Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Tutorial de Expresiones Regulares (190506 lectures)

Per **Daniel Rodriguez**, ***DaniRC*** (http://www.ibiza-beach.com/)

Creado el 30/07/2001 12:25 modificado el 15/02/2002 12:45

Había escrito otro articulo sobre los conceptos avanzados de las Expresiones Regulares[1] Pero seguía faltando un tutorial que las acercara un poco más a todos nosotros.
SUPER Actualizado: Mas de 6 páginas nuevas, mas de 50 ejemplos nuevos, tutoriales completos de GREP, SED, AWT, y mucho más! Todo gracias a la colaboración especial de un lector.
Nota: Todos los textos usados en el articulo son propiedad de sus respectivos autores y este artículo unicamente pretende ser una via para su divulgación.

Tutorial de Expresiones Regulares

No hace mucho publique un artículo sobre **la parte complicada de las expresiones regulares**, pero no caí en la cuenta de que no habia publicado nadie en Bulma un **tutorial para no iniciados**. Así que me he puesto manos a la obra. Esta vez no es una traducción, pero poco le falta, despues de todo no es que exista mucha documentación en castellano al respecto, ¿verdad?.

Introducción

Las *expresiones regulares vienen a ser una forma sofisticada de hacer un bucar&reemplazar.* En el mundo windows no tienen mucho sentido, despues de todo alli casi todo va a base de clicks. Pero en el mundo Unix/Linux, en el que casi todo son ficheros de texto, son casi una **herramienta imprescindible**. No tan solo de cara al administrador, sino tambien de cara a cualquier otro programador que puede ver como las expresiones regulares le salvan la vida en mas de una ocasión.

Particularmente llevo un tiempecillo dedicado a la programación y diseño web. No es de extrañar que un cliente que insistió en que su e-mail estuviera en cada página con un mailto: cambie de mail. Tampoco es de extrañar que hallas escrito una palabra mal 2 veces en 50 páginas distintas ... Y tampoco es especialmente raro que tengas que ir con cuidado para que la informacion que sacas no esté en el tag html adecuado. Por ejemplo ... hay clientes muy raritos que quieren que todo lo que hasta ahora era cursiva ... se vuelve negrita, pero solo si el contenido de la frase usa la palabra "clave".

¿**Os imaginais este problema en MSWord**? Busca una *frase que contenga la palabra "clave"*, ahora *mira si esta entre tag's de cursiva * y ahora *reemplaza por *

Pues para estas cosas se inventaron las expresiones regulares ;)

Mientras espero con vosotros ese dia en que los clientes no me compliquen la vida... es un consuelo saber que existen las **¡EXPRESIONES REGULARES!**

Nota: Todo lo que explico esta basado en las expresiones regulares de **PERL**. **Sed** por ejemplo no tiene porque funcionar exactamente igual. Pero la idea es basicamente la misma.

Presentando los carácteres especiales

```
[ ] cochetes
() parentesis
{} llaves
- guión
+ más
* asterisco
. Punto
^ circumflejo
$ dolar
? interrogante cerrado
| tuberia unix
\ barra invertida
     (se usa para tratar de forma normal un caracter especial)
/ barra del 7
```

Mención aparte para / puesto que es el simbolo que se usa para indicar la búsqueda. El resto son todo modificadores y se pueden usar sin restricciones.

Definiendo Rangos

```
/[a-z]/ letras minusculas
/[A-Z]/ letras mayusculas
/[0-9]/ numeros
/[,'¿!¡;:\.\?]/ caracteres de puntuacion
                -la barra invertida hace que
                no se consideren como comando
                ni en punto ni el interrogante
/[A-Za-z]/      letras del alfabeto (del ingles claro ;)
/[A-Za-z0-9]/   todos los caracteres alfanumericos habituales
                -sin los de puntuacion, claro-
/[^a-z]/        El simbolo ^ es el de negación. Esto es decir
                TODO MENOS las letras minusculas.
/[^0-9]/        Todo menos los numeros.
```

Para definir otros rangos, no dudeis en usar el operador de rangos "-" por ejemplo de la h a la m [h-m] ¿vale?

