Periódico de la Comunidad Perl de Capital Federal

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

Cafe Perl v0.6

Table of Contents
Cafe Perl v0.6

CaFe Perl v0.6 - Periódico de la Comunidad Perl de Capital Federal

Editorial

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

Este mes comenzamos un nuevo camino en la sección Mordiditas de aquí y de allá. La idea es poder seguir llegando llegar a la mayor cantidad de gente con los conocimientos más dispares, entonces la metodología usada es primero presentar el problema junto con la solución para poder usarlo como si fuera una receta (para los que quieren usarlo sin mas vueltas), y después la explicación de cómo se llega a esta receta final pero aumentando la complejidad progresivamente dentro del artículo. Igual que Perl : "one size fits all".

Se va a seguir respetando esa idea, pero esta vez vamos a usar un poco mas de metodología, y para esto nos vamos a guiar por los capítulos de "Programming Perl - 3rd. Edition". Esto no es casual, primero porque es la forma en que Larry Wall va generando sus "especificaciones" de Perl 6 (Apocalipsis - http://dev.perl.org/perl6/doc/apocalypse.html), y además por una razón pedagógica.

La razón para este cambio obedece, en cierta forma, a un e-mail que allá por el 2001 escribió una persona dedicada al QA (Quality Assurance) de software (pueden leerlo en http://www.underlevel.net/jordan/erik-perl.txt) y que en la lista de discusión del libro High Order Perl (http://hop.perl.plover.com/ de Mark Jason Dominus) generó todo un revuelo y debate. Uno puede o no estar de acuerdo con ese e-mail, pero hay algo cierto y es que se ve a Perl como un lenguaje incapaz de generar aplicaciones mantenibles, y aquí la palabra clave es mantenibles, y no aplicaciones. En este punto me quedo con el más brillante de los e-mails de ese thread (http://hop.perl.plover.com/~alias/list.cgi?2:mss:340:eabblnfenfegfacjkjhd).

Como conclusión Mordiditas de aquí y de allá va a mejorar metodológicamente, y para eso se va a orientar en la búsqueda de las mejores prácticas para obtener aplicaciones mantenibles, robustas, seguras y que hagan buen uso de sus recursos, utilizando para esto los principios establecidos de Ingeniería de Software (modularidad, orientación a objetos, etc.) y toda herramienta que tengamos a la mano para hacerlo. No es casual que en este camino también se encuentre Perl 6, con lo cual también nos va a servir para afianzarnos en Perl 5 y poder hacer una transición lo más cómoda posible.

Como última novedad les cuento que momentáneamente Andrés no nos va a poder acompañar en la sección Peace of Code, así que esta sección no va a ser fija sino que estará en los números que alguno de ustedes, queridos integrantes de la comunidad Perl, me acerque algún código que hayan hecho, que desafíe las leyes de la gravedad o que simplemente tenga una duda y busquemos la forma de llevarla adelante como un divertimento.

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

Víctor A. Rodríguez (Bit-Man)

PERlitas

Entrevista realizada y traducida al español por Víctor A. Rodríguez

gtk2-perl es el nombre dado a un conjunto de perl bindings para Gtk+ 2.x y varias librerías relacionadas. Estos hacen que sea mucho más fácil escribir aplicaciones Gtk y Gnome usando una sintaxis más orientada a Perl y objetos.

Por favor muppet, una presentación para el grupo CaFe.pm

Soy un músico y programador de 30 años de edad que vive en Lexington, Kentucky, USA con mi esposa y mis hijas mellizas. Tengo un título universitario en Ingeniería Eléctrica, pero trabajé casi exclusivamente en software desde mi graduación. Compongo y toco música rock ecléctica (http://seasonone.org/) cuando no estoy persiguiendo a mis hijas, hackeando software, o trabado en una conversación geek con mi esposa.

Puede servir el saber que mi apodo ?muppet?[1] proviene de mis días en los dormitorios de la Universidad de Kentucky, donde era conocido por mi largo cabello rubio al viento y mi inclinación por el humor de vaudeville.

Cuál fue la motivación para construir Gtk2-Perl ??

Versión corta: lo necesitaba y no existía.

Versión larga: Hice mucho procesamiento de datos de texto en mi trabajo y perl fue la herramienta correcta para ese trabajo. También era un usuario ávido de Linux y Gnome, y cuando necesité ponerle un interfaz gráfica (GUI), elegí Gtk-Perl. También estaba trabajando en una librería de C para análisis de imágenes, y conectarla con perl usando XS, de tal manera que podría usar DBI para poner los resultados del análisis directamente en una base SQL. Las combinación de C para el trabajo de bajo nivel y Perl para la lógica de aplicación fue una combinación ganadora.

