Periódico de la Comunidad Perl de Capital Federal

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

CaFe Perl v0.B

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

Editorial

Hola Perl Mongers, bienvenidos a un nuevo número de CaFe Perl !!

Esta vez estamos ante un número bizarro como pocos. No me creen ??

- en el PoC de este mes Martín Ferrari encontró esoterismo, travestismo y magia negra en los one-liners
- Una entrevista a un programador de Perl sobre Mac OS X que los va a dejar embrujados
- Nos perdemos entre los loops y las cintas de Möbius de Perl

Todavía incrédulos ?? empiecen a leer este número con una estaca de plata cerca y una ristra de ajo en la ventana ... y después no digan que no tuvieron aviso.

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

Víctor A. Rodríguez (Bit-Man)

POC (peace of code)

Por Martín Ferrari

Todos los días se aprende algo nuevo, dice el dicho. Pero esto se torna casi literal con ciertas cosas. En mi experiencia, VIM y Perl no dejan de sorprenderme. Como lo que nos interesa acá es extraer jugo de joroba de camélido, hablaremos de lo segundo.

El otro día nuestro querido Víctor me acercó un post en PerlMonks (lo conocen?) sobre one-liners, para darme ideas para la PoC de este mes, y leyendo comentarios me enteré de algunas opciones de línea de comandos que harían encolerizar a todos los enemigos de Perl.

Esoterismo

Para empezar,

```
perl -pe1 archivo1 archivo2
```

"EH?" Me dirán. "Es otro reemplazo de cat!" Les diré.

Veamos en detalle. Estamos pasando las opciones `-p` y [-e 1]. La opción `p` tiene mucha magia detrás: encierra nuestro programa en un bloque de este tipo (info sacada de perlrun(1):

```
LINE:
 while (<>) {
   ...             # Acá va el programa
 } continue {
   print or die "-p destination: $!\n";
 }
```

Como resultado, cada línea de la entrada es impresa luego de pasar por nuestro programa, que no hace nada. De manera análoga, y siguiendo la sintaxis de sed, la opción `-n` itera sin imprimir por defecto. Poniendo esto alrededor de nuestro programa:

```
LINE:
 while (<>) {
   ...             # Acá va el programa
 }
```

Hasta hoy, yo hacía cosas como:

```
perl -e 'while(<>) { s/sasa/lala/gi; print }'
```

Pero es mucho mejor así:

```
perl -pe 's/sasa/lala/gi'
```

Travestismo

Y el camello de 2 toneladas sigue haciéndose pasar por el ratoncito sed, con la opción `-i` introducida recientemente en GNU sed, para hacer la edición in-place, es decir, no de standard input a standard output, sino dentro de cada archivo nombrado. De manera opcional nos permite decirle que haga un backup:

```
perl -i -pe 's/sasa/lala/gi' archivo
perl -i.bak -pe 's/sasa/lala/gi' archivo
```

Atención con no mezclar la opción `-i` con otras, ya que lo que esté a continuación se toma como la extensión a agregarle al archivo original para guardar un backup. Para imprimir rangos, igual que en sed:

```
perl -ne 'print if /^DESDE$/ .. /^HASTA$/'
```

Para parecerse al viejo y querido awk (sabe usted qué significa el nombre awk?), tenemos la opción `-a`, autosplit, que nos deja en @F el contenido de cada línea cortado en los espacios en blanco, o lo que pongamos como argumento de la opción `-F`:

```
perl -lane 'print "$F[0] $F[3]"'
perl -F: -lane 'print "$F[0] $F[4]"' /etc/passwd
```

Otra interesante: `-l`, en combinación con `-n` o `-p` recorta los enter (como `chomp`) a la entrada y vuelve a ponerlos en la salida.

Magia negra

Aprovechándose de conocer exactamente el código que se agrega al usar la opción `-n`, hay gente que hace cosas como esta:

```
perl -lne'$x+=$_}{print$x'    # sum
perl -lne'$x+=$_}{print$x/$.' # avg
```

Claro que podríamos ser más elegantes y claros, y hacer lo que nos recomienda perlrun(1) y usar los bloques BEGIN y END, poco conocidos pero no tan esotéricos:

```
perl -lne '$x += $_; END { print $x }'
```

Noten la ausencia de `\n` en el `print`.

