Periódico de la Comunidad Perl de Capital Federal

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

CaFe Perl v1.2

CaFe Perl v1.2 - Periódico de la Comunidad Perl de Capital Federal

Editorial

Hola !!

Novedades, novedades y más novedades. Esta vez es el primer número de CaFe Perl que sale en colaboración, y nada más ni nada menos que con N3krodamus, y como si eso fuera poco con un artículo de su autoría: Usando el perl debugger. Bienvenido a la publicación !!!

En otro orden de cosas a Martín Ferrari no lo tenemos por un tiempo, así que esta vez el PoC quedó en mis manos. También tenemos un reportaje a Steve Hancock (creador de Perltidy) y nuestra ya conocida sección de novedades.

Hasta la próxima taza de CaFe Perl !!! ... eso sí, café del bueno ;-) .... no se vayan, no se vayan, que ahora sigue la editorial de N3krodamus.

Enjoy !

Víctor A. Rodríguez (Bit-Man)

Buenas...

Antes que nada quería agradecerle a Víctor por lo que hizo por el grupo y el boletín todo este tiempo. Intentaré seguirle los pasos en este camino y siempre mejorar en todo lo que se pueda.
No me quiero extender mucho en esta editorial así que simplemente les digo que espero disfruten este boletín y que siempre estaré a la escucha de sugerencias y/o colaboraciones para mejorar el mismo, por el momento es todo.

Que lo disfruten.

Marcelo A. Liberatto (N3krodamus)

POC (peace of code)

Autor: ~~Martín Ferrari~~ esta vez Víctor A. Rodríguez (Bit-Man) X-D

Durante este último tiempo Martín nos deleitó con una serie de one-liners para resolver más de una de nuestras necesidades. Comencé a buscar un poco por la web y me encuentro con que hay toda una serie de switches, teorías y demás que soportan a los one-liners.

Hay una serie de dos artículos escritos en el 2001 y 2003. En el primero (http://www-128.ibm.com/developerworks/linux/library/l-p101/) se tratan los temas básicos de este arte tales como el uso de comillas simples y dobles, el uso de los switches más importantes (-M para usar un módulo, etc.). Los dos puntos que me parecieron más importantes :

- Hay que hacer un balance entre potencia y legibilidad (los one-liners son potentes pero casi ilegibles)
- Los one-liners son para usar y tirar (son construcciones pasajeras, no son pirámides)

El segundo artículo escrito dos años después, en el año 2003 (http://www-128.ibm.com/developerworks/linux/library/l-p102.html) se trata más de una colección de one-liners, algunos muy prácticos y otros demasiado simples.

Asimismo, si nos fijamos en un artículo de la publicación "**The Perl Review"** (http://www.theperlreview.com/Articles/v0i1/one-liners.pdf) hay una forma muy completa de cómo usar los switches, módulos y otra serie de trucos.

Finalmente, podemos visualizar en la URL http://sial.org/howto/perl/one-liner/ todo un compendio y resumen de los anteriores, con muchos ejemplos y formas de convertir un programa en un one-liner !!!

Finalmente si quieren unos cuantos one-liners para divertirse, fíjense en estas direcciones :

- http://www.unixguide.net/unix/perl_oneliners.shtml
- http://www.ajs.com/ajswiki/Perl_one-liners
- http://www.visualgenomics.ca/gordonp/oneliners.html
- http://linuxgazette.net/issue91/okopnik.html

Que lo disfruten !!!!

Entrevista y traducción :
Víctor A. Rodríguez

Para los que no lo conocen Perltidy (http://perltidy.sourceforge.net/) es un script Perl que nos ayuda en la lectura de otros scripts Perl, indentando y reformateándolos. Ideal para mejorar nuestro estilo de escritura en Perl. Hoy está con nosotros Steve Hancock, su autor.

Por favor Steve, una introducción para el grupo CaFe.pm

Hola! Soy un Ingeniero Mecánico en el área de la Bahía de San Francisco, California.

Trabajo analizando la seguridad del lanzamiento de los vuelos espaciales. Esto involucra el análisis de muchos datos y su modelado.

Cuál fue su motivación para construir Perltidy ??

Lo recuerdo muy claramente. Un día en 1999 después de una larga sesión de hacer debugging en un programita de transferencia de calor, me di cuenta que había hecho tal desquicio que realmente no sabía que es lo que hacía el programa. Estaba bastante deprimido, y así busqué un programa que pudiera ordenarlo para poder leerlo de nuevo. No pude encontrar uno, así que pensé que mejor me detenía a hacer uno porque este problema continúa existiendo y malgastando mi tiempo.

Tuvo alguna ayuda de otros proyectos Open Source ??

No directamente, pero por supuesto me beneficié ampliamente de todas las herramientas a mi disposición.

Qué habilidades (relacionadas a Perl o no) adquirió mientras construía Perltidy ??

Bueno, realmente aprendí mucho sobre Perl a medida que trataba de hacer el parsing de su sintaxis.

Como resultado me convertí en un mejor programador. He estado usando Perl sólo desde hace alrededor de un año en una forma muy primitiva, y no soy un *experto en ciencias de la computación*[1]

Qué consejo le daría a los futuros diseñadores y hobbistas que encaran un nuevo proyecto ??

