Periódico de la Comunidad Perl de Capital Federal

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

CaFe Perl v1.0

CaFe Perl v1.0 - Periódico de la Comunidad Perl de Capital Federal

Editorial

Sorpresa !!

Año nuevo vida nueva ... o por lo menos formato nuevo.

Y sí, en este número cumplimos un año de publicación así que van a encontrar un diseño renovado y mucho más visual que el anterior, con muchas más oportunidades para sentirse cómodos :

- el cambio a dos columnas permite un revisión más rápida y una lectura más fluida
- Los artículos van a incorporar facilidades de interpretación
- el motivo y los colores son similares al de nuestro site (http://cafe.pm.org)

Y todo esto con el más absoluto costado Open Source, ese que empezó allá en los '50 con los primeros hackers del MIT, que siguió con el libre flujo de información y desemboca en la aparición del Open Source , ese mismo que hoy usamos y nos da tanta libertad y satisfacciones.

Y de todo ese camino recorrido por el Open Source nos cabe preguntar : qué le devolvemos nosotros a la comunidad Open Source en retribución de todo lo que nos da ?

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

Víctor A. Rodríguez (Bit-Man)

POC (peace of code)

Autor: n3krodamus

Acá va un breve truco para la linea de comandos de UNIX.

Los otros días se me presento la tarea de procesar unas fotos que venían de una cámara digital para ser publicadas en una web, debido a que soy extremadamente vago para hacer las cosas y no me gusta hacer nada repetitivo mas de una vez; me plantee como hacerlo en un *"quick and dirty"* con perl en la linea de comandos. La tarea era simple, dado un grupo de imágenes con extensión .JPG tenia que renombrarlas con un numero cada una y pasar sus nombres a minúsculas. Las imágenes provenientes de estas cámaras digitales ya de por si tienen una numeración pero a mi no me servía porque yo tengo un template armado que lee las imágenes desde *1..n* y con las extensiones en minúsculas.

Entonces paso a explicar la tarea a realizar :

1. Tengo que recorrer el listado de las imágenes en el directorio para poder numerarlas.
2. Genero el nuevo nombre para la imagen, en este caso el numero de orden (Ej: 1, 2, 3, n)
3. Armo la línea que el shell va a leer con el comando `mv` (Ej: `mv DSC435.JPG 1.jpg`)
4. Ejecuto la línea de comando con una llamada al `system`
5. Me tomo un café viendo como la máquina hace el trabajo por mi.

Acá esta la linea de código que corrí en el shell :

```
perl -e 'foreach(`ls *JPG`)
{chomp; ++$a; $b=$_; s/^(\w)
+/$a/;system "mv $b ".lc($_)."
\n"; }'
```

Explicación: Primero se arma un bucle `foreach` que recorre como un array el resultado de un `'ls -l *JPG'`, ojo no confundir **"`"** con **""** ya que el primero lo que hace es redireccionar la salida del comando ejecutado hacia una variable y el otro simplemente se usa para declarar strings de texto, de esta manera leo los nombres de todas las imágenes en el directorio de extensión .JPG . Luego hago un `chomp` para eliminar el `"\n"` al final de la línea de los nombres de las imágenes, de esa manera cuando armo la linea de comandos no me inserta un **"enter"** al final del nombre viejo. Lo que sigue es incrementar un contador para generar la numeración nueva, después de eso me guardo el nombre viejo de la imagen que quiero renombrar. Finalmente con una expresión regular cambio el nombre viejo de la imagen por el numero para armar el nombre nuevo (recuerden que el `\w` lee caracteres `[a-zA-Z]` y de esa manera en el punto que separa la extensión se detiene). Como último comando hago la llamada al `system` armando el comando `mv` con el nombre viejo primero (que lo habia guardado en `$b`) y el `lc` (lowercase) del nombre nuevo; el lowercase lo puse porque la extensión todavía esta en mayúsculas.

Voila!! en una linea de código hice todo el trabajo sin mayores esfuerzos.