Este me gusta por simpático e inesperado, la gracia es darse cuenta de las opciones que estamos pasando. Además es útil, claro, como intérprete interactivo:

```
perl -demo
```

Más ejemplos

Uno que yo siempre hago en bash:

```
for i in *.JPG; do mv -i "$i" "${i/.JPG/.jpg}"; done
```

En perl puede hacerse así:

```
perl -le'foreach(<*.JPG>) {$o=$_;s/JPG$/jpg/; rename $o, $_
unless -e }'
```

Sí, yo también prefiero seguir haciéndolo en bash :)

W00t!

```
perl -i.bak -pe 's/Mozilla/Slopoke/g' /usr/bin/netscape
```

Palíndromos (palabras capicúa):

```
perl -lne 'print if $_ eq reverse' /usr/share/dict/spanish
```

Nota: hay código robado sin culpa del artículo de PerlMonks y de la lista de one-liners de Tom Christiansen

PERlitas

Entrevista y traducción realizada por Víctor A. Rodríguez

Esta vez es la hora para la plataforma Apple y Mac OS X, así que es tiempo para PerlPad (http://perl-pad.sourceforge.net/), que ofrece ejecución de código Perl como un servicio de Mac OS X (system service). De esta forma se puede ejecutar código Perl desde casi cualquier aplicación Mac OS X.

Pero dejemos a su creador, Thilo Planz, contarnos su historia

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

Soy un hacker Alemán en mis último, muy último veinte años. Después de graduarme de la Universidad con un Master en Ciencias de la Computación y otro en Business IT, me mudé a Tokyo a trabajar para una compañía que crea websites de comunidades. En ese entonces fui introducido a Perl, y llegué a preferirlo sobre Java (que fue el lenguaje principal usado en la Universidad)

Encontró algún otro lenguaje similar a Perl en la Universidad ?? Aprendió alguna teoría que más tarde fuera empleada en Perl ??

La programación en la Universidad fue casi exclusivamente en Java, así que ahí no tuve exposición a los lenguajes de scripting.

Aunque que eran específicos de Perl, los cursos de Ingeniería de Software y temas como las introducción a estructuras de datos (lo que significa que realmente se qué es un hash y cómo funciona antes de usarlo en Perl) han sido valiosos, por supuesto.

Cuál fue la motivación para construir "PerlPad" ??

El aprender y jugar con nuevas herramientas. Quería aprender cómo crear aplicaciones Cocoa para Mac OS X[1]. Adicionalmente quería tener alguna especie de calculadora/herramienta de texto/línea de comandos con capacidades Perl.

Obtuvo alguna ayuda de otros proyectos Open Source ?? (código, consejos, etc.)

PerlPad no habría sido posible (y yo no habría tenido la idea para hacerlo) sin CamelBones[2]. Este es un bridge entre Perl y Objective-C[3] y permite realizar aplicaciones Cocoa completamente en Perl. Provee una increíble cantidad de magia “detrás de escena”

Qué ambiente de desarrollo utiliza ??

Para PerlPad uso Xcode de Apple (ver más abajo).

Para otras tareas de edición Perl, simplemente un editor de texto común, cualquier cosa que pueda editar múltiples archivos y resaltar la sintaxis. Me gusta jEdit.

Cunado hago aplicaciones web, la plataforma Mozilla tiene herramientas grandiosas (DOM

1 Cocoa es un framework orientado a objetos para realizar aplicaciones Mac OS X
2 http://camelbones.sourceforge.net/
3 Lenguaje C orientado a objetos (http://en.wikipedia.org/wiki/Objective_C) mayormente usado en Mac OS X y GNUstep

inspector, debugger de JavaScript, muchos plugins).

Prefiere un IDE Xcode, o cree que el "estado del arte" de IDEs Perl es suficiente ??

Uso Xcode para PerlPad sólo porque es el ambiente mejor soportado para el desarrollo en Cocoa. Camelbones viene con plantillas para este, las integra con Interface Builder, puede crear y ejecutar makefiles que juntan todo en una aplicación Mac.

Para los proyectos que no son Cocoa, alcanza con un simple editor de texto, especialmente si no hay necesidad de hacer un script (y con Perl usualmente no la hay)

Qué habilidades (relacionadas con Perl o no) obtuvo mientras construía "PerlPad" ??