No sobre analizar el problema al principio, sólo buceá y comenzá a trabajar en él, sabiendo que vas a tener que reescribirlo todos algunas veces en la medida que vas ganando nuevo conocimiento. Amalo como un científico de cohetes haría un nuevo vehículo de lanzamiento. Armá el testing y el debugging dentro del código desde el principio, así podes saber exactamente qué está pasando en cualquier caso. Hacé el testing en cualquier forma en la que puedas pensar, tratando de romperlo. Es divertido ! Si procesa archivos alimentalo con basura para ver qué pasa. Con Perltidy, comencé con un pequeño script que sólo buscaba *símbolos de apertura y cierre de bloques*[2] y agregaba a la indentación correspondiente. Probándolo en una gran cantidad de scripts que recolecté, gradualmente evolucionó en un programa útil.

En qué partes del código aconsejaría mirar, a un recién iniciado en Perl, para tener una experiencia de aprendizaje

1 N. del T. : Computer scientist
2 N. del T. : "left and right braces" o símbolos como corchetes y paréntesis que delimitan un bloque de código

placentera ??

En lugar de mirar el código, es interesante ejecutar 'perltidy -D' sobre un programa pequeño para ver como Perltidy lo separa en tokens. Esto creará un archivo .DEBUG y un . LOG . Estos fueron muy útiles durante el desarrollo de Perltidy. El archivo . DEBUG muestra cada línea, y debajo de esta hay una de exactamente la misma longitud mostrando los tokens.

Cómo se puede colaborar con Perltidy ??

Perltidy está muy maduro, pero siempre estoy interesado en reportes de bugs y sugerencias de nuevas funcionalidades.

Qué funcionalidades cree que no están,y cuáles agregará pronto ??

Tiene una buena cantidad de funcionalidades, aunque siempre habrá gente cuyos estilos personales no están cubiertos por las opciones disponibles. Debería agregar una para permitirle a perltidy el saltearse secciones de código.

Qué limites impuso Perl al proyecto ??

Ninguno que se me ocurra

Alguno opinión o consejo acerca de Perl 6 ??

No, pero debería señalar que no hice el cambio de Fortran Perl hasta 1998, así que no puedo ser considerado un "early adopter".

Algún módulo favorito de CPAN ??

Actualmente mi módulo favorito es Spreadsheet::WriteExcel. La gente con la que trabajo le gustan las hojas de cálculo, y es maravilloso ejecutar un script Perl que crea las hojas de cálculo directamente.

Algún programador de Perl favorito, o miembro de la comunidad ??

Mi libro favorito es Perl Cookbook de Tom Christiansen y Nathan Torkington. Aprendí Perl con sus ejemplos, y aún lo tengo como referencia frecuentemente.

Hay un Grupo Perl en su ciudad ??

Estoy seguro que hay alguno pero mi tiempo es tan limitado que no he tenido tiempo de unirme a ellos. Disfruto visitando ocasionalmente el site de los Perl Monks en http://www.perlmonks.org/

Tiene alguna experiencia (divertida o no tanto) de cuando construyó Perltidy y que quiera compartir con nosotros ??

El desarrollo de Perltidy fue distinto al de cualquier otro programa en el que haya trabajado. La sintaxis de Perl no está precisamente escrita en ningún lado, así que cómo podría esperar a hacer el parsing de código Perl y darle un nuevo formato ? Decidí que la mejor forma de hacerlo sería el conseguir tanto código como sea posible y procesarlo. Así que coleccioné ciento de megabytes de código y corrí jobs batch cada noche después de una actualización del código para procesarlos y reportar todos los errores (tales como paréntesis desbalanceados, terminación de una comillas o here-doc). Por lejos los scripts más útiles para este propósito fueron las colecciones de script Perl ofuscados. Fueron muy retadores, y cada vez que obtenía que uno funcionara después de darle un nuevo formato me sentía muy complacido. En algunos casos se leían a si mismos, así que esos scripts no funcionaban de la misma forma que lo hacía el original. Eventualmente tenía la suficiente confianza en Perltidy

para liberar la versión.

Algo más que quiera contarnos y no le preguntamos ??

Cómo comencé a usar Perl: quería crear un álbum de fotos digitales para un grupo de jóvenes local, y debido a mis conocimiento de ingeniería estaba programando en Fortran. Creí que iba a tener algo funcionando en unas pocas horas. Mi hijo, que estaba todavía en la secundaria, se sentó y tuvo algo funcionando en alrededor de veinte minutos usando un lenguaje llamado Perl. Decidí que era mejor que lo aprendiera.

Nuevos libros

- **Steal This Computer Book 4.0** de Wallace Wang (Editorial *No Starch Press*) examina qué es lo que hacen los crackers, cómo lo hacen y cómo protegerse (http://www.oreilly.com/catalog/1593271050/)
- **Building Scalable Web Sites** de Cal Henderson (Editorial *O'Reilly*) cubre todo lo relacionado con el diseño de hardware y software para aplicaciones web (http://www.oreilly.com/catalog/web2apps/)
- **HTML Utopia, Second Edition** de Rachel Andrew (Editorial *O'Reilly*) ideal para todo el que quiera usar CSS y hacer más fácil su trabajo y el de los demás (http://www.oreilly.com/catalog/0975240277/)
- **Enterprise SOA** de Dan Woods y Thomas Mattern (Editorial *O'Reilly*) ideal para arquitectos, desarrolladores y profesionales de IT que quieran entender el valor tecnológico y de negocios relacionado con SOA (http://www.oreilly.com/catalog/enterprisesa/)
- **Perl Hacks** de chromatic, Damian Conway y Curtis "Ovid" Poe (Editorial *O'Reilly*) es el libro perfecto para quienes desean saber más sobre Perl (y sacarle más el jugo) o para programadores avanzados que quieren ahorrarse un precioso tiempo de programación (http://www.oreilly.com/catalog/perlhks)
- **Learning UML 2.0** de Russell Miles y Kim Hamilton (Editorial *O'Reilly*) es una mirada sobre cómo comunicar y documentar un diseño (http://www.oreilly.com/catalog/learnuml2)

