Guido Socher

Acerca del Author: Le gusta Linux porque es un sistema libre y se diverierte trabajando con gente de la comunidad Linux de todo el mundo. Pasa su tiempo libre con su novia, escuchando la radio de la BBC World Service, montando en bicicleta por el monte y disfrutando con Linux.

Escribe al autor

Índice de contenidos:



# Expresiones Regulares


Resumen: Las expresiones regulares se utilizan para hacer búsquedas contextuales y modificaciones sobre textos. Se pueden encontrar en muchos editores de textos avanzados, en programas de análisis gramatical y en muchos lenguajes.

---

## Introducción

Las Expresiones Regulares se pueden encontrar en muchos editores como vi, emacs, en programas como grep, egrep y en lenguajes de programación como awk, perl y sed.

Las Expresiones Regulares se usan para busquedas avanzadas dependientes del contexto. Por eso se puede decir que las Expresiones Regulares no sólo son un buscador de cadenas de texto, sino que son la descripción formal de un patrón de texto.

Cuando vi por primera vez cómo se podían editar textos con ayuda de Expresiones Regulares, me quedé fascinado. Modificaciones en un texto, para las que hubiese necesitado horas, quedaban listas en pocos segundos. Lo que sin embargo veía en el monitor, parecía un conjunto de puntos, rayas y otros símbolos. Sin embargo algo tenía claro: Quería aprender el lenguaje de las Expresiones Regulares y me quedé impresionado de lo fácil que era de entender. Sigue unas cuantas reglas de sintaxis muy sencillas.

A pesar de que las Expresiones Regulares estén muy extendidas por el mundo de Unix, no existe un lenguaje estándar de Expresiones Regulares. Más bien se puede hablar de diferentes dialectos. Existen por ejemplo dos representantes del conocido programa grep, egrep y grep. Ambos usan Expresiones Regulares con capacidades ligeramente diferentes. Perl se puede calificar como el lenguaje con la sintaxis de Expresiones Regulares más desarrollado. Por suerte todos estos dialectos siguen los mismos principios y en el momento que se han entendido, el resto es sencillo.

En este artículo me dedicaré únicamente a los principios básicos, los detalles se pueden consultar en las man-pages.

## Un pequeño ejemplo

Supongamos que tenemos la siguiente lista de teléfonos de una empresa:

```
Phone Name  ID
     ...
     ...
3412    Bob 123
3834  Jonny 333
1248   Kate 634
1423   Tony 567
2567  Peter 435
3567  Alice 535
1548  Kerry 534
     ...
```

Se trata de una empresa con 500 personas y los datos están almacenados en un fichero ASCII normal. Los registros de personas cuyo teléfono comience con un 1, trabajan en el edificio 1. ¿Quién trabaja en el edificio 1?

Una Expresión Regular puede responder a eso:

```
grep '^1' phonelist.txt
o
egrep '^1' phonelist.txt
o
perl -ne 'print if (/^1/)' phonelist.txt
```

En palabras normales, esto significa: Busca todas las líneas que comiencen con un 1. El símbolo "^" es el encargado de indicar que sólo se busquen los números 1 que se encuentren al principio de la línea.

## Reglas de sintaxis

### Patrón de un solo símbolo

El elemento básico de una expresión regular es el patrón de un solo símbolo. Éste patrón sólo es efectivo cuando este símbolo se puede encontrar exactamente en el texto. Un ejemplo lo podemos encontrar en el número 1 del ejemplo de arriba.

Otro ejemplo para el patrón de un solo símbolo es:

```
egrep 'Kerry' phonelist.txt
```

Este patrón se compone de patrones de un solo símbolo (la letra K, e, etc.)

Varios signos se pueden agrupar en un conjunto. Un conjunto se representa por un par de corchetes (el de abrir y el de cerrar) y una lista de caracteres entre ellos. Un conjunto se considera también como un patrón de un solo símbolo. La búsqueda de este conjunto es efectiva cuando se encuentre uno y solo uno de los signos del conjunto en el texto. Un ejemplo:

**[abc]** Un patrón de un solo símbolo con el que se puede encontrar el símbolo a, b o c.
**[ab0-9]** Un patrón de un solo símbolo en el que se busca una a o una b o un número que se encuentre entre el 0 y el 9 en el código ASCII.
**[a-zA-Z0-9\-]** Esto busca una letra mayúscula o minúscula o un número o el signo -.
En la lista de teléfonos:

```
egrep '^1[348]' phonelist.txt
```

Esta expresión busca líneas, que comiencen con 13, 14 o 18.