Del lado de Perl, tuve que empaparme con tablas de símbolos, y cómo funciona el sistema Unicode.

Del lado de Mac, aprendí cómo trabajar con las herramientas de desarrollo de Apple, el API Cocoa y algo de ObjectiveC.

Qué consejo le daría a los futuros diseñadores y hobistas que afrontan un nuevo pryecto ??

Creo que "release often, release early" es un consejo que suena bastante para los programadores hobistas. Siempre temí que esto llevaría a código no maduro y proyectos abandonados, y que alguien eventualmente podría esto en mi contra. Pero esto no ocurre. Todos saben que los proyectos de hobistas son proyectos de hobistas. Algunos se llegan a completar o ni siquiera a levantarse del piso en primer lugar. Pero si no lo mostrás al mundo, perdés la oportunidad de generar el interés que podría hacer de tu proyecto uno realmente exitoso.

En resumen, seguí adelante, sacá tus cosas a la calle. No hay nada más gratificante que la respuesta de los usuarios.

En qué parte del código le aconsejaría mirar al programador que se inicia en Perl, para tener una experiencia de aprendizaje placentera ?

No estoy seguro que el código de PerlPad sea muy bueno desde el punto de vista de ingeniería de software o educacional, realmente. Comenzó desde una plantilla de una aplicación "Hello World" y orgánicamente (opuestamente a "de acuerdo al plan") creció desde ahí.

Cómo se puede colaborar con "PerlPad" ??

Es un proyecto muy pequeño, tanto en tamaño de código como en gente (sólo yo, básicamente), así que la colaboración puede ser completamente sin ceremonias. Simplemente envíenme ideas, reportes de bugs, pedazos de código, preguntas a mi o a la lista de correo[1].

Qué funcionalidades cree que no están, y cuáles se agregarán pronto ??

1 http://sourceforge.net/mail/?group_id=72650

“Pronto” es un concepto difícil aquí. El desarrollo ocurre en ráfagas.

0.2.2 con compatibilidad Intel está finalizado, solamente espera por los testers.

He estado con una versión 0.3 casi finalizada por dos años. Sólo necesita arreglos en la documentación, y algo de testing final que, por supuesto, son tareas un tanto aburridas, y como tales se retrasan una y otra vez... La versión actual (que yo mismo uso) está disponible como un “nightly build (más bien mensual)

Las nuevas funcionalidades principales son una interfaz de metal bruñido[1], un sistema de captura de expresiones regulares y manejo automático de pedazos de código (mayormente eliminando la necesidad de mantener manualmente un archivo de arranque para PerlPad).

Lo que eventualmente quisiera tener es integración con AppleScript. Es una gran forma de controlar otras aplicaciones. Se puede, por ejemplo, decirle a iTunes que ejecute cierta canción. Denle una mirada a Glue::Mac para ver que es posible hacer.

Que similitudes/diferencias encuentra con proyectos similares ??

Hay algunos programas que evalúan código a través de los Servicios del Sistema (tales como el servicio de AppleScript). Ninguno de ellos para Perl, creo, y todos trabajan de una sola vez, mientras que PerlPad recuerda las variables globales entre invocaciones.

Qué límites impuso Perl al proyecto ??

Todo lo contrario. PerlPad sólo pudo ser escrito en Perl, ya que es básicamente una capa sobre el intérprete de Perl.

En tanto que para hacer aplicaciones Cocoa, realmente no hay límites, gracias a CamelBones se puede hacer en Perl todo lo que se hace con ObjectiveC (el ambiente nativo).

Alguna opinión o consejo relacionado con Perl 6 ??

Antes que nada, aún estamos en tiempo muerto[2]

Entonces, creo que es tan diferente a Perl 5 que debería ser considerado otro lenguaje. Perl 6 no reemplazará simplemente a Perl 5 de la forma que 5.6 reemplaza a 5.8, van a coexistir por mucho tiempo, lo que significa que sus habilidades de Perl 5 van a permanecer válidas.

El proyecto de Perl 6 es la parte más excitante de la comunidad Perl, y atrae a gente extremadamente lista. Así que si querés aprender algo, andá y jugá con él, realmente hay algunas cosas cool. Pero específicamente para Perl programadores que se inician es probablemente demasiado confuso.

