Periódico de la Comunidad Perl de Capital Federal

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

CaFe Perl v1.1

CaFe Perl v1.1 - Periódico de la Comunidad Perl de Capital Federal

Editorial

Un día más, un día de gracia.
Es todo lo que pido, es lo que nunca pedí, es lo que siempre se debe pedir.

Como todos ya saben este es mi último número como responsable directo a la edición de CaFe Perl porque a partir del próximo número, y por espacio de dos meses, la voy a compartir con N3krodamus y finalmente él será quien quede a cargo de la edición.

Ahora los dejo con lo interesante, que es la revista misma :

- Unos interesantes acertijos plateados por Martín Ferrari en el **PoC**
- Una entrevista a Jochan Luell, uno de los creadores de EPIC
- Las novedades del mes
- Un artículo de **Mordiditas...** con todo lo que querés saber sobre las expresiones regulares

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

Víctor A. Rodríguez (Bit-Man)

POC (peace of code)

Autor: Martín Ferrari

Volviendo al formato al que nos tenía acostumbrados el amigo Víctor, les traigo un casi-acertijo y un no-acertijo-pero-cuánto-me-reí!!.

A ver quién dice qué hace esto sin ejecutarlo? [1]

[1] Inspirado en:
http://www.rollmop.org/oneliners/perl.html

```
fortune | perl -lpe
 'lc;y/aetios/43710z/;
 s/(\w)/((int rand 3) % 2)?uc($1):$1/eg'
```

Parece candidato al Obfuscated Perl contest, lástima que ya no se hace más...

Y este otro?

```
perl -le '$^=q,161/A@-> /;053; ?5: ?-.1> =A1 4-/1 1?
<1853>;?;!,,$_=qq#$^#,y,--A,a-u,,print'
```

OJO, que muerde! Espero sus explicaciones, que este ya casi está para el OPC!

Y finalmente, les dejo para que se rían un buen rato, un pedazo de código, muy divertido. Les recomiendo leer el `man` y el código fuente, del módulo Sex, no tiene desperdicio:

```
package Catholicism;
use Sex qw(strict Religion);

package Mormonism;
use Sex qw(Catholicism Sex);
```

Entrevista y traducción :
Víctor A. Rodríguez

EPIC (http://e-p-i-c.sourceforge.net) es un IDE Perl open source hecho para la plataforma Eclipse. Las funcionalidades soportadas son resaltado de sintaxis[1], chequeo de sintaxis[2], asistente de contenido, soporte para `perldoc`, formateo de código, soporte para templates, plug-in para expresiones regulares y un debugger de Perl.

Por favor Jochen, haga una introducción al grupo CaFe.pm

Hola, mi nombre es Jochan Luell y soy uno de los dos miembros fundadores del proyecto EPIC. En la "vida real" trabajo como desarrollador/administrador para una gran compañía alemana (cerca de Heidelberg) que construye prensas de impresión. Desafortunadamente no estamos haciendo mucho desarrollo en estos días.

Cuál fue su motivación para construir EPIC ?

En Marzo del 2003 mi hermano Stephan vino con la idea de construir un IDE para Perl basado en Eclipse. Él estaba haciendo un proyecto privado y pequeño en Perl y no había un IDE para Perl gratis o de muy bajo costo. Él dijo "*no estaría bueno que tener un IDE de Perl basado en Eclipse, incluyendo un debugger ?*" En ese momento no conocía mucho sobre Eclipse, pero sonaba interesante. Así que comenzamos el proyecto. Mi tarea era la de trabajar con el editor y Stephen comenzó a construir el debugger.

El objetivo era construir un editor con capacidad de resaltado y chequeo de sintaxis y tener un debugger.

En el estado actual (0.3.12) EPIC tiene mucho más funcionalidad que lo que se planeó originalmente.

Tuvo alguna ayuda de otros proyectos open source ?

1 Syntax highlighting
2 Syntax checking

Debido a la falta de conocimiento sobre Eclipse fue muy útil no sólo mirar los conceptos/documentación de Eclipse, pero también mirar el código ya existente. Al principio fue bastante frustrante y por lo tanto muy importante tener código funcionando al que mirar.

Qué habilidades, relacionadas con Perl o no, obtuvo mientras construía EPIC ?