Después de un par de años de usar Gtk-Perl en aislamiento, me preguntaba cuál sería el soporte para gtk+ 2.x, y me uní a gtk-perl-list en gnome.org. Resultó que quienes lo mantenían originalmente abandonaron el proyecto en forma sigilosa, y algunos miembros de la lista estaban discutiendo comenzar un nuevo proyecto.

Pasaron varios meses, mientras mi software casero creció alrededor de GObject y gtk+ 2.0. Cuando llegó el tiempo de hacer funcionar mi código con perl, encontré que el Gtk2-Perl inline (por entonces versión 0.12) no era suficientemente extensible para permitirme conectar con perl, de una manera interoperable, mi propia librería basada en GObject. En Abril de 2003 hice un hack de un prototipo de nueva arquitectura para los bindings, y lo presenté a Göran y Guillaume en la lista (http://mail.gnome.org/archives/gtk-perl-list/2003-April/msg00015.html). Para mi sorpresa me respondieron dándome el mantenimiento del proyecto.

En honor a la verdad, yo no construí por mi mismo a Gtk2-Perl. Hubo mucho de arte previo; tomé mucho prestado del Gtk-Perl original, de la versión Gtk2- Perl inline y de pygtk. Ross McFarland hizo mucho del trabajo pesado para cambiar el prototipo original en algo útil, y Torsten Schönfeld nos llevó directo a los bindings de TreeView y creó la mayoría de nuestra suite de testing.

No me queda claro que son los GObjects

gtk+ es un sistema de widgets basado en objetos; GObject es la clase base abstracta de la jerarquía de objetos y también la librería que exporta este objeto. Todo Gnome se basa en él (http://developer.gnome.org/doc/API/2.0/gobject/index.html). Son referencias-contadas a objetos C que soportan propiedades tipadas y notificaciones. Perl no tiene un objeto base, así que no tengo realmente una analogía.

Y cómo se sintió cuando Göran y Guillaume le dieron el rol de mantenimiento ?? Lo aceptó inmediatamente ?? Se sintió cómodo ?? Qué vacilaciones tuvo ??

Estaba sorprendido, para ser honesto. No lo esperaba, no fue nada formal. Ya tenía permiso para hacer commit, y GC un día me dio permiso de mantenimiento y dijo, ?aquí vas?[1] (sourceforge tiene un rol específico de mantenimiento).

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

Sí. Marc Lehmann ayudó a perfeccionar la conexión entre los GObject en C y las referencias de perl. Los desarrolladores de gtk+ siempre ayudaron a determinar dónde estaban los bugs, ya sea en los bindings o en el gtk+ mismo. Frecuentemente he mirado el código del viejo Gtk-Perl y pygkt como guía.

Qué habilidades, relacionadas a Perl o no, adquirió mientras construía Gkt2- Perl ??

Aprendí mucho sobre el manejo de proyectos, cuánto trabajo toma el hacer que un software sea estable, qué hace buena a un API, sin mencionar las idas y vueltas de GObject.

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

No subestimes el valor de tu comunidad de usuarios.

En qué partes del código le aconsejaría ver a quien recién comienza en Perl, para tener una experiencia placentera ??

Presumo que quien está comenzando querrá conocer cómo usar Gtk2-Perl. El mejor lugar para comenzar serían los ejemplos y demos que vienen con el código fuente del módulo Gtk2, los tutoriales varios y la documentación cuyos links están en http://gtk2-perl.sourceforge.net/

Para alguien que busca crearon o mantener bindings del lenguaje, es algo diferente. Los bindings son casi todos XS, el lenguaje de extensión de subs externas de Perl (es un dialecto de C y directivas de preprocesamiento). El trabajo pesado ocurre en el módulo Glib. Las demás extensiones son sólo plantillas, que usan utilidades provistas por Glib.

Cómo se puede colaborar con Gtk2-Perl ?

La lista de e-mail, gtk-perl-list@gnome.org, es el mejor lugar para comenzar. Allí se toman patches y se discute en general. También estamos en #gtk-perl en irc.gnome.org.

Tenemos un pequeño grupo de desarrolladores para los módulos base. Ross McFarland y yo hicimos la mayor parte del trabajo durante el verano del 2003. Sobre el final de ese verano comenzamos a tener una cantidad de muy buenos patches de un estudiante alemán de física llamado Torsten Schönfeld; de hecho tantos patches y de tal calidad que le dimos permiso de commit y comenzamos a pedirle opinión sobre nuestros cambios. El año pasado, cuando el trabajo comenzó a manifestarse en la forma de un mantenimiento efectivo tanto para Ross como para mi, le pasamos la mayor parte de la responsabilidad a Torsten, que hizo un gran trabajo.