GRACIAS Larry Wall.

Entrevista y traducción :
Víctor A. Rodríguez

Todos conocemos la importancia de un una interfaz de línea de comandos, pero hay alguien que no sólo lo sabe sino que también añoraba un shell extinto hace mucho denominado Visual Shell (vsh), tanto como para devolverlo a la vida valiéndose de la magia de Perl (http://www.cs.indiana.edu/~kinzler/vshnu). Esa persona es Steve Kinzler.

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

Es un honor y un placer hacerlo.

He estado programando Perl como parte central de mi trabajo por cerca de 17 años, primero para administrar sistemas en la Universidad de Indiana (IU), y ahora para aplicaciones Web/Internet en la IU.

También lo enseño como parte de un curso que desarrollé en la IU en 1996.

Perl es, en resumen, como lo llamó un colega, el "lenguaje de los dioses". No hay otro lenguaje que yo conozca que nos pueda acercar tanto al regocijo creativo y la alquimia cognoscitiva inherente en la programación de computadoras.

También tengo cierta notoriedad en ciertos círculos por crear y llevar adelante el Oráculo de Internet (http://www.internetoracle.org) y otros proyectos que datan de mis años de juventud (http://www.kinzler.com). De hecho, una Oráculo en Español fue establecido en Argentina después de mi visita, aunque no estoy seguro de su estado actual[1].

Cuál fue la motivación para construir "vshnu" ??

Esto viene de hace mucho. Yo comencé con Unix en 1982 con Bell Labs Version 7.

Un día obtuvimos una cinta de software cool para agregarle desde Berkeley. Ahí, junto con un nuevo shell llamado "csh", había un shell orientado a pantalla llamado "vsh". Yo cambié de "sh" a estos dos. Ahora adelantando 18 años ... csh prosperó y evolucionó en "tcsh", pero "vsh" está perdido y olvidado por todos excepto por mi. No me quería dar por vencido, pero el código fuente era tan obsoleto que no me era posible seguir portándolo a los sistemas operativos modernos. También recolecté muchas mejoras a las ideas de "vsh". Así que puse a un lado mi holgazanería y abordé una reimplementación completamente nueva. Después de varios experimentos, sabía que Perl encajaría muy bien, aunque sabía iba a ser algo no convencional.

Decidí tratar de hacer este proyecto mi pieza maestra de Perl (suceso que debo dejar a otros juzgar).

Obtuvo alguna ayuda de otros proyectos Open source (código, consejos, horas de programación, etc.) ?

Por supuesto -- ningún proyecto open source se crea en el vacío. Pero el proyecto vshnu está tratando de reclamar algo del territorio perdido y expandir y reintroducirlo, así que la ayuda fue más en término de herramientas, ambiente e ideas prestadas que código, consejos u horas de programación.

Qué habilidades (relacionadas con Perl o no) ganó mientras construía "vshnu" ?

Habiendo aprendido Perl 3 y 4, adopté tardíamente los estilos y convenciones de Perl 5. Con vshnu, verdaderamente traté de escribirlo a la forma de Perl 5, aunque espero a que mucho del viejo estilo vaya desapareciendo.

1 N. del T. : la dirección de e-mail para suscribirse al Oráculo de Internet que figura en http://cgi.cs.indiana.edu/~oracle/intl.cgi#argentina no puede ser alcanzada, con lo que se considera que está inactiva

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

Depende mucho de la personalidad de cada uno. Para mi la visión del proyecto es lo que me guía y me hacer meterme y atravesar el proyecto independientemente de su alcance y los desalentadores standards. Me imagino a otras personas buceando en él y trabajando con otros haciendo una evolución dinámica sería otra aproximación. Por supuesto, manteniéndolo divertido y compensador.

En qué partes del código aconseja mirar a los recién iniciados en Perl, para tener una experiencia educacional placentera ?