Tanto Stephan como yo no somos desarrolladores hardcore de Perl en absoluto (lo deberías notar durante la entrevista). Por mi parte estaba trabajando en la administración de sistemas Unix y solamente solía programar pequeños scripts en Perl (usando `vi`). Por lo tanto mi conocimiento de Perl es bastante limitado. Estoy más en la programación en Java. El proyecto está principalmente hecho en Java, porque es el lenguaje usado para construir un plug-in basado en Eclipse. Concerniente a Perl siempre es interesante ver con qué ideas viene la gente y estoy aprendiendo mucho en respecto a eso.

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

Desde mi punto de vista lo más importante es tener un buen plan. Tomate tu tiempo. A veces uno tiene que ir lento al principio para ir rápido al final. Sin un buen plan las cosas se pueden ir de las manos cuando un proyecto se hace más y más complejo.

En qué partes del código le aconsejaría mirar al programador Perl recién iniciado, para tener una experiencia de aprendizaje placentera ?

Todos los plugins de Eclipse principalmente están escritos en Java y EPIC no es la excepción en lo que a esto respecta. Así que idealmente los desarrolladores involucrados en el proyecto deberían tener conocimiento de ambos lenguajes, Java y Perl. Para los programadores Perl las partes más interesantes seguramente son en las que Perl es llamado directamente desde dentro del plugin. La validación de sintaxis

es un ejemplo.

Cómo se puede colaborar con EPIC ?

Cualquiera puede hacer contribuciones al proyecto. El CVS anónimo está disponible. Para cambios pequeños pueden enviarme un patch. Si alguien quiere contribuir más código envíenme un e-mail y eventualmente la/lo pondré en la lista de desarrolladores.

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

Como mencioné antes hay más incluidas hoy de las que originalmente se pensaron. Así que estoy muy satisfecho con la cantidad de funcionalidades. Una parte muy importante para mi son la estabilidad y performance. Esto debería ser tenido en cuenta al principio. Lo que es muy importante es extender el soporte de refactorización, que es un tanto débil de momento. Estamos trabajando en eso y esperamos poder proveer más funcionalidades de refactoring en el futuro.

Qué similitudes/diferencias encuentra con proyectos similares ?

Todos los IDEs basados en Eclipse comparten una base común que es la funcionalidad provista por el Framework Eclipse, que es usado para el resaltado de la sintaxis, poner marcas dentro del editor y demás. El que los hace todos diferentes son las partes específicas a cada lenguaje. Estas partes (en nuestro caso Perl) hacen cosas con un tanto de trucos, porque no puede basarse en la funcionalidad provista por el Framework Eclipse.

Qué limitaciones impuso Perl al proyecto ?

Los lenguajes dinámicos siempre son difíciles de manejar por un IDE. Perl no es la excepción (y tal vez más difícil que otros lenguajes). Para obtener resultados correctos (chequeo de sintaxis y otras cosas) llamamos a Perl directamente desde Eclipse. Esto nos dio un montón de dolores de cabeza al principio. Ahora las cosas están mejor ahora, pero hay algunas que no están funcionando como quisiéramos (por ejemplo asistencia en el contenido de objetos[1]

Alguna opinión o sugerencia respecto a Perl 6 ?

No tuve mucho tiempo para vérmelas con Perl 6. Espero ver algo como el Reflection API en Java, que haría las cosas más simples para nosotros. No puedo decir mucho sobre Perl 6. Veremos ...

Algún módulo favorito de CPAN ?

No tengo uno, hay tantos que son tan útiles. Uno muy interesante es PPI (Parse Perl Independently). Debería hacer la manipulación de código Perl más fácil y así ayudarnos en proveer un mejor soporte de refactoring. Es muy interesante. Un artículo sobre este módulo puede encontrarse en perl.com (http://www.perl.com/pub/a/2005/06/09/ppi.html).

Algún programador favorito de Perl o miembro de la comunidad ?

Quiere decir además de Larry Wall ;)
Hay un montón de buena gente en la comunidad que ofrecen ayuda y provee información preciosa.

Un muy buen contacto que tuve fue Matisse Enzer que escribió un buen artículo llamado "Perl Needs Better Tools", que puede encontrarse en el site de perl.com (http://www.perl.com/pub/a/2005/08/25/tools.html).

Hay algún grupo Perl en su localidad ?? (si es así, por favor, denos la URL de sus website)

No hay grupos Perl en mi localidad (hasta donde se). Los grupos más próximos son los de Frankfurt (http://frankfurt-pm.org/) or Stuttgart (http://stuttgart.pm.org/).