Podcasts

- Perl Best Practices, de Damian Conway, pasado a una charla de una hora por Randal L. Schwartz, con slides y audio (http://www.ourmedia.org/node/225170 y (http://www.podasp.com/episodes/p/pd/pdxlug/599.mp3). Ambas unidas en formato Quicktime : http://www.media-landscape.com/demos/PBP/Schwartz_PBP_800x600.mov
- Entrevista a Jesse Vincent creador de Jifty (http://www.jifty.org/) otra forma de construir aplicaciones web. Pueden escucharla en http://www.perlcast.com/audio/Perlcast_Interview_027_Jifty.mp3
- Entrevista a JT Smith de Plain Black creador de WebGUI (http://www.plainblack.com/webgui) en http://perlcast.com/2006/05/25/interview-with-jt-smith-of-plain-black-creators-of-webgui/

Entrevistas

- **Allison Randall :** The Perl Journal entrevistó a este personaje del mundo Perl, autor de *Perl 6 and Parrot* entre otros (http://www.theperlreview.com/Interviews/allison-randal-20060412.html)

OpenOffice standard

A partir del 1ro. de Mayo la ISO ha aprobado el formato OpenDocument como un standard para documentos denominados “office” (http://www.consortiuminfo.org/standa

rdsblog/article.php?story=20060503080915835)

Tutoriales interactivos

En el site Perl en Español de Uriel Lizama se está proponiendo el hacer tutoriales interactivos, a través de IRC (http://perlenespanol.baboonsoftware.com/foro/about924.html)

Tutoriales interesantes

Otra vez en Perl en Español una lista de tutoriales nuevo para sacarle más el jugo a Perl :

- Vender soluciones con PayPal : http://perlenespanol.baboonsoftware.com/archives-tut/000166.html
- Perl y AJAX : http://perlenespanol.baboonsoftware.com/archives-tut/000167.html
- IRC Bot : http://perlenespanol.baboonsoftware.com/foro/about935.html

Este dato te va a helar la sangre

Alex Gough tiene un interesante trabajo en la Antártida obteniendo datos de experimentos que se realizan, mayormente sobre el clima. Si querés enterarte fijate en http://www.perl.com/pub/a/2006/05/04/charting-data.html

Recursos open source de Motorola

Motorola acaba de anunciar el lanzamiento de un nuevo sitio web dedicado a los desarrolladores : http://opensource.motorola.com

Construyendo aplicaciones empresariales

Si bien este es un artículo que proviene del mundo Java (http://dev2dev.bea.com/pub/a/2006/05/agile-modeling.html) es lo suficientemente genérico como para poder leerlo abstrayéndose de su origen

Perl fuera de la ley en UK ?

Una propuesta de ley en Gran Bretaña daría nuevo poderes a la justicia para combatir el ciber-crimen, incluyendo un nuevo delito que es el de “fabricar, proveer u obtener artículos para su uso en el abuso de computadoras”. Claramente herramientas como nmap o Perl caen en esta categoría. No me creen ?? Vean lo que dijo el Dr. Richard Clayton de la Universidad de Cambridge : según él “... casi todo hacker que comete una ofensa bajo la sección 1 o 3 del CMS usará Perl como parte de su toolkit”. Si señoras y señores, todos ustedes que leen esta publicación son **innegablemente culpables** ante la futura modificación a la ley en Gran Betaña. Fuera bellacos !!!!!

Las 10 cosas que cada Perl hacker debe conocer

Algo simple sencillo y claro como para empezar a conocer, o debatir, qué es Perl (http://builder.com.com/5100-6374_14-6077064.html)

El futuro de Perl en PostgreSQL

Este artículo trata de cómo las versiones 8.0 y 8.1 agregaron y mejoraron características de soporte al lenguaje Perl dentro del motor (http://www.oreillynet.com/pub/a/databases/2006/05/25/the-future-of-perl-in-

postgresql.html)

Cómo usar scripts Perl o PHP en tu site

Un simpático artículo sobre cómo dar tus primeros pasos para poner un script en tu site (http://www.whatpc.co.uk/personal-computer-world/features/2155260/processing-web)

La caja de herramientas del sysadmin

En ninguna caja de herramientas debe falta la famosa “Duct Tape” (la de color aluminio) que de todo nos salva, entonces ... por qué iba a faltar Perl en la caja de herramientas del sysadmin ?? (http://software.newsforge.com/software/06/05/01/1630200.shtml)

Gantry, otro framework para aplicaciones web

Especialmente hecha para Perl, Apache, mod_perl, CGI y FastCGI soporta el modelo MVC. Pueden darle una mirada en http://www.perlmonks.org/?node_id=548551 o http://www.usegantry.org/

Parrot 0.4.4 liberado

Como de costumbre, una nueva versión de Parrot (http://www.parrotcode.org) la máquina virtual que soportará Perl 6 ya está disponible en http://www.cpan.org/authors/id/L/LT/LTOETSCH/parrot-0.4.4.tar.g

Licenciamiento de Pugs

Audrey Tang, Allison Randal, Jesse Vincent y otros actores de Perl 6 y Pugs están proponiendo una nueva forma de licenciamiento de Pugs (http://pugs.blogs.com/pugs/2006/04/licensing_clari.html)

Cambios en Catalyst

Adam Kennedy anunció algunos cambios en el proyecto Catalyst, siendo el mayor que el fundador del proyecto, Sebastian Riedl, no pertenece más al grupo de desarrolladores core, y comenzó a trabajar en un nuevo framework llamado Woodstock (http://lists.rawmode.org/pipermail/catalyst/2006-May/007250.html)

Begins : La Revista de software libre y código abierto.

http://www.linuxchillan.cl/?q=node/155

Vim 7 trae de todo

Pónganse contentos !!!! Ya salió la nueva versión de Vim (http://www.vim.org/) el editor usado por muchos de los que leen este periódico !!!