Algunos de los módulos en la colección aún necesitan de quienes los mantengan, y siempre estamos en la búsqueda de voluntarios.

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

No importa que tan completo creemos que esté, siempre alguien reporta que no está alguna oscura API que necesita.

Estamos revisando las API para hacer la conexión entre los bindings Cairo de Ross y Gtk2.

El equipo gtk+ está trabajando en agregar una introspección de toda el API, y estamos preparando nuestro propio soporte para esto. Esto nos llevará, esperamos, a algo muy cercano a la automatización completa de los bindings, menos problemas relacionados a las versiones para los usuarios, y menos trabajo para nosotros de mantenimiento.

Qué tipo de introspección están realizando y cómo impactaría en Gtk2-Perl ??

Ahora GObject te permite preguntar a cada objeto ?qué propiedades soporta ? cuáles son sus tipos ?? y ?qué señales soporta ? Cuáles son sus firmas ??, pero eso es todo. Las demás funciones deben explícitamente tener bindings con algo de XS y eso es verdad para todos los lenguajes, y no sólo para Gtk2-Perl.

La propuesta de la introspección es agregar la habilidad de preguntar a la librería por información acerca de cada función y tipo de dato que exporta. De esa forma los bindings del lenguaje no tienen que tener demasiado mantenimiento manual. Estamos planeando soportarlo con AUTOLOAD y libffi... (http://sourceware.org/libffi/). Si tu script pregunta por una función que no existe, AUTOLOAD se fija si la librería la exporta; si es así se fija cómo llamarla, entonces crea y guarda un stub para llamarla la próxima vez. Si funciona va a estar muy bueno, porque cuando una nueva versión de gtk+ salga, la versión de Gtk2 que ya tenés mágicamente soportará estas nuevas funciones (actualmente tenemos que agregar soporte para las nuevas funciones según son agregadas a gtk+).

Y no va a ser necesario que seas un mago del XS para crear perl wrappers para tu librería favorita basada en GObject !!!

Algún módulo de CPAN favorito ??

Debería ser Acme::Scurvy::Whoreson::BilgeRat. :-) (http://search.cpan.org/dist/Acme-Scurvy-Whoreson- BilgeRat/lib/Acme/Scurvy/Whoreson/BilgeRat.pm)

Tiene un Grupo Perl en su área ?? Participa en él ??

El grupo local, Lexington.pm es algo, ejem, de bajo tráfico. http://lexington.pm.org/

Parece que mucha de la gente que participaría están involucrados con el Lexington Professional Linux Users Group: http://lplug.org/

En su lugar, estuve suscripto a London.pm por algún tiempo, a pesar de estar a varios miles de millas de distancia y nunca haberlos visitado. Hay mucho de superposición entre london.pm y (void), así que no me sentí demasiado fuera de lugar. Al menos no me echaron, todavía =)

Tiene alguna experiencia, divertida o no tanto, mientras hacía Gtk2-Perl y que quiera compartir con nosotros ??

La razón principal por la que estaba dispuesto a pasar varios meses a costa de mis tardes y fines de semana sobre Gtk2-Perl, fue que mi esposa estaba embarazada y no se sentía bien como para salir de la casa.

Excepto por Ross, nunca me reuní con ningún usuario de Gtk2-Perl en la vida real.

Algo más que quiera contarnos y no le preguntamos ??

Usé Perl para hacer muchas cosas, pero ?no? usé perl para desarrollos web.

Para qué usa Perl ??

De todo un poco... por un largo tiempo, estuve acomodando números en archivos de texto, entonces hacer data crunching y ponerlos en una base de datos SQL. Ahora tengo herramientas para hacer scan de archivos de logs y enviarme e-mails en eventos interesantes, algo para buscar en código por la palabra FIXME en los comentarios cada noche, otra herramienta que genera stubs para un proyecto en C...

De todo un poco :-)

SudorNews

Internacionalización de Perl

En la primer página de la interesante entrevista a Autrijus Tang (http://www.perl.com/pub/a/2005/09/08/autrijus-tang.html) generador de una implementación de Perl 6, habla sobre las herramientas para convertir a cualquier programa en Perl a multi-lenguaje. Un buen lugar para empezar.

Los mejores scripts de Perl ... en Microsoft !?