Ninguno en particular me viene a la mente. Sugiero desmenuzar ambos archivos (vshnu y vshnucfg.pl) y encontrar lo que les atrape la curiosidad o el interés. Mi estilo de Perl presta mucha atención a la disposición creativa del código y el alineamiento vertical, que encuentro ayuda a crear código correcto y proveer satisfacción estética, aunque a menudo puede romper con las “mejores prácticas”.

Cómo se puede colaborar con "vshnu" ?

Contribuciones y colaboraciones en cualquier forma son bienvenidas, por supuesto.

Hay areas subatendidas porque personalmente no me encuentro con ellas, tales como la integración con otros shells y acciones sobre distintos tipos de archivos con los que no trabajo.

Hay oportunidad para mejorar el diseño en las claves y comandos. También, y ahora que está presente el reconocimiento de eventos de mouse, toda el area de comandos con mouse y acciones está abierta para el diseño e implementación. Más documentación también sería útil.

Qué funcionalidades cree que faltan, y cuáles agregará en el corto plazo ?

Aparte del ya mencionado soporte de mouse, estoy trabajando en mejorar el sistema de ayuda, agrupando los comandos por funciones, y generando una guía de referencia (incluidas las personalizaciones propias).

También estoy considerando opciones por directorio, así como otras ideas misceláneas.

Qué similitudes/diferencias encuentra con proyectos similares ?

Mientras supongo que vshnu podría ser comparado con Midnight Commander, el dired de Emacs y otros, creo que es lo suficientemente diferente en aproximación y diseño como para estar en una categoría propia. La integración de vshnu con un shell común como bash o tcsh es importante, ya que no se tiene por qué perder todo por usar vshnu, debería agregar más poder al que ya se está usando. Yo todavía hago tanto en tsch como en vshnu, y trabajo con ellos como dos modos de un solo ambiente shell.

Qué límites impuso Perl al proyecto ?

Límites? Perl? Eso no conjuga.

Alguna opinión o consejo sobre Perl 6 ?

Aquellos que aman Perl ahora pueden amarlo unas magnitudes más. Los que lo odian pueden hacerlo unas magnitudes más.

Sobre los que están trabajando en él, que se tomen todo el tiempo que sea necesario y que sus almas sean bendecidas.

Algún módulo favorito de CPAN ?

Hmmm, estaría en problemas si Mail::Audit no filtrara mis e-mails.

Algún programador de Perl o miembro de la comunidad favorito ?

Larry Wall, por supuesto. Randal Schwartz por escribir el script que hace el parse de

los votos por e-mail del Oráculo de Internet durante el almuerzo, estudio del cual me puso sobre la curva de aprendizaje de Perl y en su camino desde entonces.

Damian Conway por su genio y por mostrarnos que no hay límites.

Hay algún grupo de Perl cerca suyo ?

http://annarbor.pm.org, aunque no se si actualmente está activo.

Participa en algún grupo de Perl ?

Lo hice en el OSCON de Portland el año pasado, donde todavía Perl es un tema central. Altamente recomendado – ambos, la convención y la ciudad (la convención anual de O'Reilly sobre Perl evolucionó en su convención anual Open Source (Open Source Convention - OSCON).

Cualquier cosa que quiera decirnos y no le preguntamos ?

Quiero reiterar mi agradecimiento a Víctor y el CaFe.pm por compartir estos comentarios sobre Perl y vshnu. Mis mejores deseos para el suceso del grupo y de todos sus proyectos Perl.

El software open source, está listo para la educación ?

Una serie de respuestas a esta pregunta pueden encontrarse en http://insidehighered.com/news/2006/03/01/open, http://www.a-hec.org/open_source_state.html y http://kairosnews.org/inside-higher-ed-how-open. Juzguen por ustedes mismos.

