

Perl para administradores



por
Matías “nsss” Palomec
matias@cafelug.org.ar



Perl

Perl es un lenguaje creado por Larry Wall

En una primera instancia lo utilizó para crear reportes y trabajar con texto, por eso desde un principio se lo oriento al manejo de texto y es tan útil para Unix (donde todo es un archivo de texto)

buscar a root

Una tarea simple es el parsear texto y mostrar una parte en Perl, por ejemplo, buscar usuarios con uid o gid "0" (usuarios con permisos de root/wheel):

```
perl -F: -ane 'print "$F[0]\n" if(($F[2] eq "0") || ($F[3] eq "0"));' /etc/passwd
```

command line options

- **-F:** toma como separador de campos el “:” (lo que siga a la F)
- **-a** activa el modo de auto separación (los valores quedan en el array @F)
- **-n** hace que el comportamiento sea parecido a awk/sed (abre el archivo y lo lee directamente)
- **-e** en vez de ejecutar un archivo, se pasa el código desde la línea de comandos (para los llamados POL -Perl One Liner-)

modificación masiva de archivos

Continuando con el parseo de texto se puede intentar buscar una IP (por ejemplo la 10.65.33.39) vieja que por alguna razon se la ha cambiado por otra (por ejemplo la 10.33.23.92) en muchos lugares (por ejemplo todo el /etc)

```
find /etc -type f | perl -i.bak -pe  
's/10.65.33.39/10.33.23.92/g;'
```

command line options

- **-i.bak** hace una copia de seguridad de los archivos que se pasan como argumento dejandoles la extension que se indique (se puede agregar un simple ~)
- **-p** una de las opciones mas utiles, es el parser

command line options

El modificador que hace a perl un parser es una simple forma de ahorrar código, reemplaza a:

LINE:

```
while(<>){  
  # código  
}continue{  
  print or die "-p destination: $!\n";  
}
```

-ane + logs

Hay varias formas de interpretar logs, muchos de los cuales necesitan que la información se ingrese de cierta forma. Con Perl se puede modificar (por ejemplo el log del apache):

```
cat nns.access.log |perl -ane 'print "$F[0] -> $F[6]\n"
if(/mygal/ && !(/10\.\/));'
```

la explicación

El modificador **-ane** hace que quede disponible la variable **@F** conteniendo los valores de cada línea separados por espacios. Con lo cual se puede mostrar en pantalla solo el primer valor (el **\$F[0]**) y el séptimo valor (el **\$F[6]**) de todas las líneas que contengan el string mygal pero que no contengan algo que sea (literalmente) un "10."

totalizando valores

Una cosa bastante util es totalizar valores para hacer reportes rapidos:

```
cat nss.access.log |perl -ane 'BEGIN{%con;} if(/\/mygal\/\&& !(\/^10\./)){$con{$F[0]}=0 unless( defined($F[0]) ) ; $con{$F[0]}++;} END{print "total:\n";my @o = sort{ $con{$a} <=> $con{$b} } keys %con; foreach my $key (@o){print "$key -> $con{$key}\n";}}'
```

explicación

Perl, dado que puede comportarse como parser, tiene la posibilidad de definir bloques que se ejecutarán una sola vez, estos bloques son: BEGIN y END (BEGIN al principio y END al terminar la ejecución). En este caso tiene poca utilidad dado que el conteo es más complejo y necesita un hash, pero puede iniciarse un contador ($\$c = 0$).

explicación (cont.)

El bloque END tiene el código que tiene mas utilidad del ejemplo, y es donde se visualizan los totales ordenados por repetición.

El código del medio es simple, si el hash para la posición dada no esta definido, lo inicia en cero, y luego suma uno en la posición dada.

control remoto (sockets)

Una facilidad que tiene Perl es manejar sockets. Una forma simple de controlar un WebServer (o algún AppServer, etc) y verificar su funcionamiento “esperado”. Por ejemplo, buscar en un rango dado si un socket esta abierto:

```
use Socket;  
my %host;
```

```
for my $i (1..25){  
    my $sin = sockaddr_in(22,inet_aton("10.0.0." . $i));  
    socket(SH,PF_INET,SOCK_STREAM,getprotobyname("tcp"));  
    connect(SH,$sin) || next;  
    $host{"10.0.0.$i"} = 1 unless($host{"10.0.0.$i"});  
    close(SH);  
}
```

```
foreach my $k (sort keys %host){ print "$k\n"; }
```

explicación

La idea del script es hacer un loop de 1 a 25 (podría ser 254 o lo que sea), e intentar abrir un socket tcp al puerto 22 (indicado en sockaddr_in). Y si la conexión es exitosa, guarda la IP para luego informarla.

Este script puede variar y quedar:

```
use Socket;  
my %host;
```

```
for my $i (1..25){  
    my $sin = sockaddr_in(80,inet_aton("10.0.0." . $i));  
    socket(SH,PF_INET,SOCK_STREAM,getprotobyname("tcp"));  
    connect(SH,$sin) || next;  
    my $msg = "GET / HTTP/1.0\n\n";  
    send(SH,$msg,PF_INET);  
    while(my $l = <SH>){  
        if($l =~ /^Server:.*Apache/){  
            $host{"10.0.0.$i"} = 1 unless($host{"10.0.0.$i"});  
            last;  
        }  
    }  
    close(SH);  
}  
foreach my $k (sort keys %host){ print "$k\n"; }
```

explicación

Advertencia: En rangos grandes este ejemplo tarda MUCHO

La primera variación es el agregarle el “send” para enviarle un comando al server que nos conectamos para luego leer la respuesta. Dentro de la respuesta se busca la linea que empieza con Server y tiene luego Apache. Esto es el RFC 1945

explicación (cont.)

La sintaxis del send es:

```
send(Socket_Handler,$msg,FLAG)
```

Los flags son flags los flags que se encuentran en socket.h de C (en linux el archivo es /usr/include/linux/socket.h y en FreeBSD es /usr/include/sys/socket.h). El el archivo en C hay una descripción de cada flag a utilizar.

explicacion (cont)

La respuesta del socket se puede leer como un archivo (`my $line = <Socket_Handler>`). O puede usarse (mas conveniente para leer archivos binarios) el comando `recv`. La sintaxis es:

```
recv(Socket_Handler,$var,$length,FLAGS)
```

La variable donde se aloja la informacion es `$var`, `$length` es el tamaño a leer.

¿dudas, preguntas?



sobre la presentacion

Creado por Matías Palomec
Licencia

<http://creativecommons.org/licenses/by-nc-sa/2.5/ar/>