Personalmente, la parte con la que estoy más interesado es en la máquina virtual Parrot, que está siendo desarrollada para Perl6. También soportará Perl5, y muy seguramente Python, Ruby y PHP también. Esto debería poner a la fracturada comunidad de scripting en una mejor posición contra Java y .NET.

Algún módulo favorito de CPAN ??

Como una persona de base de datos, voy a decir “DBI”. Hace un gran uso de las flexibilidades inherentes en Perl para minimizar la cantidad de código que hay que escribir

1 (N. del T.) brushed-metal

2 (N. del T.) original : “time off”. También puede interpretarse como “entretiempo”

para disparar sentencias SQL. En contraste con la verborragia y torpeza de JDBC.

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

Sherm Pendley por crear CamelBones

Audrey Tang por revivir el interés en Perl 6

Tim Vroom por perlmonks.org

Hay algún Perl Group en su ciudad ??

Tokyo Shibuya PerlMongers http://shibuya.pm.org/

Participa en alguno ??

Voy a las charlas de Shibuya.pm, pero soy sólo uno en una audiencia de más de cien.

Alguna experiencia (graciosa o no) que tuvo mientras construía "PerlPad" y que quiera compartir con nosotros ??

Lo más sorprendente que me pasó fue que un Perl Monger de Argentina quisiera entrevistarme. Me siento adulado ;-)

SudorNews

Nuevo Pugs 6.2.11

A principios de Febrero Audrey Tang liberó esta nueva versión de Pugs. Pueden bajarlo desde http://pugscode.org/dist/Perl6-Pugs-6.2.11.tar.gz y ver el ChangeLog (con todos los cambios todos) en http://pugs.blogs.com/pugs/2006/02/changes_for_pug.html

... y también de Perl !!

Casi enseguida salió a la luz la versión 5.8.8 y puede bajarse desde ftp://ftp.cpan.org/pub/CPAN/src/perl-5.8.8.tar.gz . Antes de bajarlo tengan en cuenta que este es un release sólo del source, con lo cual para generar el binario deben tener un ambiente de desarrollo del lenguaje C instalado (típicamente gcc, make y herramientas por el estilo)

Advanced Perl Programming

Este nuevo libro de Simon Cozens toma la primer versión de este libro (que data de cuatro años atrás) y la actualiza de una manera más a como hoy en día se usa Perl con la ayuda de CPAN. Pueden ver un análisis completo, capítulo a capítulo, en http://www.perl.com/pub/a/2006/01/26/more_advanced_perl.html

Probando aplicaciones gráficas en X11

Una cosa es automatizar las pruebas de cada sub y cada parte del código que hacemos en Perl, o cualquier otro lenguaje, y otra muy distinta es automatizar la prueba de una interfaz gráfica. Para esto nada mejor que leerse este artículo (http://www.perl.com/pub/a/2006/02/02/x11_gui_testing.html) y usar el módulo X11::GUITest ... al menos para empezar. También para los que tengan la necesidad existe el módulo Win32::GUITest

GPLv3

Y nuestro amigo sigue dando que hablar :

- Linus Thorvalds explicó qué es lo que no le gusta de GPLv3, y básicamente tiene una base muy sólida, y buenos argumentos, para discutirlo y hacer una comparación con Creative Commons (http://trends.newsforge.com/article.pl?sid=06/02/02/1636216 y http://linux.slashdot.org/article.pl?sid=06/01/26/1418220&tid=190)
- Por otra parte Richard Stallman no adhiere a Creative Commons, o por lo menos a algunas de las licencias que están bajo este nombre (http://www.linuxp2p.com/forums/viewtopic.php?p=10771)

Juegos con Perl y SDL

La primera entrevista de CaFe Perl fue a Guillaume Cottenceau por la creación de su juego Frozen Bubble (http://www.frozen-bubble.org/), y ahí conocimos que es SDL. Ocurre que en

este mes Arstechnica publicó una muy completa guía de como hacer juegos con las mismas herramientas de Frozen Bubble : Perl y SDL (http://arstechnica.com/guides/tweaks/games-perl.ars)

Debugging y profiling de aplicaciones mod_perl

Programar en Perl es una cosa, con mod_perl requiere desarrollar ciertas habilidades, pero hacer debugging con mod_perl es algo de otro mundo. Un artículo sobre este tema (http://www.perl.com/pub/a/2006/02/09/debug_mod_perl.html) hace que no sólo no sea vea tan difícil sino que nos abre las puertas un poco más hacia el mundo de mod_perl y el mundo de la performance.