Libros recientes

- *UML 2.0 Pocket Reference* de Dan Pilone (http://www.oreilly.com/catalog/uml2pr/)
- *Intermediate Perl* de Randal L. Schwartz, brian d foy y Tom Phoenix (http://www.oreilly.com/catalog/intermediateperl)
- *Wicked Cool Perl Scripts* de Steve Oualline (http://www.oreilly.com/catalog/1593270623/)
- *Ajax Hacks* de Bruce W. Perry (http://www.oreilly.com/catalog/ajaxhks/)

Usando Ajax desde Perl

Para usar Ajax cualquier programador tiene que adentrarse en JavaScript le guste o no ... y si no te gusta estás de suerte porque lo mínimo indispensable para usarlo desde Perl está en http://www.perl.com/pub/a/2006/03/02/ajax_and_perl.html

GPLv3 sigue dando que hablar

- Se pueden ver videos, transcripciones de charlas y slides en http://gplv3.fsf.org/av
- Comparación con tragedias griegas (http://news.com.com/GPL+3.0+A+bonfire+of+the+vanities/2010-7344_3-6047707.html)
- Diferencias fundamentales con Linus en una entrevista de Forbes (http://www.forbes.com/technology/2006/03/09/torvalds-linux-licensing-cz_dl_0309torvalds1.html)
- Reaction to the DRM clause in GPLv3 (http://www.fsf.org/blogs/licensing/gplv3-drm)
- Free software without the freedom? (http://www.fsf.org/blogs/community/drm-carroll)

Ni Cisco se salva del open source

El producto se llama XORP (http://www.xorp.org/) y es una versátil, flexible y extensible plataforma que sería vendido a partir de Octubre en una máquina Intel dual (ver detalles en http://money.cnn.com/2006/03/03/magazines/business2/telecomopensource/)

Open source es free también de bugs

Realmente ningún software está libre de bugs, pero según el Department of Homeland Security se está en el orden de de los 0,434 bugs cada 1000 líneas de código. Para LAMP el error es aún menor : 0,29 cada 1000 líneas !!! (http://www.zdnetasia.com/news/security/0,39044215,39315781,00.htm)

Parser de nmap

Ha salido un parser de nmap hecho en Perl. Lo pueden encontrar en http://www.net-security.org/software.php?id=532

Estadísticas de uso de lenguajes de programación

Si bien estas cosas hay que tomarlas con pinzas es un buen ejercicio de cómo los se construye un modelo o un ranking (http://jaimecid.blogspot.com/2006/03/tpci-ranking-tiobe-de-lenguajes-de.html)

Se liberó ActivePerl 5.8.8.816

La distribución de Perl para Windows se

puso a la par de la versión salida el mes pasado (5.8.8) y también posee versiones beta para Mac OS X, Windows 64-bits, Linux y SPARC Solaris.

Falleció Slava Bizyayev

Les paso una respetuosa traducción de lo que apareció en use perl aparecida el 14 de Marzo (http://use.perl.org/article.pl?sid=06/03/14/2329207) :

Hace poco murió repentinamente Slava Bizyayev. Slava contribuyó varios módulos a CPAN, incluyendo el popular filtro de compresión para mod_perl Apache::Dynagzip. Dan Hansen, un amigo cercano de Slava, escribió una nota en memoria y pidió que lo compartamos con la comunidad Perl. Como comunidad, sentimos profundamente la ida de los nuestros. Si estás interesado en continuar el trabajo de Slava en CPAN, por favor contacte a la lista de módulos a través de modules@perl.org

XML::Atom::Syndication

Timothy Appnel abrió una lista para discusión y ayuda sobre este módulo y XML::Atom (http://code.appnel.com/changelog/2006/03/000022.html) los interesados pueden suscribirse enviando un e-mail a atomic-perl-subscribe@yahoogroups.com o visitando http://groups.yahoo.com/group/atomic-perl/

Revisión de módulos

El editor de temas Open Source y Free Software de O'Reilly Network (chromatic - http://wgz.org/chromatic/), hizo el review de algunos módulos :

- XML::XPathEngine - http://www.oreillynet.com/onlamp/blog/2006/03/cpan_module_review_xmlxpatheng.html
- KinoSearch - http://www.oreillynet.com/onlamp/blog/2006/03/cpan_module_review_kinosearch.html
- Jifty::DBI : http://www.oreillynet.com/onlamp/blog/2006/03/cpan_module_review_jiftydbi.html

La historia de Tron

Quien no vio esta película, y quien no puede citar de memoria incontables pasajes y anécdotas basadas en Tron. Finalmente tenemos algo así como la versión **E! True Hollywood Story** de Tron en http://www.tomshardware.co.uk/2006/03/16/tron_uk/. No se la pierdan !!!

