRAY) seguido de un número que es la dirección en memoria que tiene el mismo, con lo cual si hacemos caso que una referencia en realidad es un escalar y lo manejamos como tal, podremos manipular esa referencia (como un escalar) para convertirlo de ARRAY a HASH :

```
use warnings;
use strict;

my @data = qw/Nombre Victor Edad 39/;
my $refArray = \@data;

print $refArray . "\n";
$refArray =~ s/ARRAY/HASH/;
print $refArray . "\n"
print "Nombre (como array): " . $refArray->{'Nombre'} . "\n";
```

lo que nos da como salida :

```
ARRAY(0x809f18)
HASH(0x809f18)
Can't use string ("HASH(0x809f18)") as a HASH ref while "strict refs" in use at t
```

Cómo ?? qué es esta rebelión ?? Acaso las referencias no son un escalar que podemos manejar como escalares ?? Bueno, parece que no es así porque luego de manejarla como un string y querer volver a usarla como una referencia, entonces nos dice que no puede usar un string como referencia, con lo cual parece haber una cierta conversión implícita y no especificada.

Para ayudarnos a ver un poco dentro de los distintos tipos de datos, vamos a recurrir a nuestro viejo amigo CPAN. Hay un módulo en particular que nos puede ayudar y se llama Devel::Peek. Básicamente lo que vamos a hacer es mirar dentro de \$refArray antes y después de la transformación :

```
use warnings;
use strict;
use Devel::Peek 'Dump';

my @data = qw/Nombre Victor Edad 39/;
my $refArray = \@data;

Dump $refArray;
$refArray =~ s/ARRAY/HASH/;
print "-" x 20
Dump $refArray;
print "-" x 20
print $refArray;
```

que nos da como salida :

```
SV = RV(0x80e240) at 0x809f90
  REFCNT = 1
  FLAGS = (PADBUSY,PADMY,ROK)
  RV = 0x809f18
  SV = PVAV(0x80263c) at 0x809f18
    REFCNT = 2
```

```

    FLAGS = (PADBUSY,PADMY)
    IV = 0
    NV = 0
    ARRAY = 0x1019a0
    FILL = 3
    MAX = 3
    ARYLEN = 0x0
    FLAGS = (REAL)
    Elt No. 0
    SV = PV(0x801460) at 0x801180
        REFCNT = 1
        FLAGS = (POK,pPOK)
        PV = 0x100ca0 "Nombre"\0
        CUR = 6
        LEN = 7
    Elt No. 1
    SV = PV(0x801484) at 0x801234
        REFCNT = 1
        FLAGS = (POK,pPOK)
        PV = 0x101bc0 "Victor"\0
        CUR = 6
        LEN = 7
    Elt No. 2
    SV = PV(0x801508) at 0x801288
        REFCNT = 1
        FLAGS = (POK,pPOK)
        PV = 0x105160 "Edad"\0
        CUR = 4
        LEN = 5
    Elt No. 3
    SV = PV(0x801448) at 0x8012b8
        REFCNT = 1
        FLAGS = (POK,pPOK)
        PV = 0x105170 "39"\0
        CUR = 2
        LEN = 3
-----
    SV = PVIV(0x801820) at 0x809f90
    REFCNT = 1
    FLAGS = (PADBUSY,PADMY,POK,OOK,pPOK)
    IV = 1 (OFFSET)
    PV = 0x1050f1 ( "A" . ) "HASH(0x809f18)"\0
    CUR = 14
    LEN = 15
-----
    HASH(0x809f18)

```

Pero si está clarísimo , no ?? Bueno, para mi tampoco así que empecemos a desenmarañarlo. En principio las dos son SV (primera línea de ambas) lo que nos indica que son escalares (nada nuevo), en tanto que antes del cambio se trata de un array (ARRAY = 0x1019a0) compuesto de 4 elementos (MAX = 3) y que nos son mostrados como Elt. No. del 0 al 3, y después del cambio se convirtió en un escalar hecho y derecho sin nada de referencias (el flag POK nos indica que es un string), además de poder ver el contenido en PV. Curiosamente van a ver que hay una "A" delante del HASH (que no es mostrada al hacer el print) y que es la que corresponde al texto "ARRAY" que existía antes del reemplazo. La leyenda OFFSET en IV nos indica que en realidad hay que leer el contenido a partir del primer caracter (o byte ??).

Resultado, si bien las referencias se comportan como escalares cuando las accedemos en ese contexto, pierden su propiedad de referencias y se transforman en un simple escalar cuando las modificamos con herramientas para este tipo de datos. Cuando las escribimos con herramientas que denotan referencias, entonces mantienen su esencia.