Participa en algún grupo Perl ?

No, no participo en un grupo Perl. No estoy tan involucrado en la programación Perl y mi

1object content assist

tiempo es muy limitado por el momento.

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

Tuve una experiencia realmente divertida. Al principio del proyecto entré en contacto con una persona de USA. Tenía algunas preguntas y traté de ayudarlo. En uno de sus e-mails me preguntó por mi dirección postal. Estaba sorprendido y le pregunté cuál era su intención. Me dijo que le gustaría enviarme un paquete con galletas (especialidades de Idaho). Algunas semanas más tarde recibí el paquete. Fue muy lindo de su parte y realmente divertido.

Algo más que quiera decirnos y que no le preguntamos ?

Gracias pro la entrevista y el interés en el proyecto. Espero que EPIC sea de uso para todos los lectores.

"The Perl Journal" deja de existir

Fundado en 1996 por Jon Orwant (http://www.tpj.com) ha dejado de existir, siendo su última publicación en Enero de 2006. Actualmente ha pasado a formar parte de la serie de lenguajes livianos de la publicación Dr. Dobb's Journal (http://www.ddj.com/dept/lightlang/tpj.jhtml). Pueden verse comentarios sobre este suceso en http://use.perl.org/comments.pl?sid=06/04/13/1942216

Se lanza Summer of Code de Google 2006

Un nuevo verano en el hemisferio Norte se acerca, y con el una nueva edición de este ya famoso ... concurso ?? (http://code.google.com/soc/) Como de costumbre también participa The Perl Foundation (http://www.perl.org/advocacy/summerofcode/) así que cualquiera que quiera participar puede anotarse y colaborar con algo a la comunidad open source a cambio de dinero y algo de fama

Nuevos libros

- **Wicked Cool Perl Scripts** (http://books.perl.org/book/243) de Steve Oualline, editorial No Starch Press : quizás algo nuevo bajo el sol
- **Micro-ISV: From Vision to Reality** (http://www.apress.com/book/bookDisplay.html?bID=10057) de Bob Walsh y editorial Apress : explica qué funciona y por qué en el mercado de los micro-ISV (Independent Software Vendors)
- **Practices of an Agile Developer** (http://www.pragmaticprogrammer.com/titles/pad/index.html) por Venkat Subramaniam y Andy Hunt, editorial Pragmatic Bookshelf : este libro colecciona los hábitos personales e ideas de desarrolladores que usan métodos ágiles con éxito
- **Learning UML 2.0** (http://www.oreilly.com/catalog/learnuml2/) por Russell Miles y Kim Hamilton, editorial O'Reilly : no solo explica UML 2.0 sino también como usarlo

... y también los podcasts del mes

- A Steve Oualline sobre su libro **"Wicked Cool Scripts"** (http://www.perlcast.com/audio/Perlcast_Interview_026_Oualline.mp3)
- a Bob Walsh, autor del libro **"Micro-ISV: From Vision to Reality"** (http://perlcast.com/2006/04/11/interview-with-bob-walsh/)

Parrot 0.4.3 ve la luz

La nueva versión de la máquina virtual que correrá Perl 6 puede bajarse desde http://www.cpan.org/authors/id/L/LT/LTOETSCH/parrot-0.4.3.tar.gz. Más información en http://www.parrotcode.org/

People of Perl

En el blog de O'Reilly sobre LAMP (http://www.oreillynet.com/onlamp/blog/) ha comenzado, hace un tiempo, una serie de entrevistas con gente del ambiente Perl : desarrolladores core, aplicaciones novedosas y en general todo aquel que haga resaltar a Perl de alguna u otro forma. Las últimas entrevistas fueron :

- Nicholas Clark (http://www.oreillynet.com/onlamp/blog/2006/04/people_of_perl_nicholas_clark.html)
- Guillaume Cottenceau (http://www.oreillynet.com/onlamp/blog/2006/04/people_of_perl_guillaume_cottenceau.html)

The World's Most Maintainable Programming Language

El ya conocido chromatic ha generado una serie de seis entradas en el blog ONLamp de O'Reilly sobre este interesante tema, con una parte dedicada a cómo reforzar la buena práctica de programación. (http://www.oreillynet.com/onlamp/blog/2006/03/the_worlds_most_maintainable_p.html)

Punto de vista de Richard Stallman Views sobre Linux, Java, DRM y Open Source

Una más que interesante transcripción de una charla que el citado dio en la Australian National University. Disfrútenlo en http://linuxhelp.blogspot.com/2006/04/unabridged-selective-transcript-of.html

... y Bruce Perens también tiene algo que decir

En http://technocrat.net/d/2006/4/5/2073 pueden encontrar el texto de su última conferencia de prensa en LinuxWorld Expo Boston.

