

Resolución de problemas con Vim

# ENTREVISTA DE TRABAJO

Christos Georgiou, Fotolia

El editor Vim soporta plugins en Perl que permiten al usuario manipular cualquier texto recién editado. Se pueden desarrollar complejas funciones de manera mucho más rápida que con el lenguaje de script integrado en Vim. **POR MICHAEL SCHILLI**

**S**i alguien solicita un puesto de empleo básico en la división Perl de Yahoo y tiene el placer de visitarme para una entrevista de trabajo, seguramente le preguntaría la siguiente cuestión: “¿Cómo añadirías números de línea a un listado en Perl como los que se suelen usar en las revistas de informática?”

## Estrés

Añadir números de línea a listados en Perl es una tarea bastante sencilla, y probablemente la mayoría de los candidatos sabría resolverla. Pero otros seguramente tendrían dificultades a la hora de numerar las líneas. Si el listado tiene nueve líneas, todas tienen un único dígito, pero los números tienen dos dígitos para las líneas 10 a 99, y podemos entonces añadir un cero a la izquierda (01 a 09). De este modo los listados aún mayores, de más de 100 y menos de 1000 líneas necesitan números de tres dígitos, con lo que la numeración comienza con 001.

La función Perl *printf()* tiene una opción para dar formato a los números con ceros adyacentes. Una cadena para formatear tal que *%03d*, convierte el entero 3 en la cadena *003*. 99 queda convertido en *099*, y 100 se queda en *100*.

¿Pero cómo se las arregla *printf()* para rellenar la cadena con una longitud variable? Si alguien me sugiere una construcción *if/elsif* que verifica un número limitado de longitudes variables de los números, entonces suena una alarma y se abre la compuerta de la piscina con tiburones...

Si el ancho de la columna de mayor valor se guarda en la variable *\$numlen*, podríamos simplemente adjuntar la cadena formateada antes de pasarla a *printf*. Lo hacemos concatenando *"%0" . \$numlen . "d"*.

Nótese que si hemos escrito todo en una sola cadena sin prestar mayor atención, Perl interpreta que *"%0\$numlend"* hace referencia a una inexistente variable denominada *\$numlend*. Los iniciados en Perl saben

```
mschilli@mybox:/home/mschilli/DEV/articles/vimscript/e
#!/usr/bin/perl -w
#####
# linenum - Print listing with numbers
# Mike Schilli, 2007 (m@perlmeister.com)
#####
use strict;

my @lines = <>;

my $numlen = length scalar @lines;

my $num = 1;

for my $line (@lines) {
    printf "%0" . $numlen . "d %s",
        $num++, $line;
}

1,1 All
```

Figura 1: El listado sin...

```
mschilli@mybox:/home/mschilli/DEV/articles/vimscript/eg
01 #!/usr/bin/perl -w
02 #####
03 # linenum - Print listing with numbers
04 # Mike Schilli, 2007 (m@perlmeister.com)
05 #####
06 use strict;
07
08 my @lines = <>;
09
10 my $numlen = length scalar @lines;
11
12 my $num = 1;
13
14 for my $line (@lines) {
15     printf "%0" . $numlen . "d %s",
16         $num++, $line;
17 }
:silent :1,$!linenum 1,1 All
```

Figura 2: ...y con los números de línea alineados, con sólo presionar un botón.

que un escalar puede escribirse como `${numlen}` en lugar de `$numlen`, y esto salva la situación: `"%0${numlen}d"` refiere correctamente a `$numlen` y añade "d" al final.

Aquellos que crecieron con C pueden que recuerden que `printf()` soporta campos de formato variables mediante el asterisco (`*`) como comodín y un parámetro adicional. Una expresión como `printf("%0.*d", 3, 1)` añade al número 1 tres dígitos mediante ceros adyacentes.

Así mismo, los programadores pueden reemplazar el 3 con una variable para añadir un número de línea con un ancho que varía dinámicamente.

## De Vuelta Al Cole

¿Cómo podríamos saber la longitud de `$numlen` del último número de línea `$num`? Es decir, ¿cuántos dígitos tiene `$num`? Como sabemos, Perl no tiene problema alguno en convertir números a cadenas, y la función integrada `length()` nos dará la longitud de la cadena. Una llamada a `length($num)`, por tanto, devuelve el valor que necesitamos para `$numlen`.