En http://www.microsoft.com/technet/scriptcenter/scripts/perl/prlindex.mspx vas a poder a encontrar una serie de scripts Perl para administrar máquinas Windows : el gigante de Redmond no puede resistirse a Perl ni al Open Source

Se lo que hicieron el verano pasado

Además del título de una película puede ser una afirmación que muchos de los asistentes al Google Summer of Code (http://code.google.com/summerofcode.html) pueden tomar de muy buena manera. En este publicitado programa Google aceptó a ocho estudiantes para proyectos relacionados con Perl a

través de The Perl Foundation (http://www.perlfoundation.org/news/2005/googlesoc2.html). Google dará U$S 4500 a cada estudiante que complete el proyecto propuesto, y U$S 500 a The Perl Foundation por cada proyecto.

GPL 3 ya da que hablar

La presunta inclusión de penalidades para el software propietario y DRM en la nueva versión de GPL lanzó una avalancha de comentarios. La nueva versión aún está en sus primeros pasos (http://www.fsf.org/news/gplv3, fechado el 6 de Septiembre ppdo.). La verdad es que no había nada cierto sobre esta restricción para principios de Septiembre, cuando MSNBC se hizo eco de esta posibilidad, más especulativa que otra cosa (http://www.msnbc.msn.com/id/9225821/). Esto también tuvo su pronta respuesta por parte de la Free Software Foundation (http://mail.fsfeurope.org/pipermail/discussion/2005-September/005190.html) donde claramente que eso sólo podría estar en la cabeza de Richard Stallman, o sea que cualquier cosa que se diga es pura especulación.
Finalmente en una entrevista a este personaje fundamental del GPL (http://www.onlamp.com/pub/a/onlamp/2005/09/22/gpl3.html) establece que hacerlo no ayudaría en nada.

Para prácticas, las mejores !!

Del ya conocido Damian Conway salió el libro Perl Best Practices (http://www.oreilly.com/catalog/perlbp/), una colección de 2^8 lineamientos que prometen un jugoso desempeño, a juzgar por el capítulo de ejemplo (http://www.oreilly.com/catalog/perlbp/chapter/ch09.pdf) o por la review de Linux Journal (http://www.linuxjournal.com/article/8567) ... o será que mi tópico preferido es el de la modularización ??

Qué es el Open Source ?

Antes explicar qué era el Open Source era más fácil, ahora involucra muchos más factores que hacen que esta tarea quede para unos pocos (http://www.onlamp.com/pub/a/onlamp/2005/09/15/what-is-opensource.html)

Otra vez la cebolla !!

Son famosas las ?State of the Onion? de Larry Wall donde hace referencia a Perl y el movimiento Open Source. Esta vez lo hace a través de una serie de personajes (http://www.perl.com/pub/a/2005/09/22/onion.html). Algo entretenido para leer

YAPC:EU::2005

Están comenzando a estar disponibles las charlas de esta Perl Conference (http://perl-hackers.net/YAPC-05/). Próximamente también van a estar disponibles dos versiones de cada charla (una grande y otra más pequeña)

Nuevo site para comenzar a aprender Perl !!

En http://perlmeme.org/ vas a poder encontrar FAQ, HOWTOs y tutoriales (todos en inglés). Hay para todos los gustos : para quién recién empieza, para quién ya sabe y para el que quiere saber aún más. También sirve para saber qué es un meme.

Lo básico (Mordiditas de aquí y de allá)

En esta entrega empezamos con lo básico del lenguaje Perl, pero a ver, que se entienda que no vamos a empezar con el típico ?los tipos de datos de Perl son? o ?la sentencia if ...? sino que vamos a empezar por algo más básico y más general : la filosofía detrás de Perl.

Si bien parece algo abstracto vamos a ver que es absolutamente tangible y de mucho de lo que aquí se vea, seguramente, van a recordar situaciones en las que se encontraron con código que lo ejemplifique.

Hay muchas evidencias que la tendencia es que todo el que trabaje con Perl se sienta cómodo : comenzando con Perl[1] , siguiendo por perl[2] y finalizando con la comunidad que los soporta y sus distintas manifestaciones (CPAN, Perl Mongers, Perl Monks, etc.). En particular Perl permite una cantidad de usos del lenguaje que hacen que se pueda empezar a aprender el lenguaje de una forma casi

inmediata :

- - no hay tipos de datos estrictos : al momento de empezar a programar no hay nada más insidioso que encontrarse con la conversión de tipos de datos y esos molestos mensajes sobre la imposibilidad de establecer una conversión implícita entre tipos de datos
- - hay sólo tres estructuras de datos : son simples, poderosas y sumamente dinámicas (en todo sentido) que nos hacen que no tengamos que desviar nuestra atención en problemas de implementación tales como alocar memoria, especificar longitudes de strings, saber de antemano la cantidad de dimensiones y el tamaño de un array, etc. Simplemente usar sin preocupaciones
- - posee las sentencias más usuales : lo más común de la programación imperativa (condicionales y loops), y las herramientas para modularizar más usuales (subrutinas y módulos), lo que hace que todo conocimiento que tengamos de un lenguaje imperativo lo podamos reutilizar, resultando en una ganancia de tiempo y una reducción del stress propio de comenzar a aprender un nuevo lenguaje.