Open Source analizado

The Economist tiene un artículo donde se analiza , en cierta forma, cómo debe ser un negocio exitoso en el ámbito open source http://www.economist.com/business/displaystory.cfm?story_id=5624944

Otra vuelta de St. Patrick !!

Muchos de nosotros hemos seguido la tradición de la cerveza verde en St.Patrick's day, y sabemos que lo verde es un tinte (o al menos eso nos dicen). Bueno, que mejor que festejar este día con cerveza verde peor de verdad !!!! http://www.wired.com/news/technology/0,70361-0.html?tw=wn_index_4

Comunicación Open Source con dificultades

Parece ser que, según un artículo de NewsForge (http://software.newsforge.com/software/06/03/13/1628249.shtml?tid=150) falta comunicación entre nuestra comunidad y las personas con capacidades disminuidas. A poner manos a la obra, entonces !!

Programadores matando dragones

Una humorada de los más graciosa sobre cómo cumplen un objetivo cada uno de los lenguajes disponibles (no se pierdan el de Perl). El artículo está en http://rebotacion.blogspot.com/2006/03/programadores-matando-un-dragn.html pero si quieren leer el original en Portugués entonces adelante (http://www.amauta.inf.br/index.php?option=

com_content&task=view&id=1048&Itemid=29)

Sobre ofuscación

Un buen comienzo en http://domm.zsi.at/talks/2005_brussels_dark_art_obfuscation/

Frivolity ... lo más !!

Si te interesa crear juegos multiplayer basados en Internet, no te pierdas esta plataforma creada en Perl (http://volity.org/projects/frivolity/)

Perlcast con Audrey Tang

Ya está disponible un nuevo Perlcast, y esta vez con el creador de Pugs. Apunten sus oídos hacia esta dirección : http://perlcast.com/2006/03/29/interview-with-audrey-tang/

Regexp ... ahora recursivas

Vamos, no me vas a decir que no se te ocurrió !! Fijate en http://use.perl.org/article.pl?sid=06/03/30/1337208 y vas a ver que fácil que la hizo Dave Mitchell

Qué está mal con ORM ?

Eso es lo que se preguntó un día Dave Cross, se puso a trabajar y plasmó sus ideas en estos slides : http://dave.org.uk/talks/lpm/2006/orm/

Herramientas de Parrot

Como todo buen lenguaje Parrot tienen sus propias herramientas de compilación. Para adentrarse : http://www.oreillynet.com/onlamp/blog/2006/03/inside_parrots_compiler_tools.html

El mantenimiento, una preocupación

Por qué los lenguajes fallaron en ser aprendidos y mantenidos. Comiencen a disfrutar una serie jugosa en http://www.oreillynet.com/onlamp/blog/2006/03/the_worlds_most_maintainable_p.html

Mordiditas de aquí y de allá

Autor : Víctor A. Rodríguez

Hay algo realmente fascinante, y es cuando un concepto puede ser abordado de no importa cuantas maneras que siempre se llega al mismo resultado : eso es coherencia, robustez y por sobre todo elegancia. Tomemos, por ejemplo, la tarea de hacer la siguiente operación :

$$\frac{3}{4}+\frac{3}{4}$$

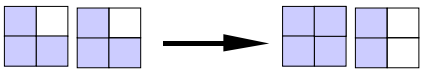
Si decidimos operar con fracciones podemos hacerlo así :

$$\frac{3}{4}+\frac{3}{4}=\frac{6}{4}=1\frac{1}{2}$$

Ahora, si queremos hacerlo con números decimales :

$$\frac{3}{4}=0{,}75\ldots\frac{3}{4}+\frac{3}{4}=0{,}75+0{,}75=1{,}5$$

Pero también podemos decidir hacerlo en forma gráfica donde ¾ significa, nada mas ni nada menos, que de un total de 4 partes (que forman un entero) tomamos sólo 3. A su vez la operación adición es el equivalente a *agrupar* o *agregar*, entonces nos bastará con juntar las partes indicadas y verificar cuál es el resultado :