Criptografía manuscrita

Ocurre que a Bruce Shneier (brillante personaje del mundo de la seguridad) le pasó una tragedia abominable. Más precisamente una amiga de él y su hijo fueron asesinados por su marido (y a la vez padre adoptivo de la criatura) quién después se colgó. A qué viene esta nota de tinte negro en CaFe Perl, dirán ustedes ??

Sencillamente que esta persona dejó

una especie de criptograma manuscrito el cual fue publicado en el blog de Schneier (http://www.schneier.com/blog/archives/2006/01/handwritten_rea.html) pidiendo ayuda para descifrarlo y ver si realmente se podía saber algo más sobre el móvil de esta persona. Si ven la entrada del blog se van a maravillar de cómo un montón de gente propuso y analizó cada pedazo (hasta los pliegues del papel) para ayudar en esto. Algo muy común en la comunidad Open Source, pero que en otras lides asombra mucho.

Mordiditas de aquí y de allá

Usando el Debugger de Perl (Parte 1)
Autor : N3krodamus

Introducción:
El sentido de este artículo es mostrar de una forma empírica el uso del debugger de perl, para ello me voy a valer de un par de scripts; el archiconocido "Hola Mundo" y script un poco mas complejo que se conecta a una base de datos MySQL y hace una consulta.
Porque una conexión a una base de datos? Bueno porque de esa manera se puede apreciar el funcionamiento al acceder a una librería de funciones externa.
También puse un apartado extra para aquellos que no tienen conocimientos de MySQL y puedan de paso generar la tabla de pruebas sin mayores dificultades.
Espero que este artículo sirva para esclarecer el uso de esta herramienta y quitarle el miedo al mismo, ya que por experiencia propia se que es bastante intimidante la primera vez que se la usa.
Este artículo lo escribí en mi pc Linux, seguramente existan diferencias en algunos comandos ejecutados desde el prompt que son propios de Linux y utilicen convenciones propias del sistema operativo, desconozco los comandos equivalentes para la versión de Windows.

Enjoy it.

```
#!/usr/bin/perl

my $str = 'Hola Mundo';
print "$str \n";
exit;
```

Primero que nada vamos a presentar el script a depurar.
Vamos a empezar con los comandos básicos del `debugger` y una vez que los hayamos visto podremos pasar al siguiente script que es un poco mas interesante que este de aquí. Primero que nada grabaremos el script en un archivo de nombre `hola_mundo.pl`.

Nota:
Verifiquen la ruta correcta del interprete de perl cuando graban el archivo en sus equipos porque puede estar ubicado en otra ruta que no es la que yo pongo en mi script, en Linux lo pueden hacer con el comando *`'whereis perl'`*.

Para ingresar al debugger tenemos que invocar el script con el interprete perl delante mas el parámetro -d, lo que nos va a dejar dentro de la consola del debugger.

```
nekrodamus@daimaku:~$ perl -d hola_mundo.pl

Loading DB routines from perl5db.pl version
1.25
Editor support available.

Enter h or `h h' for help, or `man
perldebug' for more help.

main::(hola_mundo.pl:3): my $str = 'Hola
Mundo';
DB<1>
```

Que información vemos aquí? Bien en las primeras lineas nos informa que la versión del debugger es la 1.25, mas a abajo nos indica como podemos entrar al help del debugger; un comando que vamos a usar seguido es *`'h'`* help o *`'h h'`* dentro del debugger; si queremos ver la man page del debugger tenemos que tipear desde el prompt *`'man perldebug'`*.
También vemos la primera linea de nuestro script `'hola_mundo.pl'`, en

nuestro caso la número 3, con la primer instrucción ejecutable del mismo. Una confusión que todos tenemos la primera vez es pensar que esta linea ya se ejecutó, pero en realidad nos esta diciendo que la próxima linea a ejecutar es la que nos muestra.
Este enunciado *'main::(hola_mundo.pl:3):'* nos esta dando un poco mas de información además del numero de linea dentro del script, nos esta mostrando el namespace en donde estamos parados. Que es el namespace? Bueno no esta en el alcance de este artículo explicar namespaces de perl pero a modo resumido podemos decir que es el espacio donde existen las variables, cuando trabajamos con módulos cada modulo tiene su namespace evitando que haya choques entre variables de distintos módulos, al modulo principal perl le asigna siempre el namespace main. En nuestro caso el script que tenemos no usa ningún modulo externo y la única variable que usa es declarada en el modulo principal. La linea que vemos aquí se leería así, la linea 3 del modulo hola_mundo.pl en el namespace main. Sigamos entonces leyendo lo que sigue.
En la pantalla vemos 'DB<1>', este es el prompt del debugger, acá se pueden ingresar comandos tanto de perl como del mismo debugger. El numero entre los signos *'<''>'* es el numero de linea dentro del archivo histórico de comandos, a medida que vayamos ingresando comandos ese numero se va incrementando.

Vamos a hacer una prueba.