Pero esto no se detiene en este punto, simplemente porque el diseño está basado en los Principios de un Lenguaje Natural (http://www.wall.org/~larry/natural.html) y eso entre otras cosas significa que se va a prendiendo mientras se avanza, se reusa el conocimiento y que inevitablemente hay cierta redundancia y varias formas de decir lo mismo, lo cual por un lado esto no es bueno al momento de usar los recursos computacionales, pero se produce una devolución de ese valor en una forma no esperada y es que al no haber una sola forma de hacerlo no es necesario obtener un titulo de ?master? para empezar a programar, con lo que tanto la ecuación económica como de satisfacción personal dan un balance positivo casi de inmediato (la clave motora de Perl es TMTOWTDI : There‘s More Than One Way To Do It ? hay más de una forma de hacerlo).

Avanzando un paso más también vemos que Perl soporta múltiples sintaxis, con lo cual nos permite expresarnos de la forma más conveniente a nuestro estilo, nivel de conocimiento o reglas determinadas por el ambiente (reglas de programación, de estilo, etc.).

Para dar un ejemplo de esto que acabo de comentar introduzcámonos en una tarea bastante común, que puede ser la de leer un archivo y procesar cada línea . Normalmente abro el archivo, hago un loop y en cada iteración lo proceso, mas o menos de la siguiente forma :

```
my $line;
while $line (<$arch>){
process_line( $line );
}
```

en tanto que un loop también lo puedo hacer con un for/foreach y utilizando la variable default $_ :

```
process_line( $_ )
       for ( <$arch> );
```

Y si aún esto no nos satisface podemos desligarnos completamente de la forma de loop evaluando igualmente <$arch> en un contexto de array pero sin que sea explícitamente reconocido como un loop a primera vista, al menos no para el ojo no entrenado.

```
map process_line( $_ ), <$arch>;
```

Si miramos lo anteriormente desde otro punto de vista podemos vislumbrar otra de las características de Perl, y es que soporta múltiples paradigmas, y si bien es un lenguaje imperativo puede dar cabida a otros paradigmas como el de orientación a objetos. Por supuesto puede resultar mas o menos difícil según las características del paradigma a implementar en relación con las de Perl mismo. Adicionalmente a medida que uno se va sintiendo cómodo con el lenguaje comienza a tomar conciencia de estos otros aspectos y formas de trabajo que, para ciertas situaciones son más útiles y podrían ser utilizadas, pero que en otros lenguajes sería imposible de utilizar a menos que se cambie a otro lenguaje.

Para que tengan una idea más acabada que mejor que usar las propias palabras de Larry Wall acerca de Perl. Lo que sigue es una traducción del documento Natural Language Principles in Perl.

Apréndalo una vez, úselo muchas veces

Se aprende un lenguaje natural una vez y se lo usa muchas. La lección para una diseñador de lenguajes es que un lenguaje debería estar optimizado para potenciar su expresividad en lugar de para su facilidad de aprendizaje. Es fácil aprender a manejar un carro de golf, pero es difícil poder expresarse con uno de ellos.

Aprenda mientras avanza

Uno no aprende un lenguaje natural ni siquiera una vez, en el sentido que nunca termina de aprenderlo. Nunca nadie ha aprendido un lenguaje natural completamente. Desafortunadamente, en el interés de la ortogonalidad, muchos lenguajes de computadora son diseñados de tal forma que cada grado de libertad (dimensión) está disponible en todo lugar. Esto tiene sus puntos a favor si se comprende todo el lenguaje, pero de lo contrario puede llevar a confusión. Uno quisiera ignorar algunas de estas dimensiones para empezar. Uno quisiera poder hablar como los bebés y ser entendido. Está bien si un lenguaje es difícil de aprender, en tanto y en cuanto no deba aprenderse todo el lenguaje a la vez.

Muchos niveles aceptables de competencia

Es mas una característica social, comparado con "aprenda mientras avanza", que una característica psicológica. A la gente no le importa si se habla un subconjunto de un lenguaje natural, especialmente si se es un niño o un extranjero (excepto en París, por supuesto). Si un lenguaje es diseñado de tal forma que se "aprende mientras se avanza", entonces se espera que todos estén aprendiendo, y eso está correcto.