Ajhá !!!! No importa el camino, el resultado es el mismo : las técnicas son consistentes entre si y con la solución real, que giran todas alrededor de un concepto. Coherencia, robustez y elegancia.

Que tiene esto que ver con Perl 6. Bueno, la verdad que mucho.

En principio podríamos decir que el diseño de Perl 6 se rige por el concepto DWIM[1] donde si una nomenclatura de operadores nos sugiere algo entonces hará lo que realmente nos sugiere (ver apartado **DWIM, el camino del Zen**).

Ahora bien, el concepto de *bloque* está presente en Perl 5 (un bloque está definido entre dos llaves), aunque no siempre de una forma muy consistente. No se asusten que en Perl 6 los bloques siguen existiendo, pero con reglas más consistentes :

- las funciones internas usarán las mismas reglas de parsing que las definidas por cada programa, lo que aumenta la extensibilidad y consistencia del lenguaje

1 Do What It Means : Que hace lo que significa

DWIM, el camino del Zen

Siempre que el decir y el hacer van de la mano no sólo es un signo de coherencia, sino también una buena forma de no ganarse enemigos. Esto trasladado a Perl significa que si una nomenclatura de operadores nos sugiere algo, entonces debe hacer exactamente eso ... aunque a veces deban respetarse ciertas pautas por compatibilidad.
Por ejemplo `@array1 + @array2` nos sugiere que nos va a ser devuelto un array cuyos elementos es la suma de los elementos correspondientes de cada array. Sabemos que en Perl 5 esto significa devolver la suma de la cantidad de elementos de cada array, ya que al evaluarse en contexto scalar un array devuelve la cantidad de elementos que posee. Por compatibilidad en Perl 6 se mantiene esto, pero existe el operador ***element-wise*** que aplica el operador a cada elemento de los arrays. Un verdadero operador DWIM.
Por ejemplo `@array1 >>+<< @array2` devuelve un array donde cada elemento es la suma de los elementos correspondientes de cada array.
DWIM, pequeño saltamontes, y estarás en el camino del Zen.

Mi nombre es BOND

Hay ciertos bloques en Perl 5 que son tan especiales que tienen su nombre propio, siendo los más conocidas :

- **BEGIN** : este bloque se ejecuta tan pronto sea posible, que es ni bien se termina de definir esta y antes que se haga el parse del resto del archivo
- **END** : se ejecuta en último termino, que es cuando se termina de ejecutar el programa y antes que termine el intérprete de perl, aún si termino debido a la ejecución de un `die ()`.

Por ejemplo :

```
 1 #!/usr/bin/perl
 2
 3 my $XML;
 4
 5 BEGIN {
 6    my $file = $ENV{'TREX_CONFIG'};
 7    $XML = XML::Smart->new( $file );
 8 };
 9
10 END {
11    my $bkpFile = $ENV{'TREX_CONFIG'} . ".bkp";
12    $XML->save( $bkpFile );
13 };
```

- Cada bloque es un tipo de closure
- Las declaraciones dentro de un lexical scope tienen efecto desde el momento de su declaración hasta el final del bloque, lo que implica que no podemos tener construcciones definidas dentro de un bloques y que sean traspasadas a otro (ver el **Mordiditas...** anterior publicado en nuestro site http://cafe.pm.org/boletines/CaFePerl-Issue0b.html#mordiditas)