Un buen ejemplo de este comportamiento un tanto esquizoide se puede ver en `Object::MultiType`, donde según el contexto algo puede ser visto como un array, hash, escalar, código o un glob, pero lo interesante es que no son todas distintas vistas de un único elemento (u objeto) sino que según se lo invoque el mismo es equivalente a distintos tipos de variables.

La caja que todo lo esconde

Muchas veces estamos acostumbrados a que cuando necesitamos usar cierto paquete de software (library, server, etc.), y a través de la disponibilidad del código, nos es posible descubrir el uso de ciertas variables o de ciertas rutinas del código de una forma no descrita por la documentación, e incluso a veces no pensada. En particular uno de los puntos fuertes del open source puede hacer que a la hora de ser elegido, y manejado de esta forma, se convierta en una debilidad. Por qué ?? básicamente porque si bien esta forma de trabajo nos deja sacar una ventaja, también nos genera un problema al momento de hacer un cambio de versión (upgrade) del paquete de software, porque es posible que esas funcionalidades ocultas (que no son otra cosa que nuestros amados hacks) hayan desaparecido. Después de todo no hay nada dicho en la documentación sobre su existencia, y por lo tanto no necesariamente uno puede pensar en que van a seguir siendo mantenidas ad infinitum (si no está estipulado en el contrato, entonces no se puede reclamar).

Tomemos un ejemplo simple para ilustrar este problema. Supongamos que hay un módulo el cual deseamos sea usada a través de un API (una serie de reglas), en el que se generan ciertos datos y estos son accesibles a través de ciertas subs o funciones.

Supongamos que realizamos un software para su uso en seguridad, donde se almacenan datos de usuario (en particular user y password) de tal forma que sólo se pueda obtener el nombre del usuario pero no su password. Una forma de implementación sería a través de un package con los datos puestos como privados del mismo package :

```
package myData_0;

use Exporter;
@EXPORT = qw(getUser);
@ISA = qw(Exporter);

my $data = { 'user' => 'bit-man',
            'password' => 'chupitanga2005' };

sub getUser() {
    return $data->{'user'};
}

1;
```

se generan una sub llamadas y `getUser()` para que esta pueda acceder al nombre de usuario. Un ejemplo de uso sería :

```
use warnings;
use strict;

use myData_0;

print "El usuario es : " . getUser() . "\n";
```

Ahora, sabiendo que los datos se almacenan en un hash referenciado por \$data podríamos intentar acceder al dato 'password' para poder, al menos, leerlo :

```
use warnings;
use strict;

use myData_0;

print "La password es : " . $myData_0::data->{'password'} . "\n";
```

con lo que obtenemos :

```
Name "myData_0::data" used only once: possible typo at test.Data_0.pl line 6.
Use of uninitialized value in concatenation (.) or string at test.Data_0.pl line 6.
La password es :
```

Como era previsible, al ser \$data un escalar válido dentro del scope del package myData_0, entonces los datos no están disponibles. El problema que tiene esta implementación es que si queremos tener más de un juego de datos, por ejemplo porque se trata de parte del código de un daemon que atiende varios requerimientos simultáneos, entonces necesitamos poder hacer que cada uno tenga su juego de datos para, por ejemplo, cambiar la password. En este caso deberíamos generar un juego de datos para cada nueva invocación que se haga, pero para evitar un acceso directo a los datos (como en el caso anterior) mantendremos el mismo hash dentro del scope del package, pero por cada nuevo juego de datos entregaremos un nuevo identificador (o index) para ubicar los mismos. Un código que podría implementar sería :

```
package myData_2;

use Exporter;
@EXPORT = qw(new getUser setPasswd);
@ISA = qw(Exporter);

my $data;

my $nextInstance = 1;

sub new {
    my $index = $nextInstance++;

    my $access = { 'index' => $index };
    ## No se entrega una referencia a los datos sino un identificador

    ## ... y un nuevo juego de datos por cada invocación de new()
    $data->{ $index } = { 'user' => 'bit-man' . $index,
                        'password' => 'chupitanga2005' };

    return $access;
};

sub getUser($) {
    my $access = shift;
    my $index = $access->{'index'};

    return $data->{ $index }->{'user'};
};

sub setPasswd($$$) {
    my $access = shift;
```

```

my $oldPasswd = shift;
my $newPasswd = shift;
my $index = $access->{'index'};

return 1
    if ( $data->{ $index }->{'password'} ne $oldPasswd );

    $data->{ $index }->{'password'} = $newPasswd;
    return 0;
}

1;