Muchas formas de decir lo mismo

Esta es mas una característica antropológica. La gente no solo aprende mientras avanza, sino que además proviene de distintos ambientes, y aprenderá primero un subconjunto distinto del lenguaje. Está oficialmente bien en Perl programar en el subconjunto de Perl correspondiente a sed, awk, C, shell, BASIC, Lisp o Python. O aún FORTRAN. Sólo porque Perl es un crisol lenguajes de computadora no significa que uno tenga que convertirse en una mezcla.

No hay vergüenza en pedir prestado

En Inglés[1] (y otros idiomas que no sufren una crisis de identidad), a la gente no le importa tomar ideas de otros idiomas y hacerlas parte del propio. Esfuerzos para mantener la "pureza" de un idioma (sea este natural o no) sólo son exitosos en establecer una elite de personas que conozcan "la jerga". La gente común conoce mejor al idioma, aunque no conozcan "la jerga".

Dimensionalidad indeterminada

A los científicos les gusta ubicar las cosas dando un "vector" que es, una lista de coordenadas en un espacio de una dimensionalidad dada. Esta es una de las razones por las que les gusta la ortogonalidad: significa que los distintos componentes del vector son independientes entre si. Desafortunadamente, el mundo real no está hecho de esta forma. La mayoría de los problemas, incluyendo los lingüísticos, son una cuestión de ir de aquí hasta allí, y la geografía que hay entre estos tiene una influencia muy grande sobre qué soluciones son prácticas. Los problemas tienen una disposición a ser resueltos en varios niveles. Un viaje típico podría involucrar sus piernas, su auto, una escalera mecánica, una cinta transportadora, un avión, un taxi, y un ascensor. En cada uno de estos niveles, no hay muchos "ángulos rectos", y todo es mas de naturaleza fractal. En términos del idioma, uno dice algo que se aproxima a lo que quiere decir, y entonces comienza a refinarlo, de la misma forma en que planeó su itinerario entre los aeropuertos, y solamente más tarde se preocupó en como llegar y salir de cada aeropuerto.

La ambigüedad local está bien

La gente prospera sobre la ambigüedad, siempre y cuando sea resuelta rápidamente. Generalmente, dentro de un lenguaje natural, al ambigüedad se resuelve rápidamente usando temas y palabras de uso reciente. Pronombre somo "eso" se refieren a cosas que están cercanas, sintácticamente hablando. Perl está repleto de pequeñas ambigüedades que la gente nunca nota porque se resuelven en forma rápida. Por ejemplo, muchos términos y operadores en Perl comienzan con los mismos caracteres. Perl lo resuelve basado en lo que espera encontrar, si es un término o un operador, igual que lo haría una persona. Si uno

codifica 1 & 2, Perl sabe que & es una operación AND, pero si uno dice &foo, entonces sabe que estamos llamando a la subrutina "foo"

En contraste, muchos lenguajes fuertemente orientados al tipo de datos tienen una ambigüedad "distante". C++ es uno de los peores en este sentido, porque uno puede mirar a "a + b" y no tener idea de qué está haciendo el operador +, y menos en dónde está definido. Enviamos gente a estudiar para aprender a resolver esta ambigüedades distantes.

Puntuación por prosodia[2] en Inflexión

El lenguaje natural es naturalmente puntuado por los cambios de tono, énfasis y pausas que usamos para indicar cómo se relacionan las palabras. El llamado "lenguaje corporal" también está en juego aquí. Alguna de esta puntuación está escrita en Español[1], pero mucha de esta no lo está, o lo es en forma aproximada. La tendencia en las comunicaciones electrónicas
ha sido inventar varias formas de puntuación :-)

Algunos diseñadores de lenguajes de computadoras tienden a pensar que la puntuación es dañina. No creo que sus maestras/os de idioma piensen lo mismo.

Quitando la ambigüedad a través del número, las mayúsculas y el orden de las palabras