En el prompt ingresemos lo siguiente:

```
DB<1> print $str;
```

Que valor nos va a devolver?
Si pensaron 'Hola Mundo' esta mal. Por qué?. Porque la variable $str todavía no se asigno debido a que la linea que estamos viendo no se ejecuto aun.
Vamos a ejecutar nuestro script linea por linea para ver que sucede, esto lo hacemos con el comando *'s'*.

```
DB<2> s
main::(hola_mundo.pl:5):print
"$str \n";
DB<2> s
Hola Mundo
main::(hola_mundo.pl:7): exit;
DB<2>
```

Bien, que se ve aquí? Bueno como verán el numero de linea del histórico de comandos se incremento en 1, eso es porque ahora en nuestro histórico de comandos esta guardado el comando 'print $str;' que anteriormente ejecutamos. Vamos a repetir la prueba anterior, pero ahora en vez de escribir toda la linea de nuevo ingresamos lo siguiente '!1'.

```
DB<2> !1
print $str;
Hola Mundo
DB<3>
```

El comando *'!'* seguido de un numero nos ejecuta la instrucción que tenemos en esa linea del histórico de comandos, en nuestro caso el print de la variable $str, y ahora si obtenemos un resultado. En caso de que queramos ingresar un comando de varias lineas podemos hacerlo agregando una contra barra al final de la linea para continuar la instrucción en el renglón que sigue.

Por ejemplo.

Acá cree un pequeño script en el prompt del debugger declarando una variable y todo, si en un futuro quiero volver a correrlo simplemente llamo a la linea con el comando '!'.

Después de tantas pruebas me olvide de como era el listado del script, y no

se en que linea estaba, ingresemos en el prompt el comando 'l' L minúscula.

```
DB<3> my @pp = ('a','b','c');\
  cont: foreach (@pp) {\
  cont: print $_;\
  cont: }
abc
DB<4>
```

Este comando me dice en que linea estoy parado esperando ejecución, en este caso la linea 7 que tiene como

```
DB<4> l
7==>    exit;
DB<4>
```

instrucción el comando 'exit;', pero no me acuerdo lo anteriormente ejecutado entonces puedo agregar un par de modificadores al comando.

Acá le indique al debugger que me muestre un listado de las lineas entre la 1 y la 8, por supuesto el script solo tiene 7 lineas por lo tanto veo todo el listado. Si quisiera ver solamente la

```
DB<4> l 1-8
1 #!/usr/bin/perl
2
3: my $str = 'Hola Mundo';
4
5: print "$str \n";
6
7==> exit;
DB<5>
```

linea 5 entonces haría 'l 5'.

Continuamos la ejecución del script con el comando *'s'*.

Bien, la ejecución termino y nos deja una leyenda con algunas sugerencias para continuar depurando el script. Mas adelante veremos estos comandos.

```
DB<5> s
Debugged program terminated.
Use q to quit or R to restart,
  use O inhibit_exit to avoid
stopping after program
termination,
  h q, h R or h O to get
additional info.
DB<5>
```

Hasta acá vimos algo sencillo, vamos a ver ahora un script un poco mas complicado para que vean el uso del debugger un poco mas a fondo y sea mas provechoso.

Antes que nada lo que vamos a necesitar:

Una tabla MySQL de la cual vamos a leer datos.

Copien esto a un archivo aparte y

```
create database emails;

use emails;

CREATE TABLE subscripciones (
  id int(7) NOT NULL
       auto_increment,
  Email varchar(100)
      default NULL,
  PRIMARY KEY  (id)
);

INSERT INTO subscripciones
VALUES (null,'aa@hotmail.com');
INSERT INTO subscripciones
VALUES (null,'bb@hotmail.com');
INSERT INTO subscripciones
VALUES (null,'cc@hotmail.com');
INSERT INTO subscripciones
VALUES (null,'dd@hotmail.com');
```

guárdenlo con nombre tabla.sql. A continuación en el prompt de su maquina ejecuten lo siguiente:

```
MySQL -u root -p < tabla.sql
```

Esto les dejará la base de datos 'emails' y la tabla 'subscripciones' creada en el MySQL.

Explicación del comando:

-u Indica al MySQL que usuario ejecuta los comandos, en este caso root (ojo es el root del MySQL que no tiene nada que ver con el root del equipo).

-p Indica que la clave sera pedida antes de conectarse al MySQL, en la instalación por defecto del MySQL el root esta sin clave y si fuera ese el caso no debemos poner este parámetro.

```
#!/usr/bin/perl

use DBI();

unless($ARGV[0])
	{print "ERROR: inserte un numero valido\n\n";exit;}


# Guardamos el input en una variable
my $limit = $ARGV[0];

# connect
my $dbh = DBI->connect("DBI:MySQL:database=emails;host=localhost",
"root", "", {'RaiseError' => 1})
	or die ("Cannot connect to database");

# Obtenemos la lista de emails
my $sth = $dbh->prepare("SELECT Email FROM subscripciones  limit
0,$limit");
$sth->execute();

while(my $ref = $sth->fetchrow_hashref())
{
	my $to = $ref->{'Email'};
	print "$to\n";
}

# clean up
$dbh->disconnect();
```

Explicación:

Este script es bastante sencillo, pide un argumento vía linea de comandos que es la cantidad de registros a obtener de la base de datos y luego entra en un ciclo hasta llegar a ese limite. A medida que va obteniendo los resultados de la base, que son direcciones de email, las va imprimiendo en pantalla. Como antes guardamos el script en un archivo a parte con el nombre de prueba.pl y le damos permisos de ejecución.