Coincidencia total

La expresion regular suele funcionar con que solo coincida un caracter de los muchos listados. Pero en ocasiones queremos que la coincidencia sea total.

Imaginemos que nos hemos dejado todas las tildes de palabras acabadas en ion. Se nos ocurre usar /[ion]/ como patron de busqueda.

Pero esto daria concordancia positiva con Sol -por la o- con noche -por la n- ... etc.

Para que la concordancia sea de cada caracter y en el orden adecuado, necesitamos el operador Punto "."
Por ejemplo [.ion] solo concordaria con cosas como camion o salsa lionesa

Si bueno, la "ion" de l**ion**esa no esta a final de palabra ... y eso tambien hemos de retocarlo, pero todo a su tiempo.

Aunque se os podria ocurrir que algo del tipo [.ion[^a-zA-Z]] tal vez funcionase ... ya veremos ;)

Por cierto que el punto "." no concuerda con \n \r \f \0 (newline, return, line feed, null respectivamente)

Agrupando, referenciando y extrayendo

Agrupación

En ocasiones nos interesa usar el resultado de una parte del patrón en otro punto del patrón de concordancia.

Tomemos por ejemplo una agenda en la que alguien puso su telefono en medio de su nombre.(Pepe 978878787 Gonzalez) Queremos extraer el telefono de esa frase y guardarlo para introducirlo en otro sitio. O para usarlo como patron de busqueda para otra cosa.

Lo primero que hay que hacer es agrupar. Agrupar permite que el resultado del patron se almacene en una especie de registro para su uso posterior. El **operador de agrupamiento son los parentesis ( )**

Los registros se llaman 1, 2, 3, ... 9 segun el orden del agrupamiento en el patrón.

```
Ejemplo (del libro de perl): /Esto es ([0-9]) prueba/
si el texto fuente es: Esto es 1 prueba.
El valor 1 seria almacenado en el registro $1
```

Referencias

Tenemos una serie de elementos agrupados, a los cuales se les ha asignado un valor. Ahora queremos usar ese valor en algun otro punto del patron de concordancia. Pues para eso existen las referencias.

\1 representa la referencia al grupo 1

```
Ejemplo (del libro de perl): /([0-9]) \1 ([0-9])/
si el texto es : 3 3 5
El resultado es: $1 --primer grupo-- vale 3
\1 --hace referencia al resultado del primer grupo-- vale 3
$2 --segundo grupo-- vale 5
```

En este caso hubiese habido concordancia, no asi si el texto hubiese sido 3 5 3

Extracción

La extracción es simplemente usar el **$**1 en los mensajes que se generan despues de haber usado la expresion regular.

Expresiones opcionales y alternativas.

Si os habeis fijado hasta ahora todo lo que hemos visto han sido condiciones mas bien del tipo AND (Y)

Si hay un numero Y hay un texto Y no hay otro numero ... blablabla. Siempre Y, es evidente que nos faltan las condiciones de tipo opcional y alternativo.

Opcionales

el simbolo que se emplea para indicar que una parte del patron es **opcional es el interrogante. ?**

```
/[0-9]? Ejemplo/
Esto concuerda tanto con textos del tipo
1 ejemplo, como con
ejemplo
```

Alternativas

No confundamos alternativas con opciones, la alternativa es el equivalente a la OR no lo era asi el operador opcional. Veamos la diferencia. Por cierto, el simbolo de la **alternativa es |**

```
/[0-9]? (EJEMPLO|ejemplo)/
a: 1 ejemplo
b: 1 EJEMPLO
```

```
c: 1
d: EJEMPLO
```

a,b,d concuerdan con el patron. Sin embargo c no concuerda con el patrón! Porque ejemplo o EJEMPLO son alternavitas válidas, pero tiene que haber uno de los 2. No ocurre lo mismo con la opcionalidad, el 1 puede estar o no estar. Esa es la principal diferencia.

Contando expresiones

A estas alturas te estas preguntando qué mas se puede hacer con una expresión. Pues la verdad es que ya queda poca cosa, pero lo de contarlas! vamos ... eso es imprescindible.

El operador para esta ocasión son las llaves {m,n}

Donde m, n son 2 enteros positivos con n mayor o igual que m. Esto se lee asi ... la expresion debe cumplirse al menos m veces y no mas de n veces.