Problemas de salud y el estilo de vida Geek

Una breve, pero interesante, lista de los problemas que pueden aquejarnos, incluso con comentarios más que interesante por parte de los visitantes (http://www.carotids.com/lifestyle/health-problems-related-to-the-geek-lifestyle/)

Exploración espacial open source

Como muchos la NASA también se ha volcado al open source, y muchas veces este revolucionó la forma en que se hacían las expediciones. Polvo de estrellas en http://www.onlamp.com/pub/a/onlamp/2006/03/30/software-of-space-exploration.html

V de Vendetta, en Scifiworld Magazine

Está disponible para descarga un el nuevo número de la revista gratuita *Scifiworld Magazine*, en particular el número 3 pone especial atención al estreno de la adaptación de *V de Vendetta* entre muchos otros contenidos de Ciencia Ficción (http://www.scifiworldmagazine.com/)

Por qué importan los standards abiertos

Todos lo sabemos, pero no todos sacan la ventaja necesaria, y si hay alguien que debería estar en pos de hacerlo son las organizaciones gubernamentales http://politics.slashdot.org/article.pl?sid=06/04/10/0439242

El debugger, ese aliado olvidado

Daniel Allen nos muestra cómo usar esta herramienta, muchas veces olvidada, en http://www.perl.com/pub/a/2006/04/06/debugger.html

Refactoring Everything

Fácil de decir, difícil de hacer, donde se narran las peripecias, en 30 capítulos, del refactoring de un legacy hecho en Perl (http://www.oreillynet.com/onlamp/blog/2006/04/refactoring_everything_day_1.html)

Una vuelta de Synthehol para todos !!

Parece ser que, al fin y al cabo, otra de las visiones de Star Trek se podría concretar : un sustituto del alcohol que no deja resaca (http://www.livescience.com/scienceoffiction/060412_synthehol.html)

Perl y Ajax

Una charla interesante sobre este tópico en

http://www.houseabsolute.com/presentations/ajax-perl/

Class::CGI tiene lista

y la pueden encontrar en http://groups.yahoo.com/group/class_cgi/, para enviar parches, ayudar en el desarrollo o simplemente plantear dudas y discusiones

Artistic License 2.0

Así como GPL está madurando su nueva versión, también lo hace Perl con su licencia. Pueden interiorizarse en http://www.cgidir.com/news/news/060421Perl6Licenses.html o también en http://trends.newsforge.com/trends/06/04/24/1449208.shtml?tid=147

Balanceador de carga LAMP

Todo sobre cómo configurarlo en http://www.linux.com/article.pl?sid=06/04/12/1824235. Una muy opción a tener en cuenta a la hora de elegir

Web sites seguros con Perl

Developerworks es famoso por sus artículos técnicos, y este le hace más que justicia. Disfrútenlo en http://www-128.ibm.com/developerworks/web/library/wa-perlsecure.html

Autor : Víctor A. Rodríguez

El motor de expresiones regulares

Voy a comenzar con la mejor introducción que he visto sobre expresiones regulares. Por supuesto no es mía sino de "Programming Perl, 3rd. Edition" :

"Zoológicamente hablando, los operadores para realizar manejo de patrones[1] funcionan como una especie de jaula para las expresiones regulares, para impedir que se salgan. Esto es por diseño; si dejáramos a las bestias de expresiones regulares vagar por el lenguaje, Perl sería una jungla. El mundo necesita sus junglas, por supuesto-- son los motores de la diversidad biológica, después de todo-- pero las junglas deben estar en donde pertenecen. Similarmente, a pesar de ser los motores de la diversidad combinatoria, las expresiones regulares deberán permanecer dentro de estos operadores, donde pertenecen. Es una verdadera jungla."