Estas son las novedades a las que se refiere Perl 6 para las declaraciones, pero en lo que es operaciones de control del flujo del programa que permitan salirse de los bloques de ejecución, esto aún deberá ser permitido. Un clásico ejemplo de estas son `return`, `next`, `last`, `redo`, y `die` . Estas pueden ser vistas como excepciones a tratar en el tema de los bloques. Excepciones ... me suena ... si, lo vimos hace como 8 meses en otro **Mordiditas...** (ver http://cafe.pm.org/boletines/CaFePerl-Issue03.html#manejo_de_excepciones__mordiditas_de_aqu%ED_y_de_all%E1_)

Pero que tiene que ver esas excepciones con estas ?? No es un mecanismo un poco más complejo ?? En realidad la respuesta es que son excepciones al flujo normal (igual que las que vimos en ese artículo) pero que además son un poco más simples ya que `return`, `next`, `last`, `redo`, y `die` vuelven siempre a un punto bien determinado en tiempo de ejecución (`return` lo hace a la próxima instrucción de donde fue llamada una sub, `next` y `redo` al principio del loop, `redo` al final del loop y `die` al final del programa), con lo cual establecer un mecanismo más complejo y después, para estas últimas, usarlo en forma degradada no suena tan mal. Después de todo en Perl 5 no hay un mecanismo de manejo de excepciones del estilo mostrado en el **Mordiditas...** mencionado anteriormente, sino que está implementado a través del módulo `Error.pm` (el manejo nativo de excepciones en Perl 5 se hace a través de `die()` y `eval()`).

Haciendo un repaso, porque lo vamos a necesitar, hay una serie de bloques que por su importancia tienen nombres propios y que cada uno tiene su rol específico (ver apartado **Mi nombre es BOND**).

Perl 6 no ajeno a este y tampoco se queda atrás sino que por el contrario definiendo una serie de nuevos bloques, y basándose en los tres reglas provistas anteriormente para los bloques, se arma un mecanismo para el manejo de excepciones. Los bloques que define para este caso son :

- **CATCH** : cuando en un bloque surge una excepción automáticamente se pasa el control al bloque CATCH que esté definido dentro de este
- **LEAVE** : una vez que se ejecuta el bloque, se haya producido o no una excepción, se ejecuta este bloque
- **ENTER** : similar a **LEAVE** pero para el inicio del bloque

Entonces la forma de manejar/atrapar una excepción sería :

```
try {
    codigoQueGeneraExcepcion();

    CATCH {
        # código que maneja
        # la excepción
    };

    LEAVE {
        # una vez finalizado los
        # bloques try o CATCH
        # se ejecuta LEAVE
    }
}
```

Un ejemplo vale más que mil declaraciones

No vamos a tratar aquí la teoría de objetos, sino simplemente vamos a dar una mínima introducción de su uso dentro de Perl 6, con algunos elementos para aprovecharlos dentro del manejo de excepciones. Supongamos que necesitamos definir un tipo de objeto básico que me permita colocar notas en cierta posición de la pantalla, algo similar a un Post-It (TM).

```
 1 class postit {
 2    has $.x is rw;
 3    has $.y is rw;
 4    has $.text is rw;
 5
 6    method clear () {
 7        $.x = 0; $.y = 0; $.text = "";
 8    }
 9 }
10
11 my $sticker = postit.new(
12     x => 20, y => 30,
13     text => "Llame ya!"
14 );
15
16 $sticker.meta.does(postit);   ## true
17 $sticker.meta.does(banner);   ## false
```

Comparado con Perl 5 hay ciertas diferencias :

- se usa class y method en lugar de package y sub
- no se usa un hash para acceder a los atributos, sino que están impílcitos dentro del objeto
- los atributos y métodos se acceden a través del objeto interponiendo un punto (.)
- No es necesario generar un constructor

Adicionalmente los objetos poseen ciertos métodos llamados metamétodos (algo asi como *generados por default*) que nos permiten trabajar sobre ciertas bases comunes. Por ejemplo . does nos indica si el objeto (`$sticker`) satisface la interfaz del objeto mencionado (`postit` o `banner`).