Otra opción: en el sistema decimal los dígitos se ponderan de derecha a izquierda:  $10^0$ ,  $10^1$ ,  $10^2$ , etc. Por tanto, el número 15 puede descomponerse en  $5 * 10^0 + 1 * 10^1$ . El número 100, un número de tres dígitos, también puede expresarse como  $1 * 10^2$ . El número 1000, un número de 4 dígitos, equivale a  $1 * 10^3$ .

Por tanto, ¿cuántos dígitos tiene el número  $N$ ?

Si en su día prestamos atención en el colegio, probablemente recordaremos que si queríamos calcular el resultado de "10 elevado a qué potencia da  $N$ ", teníamos que establecer el logaritmo decimal de  $N$ . Aunque Perl no dispone de logaritmo decimal, tiene la función `log()` para calcular el logaritmo de un número en base  $e$  (el número de Euler). Y si tenemos ya memoria de elefante, incluso recordaremos que el logaritmo de  $N$  en base  $x$ , es decir  $\log_x N$  puede calcularse dividiendo `log_y N` entre `log_y x`.

En nuestro ejemplo, podemos calcular el logaritmo decimal de  $N$  en Perl como `log(N)/log(10)`. La longitud del número  $N$  es el resultado de la opera-

ción logarítmica redondeada al entero más cercano e incrementado en 1. Nótese que, sin embargo, debido a la manera en la que el ordenador calcula los logaritmos, puede aparecer un pequeño error. Si obtenemos 10.999999999 en lugar de 11, estamos ante un error de superación de límite (off-by-one error).

## El Script Linenum

El script `linenum` (véase el Listado 1) muestra una solución al problema. Esta solución comienza cargando al array `@lines` las líneas de un script que se leen desde un archivo o desde STDIN. Por supuesto, esto sólo tiene sentido para archivos pequeños, pero si alguien es capaz de escribir scripts en Perl de más de 100.000 líneas, tal vez debería reconsiderar sus opciones profesionales. El operador punto de Perl, (`"."`) construye la cadena de formato, resultando algo tal que `"%02d %s"`.

Lo bueno de esta pregunta para una entrevista de trabajo es que no se trata simplemente de un problema sin sentido que un candidato pueda haber resuelto previamente o no, o que pueda resolverlo en ese momento de tanto estrés. Si el candidato no ve la solución inmediatamente, el examinador puede ayudarlo un poco y ver cómo reacciona ante las sugerencias.

Hay muchas soluciones posibles, cada una con sus propias ventajas e inconvenientes. ¿Qué ocurre si tenemos un archivo de 10GB? ¿Qué método es el

### Listado 1: `linenum`

```
01 #!/usr/bin/perl -w
02 use strict;
03
04 my @lines = <>;
05
06 my $numlen = length scalar
07     @lines;
08 my $num = 1;
09
10 for my $line (@lines) {
11     printf "%0" . $numlen . "d
12         %s",
13         $num++, $line;
14 }
```

más rápido? ¿Qué consideraciones añadidas se necesitan para admitir un archivo con caracteres Unicode?

## Perl Externo

¿Cómo se podría hacer esto con Vim? (presionar un botón y numerar un listado completo en único barrido). La solución más sencilla es pasar las líneas de un rango por un script como filtro.

Un comando como `:1,$!lineum` coge todas las líneas del archivo editado (desde la línea 1 a la última línea \$) y usa STDIN para enviarlas al script `lineum`, que se ejecuta como un programa externo gracias al signo de exclamación. Vim recoge la salida del script y reemplaza las líneas originales con los resultados del filtro. La siguiente sentencia `map` en `.vimrc` mapea este comando a la tecla `L` del modo normal,

```
:map L :silent Z
:1,$!lineum<Return>
```

suponiendo que `lineum` es ejecutable y en una ruta accesible por el shell que estemos usando.