Realmente esta pequeña jungla nos da que hablar, de comer y de sufrir (aunque si querés hacer un repaso fijate en los tutoriales [1] y [2] de *Baboon Software* que figuran en **Infografía**). Ahora seguramente las hemos usado, y las seguiremos usando, para nuestro deleite intelectual y por facilidad con que resuelven nuestras tareas. Pero como todo hay que saber usarlas en el contexto apropiado, y es a eso que dedicaremos este número de **Mordidtas...**

Para comenzar veamos un poco cómo trabaja el motor de expresiones regulares. Este trabaja como un muy buen empleado, porque uno no tiene qué decirle cómo hacer algo sino que simplemente le decimos "quiero ver si este string coincide con este patrón" y él se encarga del resto. Lo curioso es cómo lo hace.

En líneas generales podemos decir que se esfuerza en hacer que el string tenga un patrón como el especificado y no en buscar si hay un patrón como el especificado. La diferencia es sutil pero notable : no se da por vencido al primer obstáculo. Para hacer su trabajo se basa en estos principios :

- posee un mecanismo que le permite saber qué opciones ya probó, cuáles no
- cada vez que encuentra una coincidencia sigue con el siguiente caracter, y si esto no coincide con el patrón entonces desanda el camino (*backtracking*)
- siempre comienza desde la izquierda del patrón y trata de obtener la mayor cantidad de caracteres que coincidan

Por ejemplo :

```
"fueron fuego" =~ /ue/;
```

empezará con la primera 'f' y al no conseguirlo lo hará con la letra 'u' (*backtracks*) que al coincidir seguirá con la 'e', dando por terminada la búsqueda.

Ahora seguiremos el mismo razonamiento con el siguiente código :

```
"fueron fuego" =~ /.*ue/;
```

A primera vista debería hacer lo mismo que la anterior, pero en realidad como trata de hacer que la primer combinación tome todos los caracteres, entonces tomará todos los posibles hasta el final del string menos 'u' y 'e', con lo cual hace el matching del 'ue' que está en la segunda palabra 'fuego' en vez de la primera.

Ahora vamos a analizarlo bajo la lupa del debugger, ejecutándolo de la siguiente manera :

```
1 #!/usr/bin/perl
2 use re "debug";
3
4 "fueron fuego" =~ /ue/;
```

cuya ejecución nos da esta salida :

1 N. del T. : pattern matching

```
 1 Compiling REx `ue'
 2 size 3 Got 28 bytes for offset annotations.
 3 first at 1
 4    1: EXACT <ue>(3)
 5    3: END(0)
 6 anchored `ue' at 0 (checking anchored isall) minlen 2
 7 Offsets: [3]
 8         1[2] 0[0] 3[0]
 9 Guessing start of match, REx `ue' against `fueron fuego'...
10 Found anchored substr `ue' at offset 1...
11 Starting position does not contradict /^/m...
12 Guessed: match at offset 1
13 Freeing REx: `"ue"'
```

Simpático, no ?? Para ser sinceros estas serie de conjuros hay que interpretarlo viendo `perldoc perldebguts` (o en su defecto la entrada [4] de **Infografía**). La primer parte (líneas 1 a 8) se realizan en tiempo de compilación donde, a vuelo de pájaro[1], nos muestra la forma de la regex[2] pre-compilada (línea 1), el tamaño de la forma compilada (línea 2), información del optimizador[3] (línea 3), la lista de nodos de la regex compilada (líneas 4 y 5) y un dump de esta lista (líneas 7 y 8).

Una vez hecha esta magia viene el tiempo de ejecución (no ... acá no hay entretiempo como en el fútbol) que va desde la línea 10 a la 13. Simplemente en estas líneas no hizo ningún análisis muy complejo, se dio cuanta que 'ue' coincide (matching) en el offset 1 y allí termina su búsqueda. Básicamente digamos que lo que ocurrió en este punto fue que la optimización fue tan buena que no necesitó empezar a rastrear caracter por caracter, o mejor dicho si lo hizo pero sin usar el motor de expresiones regulares sino más bien haciendo un match de strings (simple, no ??).

Ahora lo que vamos a hacer es ejecutar la segunda expresión regular con debugging :

```
1 #!/usr/bin/perl
2 use re "debug";
3
4 “fueron fuego” =~ /(.*)ue/;
```

Dándonos la salida del **Listado 1** (*al final de la nota*). Como se puede ver las fases son las mismas que en la ejecución anterior, pero al tratarse de una expresión regular más compleja requiere ciertas idas y vueltas adicionales. Esta vez si va a necesitar hacer un poco de magia, entonces lo que hace es tratar de que el primer grupo (todo lo que está antes de 'ue') ocupe la mayor cantidad posible de caracteres y de ahí en más intenta que le resto de la regex coincida : en la línea 23 vemos como dice que REG_ANY (haciendo referencia a la forma compilada de la regex que está en las líneas 5 a 10) puede coincidir hasta en 12 caracteres (que es la longitud de todo el string) pero en realidad toma 8 (línea 25, donde <fueron f> son los caracteres que coinciden), luego toma 2 más (línea 26, correspondiente al string 'ue') donde termina la ejecución encontrando una coincidencia (línea 28).