La mayoría de símbolos ASCII nos llevan a una búsqueda exitosa cuando se encuentran en el texto, pero también encontramos símbolos en una expresión regular, que tienen un significado especial. Los corchetes hacia la derecha indican el comienzo de la definición de un conjunto. El símbolo "`-`" en un conjunto es un símbolo especial y representa un rango en el sistema de símbolos ASCII. Para indicar que nos referimos al significado normal del símbolo, se coloca una barra invertida delante "`\`". Por ejemplo, en `[a-zA-Z0-9\-]`. En algunos dialectos encontramos que la barra invertida junto con otro símbolo tienen un significado especial. En este caso se obtiene el significado normal retirando la barra invertida.

El punto también es un símbolo importante un una expresión regular. La búsqueda será efectiva, cuando el símbolo comparado sea cualquier símbolo menos el símbolo de nueva línea del código ASCII. Ejemplo:

```
grep '^.2' phonelist.txt
```

Esta expresión busca líneas que contengan el número 2 en la segunda posición, y cualquier otro símbolo delante.

Los conjuntos se pueden invertir mediante la colocación de "`[^`" en lugar de "`[`" en la expresión. En este caso el símbolo "`^`" no representa el comienzo de la línea, sino que "`[`" y "`^`" juntos representan el inverso del conjunto.

**[0-9]** Un patrón de un solo símbolo, la búsqueda será efectiva, si el símbolo en el texto es un número.
**[^0-9]** Todo lo que no sea un número.

**[^abc]** Todo lo que no sea una a, b o c. · Todo menos el símbolo de línea nueva. Es idéntico a [^\n]. \n es el símbolo para nueva línea.

Para buscar todas las líneas que NO comiencen con un 1, se escribiría:

```
egrep '^[^1]' phonelist.txt
```

## Anclas (anchors)

Antes hemos visto que "^" representaba el inicio de una línea. Las anclas son símbolos especiales en las expresiones regulares, que no denotan un símbolo, sino una posición.

```
^   Inicio de una línea
$   Final de una línea
```

Para encontrar personas en nuestra lista con un identificativo (company-ID) de 567, utilizaremos la expresión:

```
egrep '567$' phonelist.txt
```

Significa: Busca líneas que finalicen con 567.

## Multiplicadores

Los multiplicadores nos indican, cuantas veces ha de aparecer un patrón de un solo símbolo en el texto:

| Descripción | grep | egrep | perl | vi | vim | vile | elvis | emacs |
|---|---|---|---|---|---|---|---|---|
| cero o más veces | * | * | * | * | * | * | * | * |
| una o más veces | \{1,\} | + | + | | \+ | \+ | \+ | + |
| cero o una vez | \? | ? | ? | | \= | \? | \= | ? |
| de n hasta m veces | \{n,m\} | | {n,m} | | | | \{n,m\} | \{n,m\} |

Nota: Para los distintos VI tendría que utilizarse la opción magic.

Un ejemplo de la lista de teléfonos:

```
....
1248   Kate 634
....
1548  Kerry 534
....
```

Para encontrar una línea que se componga de un 1, varios números, varios

espacios y la letra K se escribirá:

```
grep '^1[0-9]\{1,\} \{1,\}K' phonelist.txt
```

o se utiliza `*` y se repite `[0-9]` y el espacio:

```
grep '^1[0-9][0-9]*  *K' phonelist.txt
```

o

```
egrep '^1[0-9]+ +K' phonelist.txt
```

o

```
perl -ne 'print if (/^1[0-9]+ +K/)' phonelist.txt
```

Los multiplicadores nos indican, cuantas veces tiene que aparecer el patrón de un solo símbolo que les precede. "`23*4`" NO significa "2, 3, quizás algo y después un 4" (¡esto sería "`23.*4`"!), sino que significa, que se está buscando "un 2, quizás algunos treses y un 4".

Es importante saber que un multiplicador es ávido. Esto significa que el primer multiplicador se extenderá lo máximo que pueda hacia la derecha.

Esto no es muy importante en grep, pero es importante en el momento de editar textos.

La expresión `^1.*4` se extendería por toda la línea 1548 Kerry 534 de principio a fin.

La expresión no se extiende sólo hasta la zona corta, o sea, 154, sino hasta el máximo que se pueda.

### Paréntesis como memoria

Los paréntesis como memoria no influyen en el tipo o símbolos por los que se buscará. Sirven para que los fragmentos de texto se almacenen y pueda accederse a ellos a través de variables.

El primer paréntesis será referenciado como variable 1, el segundo como variable 2, etc.

| Nombre del programa | Sintaxis del peréntesis | Sintaxis de las variables |
|---|---|---|
| grep | \(\) | \1 |
| egrep | () | \1 |
| perl | () | \1 or ${1} |
| vi,vim,vile,elvis | \(\) | \1 |

| | | |
|---|---|---|
| emacs | \(\) | \1 |

Un ejemplo: La expresión `[a-z] [a-z]` busca dos letras minúsculas.