Desarrollo Open Source por Mark Shuttleworth

Si, ese Mark Shuttleworth (el de Ubuntu) tuvo una experiencia adicional con el movimiento Open Source, comenzó a comprender que la necesidad es una de las esencias de nuestro movimiento y que matando esa característica no importa cuanto se gana programando, simplemente ya no queda por qué seguir adelante (http://mscom.rabbithole.co.za/archives/4)

Oracle adquiere Sleepycat

Una vez más este gigante hace lo que se le da la gana, y compra a esta empresa famosa por al creación de BerkeleyDB. No es la primera compra de una compañía que genera software open source porque compró (hace un año) Innobase que es quién creó InnoDB el cimiento de MySQL.

Entrevista a uno de los inventores de la ENIAC

Este año es el 60 aniversario de la creación de ENIAC y hay una interesante entrevista en http://www.computerworld.com/hardwaretopics/hardware/story/0,10801,108568,00.html?source=x10 a J. Presper Eckert, uno de sus inventores.

Podcasts del mes

Esta vez, los que puede apreciarse (y escucharse) son :

- bran d foy habla de sus libros “Intermediate Perl” y “Mastering Perl” (http://perlcast.com/2006/02/16/brian-d-foy-on-intermediate-perl-and-mastering-perl/)
- Tom Limoncelli, autor del libro "Time Management for System Adminstrators" (http://perlcast.com/2006/02/09/interview-with-tom-limoncelli/)

FLISOL Perú

Perú será este año sede del Festival Latinoamericano de Instalación de Software Libre (FLISOL). Más información en http://peru.flisol.net/

Estructuras de Datos Enriquecedoras

Seguro que más de una vez tuviste que guardar un hash, o alguna estructura simple de Perl, y recurriste a alguna base de datos como PostgreSQL o MySQL, aún a sabiendas que una estructura relacional a veces puede ser un estorbo. Si querés saber un poco más de cómo hacerlo fijate en http://www.perl.com/pub/a/2006/02/16/mldbm.html

La ONU recomienda el uso y fomento del software libre

Esta es una de las afirmaciones que fue hecha en la clausura de la II Conferencia Internacional de Software Libre celebrada en Málaga (http://www.opensourceworldconference.com/malaga06/es/modules/wiwimod/).

Parrot 0.4.2

Como ocurre cada mes y medio salió la nueva versión de Parrot (http://www.parrotcode.org/) la máquina virtual que ejecutará Perl 6 y otros lenguajes dinámicos. La podés bajar desde http://www.cpan.org/authors/id/L/LT/LTOETSCH/parrot-0.4.2.tar.gz

Mejoras a Perl 5

Para el primer trimestre del 2006 la Fundación Perl sólo ha otorgado fondos para un proyecto : la mejora de Perl 5. Un sólo proyecto parece poco, pero se vana considerar una sustancial cantidad de mejoras. Podés obtener más información en http://news.perlfoundation.org/2006/02/2006_q1_grant_votes.html

DJ, no me enseñás un poco de Perl !!

Aunque no es el último grito de la moda, ni fue escrito hace poco, este artículo tiene un costado interesante : generar música en vivo a través de código Perl (http://www.perl.com/pub/a/2004/08/31/livecode.html)

Técnicas avanzadas para subrutinas

El título debería ser “Cómo puedo hacer para pasar valores a una sub y eventualmente validarlos” (algo largo por cierto) pero los editores de perl.com seguramente prefirieron algo más conciso (http://www.perl.com/pub/a/2006/02/23/advanced_subroutines.html)

Las grandes empresas también pueden aprender del Open Source

O al menos a esto se refiere este largo e interesante artículo (http://www.onlamp.com/pub/a/onlamp/2006/02/27/what-corp-projects-learn-from-open-source.html). No dejen de pegarle una leída, y se van a encontrar conque pueden estar de acuerdo en muchas cosas ... y en otras tantas no.