Fíjense que lo curioso es que ahora la coincidencia fue con el 'ue' final y no con el inicial, como en el caso anterior.

Ahora vamos a ir un poco más lejos, y vamos a hacerle un pequeño truco : vamos a complicar un poquito la regex y ver si el engine se comporta igual.

```
1 #!/usr/bin/perl
2 use re "debug";
3
4 “fueron fuego” =~ /(.*)uer/;
```

Esta vez el motor debería hacer lo mismo que antes. Lo hará ??

---

1 No nos vamos a detener en cada uno de estos pasos porque sería escribir un capítulo completo del libro “cómo las expresiones regulares acabaron con mi vida”

2 regex : regular expression

3 Si señores, todo se optimiza en Perl !!

Mostrando sólo la parte correspondiente a la fase de ejecución, vemos cómo primero intenta tomar la mayor cantidad del string (nuevamente 8 caracteres) pero falla en el

```
 1 Matching stclass
 `ANYOF[\0-\11\13-\377{unicode_all}]'
 against `fueron fue'

  Setting an EVAL scope, savestack=3
   0 <> <fueron fuego>    |  1:  OPEN1
   0 <> <fueron fuego>    |  3:  STAR
                  REG_ANY can match 12
                  times out of 2147483647...
  Setting an EVAL scope, savestack=3
   8 <fueron f> <uego>    |  5:    CLOSE1
   8 <fueron f> <uego>    |  7:    EXACT <uer>
                           failed...
   1 <f> <ueron fuego>    |  5:    CLOSE1
   1 <f> <ueron fuego>    |  7:    EXACT <uer>
   4 <fuer> <on fuego>    |  9:    END
Match successful!
```

intento (`failed...`) y prueba con menos caracteres encontrando la coincidencia con 4 caracteres (última línea).

Las regex deben ocupar su lugar

Entonces dimos toda esta vuelta para qué, simplemente porque nuestro cometido es saber cuándo si y cuando no usar las regexes, y para eso tenemos que saber como funciona el motor de regexes. Entonces primer misión cumplida : ya sabemos cómo funciona y que en una búsqueda simple de strings el regex engine hace simplemente eso, buscar strings. Y si hace lo mismo no sería más apropiado usar una función de búsqueda de strings ?? Por qué ?? en qué casos ??

En nombre de la elegancia y la generalización usar una regex cuando podemos hacerlo con una función de manejos de string es lo mejor, pero cuando hablamos de la performance normalmente cuanto más específica es una funcionalidad seguramente mayor performance tendrá. Para verificarlo vamos a tomar como testigo nuestro primer caso, donde vamos a buscar si en el string 'fueron fuego' existe el string 'ue' comparando la ejecución del código :

```
“fueron fuego” =~ /ue/;
```

contra el código :

```
index(
 'fueron fuego',
 'ue'
);
```

usando nuestro viejo amigo el módulo Benchmark.pm (ver el código en **Listado 2**, al final del artículo).
Al ejecutar el benchmark en distintos equipos[1] se puede ver que nos da una ventaja de performance entre 150% y 300% a favor de `index()`. Con esto vemos que si bien el regex engine se las ingenia para poder detectar que debe usar una función de búsqueda de string, al usar `index()` nos evitamos que se genere toda la infraestructura necesaria para analizar las regex y su posterior liberación.