Es notable el cambio de sintaxis de `try{} catch{}`, propuesto en `Error.pm`, a `try{ CATCH{} }` mayormente impulsado por la imposibilidad definir construcciones en un closure/bloque y usarlas en otro.

Hasta acá no tenemos mas que una vista a vuelo de pájaro de como manejar las excepciones, pero nos quedan varias dudas :

- Cómo genero una excepción ?
- Cómo se qué excepción se generó ?
- Cómo la atrapo dentro del bloque CATCH ?

Básicamente cuando en Perl 5 ejecutamos un `eval()` cualquier error generado en el código será puesto en la variable `$@` y así podemos detectarlo y actuar en consecuencia, pero en Perl 6 esto sufre dos cambios :

- se general la clase `Exception`, que sirve para definir objetos que van a ser usados como excepciones
- el objeto usado para definir la excepción es pasado dentro de CATCH como `$!`

El problema a partir de aquí es que al definirse las excepciones como objetos necesitamos recostarnos en forma extensiva sobre estos y su implementación en Perl 6. Para esto vamos a tomar sólo algunos conceptos y diferir el tratamiento de objetos para otra edición de **Mordiditas...** (o ver un adelanto en el apartado **"Un ejemplo vale más que mil declaraciones"**)

En definitiva un ejemplo más completo sería :

```
class Err::FileOpen
   is Exception {...};

try {
    my $fh;
    open( $fh, $file ) ||
        die Err::FileOpen;

    CATCH {
        when Err::FileOpen {
            print "Oops ... " .
                "no existe $file";
            exit;
        };
    };
}
```

Entonces vemos que las tres preguntas sobre las excepciones están respondidas en este código, donde la excepción se puede generar con `die()` (también con `fail()`), usando clases especiales predefinidas llamadas `Exception`, y las detecto con un bloque `CATCH` el cual por default posee en la variable `$!` el objeto de excepción (de clase Exception) y con la función `when()` se pregunta si el objeto `$!` es de la clase `Err::FileOpen`, que en nuestro caso nos indica que hubo un error al abrir un archivo y por lo tanto la lógica que deseamos seguir es terminar el programa.

Ahora que sabemos como manejar una excepción vamos a adentrarnos un poco más en cómo funciona el bloque `CATCH`. En realidad la función `when` lo que realiza es un match de forma inteligente, de tal forma que si como argumento pasamos un string tratará de convertir `$!` en string, si es un nombre de objeto preguntará usando `$!.meta.does`, etc. Lo que llamamos algo inteligente. Ahora bien cómo hacemos para poner data en el objeto `$!` ?? Basta con asignarlo ?? Bueno. Algo así. Para esto usamos la función `given()` que nos realiza esta asignación en forma inteligente. Por ejemplo :

```
given $! {
    when Err::FileOpen { ... };
    when Other::Error { ... };
    when /not found/ { ... };
};
```

Lo cual si lo miramos un poco lo que estamos haciendo es, dado un objeto, decidir el hacer ciertas acciones dependiendo que cumpla con ciertas condiciones. Entonces, no se parece esto a un case ?? La verdad es que la acción conjunta entre when() y given() nos da exactamente eso, una funcionalidad como el case pero pero con vitaminas ... y muchas !!!

Entonces saben qué : *"Hay algo realmente fascinante, y es cuando un concepto puede ser abordado de no importa cuantas maneras que siempre se llega al mismo resultado : eso es coherencia, robustez y por sobre todo elegancia. "*

Infografía

- Apocalypse 4 : "Syntax" (http://dev.perl.org/perl6/doc/design/apo/A04.html)
- Exegesis 4 (http://dev.perl.org/perl6/doc/design/exe/E04.html)
- Synopses 4 => "Blocks and Statements" (http://dev.perl.org/perl6/doc/design/syn/S04.html)
- RFC 88 (Perl 6) : Omnibus Structured Exception/Error Handling Mechanism (http://dev.perl.org/perl6/rfc/88.pod)
- Perl CodeFetch (http://perl.codefetch.com/)