Mordiditas de aquí y de allá

Statements and declarations

A mi no me gusta dar vueltas (`next`), y vueltas (`next`), y vueltas (`next`), y vueltas (`next`) ... bueno, al menos no sin control (`last`).

Recuerdo que lo primero que aprendí de lenguajes de programación fue el if (en **Basic**), y lo que más me fascinó fue el uso de subrutinas, variables locales y recursión (todo aprendido de golpe y en **Pascal**), y ante tanta potencia los loops pasaron desapercibidos, casi nada más que para procesar arrays. En **Perl** los loops toman otro tinte, algo así como seres de una morfología apenas delineada.

Sin ir más lejos un loop es algo así como la famosa cinta de Möbius (http://mathworld.wolfram.com/MoebiusStrip.html) pero con algo más de control sobre cuándo finalizar el loop, cuántas medias vueltas se le pueden dar a la cinta y cuan rápido puedo ir sin salir volando ... como si fuera una pista de carrera de autos.

Como siempre, o como casi siempre, un loop consta de una sección de inicialización, una condición de parada y una sección de control que se ejecuta al final de cada iteración. En pseudocódigo :

```
INIT()
loop( CONDICION ) {
      hacemos_lo_que_debemos_hacer
      CONTROL
}
```

Escrito en buen perl, y ejemplificado con un while :

```
1     $i = 0;
2     while ($i < 10) {
3           # hacemos lo que debemos hacer
4           $i++
5     }
```

Claro, así es simple sencillo y claro, pero nada me impide escribir lo siguiente :

```
1     $i = -1;
2     $varNonSancta += 3;
3     @numFlat = split /:/, $strLongFromFile
4     $processNum = $numFlat[ ++$i ];
5
6     while ($i < 10) {
7           processElement ( $processNum );
8           $prev = $numFlat[ $i ];
9           $processNum = $numFlat[ ++$i ];
10    }
```

Entonces ... dónde han quedado nuestras tan bonitas estructuras de control ? Y nuestra inicialización ? Y a sus elementos, Señor, que les pasa, odian los índices, odian el control ? Ocurre que la inicialización ha quedado diluida entre las líneas 1 y 4, la condición aún está intacta en la línea 6 del `while`, y la parte de control en la línea 9 entremezclada con un acceso a un elemento de un array.

Por suerte en la comunidad Perl hay gente muy inteligente, y en la comunidad de C también,

que usan la construcción `for`:

```
for( INIT; CONDICION; CONTROL ) {
      hacemos_lo_que_debemos_hacer
}
```

Y todo queda muy bien empaquetadito y prolijito. Entonces nada, queda fuera de discusión que hay que usar `for` para hacer loops y listo pero todos sabemos que, además de que en Perl TMTOWTDI, las otras formas de loop (como `while` y `foreach`) tiene sus propósito y su encanto. Para empezar vamos a reescribir el último pedazo de código con un `for` :

```
1     $varNonSancta += 3;
2     @numFlat = split /:/, $strLongFromFile
3
4     for ( $i = -1, $processNum = $numFlat[ ++$i ];
5           $i < 10;
6           $processNum = $numFlat[ ++$i ] ) {
7
8           processElement ( $processNum );
9           $prev = $numFlat[ $i ];
10    }
```

También convengamos que aunque todas las estructuras están en su lugar este loop necesitaría una buena reescritura, pero básicamente lo que quiero mostrar es que aunque sea complicado se pueden juntar todas las estructuras del loop en el `for` y así y todo dar una mejor idea de cómo hacerlo. Pero vamos, que con `while` (o sus contrapartidas `do/while` y `foreach`) se posee de una cláusula `continue`, puesta al final de los mismos, que permite realizar una tarea similar a la del bloque CONTROL, y es ejecutada justo antes de re-evaluar CONDICION, excepto en la primer evaluación de este último (esto ocurre cuando se llega al final del loop o cuando se ejecuta la sentencia `next`).