Algunos trucos :-)

La forma más común de usar el operador match (`m//`) es la de decirle que haga matching contra un string determinado o simplemente contra la variable $_., pero no siempre se sabe que el operador match devuelve, en un contexto de lista, los valores con lo que ha hecho match que han sido colocados entre paréntesis. Esto lo hace particularmente simpático para poder leer los valores de un archivo en formato INI :

```
($key, $value) =  /(\w+)=(.*)/
```

Otro truco. Si buscamos por una coincidencia de más de un string normalmente utilizamos | para separarlos :

```
"fueron fuego" =~
      /gas|petroleo|oleoducto/;
```

Si lo ejecutamos con use re “debug” vamos a ver que tratará de hacer el matching de cada una de la opciones contra la primer

1 Gentoo Linux 2.6.15 sobre Pentium 4 2.66 Ghz, Microsoft Win XP SP2 sobre Pentium M 1.5 Ghz y Darwin 7.9.0 10.3.8 sobre PowerPC G3

letra (f) luego hará lo mismo con la segunda (u) y así hasta el final que luego de 11 intentos fallará y terminará la búsquedainfructuosamente. En cambio si las opciones las damos como una serie de opciones pero de regexes :

```
"fueron fuego" =~
      /gas/ ||
      /petroleo/ ||
      /oleoducto/;
```

sólo le tomará 3 intentos de 1 carácter cada uno (primero con 'gas', luego con 'petroleo' y después con 'oleoducto') el saber que la cosa no funciona.

Infografía

[1] Expresiones Regulares – Las Bases (http://perlenespanol.baboonsoftware.com/archives-tut/000072.html)

[2] Expresiones Regulares – Internedio (http://perlenespanol.baboonsoftware.com/archives-tut/000074.html)

[3] Programming Perl, 3rd. Edition (http://www.oreilly.com/catalog/pperl3)

[4] Perl debugging guts and tips (http://perldoc.perl.org/perldebguts.html)

[5] Apocalypse 5 : "Pattern Matching" (http://dev.perl.org/perl6/doc/design/apo/A05.html)

[6] Exegesis 5 : “Pattern Matching” (http://dev.perl.org/perl6/doc/design/exe/E05.html)

[7] Synopses 5 => "Regexes and Rules" (http://dev.perl.org/perl6/doc/design/syn/S05.html)

[8] Perl CodeFetch (http://perl.codefetch.com/)

[9] Gory details of parsing quoted constructs (http://www.annocpan.org/~NWCLARK/perl-5.8.8/pod/perlop.pod#gory_details_of_parsing_quoted_constructs)

[10]Benchmark - benchmark running times of Perl code (http://search.cpan.org/~nwclark/perl-5.8.8/lib/Benchmark.pm)

Listado 1

```
 1 Compiling REx `(.*)ue'
 2 size 9 Got 76 bytes for offset annotations.
 3 first at 3
 4 synthetic stclass `ANYOF[\0-\11\13-\377{unicode_all}]'.
 5    1: OPEN1(3)
 6    3:   STAR(5)
 7    4:     REG_ANY(0)
 8    5: CLOSE1(7)
 9    7: EXACT <ue>(9)
10    9: END(0)
11 floating `ue' at 0..2147483647 (checking floating) stclass
   `ANYOF[\0-\11\13-\377{unicode_all}]' minlen 2
12 Offsets: [9]
13         1[1] 0[0] 3[1] 2[1] 4[1] 0[0] 5[2] 0[0] 7[0]
14 Guessing start of match, REx `(.*)ue' against `fueron fuego'...
15 Found floating substr `ue' at offset 1...
16 Does not contradict STCLASS...
17 Guessed: match at offset 0
18 Matching REx `(.*)ue' against `fueron fuego'
19 Matching stclass `ANYOF[\0-\11\13-\377{unicode_all}]' against
   `fueron fueg'
20  Setting an EVAL scope, savestack=3
21    0 <> <fueron fuego>    |  1:  OPEN1
22    0 <> <fueron fuego>    |  3:  STAR
23                            REG_ANY can match 12 times
                              out of 2147483647...
24   Setting an EVAL scope, savestack=3
25    8 <fueron f> <uego>    |  5:    CLOSE1
26    8 <fueron f> <uego>    |  7:    EXACT <ue>
27   10 <fueron fue> <go>    |  9:    END
28 Match successful!
29 Freeing REx: `"(.*)ue"'
```

Listado 2

```
 1 #!/usr/bin/perl
 2
 3 use strict;
 4 use warnings;
 5 use Benchmark qw(:all) ;
 6
 7 my $count = -4;     ## ejecuta el test durante $count segundos
 8                     ## para cada sub()
 9
10 my $subString = 'ue';
11 my $myString = 'fueron fuego';
12 my $pos;
13 my $results =    timethese($count, {
14           'regex' => sub {
15                 $myString =~ /$subString/;
16              },
17                 'string' => sub {
18                 $pos = index( $myString, $subString );
19              }
20         });
21
22 cmpthese( $results );
```