Luego ingresamos al debugger:

```
nekrodamus@daimaku:~/$
                  perl -d prueba.pl
```

Lo que nos va a devolver:

Bien como antes quedamos en la primer linea ejecutable de nuestro

```
Loading DB routines from perl5db.pl version 1.25
Editor support available.

Enter h or `h h' for help, or `man perldebug'
for more help.

main::(prueba.pl:5):    unless($ARGV[0])
DB<1>
```

script, vamos a ejecutarlo linea por linea con el comando `'s'`

```
main::(prueba.pl:5): unless($ARGV[0])
DB<1> s
main::(prueba.pl:5): unless($ARGV[0])
DB<1> s
ERROR: inserte un numero valido

main::(prueba.pl:6): {print "ERROR: inserte un numero
valido\n\n";exit;}
DB<1> s
DBI::CODE(0x83dc4d8)(/usr/local/lib/perl/5.8.4/DBI.pm:466):
466:        return unless defined &DBI::trace_msg; # return unless
bootstrap'd ok
DB<1> s
DBI::CODE(0x83dc4d8)(/usr/local/lib/perl/5.8.4/DBI.pm:467):
467:        local ($!,$?);
DB<1> s
DBI::CODE(0x83dc4d8)(/usr/local/lib/perl/5.8.4/DBI.pm:468):
468:        DBI->trace_msg("    -- DBI::END\n", 2);
DB<1> s
DBI::CODE(0x83dc4d8)(/usr/local/lib/perl/5.8.4/DBI.pm:470):
470:        $DBI::PERL_ENDING = $DBI::PERL_ENDING = 1;  # avoid typo
warning
DB<1> s
DBI::CODE(0x83dc4d8)(/usr/local/lib/perl/5.8.4/DBI.pm:471):
471:        DBI->disconnect_all() if %DBI::installed_drh;
DB<1> s
Debugged program terminated.  Use q to quit or R to restart,
use O inhibit_exit to avoid stopping after program termination,
h q, h R or h O to get additional info.
DB<1>
```

Bien acá vemos algo raro...

El script devuelve un montón de errores, bueno la respuesta es simple; este script necesita ser invocado con un parámetro al final que es el limite de cantidad de direcciones que va a obtener de la base de datos. Acá yo cometí el error intencional de llamarlo sin ningún parámetro, si vemos el listado después de ejecutar la linea 5 del script, la que tiene el `'unless (ARGV[0])'`, imprime el mensaje de error `'ERROR: inserte un numero valido';` lo que quiere decir que no se ingreso ningún parámetro y si vemos el listado del script la linea 5 es así:

```
unless($ARGV[0])
      {print "ERROR: inserte un
numero valido\n\n";exit;}
```

Lo que sigue después del mensaje de error es un *'exit;'*. Pero se siguen ejecutando lineas después del exit?. Bueno en perl, como en otros lenguajes, tenemos la posibilidad de ejecutar código de limpieza después de terminar el script, algo inverso a lo que seria código de inicialización; este código se coloca en un bloque END {} y todo lo que este dentro de ese bloque sera ejecutado cuando el script termine. Una de las razones por las que elegí este script para correr dentro del debugger es porque como en este caso nos permite ver rutinas que corren y que nunca nos enteramos. Lo que esta haciendo después del exit es llamar a una rutina interna propia de la librería DBI que se encarga de la limpieza interna del modulo, si analizamos la linea :

vemos que esta llamando la linea 466 del modulo / usr/local/lib/perl/5.8.4/DBI.pm, que justamente tiene un bloque END{}.

```
DBI::CODE(0x83dc4d8)
(/usr/local/lib/perl/5.8.4/DBI
.pm:466):
```

Vamos a volver a entrar al debugger

```
nekrodamus@daimaku:~/perl/libro/paper_debugg
er$ perl -d prueba.pl 3

Loading DB routines from perl5db.pl version
1.25
Editor support available.

Enter h or `h h' for help, or `man
perldebug' for more help.

main::(prueba.pl:5):    unless($ARGV[0])
DB<1> s
main::(prueba.pl:10):   my $limit = $ARGV
[0];
DB<1> s
main::(prueba.pl:13): my $dbh = DBI->connect
("DBI:MySQL:database=emails;host=localhost",
"root", "",
main::(prueba.pl:14):
{'RaiseError' => 1})
main::(prueba.pl:15):           or die
("Cannot connect to database");
DB<1>
```

pero esta vez llamando al script en forma correcta.

```
DB<1> $limit = 10;
DB<2> print $limit;
10
DB<3>
```

Vamos a suponer que queremos cambiar el valor de $limit, que aquí tiene valor 3 ya que lo toma de $ARGV [0].

En cualquier momento podemos imprimir o modificar los valores de cualquier variable en tiempo de ejecución. Continuamos corriendo el script con el comando *'s'*

Aquí vemos algo nuevo, anteriormente hablaba del namespace del módulo y acá vemos justamente eso. Luego de que nuestro script invoca la función connect del modulo DBI el debugger se va de nuestro script y pasa a ejecutar las sentencias del modulo DBI.pm, concretamente la linea 512. Esto es particularmente útil si queremos depurar módulos nuestros y realizar un seguimiento minucioso del script, en este caso es solamente educativo y por simple curiosidad. Acá vamos a darle una mirada al listado de la función connect solo para ver que hace.