Parte de la razón por la que un lenguaje puede seguir adelante con ciertas ambigüedades locales es que otras ambigüedades son suprimidas por varios mecanismos. En Inglés[2] se usan el orden de números y palabras, con vestigios de un sistema de mayúsculas en los pronombres: ?El hombre miró a los hombres, y ellos le devolvieron la mirada". Está perfectamente claro quién le está haciendo que a quién. Similarmente, Perl tiene marcadores de número en sus sustantivos; esto es, $perro es sólo, uno y @perro es (potencialmente) más de uno, Así que $ y @ son como "este" y "estos". Perl también usa el orden de las palabras: ?sub use? significa algo totalmente distinto de ?use sub?. Perl no hace mucho con las distinción entre mayúsculas/minúsculas como lo hacen los shells que diferencian el uso de la definición usando $ como prefijo. Aunque imagino que si lo permito, se podrían contar a las comillas como un marcador. En un nivel ligeramente más rebuscado, el operador de Perl ?\? es una especie de marcador o indicativo de preposición mas que de uso, Pero como en la mayoría de los lenguajes, las nociones preposicionales comúnmente se expresan en forma posicional dentro de una lista de argumentos (aunque es altamente posible escribir llamadas usando parámetros nominales en Perl y claves de acceso a hashes, a veces funcionan como preposiciones)

```
move $rook from => $qr_pos, to => "kb3";
```

Topicalización

Con respecto a la topizalización, debería precisar que esta oración empieza con una. Un topicalizador simplemente introduce al tema acerca del que se está tratando de hablar. Hay varias formas sintácticas en inglés[3], la más simple es un sustantivo: "Zanahorias, las odio!. El lenguaje Pascal tiene una cláusula "with" que funciona como topicalizador. Los topicalizadores a veces pueden dar una lista de temas, en los que se ven palabras como "para ESTO yESTO, entonces haga ESTO". En Perl hay varias cosas que funcionan como
topicalizadores. Se puede decir :

```
foreach (@dog) { print $_ }
```

Esto también puede usarse singularmente como :

```
for ($some_long_name) { s/foo/bar/g; tr/a-z/A-Z/; print; }
```

Las coincidencias de patrones[4] (y algunos condicionales) tienden a funcionar como topicalizadores en Perl :

```
/^Subject: (.*)/ and print $1;
```

Estructura del discurso

La estructura del discurso significa cómo una alocución más grande que una oración puede ser puesta toda junta. Distintos lenguajes y culturas tienen reglas diferentes de cómo contar un chiste o una historia, por ejemplo, o cómo escribir un libro sobre Perl. Algunos lenguajes de computadoras tienen reglas fijas para grandes estructuras. Me vienen a la cabeza COBOL y Pascal. Perl tiende a ser bastante libre sobre en qué orden uno pone las oraciones, con excepción que es bastante Aristotélico al momento de requerir que uno provea una inicio y
fin explícitos para estructuras grandes, usando llaves. Pero fácilmente podría decirse que #!/usr/bin/perl corresponde a "Había una vez", mientras que __END__ significa "Y vivieron felices por siempre"

Pronominalización

Todos sabemos acerca de los pronombres y sus usos. Hay una cantidad de pronombres en Perl: $_ significa ?eso", y @_ significaría "ellos". (pero $1, $2, etc. también son referencias pronominales a substrings usados en la última coincidencia de patrones -pattern match-, que pueden funcionar como topicalizadores). Dentro de un lazo foreach o un grep $_ no es simplemente una copia del ítem en cuestión, sino un alias. Similarmente, @_ es una lista de referencia a los argumentos de la función, y los argumentos pueden ser modificados cambiando elementos de @_.

Sin armas teóricas para provocar

Los lenguajes naturales son usados por personas que mayormente no le dan importancia a cuan elegante es el diseño de este. A excepción de unos pocos escritores que ponen empeño en asegurarse en hacerlo de la forma más eficiente posible, la gente común usa toda serie de redundancias en su comunicación para asegurarse de ser comprendidas. Usan cualquier palabra que tienen a la mano para hacerse entender, y trabajan sobre ello hasta que lo logran. Comúnmente esto no es un problema. Son proclives a aprender un nuevo vocablo si ven que será útil, y en contrario que los abogados o profesionales de computación, que sienten poca necesidad de definir cantidades de nuevas palabras para decir lo que quieren.

En términos de lenguajes de computadora, esto argumenta en favor de predefinir los conceptos usados más comúnmente para que las personas no sientan la necesidad de hacer tantas definiciones. Algunos pocos scripts de Perl no contienen definiciones en absoluto. Los reto a encontrar un programa en C++ sin una definición.

Estilo impuesto sólo por presión de pares

No todos tenemos que escribir como Faulkner, o programar como Dijkstra. Le diré a cualquiera cuál es mi estilo de programación, y les diré dónde creo que su propio estilo no es claro o me hace dar vuelta a través de lazos (loops) mentales. Pero lo voy a hacer como un par, no como un Dios del Perl. Algunos diseñadores de lenguajes esperan reforzar un estilo a través de medios tipográficos tales como forzar (mas o menos) una sentencia por línea. Esto está muy bien para la poesía, pero no creo que quiera forzar a cada uno a escribir poesía en Perl. Tales límites estilísticos deberían ser auto- impuestos, o al menos a través de políticas por consenso entre sus pares.