Podemos usar {n}o bien {n,} para indicar que queremos que se repita exactamente n veces o que se repita n veces o más respectivamente.

```
De tal forma que el patrón /go{1,4}l/
a: gool
b: goooooooooool
c: gooool
concordaria con a y c pero no asi con c
```

Expresiones repetidas

Tenemos el **operador * asterisco** que representa que el patrón indicado debe aparecer **0 o mas veces** y el **operador + suma** que representa que el patrón indicado **debe aparecer 1 o más veces (pero almenos 1)**

Mucho ojo con el operador asterisco!!! leeros el articulo relacionado sobre conceptos avanzados de las expresiones regulares antes de usarlo o no me hago responsable de los resultados.

Comodines y abreviaturas

- \w para indicar word (alfanumericos y _)
- \W para indicar lo opuesto al word.
- \s para concordar con los caracteres los espacios y otros caracteres en blanco (\t \n \r y espacio)
- \S para lo contrario a \s
- \d para concordar con un digito
- \D para lo contrario al \d
- \A para empezar a mirar por el principio del string
- \Z para empezar a mirar por el final de string
- \b concuerda con las "word boundaries" los limites de palabra
- \B lo opuesto a \b

El operador ^ dice al compilador que empiece por el principio de la linea y $ indica que empiece por el final de la linea. Estos operadores cumplen este efecto cuando estan fuera de la expresion regular, evidentemente. Porque dentro de las barras tienen otras funciones ya comentadas anteriormente.

Y con esto creo que acabamos el listado de nuestras herramientas de busqueda. Queda sin embargo pendiente de explicar lo referente al reemplazo, con lo que por fin daremos solución al problema que plantee en la primera página.

Buscar y Reemplazar

El manual dice lo siguiente

```
[m]/PATRON/PATRON REEMPLAZO/[g][i][m][o][s][x]

Donde mviene a ser el ambito de la busqueda.
^ inicio de linea. $ fin de linea.
% para todo el documento.
1,15 para las lineas de 1 a 15 ...

PATRON es el patron de busqueda del que hemos
estado hablando todo el rato.

PATRON REEMPLAZO es lo mismo que el patron de
busqueda solo que este se usa para el reemplazo

g (global) reemplaza TODAS las ocurrencias
en el texto.

i (insensitive) es para evitarnos problemas
con la capitalizacion. A = a

o (interpola variables solo una vez)
        No me mireis asi que yo tampoco se lo que es eso XD

m (multiples lineas) Acepta strings de varias lineas

s (single line) solo mira el string de una linea.

x (extensiones) permite usar extensiones de expresion regular.
Insisto, yo tampoco se como se usa eso XD
```

Ejemplos en varios entornos.

Tal como prometi algunos ejemplos prácticos. Lo primero, advertiros que es una parte que ire cambiando con el tiempo. He descubierto que no todas las expresiones regulares soportan la misma notación. Si bueno ... deberia haberlo sabido, pero no lo sabia. Asi que hasta que me ponga las pilas con el sed ... voy a tocar un poco el tema del **PERL**

Lo primero es lo primero:
Tenemos el PERL en nuestra maquina muerto de risa asi que vamos a cambiar eso ;-)

editad un fichero con el nombre "ejemplo.pl" -por ejemplo-
En la primera linea poneis:

```
!#/usr/bin/perl
```

Si el perl no esta ahi, buscadlo y poned el path correspondiente en general */usr/local/bin/perl*
Ahora podriamos hacer el tipico **print "Hello Wordl\n";** tras lo cual cerramos el fichero e intentamos ejecutarlo.
Necesitaremos hacer **chmod u+x ejemplo.pl** para dar permiso de ejecucion al usuario en curso sobre el fichero.
Ahora tecleando **./ejemplo1.pl** ... deberia saliros en pantalla "Hello World". Lo que indicaria que el script funciona. De **no ser asi** os sugiero que repaseis los pasos y que os asegureis de que el perl existe en el directorio que habeis pasado en el path del principio del fichero.

Ejemplo 1 en PERL

```
#!/usr/bin/perl5
$texto="hola clave buena";
if ($texto=~ /<i>([a-z\s]{1,})
```