```

Como vemos, este package me permite ver el usuario (ahora se genera uno distinto por cada nuevo paquete de datos) y cambiar la password siempre y cuando conozca la password anterior :

```

use warnings;
use strict;

use myData_2;

print "----- Generación de datos -----\n";
my $data = new();
print "Primer user : " . getUser($data) . "\n";
my $data2 = new();
print "Segundo user : " . getUser($data2) . "\n";

print "----- Cambio de password -----\n";

my $error = setPasswd( $data,
                      'chupitanga2005', ## password vieja
                      'mypasswordeslargaperomala');
die ("No puedo cambiar la password")
    if $error;

$error = setPasswd( $data2,
                  'chupitanga2005', ## password vieja
                  'mypasswordeslargaperomala');
die ("ERROR : No puedo cambiar la password (1) !!!!!!!")
    if $error;

$error = setPasswd( $data2,
                  'mypasswordeslargaperomala', ## password vieja
                  'nuevapasswordJA');
die ("ERROR : No puedo cambiar la password (2) !!!!!!!")
    if $error;

print "OK !!!\n";

```

Simplemente crea dos juegos de datos, muestra los nombres de usuarios generados (para ver la independencia de los datos) y luego cambio la password de ambos juegos. Si se genera algún error en esta segunda fase mostraría que hay algo que no funciona en la independencia de datos, pero no es así. La ejecución arroja el siguiente resultado :

```

----- Generación de datos -----
Primer user : bit-man1
Segundo user : bit-man2

```

```
----- Cambio de password -----
OK !!!
```

El inconveniente que tiene este sistema es que para acceder al juego de datos solicitado necesitamos decirle a cada una de las subs el indicador (index) propio del juego de datos, que se encuentra dentro del hash reference devuelto por new(), pasando esta referencia a cada una de las subs como primer parámetro. Esto nos genera un problema, y es que sabemos que el index es un número secuencial que debemos pasárselo a cada sub para que esta opere sobre el mismo, con lo cual en algún momento tenemos el control del mismo.

Supongamos que administramos un equipo con un sistema de usuarios generados con este package y necesitamos ver qué usuarios poseen logon en determinado momento. Consultamos la documentación y vemos que no está establecida, así que hacemos lo siguiente : analizamos el código, generamos nuestro propio usuario (digamos que nos genera un index = 100) y a partir de aquí comenzamos a variarlo desde 1 hasta 100 invocando la sub getUser(). Manos a la obra con nuestro hack !!!

Simplemente agreguemos estas líneas a nuestro código anterior :

```
my $data3 = new();

for my $index ( 1 .. $data3->{'index'} ) {
    my $testData;
    $testData->{'index'} = $index;
    print "Juego de datos '$index', usuario "
        . getUser( $testData ) . "\n"
};
```

obteniendo esta salida adicional :

```
Juego de datos '1', usuario bit-man1
Juego de datos '2', usuario bit-man2
Juego de datos '3', usuario bit-man3
```

En este caso sólo podemos obtener el usuario, pero cualquier datos que pueda ser accedido a través de las subs está accesible para el resto de los usuarios. Entonces, cómo evitar este uso ?? Una referencia cuasi infranqueable es la que se hace a través del código. En estas simplemente se nos da una referencia a la que podemos pasarle parámetros y ejecutará el código pre-establecido sin que podamos cambiarlo ni acceder a sus detalles más íntimos.

Básicamente cuando usamos una sub (ya sea de un package o dentro del programa principal, como en el caso anterior) estamos usando una especie de referencia a un código, tanto que getUser() y new() son un excelente ejemplo de esto, con lo que en principio no se ve que la solución pase por este lado. Pero que tal si, en lugar de generar una sub con código fijo, hacemos una sub con un código distinto para cada invocación y que además tuviera embebido el index de tal forma que no pudiera cambiarse.