Diseño cooperativo

Nadie diseña un lenguaje natural por si mismo, a menos que su nombre sea Tolkien. Todos contribuimos a diseñar nuestro propio lenguaje ya sea tomando prestado o acuñando nuevos términos, copiando lo que creemos es ?cool? y evitando lo que creemos es confuso. Los mejores lenguajes artificiales son colaborativos, aún con un lenguaje como Perl en el que una persona parece estar a cargo. La mayoría de las buenas ideas de Perl no son mías. Algunas de ellas vienen de otros lenguajes, y algunas fueron sugerencias hechas por varias personas
a medida que avanzamos. Si considera que el lenguaje debe incluir varias trampas culturales (bilbiotecas, directorio bin) entonces siga adelante con el lenguaje, aún lenguajes como C, Ada, C++ o aún los shell de Unix son colaboraciones de muchas personas. Perl no es la excepción.

Divergencia inevitable

Debido a que un lenguaje es diseñado por muchas personas, cualquier lenguaje inevitablemente diverge en dialectos. Es posible dilatar esto, pero para cualquier lenguaje vivo las fuerzas de la divergencia son mas fuertes que las de convergencia. POSIX trató de unificar System V y BSD, y tan pronto como estrecharon filas en esa dimensión, la cantidad de variantes de Unix explotaron en otras dimensiones. La lección para un diseñador de lenguajes es construir un mecanismo explícito que haga fácil identificar con qué variante del lenguaje se está tratando. Perl 5 tiene este mecanismo explícito de extensión el que especifica, usando cláusulas use, en qué clase de semántica o dialecto se está basando. Perl 4 no tenía

esto, y había considerablemente mayor presión para poner varias cosas en el lenguaje que no estaban en el núcleo del mismo. Afortunadamente ahora podemos estabilizar
un Perl básico de tal forma que haya menos necesidad de inventar oraperl, sybperl, isqlperl, etc.

Bueno, hasta aquí es donde llega este documento, espero que hayan captado al esencia de Perl y perl. Simplemente lo que resta es ver que nos depara perl6. Algunos de los lineamientos para perl6 :

- - las bases para perl6 son las mismas que para perl5 pero adicionalmente va a soportar múltiples sintaxis. En cierta forma ahora lo está haciendo cuando se usa un módulo o se agrega algún pragma (use warnings, use strict, etc.)
- - perl soporta múltiples paradigmas, y como tal no se inclina por ninguno en particular con lo cual el peligro que cualquiera de los paradigmas de programación avance sobre perl6 es caso inexistente. perl continuará siendo perl
- - perl seguirá siendo libre de limitaciones, tales como warnings y strict
- - la implementación de objetos de perl6 será mucho más adecuada. En particular en perl5 esta es algo pesada y si se continúa con esta en perl6 podría convertirse en el factor principal de la caída de Perl.

En general debemos decidir cuanto de nuestra cultura sobrevivirá, y cómo hacer una migración lo más simple posible hacia perl6.

Infografía

- - CPAN (http://www.cpan.org)
- - Perl Mongers (http://www.pm.org)
- - Perl Monks (http://www.perlmonks.com)
- - Natural Language Principles in Perl (http://www.wall.org/~larry/natural.html traducido al español en http://www.bit-man.com.ar/es/SoftwarePerlNatural)
- - Apocalipsis 1 : The ugly, the bad and the Good (http://dev.perl.org/perl6/doc/design/apo/A01.html)
- - Sinopsis 1 : Overview (http://dev.perl.org/perl6/doc/design/syn/S01.html)

POC (peace of code)

Este mes el acertijo lo propuso Félix Cuello.
Dado el siguiente código :

```
{
my $a;
$a = \$a;
}
```

- - viola alguna regla de perl5 ??
- - qué utilidad tiene ??
- - qué ocurre con el garbage collector ??

NOTES

[1] N. del E.: muppet es quien se puede ver en http://seasonone.org/sights/2005-06-25/scotta2.jpg] [2] N. del T. : en el original ?here you go? algo así como ... y bueno ya que empezaste a caminar podés seguir haciéndolo
[3] Especificación del lenguaje
[4] Implementación del lenguaje Perl
[5] N. del T. : en español pasa lo mismo
[6] N. del T. : Parte de la gramática que enseña la correcta pronunciación de las palabras
[7] N. del T. : en el original se refiere a que el texto está escrito en Inglés
[8] N. del T. : en español también !!
[9] N. del T. : también en español
[10] N. del T. : pattern matches