(sigue en la próxima página)

```
DB<3> s
DBI::connect(/usr/local/lib/perl/5.8.4/DBI.pm:512):
512:        my $class = shift;
DB<3> s
DBI::connect(/usr/local/lib/perl/5.8.4/DBI.pm:513):
513:            my ($dsn, $user, $pass, $attr, $old_driver) = my
@orig_args = @_;
DB<3> s
DBI::connect(/usr/local/lib/perl/5.8.4/DBI.pm:514):
514:        my $driver;
DB<3> s
DBI::connect(/usr/local/lib/perl/5.8.4/DBI.pm:516):
516:          if ($attr and !ref($attr)) { # switch $old_driver<-
>$attr if called in old style
DB<3>
```

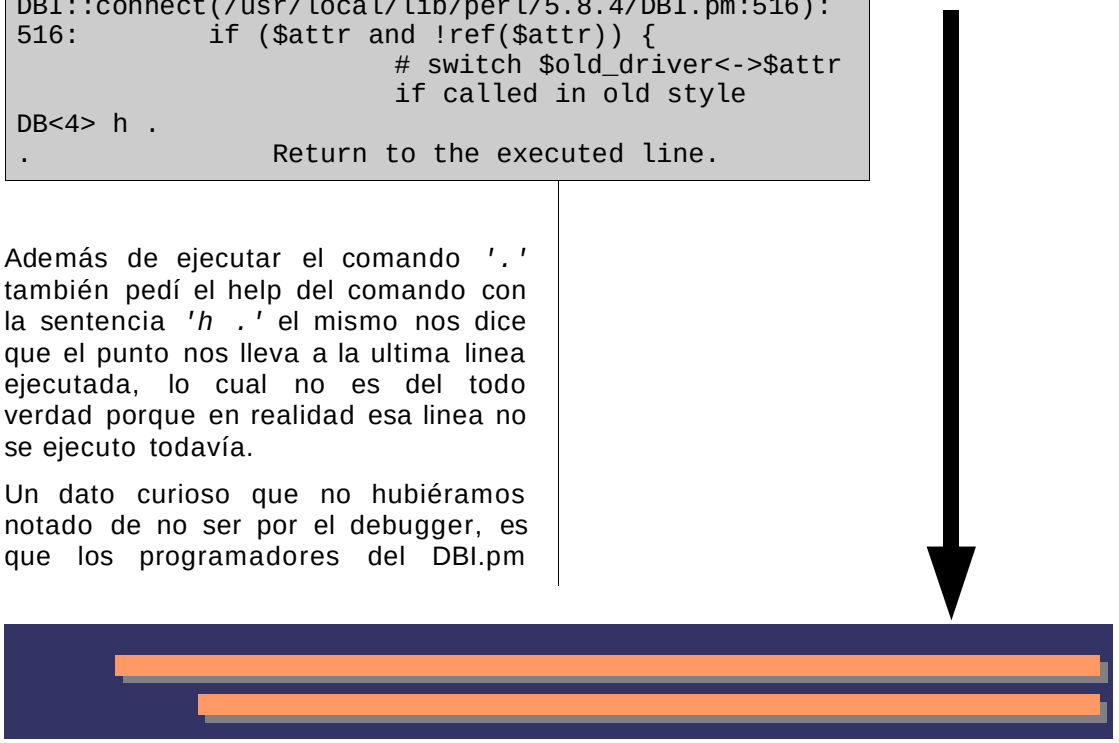
```
DB<3> l 512-525
512:        my $class = shift;
513:        my ($dsn, $user, $pass, $attr, $old_driver) = my
@orig_args = @_;
514:        my $driver;
515
516==>       if ($attr and !ref($attr)) { # switch $old_driver<-
>$attr if called in old style
517:            Carp::carp("DBI->connect using 'old-style' syntax
is deprecated and will be an error in future versions");
518:            ($old_driver, $attr) = ($attr, $old_driver);
519         }
520
521:        my $connect_meth = $attr->{dbi_connect_method};
522:        $connect_meth ||= $DBI::connect_via;        # fallback
to default
523
524:        $dsn ||= $ENV{DBI_DSN} || $ENV{DBI_DBNAME} || '' unless
$old_driver;
525
DB<4>
```

Nos esta indicando con un signo '==>' que la linea próxima a ejecutar es la 516, esto mismo lo podemos ver con el comando `'v'` que devolverá una salida similar. Y si queremos ver solamente la linea que se va a ejecutar lo podemos hacer con el comando *'.'*, acá mas que nunca se hace sentir aquella frase de 'Hay mas de una forma de hacerlo'.

```
DB<4> .
DBI::connect(/usr/local/lib/perl/5.8.4/DBI.pm:516):
516:        if ($attr and !ref($attr)) {
                        # switch $old_driver<->$attr
                        if called in old style
DB<4> h .
.               Return to the executed line.
```

Además de ejecutar el comando *'.'* también pedí el help del comando con la sentencia *'h .'* el mismo nos dice que el punto nos lleva a la ultima linea ejecutada, lo cual no es del todo verdad porque en realidad esa linea no se ejecuto todavía.

Un dato curioso que no hubiéramos notado de no ser por el debugger, es que los programadores del DBI.pm colocaron una variable dbi_debug dentro de su código, se podría probar a ponerla en 1 a ver que pasa :). Pero eso queda para que lo investiguen. Ya vimos todo lo que queríamos ver de la función DBI::connect y para salir de la misma sin ver como se ejecuta cada linea dentro de la función ejecutamos el comando *'r'* el mismo lo que hace es continuar la ejecución del script hasta terminar la función en donde estamos, en este caso nos devuelve a la linea posterior al connect.



```
DB<5> l 525-530
525
526:        if ($DBI::dbi_debug) {
527:            local $^W = 0;
528:            pop @_ if $connect_meth ne 'connect';
529:            my @args = @_; $args[2] = '****'; # hide password
530:            DBI->trace_msg("    -> $class->$connect_meth(".join
(", ",@args).")\n");
DB<6>
```

Como vemos también nos da alguna información extra, nos informa que el valor de retorno de la DBI::connect es un hash vacío en entorno scalar. Que significa esto? Bueno la función

```
DB<6> r
scalar context return from DBI::connect:   empty hash
main::(prueba.pl:19):   my $sth = $dbh->prepare(
"SELECT Email FROM subscripciones  limit 0,$limit");
DB<6>
```

connect devuelve una referencia a un hash anónimo, por eso el mensaje, nosotros guardamos el valor en un scalar que en realidad es una referencia a un hash anónimo.