Presten atención a los siguientes puntos por demás interesantes :

- - vamos a hacer que las subs generadas sean anónimas, a través de sub {}, y no accesibles directamente sino a través del mecanismo de hash que se usó para index
- - sólo vamos a exportar new, ya que el resto de las subs serán anónimas y visibles sólo desde el package
- - las subs que implementan el código quedan sin modificación, y las subs anónimas actúan como wrappers de estas

```
package myData_3;

use Exporter;
```

```

@EXPORT = qw(new);
@ISA = qw(Exporter);

my $data;

my $nextInstance = 1;

my $getUser = sub {
    my $index = shift;

    return $data->{ $index }->{'user'};
};

my $setPasswd = sub {
    my $index = shift;
    my $oldPasswd = shift;
    my $newPasswd = shift;

    return 1
        if ( $data->{ $index }->{'password'} ne $oldPasswd );

    $data->{ $index }->{'password'} = $newPasswd;
    return 0;
};

sub new {
    my $index = $nextInstance++;

    my $access = { 'getUser' => sub { $getUser->( $index ) },
                  'setPasswd' => sub { $setPasswd->($index, $_[0], $_[1]) }
                };
    ## Ya no se entrega una referencia a los datos
    ## sino un identificador embebido en el código
    ## y no modificable (bueno.. es una forma de decir)

    ## un nuevo juego de datos por cada invocación de new()
    $data->{ $index } = { 'user' => 'bit-man' . $index,
                        'password' => 'chupitanga2005' };

    return $access;
};

1;

```

La forma de uso de este package es similar al anterior, cambiando los nombres de las subs por las correspondientes claves de los hashes :

```

use warnings;
use strict;

use myData_3;

print "----- Generación de datos -----\n";
my $data = new();
print "Primer user : " . $data->{'getUser'}() . "\n";
my $data2 = new();
print "Segundo user : " . $data2->{'getUser'}() . "\n";

```

```

print "----- Cambio de password -----\n";

my $error = $data->{'setPasswd'}->( 'chupitanga2005', ## password vieja
                                   'mypasswordeslargaperomala');
die ("No puedo cambiar la password")
    if $error;

$error = $data2->{'setPasswd'}( 'chupitanga2005', ## password vieja
                               'mypasswordeslargaperomala');
die ("ERROR : No puedo cambiar la password (1) !!!!!!!")
    if $error;

$error = $data2->{'setPasswd'}( 'mypasswordeslargaperomala', ## password vieja
                               'nuevapasswordJA');
die ("ERROR : No puedo cambiar la password (2) !!!!!!!")
    if $error;

print "OK !!!\n";

```

dándonos su ejecución la siguiente salida :

```

----- Generación de datos -----
Primer user : bit-man1
Segundo user : bit-man2
----- Cambio de password -----
OK !!!

```

Una mirada más allá

Como ven el uso de referencias deja lugar a muchas implementaciones, y una de las posibles es la de objetos. Sin entrar en detalles uno de los temas que se persiguen con esta tecnología es el que un API/paquete/library/o- como-se-llame sea usada sólo de la forma especificada, exponiendo una interfaz y sin posibilidad de abuso de los elementos internos. Por ejemplo en nuestro caso debimos implementar las subs internas como una referencia de acceso interno sólo al package para evitar este tipo de accesos laterales "non sanctos"

Infografía

- - Perl builtin functions (<http://search.cpan.org/~nwclark/perl/pod/perlfunc.pod>)
- - Perl references and nested data structures (<http://search.cpan.org/~nwclark/perl/pod/perlref.pod>)
- - Mark's very short tutorial about references (<http://search.cpan.org/~nwclark/perl/pod/perlrefut.pod>)
- - Object::MultiType - Perl Objects as Hash, Array, Scalar, Code and Glob at the same time (<http://search.cpan.org/~gmpassos/Object-MultiType-0.05/MultiType.pm>)

POC (peace of code)

por Andrés Kievsky (andres.kievsky@disenoporteno.com)

El desafío Perl del mes!

```

#!/usr/bin/perl

use strict;
use warnings;

my $suma;
<solución>

```

```
$suma->{1} = 31;  
$suma->[3] = 11;  
print $suma->()."\n";
```

Agregar una línea de código al programa anterior, de forma tal que el print imprima la suma de los dos números. En este caso, la salida sería 42 (es decir, 31+11) pero tiene que funcionar para cualquier número que uno ponga en \$suma-{1} y \$suma-[3] (incluso si uno repite las últimas 3 líneas varias veces) .

La línea tiene las siguientes restricciones:

- - no se puede utilizar bless ni tie ni puede ser un objeto
- - no se pueden usar pseudohashes
- - no puede haber más de un ; en la línea
- - la solución debe tener 35 caracteres o menos
- - no debe generar warnings ni errores, usando siempre "use warnings" y "use strict"
- - no se puede agregar más código que el de esa línea.
- - no se pueden usar módulos de ningún tipo

Está probado en Perl 5.8.0 y 5.8.5.

La solución? En el próximo número :)