Ahora se pueden utilizar estas variables, para buscar patrones de texto como ?otto?:

```
egrep '([a-z])([a-z])\2\1'
```

La variable `\1` contiene la letra `o` y `\2` la letra `t`. La expresión serviría también para `anna`, pero no para `yxyx`.

Los paréntesis no se utilizan normalmente para la búsqueda de nombres como otto y anna, sino que para editar o para buscar y sustituir.

## Uso de Expresiones Regulares para el tratamiento de Textos

Para editar textos es necesario un editor de textos como vi o emacs o perl. Todos estos programas soportan expresiones regulares.

En emacs se utiliza `M-x query-replace-regexp` o `replace-regexp`. `M-x query-replace-regexp` pide confirmación en cada búsqueda y sustitución. Lo más recomendable es asignar a una tecla de función el `query-replace` o `replace-regexp`.

En vi se utiliza `:%s/ / /gc`. El símbolo `%` representa la expresión "todo el fichero" por lo que se puede reemplazar por cualquier otra expresión. En el editor vim, por ejemplo, se puede marcar una región con mayúscula-v y cursor arriba/abajo. Desgraciadamente el artículo se saldría de sus propósitos si diésemos una pequeña introducción sobre vim. `///gc` es interactivo y pregunta por una confirmación. `s///g` no es interactivo. En perl utilizamos:

```
perl -pe 's/ / /g'
```

Ahora unos cuantos ejemplos. En la empresa tienen que modificarse algunos teléfonos. Todos los teléfonos que comiencen con un 1, han de tener un 2 después de la segunda cifra. De 1423 pasaremos a 14223.

```
Phone Name  ID
    ...
3412    Bob 123
3834  Jonny 333
1248   Kate 634
1423   Tony 567
2567  Peter 435
3567  Alice 535
1548  Kerry 534
     ...
```

y esto es así de fácil:

```
vi:    s/^\(1.\)/\12/g
emacs: ^\(1.\)   sustituir por  \12
perl:  perl -pe 's/^(1.)/${1}2/g' phonelist.txt
```

La lista de teléfonos aparecerá así:

```
Phone Name  ID
   ...
3412    Bob 123
3834  Jonny 333
12248  Kate 634
14223  Tony 567
2567  Peter 435
3567  Alice 535
15248 Kerry 534
     ...
```

Perl no sólo conoce las variables `\1` `\9`. `\12` apuntaría a la variable 12. Para solucionar el problema se utiliza simplemente `${1}`.

Pero ahora ya no se encuentran las columnas bien alineadas. ¿Cómo se podría solucionar esto? Se podría comprobar si hay un espacio en la quinta posición e insertar otro.

```
vi:    s/^\(....\) /\1  /g
emacs: '^\(....\) '   sustituir por   '\1  '
perl:  perl -pe 's/^(....) /${1}  /g' phonelist.txt
```

Ahora tenemos:

```
Phone Name  ID
    ...
3412     Bob 123
3834   Jonny 333
12248   Kate 634
14223   Tony 567
2567   Peter 435
3567   Alice 535
15248  Karry 534
    ...
```

Un colega ha editado la phonelist.txt y no ha tenido cuidado. Al principio de alguna línea encontramos espacios. ¿Cómo se puede solucionar?

```
Phone Name ID
    ...
3412     Bob 123
3834   Jonny 333
12248   Kate 634
14223   Tony 567
2567   Peter 435
```

```
3567   Alice 535
15248  Kerry 534

    ...
```

Esto tendría que solucionar el problema:

```
vi:    s/^  *// (dos espacios, pues no tenemos el +)
emacs: '^ +'  sustituir por la cadena vacía
perl:  perl -pe 's/^ +//' phonelist.txt
```

Estás escribiendo un programa y utilizas las variables `temp` y `temporary`. Ahora quieres sustituir la variable `temp` por la variable `counter`. Si utilizases la opción de buscar y reemplazar, verías que la variable `temporary` se reemplazaría por `counterorary`. Esto lo quieres evitar.

Las expresiones regulares lo hacen muy fácil. Se sustituye `temp([^o])` con counter\1. Con esto se sustituye `temp` y no `o`. (Otra alternativa serían las Boundaries, que no se comentaron aquí.)

Espero que este artículo te haya despertado la curiosidad. Ahora tendrías que mirar en las man-pages de tú editor preferido.

Existen más símbolos especiales en las expresiones regulares, como por ejemplo, Alterations, Art Order, y naturalmente las Boundaries.

Que disfrutéis

---

*Traducido por Franz Jimeno*