Continuamos la ejecución del script pero esta vez con el comando 'n' que nos permite quedarnos dentro de nuestro script salteando las llamadas a funciones y subrutinas.

```
DB<6> n
main::(prueba.pl:20): $sth->execute
();
DB<6> n
main::(prueba.pl:22):   while(my
$ref = $sth->fetchrow_hashref())
main::(prueba.pl:23): {
DB<6> n
main::(prueba.pl:24): my $to =
                    $ref->{'Email'};
DB<6> n
main::(prueba.pl:25): print "$to\n";
DB<6> n
aa@hotmail.com
main::(prueba.pl:24):  my $to =
$ref->{'Email'};
DB<6>
```

Acá entramos en la iteración del while, vamos a ver un par de comandos para imprimir valores de variables. Si yo quisiera imprimir el valor de la variable $to lo podría hacer con el clásico comando de perl *'print'*.

Que paso acá? La variable $to

```
DB<6> n
aa@hotmail.com
main::(prueba.pl:24): my $to =
$ref->{'Email'};
DB<6> print $to

DB<7> s
main::(prueba.pl:25): print
"$to\n";

DB<7> s
main::(prueba.pl:25): print
"$to\n";

DB<7> s
bb@hotmail.com
main::(prueba.pl:24): my $to =
$ref->{'Email'};
```

esta vacía cuando tendría que tener el valor de la iteración anterior.... Bueno en realidad fue otro error mio, no intencional esta vez, puse un *'my $to = $ref->{'Email'}'*, cuando debería haber ido *'$to = $ref->{'Email'};'* el my declara la variable en privada dentro del bloque while, de esa manera con cada iteración vuelve a declarar la variable lo que provoca que en algún punto antes del my la misma sea blanqueada, pueden probarlo

modificando el script y sacando ese my. Sigamos adelante; puedo hacer un print de la variable cuando es un scalar común y corriente pero que pasa cuando es una referencia como en el caso de $ref ?

```
DB<7> print $ref;
HASH(0x86227fc)
DB<8>
```

No me sirve de mucho, para eso es mejor usar el comando *'x'*.

Mejor, acá nos informa que tipo de

```
DB<8> x $ref
0  HASH(0x86227fc)
   'Email' =>'cc@hotmail.com'
DB<9>
```

valor contiene esa variable, HASH, y a la vez la des referencia por nosotros; la salida de este comando es muy similar a la del modulo Data::Dumper así que aquellos que lo hayan usado se van a sentir cómodos.
En algún momento podemos querer ver todas las variables de nuestro script o de algún modulo en particular, para eso podemos usar el comando *'V'*, por ejemplo si quisiera ver las variables del modulo DBI.

(ver cuadro de la página siguiente)

Como vemos la salida es bastante extensa, pero paso a destacar algunas de las lineas que me parecieron mas útiles.

Name'=>'database=emails;host=localhost'

Vemos los datos de la conexión a la base de datos, concretamente la base de datos a la que nos conectamos y el host.

'Statement' => 'SELECT Email FROM subscripciones limit 0,10'

En la variable `'Statement'` vemos el query SQL que ejecutamos en nuestro script, se acuerdan que anteriormente modificamos el valor de la variable $limit a 10 ? Bueno acá lo vemos después del 0 .

'Username' => 'root'

Analogamente en `'Username'` se ve el usuario que esta conectado a la base de datos.

Hemos llegado al final de esta primer entrega, debido a la longitud del artículo decidí cortarlo en dos partes para que no quede tan grande el boletín. En la segunda entrega veremos algunas cosas mas acerca de las herramientas que nos brinda el debugger.

Saludos.

Marcelo A. Liberatto

N3krodamus

Agradecimientos:

A Víctor por darme una mano corrigiéndome algunas cosas del documento.
A Matías por hacerme un par de sugerencias para clarificar más el artículo.

Bibliografía

- Perldebtut : http://search.cpan.org/dist/perl/pod/perldebtut.pod
- Using the perl debugger (http://www.devshed.com/c/a/Perl/Using-The-Perl-Debugger/1/)
- Unraveling the code with perl debugger (http://www.perl.com/pub/a/2006/04/06/debugger.html?page=1)

```
DB<9> V DBI
$lasth = DBI::st=HASH(0x8622694)
   'Database' => DBI::db=HASH(0x860a4bc)
      'Driver' => DBI::dr=HASH(0x85df988)
         'Attribution' => 'DBD::MySQL by Rudy Lippan'
            .
            .
            .
            .

   'Statement' => 'SELECT Email FROM subscripciones  limit 0,10'
   'TraceLevel' => 0
Can't locate auto/DBI/FIRSTKEY.al in @INC (@INC contains: /etc/perl
/usr/local/lib/perl/5.8.4 /usr/local/share/perl/5.8.4 /
usr/lib/perl5 /usr/share/perl5 /usr/lib/perl/5.8 /
usr/share/perl/5.8 /usr/local/lib/site_perl .) at /
usr/share/perl/5.8/dumpvar.pl line 355
        ...propagated at /usr/share/perl/5.8/perl5db.pl line 2244.
       DB::DB called at prueba.pl line 25
Debugged program terminated.  Use q to quit or R to restart,
  use O inhibit_exit to avoid stopping after program termination,
  h q, h R or h O to get additional info.
DB<10>
```