Por ejemplo, el código anterior puede reescribirse como :

```
1     $varNonSancta += 3;
2     @numFlat = split /:/, $strLongFromFile
3
4     $i = -1;
5     $processNum = $numFlat[ ++$i ];
6
7     while ( $i < 10 ) {
8           processElement ( $processNum );
9           $prev = $numFlat[ $i ];
10    } continue {
11          $processNum = $numFlat[ ++$i ];
12    {
```

Ahora esto tiene un poco más de color, pero cuidado porque tanto `while` como `continue` no comparten el scope, por lo que si el scalar `$prev` estuviera definido local al `while` (y este fuera accedido dentro del bloque `continue`), esto daría un error en tiempo de compilación en case de utilizarse "`use strict`" :

```
Global symbol "$prev" requires explicit package name at test00.pl
line 30.
Execution of test00.pl aborted due to compilation errors.
```

Aunque `continue` no sólo puede adjuntarse a una sentencia de loop sino simplemente puede hacerse a un bloque, con lo cual podemos hacer cosas muuuuy locas como :

```
1     {
2           print ".";
3     } continue { redo };
```

y tener un hermoso loop infinito ya que al ejecutarse el `print` va al bloque `continue` que indica que se inicie otra vez el bloque **pero** sin evaluar la condición inicial (que no la hay) con lo cual ejecuta el bloque indefinidamente. Otra forma mucho más rara, y también mucho más propensa a errores es hacer lo siguiente :

```
1     {
2           print "Se ejecuta sólo una vez";
3     } continue {
4           print ".";        ## main body
5           next
6     };
```

En esta el cuerpo principal se ejecuta una sola vez, en cambio el bloque que acompaña a `continue` se ha vuelto el bloque principal. Si lo queremos convertir en un loop, pero no infinito :

```
1     my $i;
2
3     {
4           $i = 0;
5           print "Se ejecuta sólo una vez";
6     } continue {
7           print ".";  ## main body
8           last if ($i >= 10);
9           $i++;
10          next;
11    };
```

Como ven esto es un poco más pedestre, ya que uno mismo tiene que hacer cumplir la condición (línea 8), pero realmente no es gran cosa. Aunque recuerden, que lo más probable es que este código no lo termine cambiando quién lo hizo sino alguien más. O te puede pasar algo peor : podés tener que hacerlo vos mismo después de un tiempo, y ni siquiera entender lo que escribiste. Eso es tomar de tu propia medicina.

En Perl 6 los loops cambian un poco. Todavía uno puede reconocer uno en Perl 6, pero igual están algo cambiados. Las sentencias `while` permanecen igual y los modificadores `next, last y redo` también pero :

- la cláusula `continue` desaparece, y es reemplazada por un bloque NEXT
- ya no es necesario el uso de paréntesis para delimitar la CONDICION
- en lugar de `do/while` ahora se usa `loop/while` (esto evita que en un bloque muy grande donde ambas quedaban muy separadas no se supiera si se trataba de un loop o no)
- la sentencia `loop` también trabaja como el `for` del lenguaje C
- la sentencia `for` actúa como `foreach` de Perl 5
- `foreach` no existe en Perl 6

- un bloque `{ ... }` ya no es un loop cde una sola ejecución, ahora lo es `do { ... }`. Un bloque ahora es exclusivamente un closure, que nos delimita un scope.

Parece como si alguien se hubiera querido divertir cambiando una sentencia por otra, esta por la siguiente y al última haciéndola hacer malabares nuevos. Pero créanme que no es así, todo está orquestado como para quitar algunos vicios o permitir que cada una haga lo que tenga que hacer (DWIM – Do What It Means), brindar consistencia y dar espacio a nuevas habilidades.

Por eso, en el próximo número de CaFe Perl, esta sección va a estar por entero dedicada a Perl 6 y el manejo de bloques, loops y demás alimañas del Nuevo Mundo !!!

Infografía

[1] Codefetch (http://perl.codefetch.com/)

[2] Wikipedia (http://en.wikipedia.org/wiki/Perl_control_structures)

[3] Perlmonks (http://www.perlmonks.com/?node_id=496105)

[4] Apocalypse 4 – "Syntax" (http://dev.perl.org/perl6/doc/design/apo/A04.html)

[5] Exegesis 4 (http://dev.perl.org/perl6/doc/design/exe/E04.html)

[6] Synopses 4 => "Blocks and Statements" (http://dev.perl.org/perl6/doc/design/syn/S04.html)