La opción `:silent` suprime la salida por pantalla, facilitando al comando que se ejecute suavemente sin que Vim saque mensajes de estado por la consola y soli-

### Listado 2: vimperl

```
01 :function! Linenum()
02
03   :if !has('perl')
04     :echo "Sorry, no Perl!"
05     return
06   :endif
07
08   perl <<EOT
09     $numlen =
10     length($curbuf->Count());
11     for $num
12       (1..$curbuf->Count()) {
13         $newline = sprintf
14         "%0.*d %s", $numlen,
15         $num, $curbuf->Get($num);
16         $curbuf->Set($num,
17         $newline);
18       }
19   EOT
20 :endfunction
21
22 :command! Linenum :call
23   Linenum()
```

citando confirmación por parte del usuario. El comando `<Return>` simula la pulsación de la tecla Return/Enter. Si no usamos esta opción, Vim llena la línea de comandos y aguarda a que presionemos Enter para confirmar.

Si sólo queremos añadir números de línea a una sección del documento entre el marcador `a` y el marcador `b` en lugar de todo el documento, debemos definir el área como `'a,b` en lugar de `1,$`.

## Intérprete Perl

Vim tiene también un intérprete Perl incluido, pero debemos configurar esta funcionalidad explícitamente antes de compilar Vim. Vim sabe en tiempo de ejecución si el intérprete Perl está disponible y proporciona esta información mediante la función `has('perl')`, que devuelve el valor verdadero en caso de encontrarlo.

Ejecutando esta verificación en nuestros scripts Vim antes de usar una función Perl, aseguramos que Vim finalizará con un mensaje de error claro si es que falta Perl, en lugar de tropezar con comandos que simplemente no es capaz de entender.

## Script Vimperl

El script `vimperl` (véase el Listado 2) usa el lenguaje de script de Vim para definir una función llamada `Linenum()`, la cual inserta los números de línea en el archivo que estamos editando. Nótese que Vim requiere que las funciones definidas por el usuario comiencen con letra mayúscula. En caso de que `has('perl')` indique que no tenemos el intérprete Perl instalado, el comando `:echo` de Vim muestra un mensaje en la línea de estado.

La documentación relativa a Perl en Vim [3] indica que `$curbuf` del intérprete Perl apunta automáticamente al búfer de edición activo, que contiene las líneas del archivo que estamos editando en ese momento. El método `Count()` devuelve el número de líneas del búfer en un entero.

A continuación el bucle `for` itera por todas las líneas del búfer en uso, utilizando `$curbuf->Get($num)` para leer cada línea, donde `$num` representa el número de la línea del búfer que está siendo procesada en ese momento.

La función `$curbuf->Set()` envía entonces la línea, con su flamante

número nuevo, de vuelta al búfer. La función aguarda el número de línea y el nuevo contenido como argumentos.

En el lenguaje de script de Vim, los comentarios comienzan con comillas dobles y aplican hasta el final de la línea en cuestión.

Si hacemos la llamada a `:source vimperl` en Vim se carga el script `vimperl`, pero deberíamos añadirlo al archivo de inicialización de Vim, `.vimrc`, o bien guardarlo en uno de los directorios de plugins de Vim para su uso en producción.

## ¡Función!

Durante las pruebas, resulta bastante práctico usar `:function!` (con el signo de exclamación incluido) para definir las funciones de Vim, indicándole que sobrescriba la función sin preguntar en caso de que ya esté definida. De otra manera, el proceso de carga se cancela y se muestra un mensaje de error.

Dentro del script de Vim, el script Perl está disponible como un documento que comienza con `<<EOT` y termina con `EOT`. Nótese que el `EOT` final debe estar al comienzo de la línea para que Perl reconozca el final del script.

Después de que `endfunction` indique el final de la definición de la función de Vim, `vimperl` realiza la llamada `:command` para definir un comando `Linenum`, al que los usuarios pueden llamar tecleando `:Linenum` en la línea de comandos de Vim o incluso mapear en una tecla. Vim de nuevo requiere que la primera letra del comando definido por el usuario comience por letra mayúscula. El comando `:map L : Linenum <Return>` mapea el comando a la tecla `L` en el modo normal.

De acuerdo, ya sabéis mis trucos para las entrevistas de trabajo, por lo que tendré que buscarme nuevas preguntas. Buscaré algo de la misma dificultad, pero incluso más divertido. ■

## RECURSOS

- [1] Listados de este artículo: <http://www.linux-magazine.es/Magazine/Downloads/38>
- [2] Página del proyecto Vim: <http://www.vim.org>
- [3] Espartana documentación de Vim acerca de Perl: [http://www.vim.org/html/doc/if\\_perl.html](http://www.vim.org/html/doc/if_perl.html)