Expresiones regulares

Las expresiones regulares (o regex) nos permiten definir un patròn de búsqueda con el que analizar un conjunto de datos. Estos patrones se componen de 2 tipos de carácteres:

- literales: caracteres que representan el propio carácter en si mismo.
- operandos: caracteres especiales con un significado y una función dentro del patrón especificado

File Globbing VS Regular Expressions

Aunque se puedan parecer en ocasiones, es importante entender la diferencia que existe entre el file globbing y las expresiones regulares. En el primer caso estamos hablando de patrones de ficheros. Es decir, estamos analizando ficheros y directorios que se encuentran en nuestro sistema de ficheros, y los resultados obtenidos harán referencia a esta clase de objetos del sistema. En el caso de las expresiones regulares hablamos de filtros que aplicamos a flujos de datos, por lo que analizan cadenas de texto y nos devuelven coincidencias producidas en ellas.

Tipos de expresiones regulares

Las expresiones regulares más sencillas son aquellas que sólo incluyen literales en su patrón de búsqueda (ver ejemplo 1). Si queremos añadir complejidad (y potencia) a nuestras expresiones podemos hacer uso de los diferentes tipos que existen, que se clasifican en 2 variantes:

- Básicas
- Extendidas

Cuando queremos usar caracteres especiales con su significado literal es necesario utilizar comillas para escapar su función de la ejecución del comando.

Expresiones regulares básicas

Las expresiones regulares básicas contienen una serie de que nos permiten afinar nuestra búsqueda en un flujo de datos.

^

Este operando nos permite indicar el carácter no imprimible de inicio de línea. Por lo tanto, si lo añadimos a nuestro patrón de búsqueda podemos limitarla a líneas que empiecen con ese patrón de coincidencia (ver ejemplo 2)

$

Este operando nos permite indicar el carácter no imprimible de fin de línea. Por lo tanto, si lo añadimos a nuestro patrón de búsqueda podemos limitarla a líneas que acaben con ese patrón de coincidencia ver ejemplo 3

.

Este operando nos permite indicar que estamos buscando cualquier carácter en esa posición del patrón de búsqueda. Se puede utilizar una o varias veces, de manera consecutiva o alterna en el patrón ver ejemplo 4

[]

Este operando nos permite indicar un rango de valores que van a ser utilizados para realizar varias búsquedas, una por cada opción posible dentro del rango especificado. Así, si tenemos un rango de 3 valores, se van a utilizar 3 patrones de búsqueda ver ejemplo 5 Es importante saber que cuando utilizamos un operando dentro de [] pierde su funcionalidad, y actúa como un literal más.

*

Este operando nos permite indicar la aparición desde 0 a N veces del patrón anterior a él. Es por esto que siempre debe ir precedido de un carácter (literal u operando) sobre el que aplicar el criterio de búsqueda ver ejemplo 6

Si queremos usar alguno de los operandos de forma literal hay que escapar el carácter con una \ delante de él.

Expresiones regulares extendidas

Las expresiones regulares extendidas no siempre son reconocibles por todos los comandos, incluso algunos que sí pueden operar con ellas necesitan saberlo a través de alguna de sus opciones. También es importante delimitar (con `\'\' - comillas simples`)Dentro de las expresiones regulares extendidas podemos encontrar las siguientes:

+

El símbolo + nos indica que el patrón de búsqueda debe localizar una o varias coincidencias del literal (o subpatrón) que le precede. Es importante entender la diferencia con *m ya que es este caso el literal (o subpatrón) debe aparecer como mínimo una vez (ver ejemplo 7)

?

El símbolo ? nos permite indicar que el literal (o subpatrón) que le precede puede aparecer 1 o ninguna vez en la búsqueda. Dicho con otras palabras, es indicar que un literal del patrón es "opcional" (ver ejemplo 8)

{}

Las llaves {} nos permiten poder indicar dentro de nuestro patrón de búsqueda cuantas veces dede repetirse el literal (o subpatrón) que le precede (ver ejemplo 9) Con este tipo de expresión podemos realizar patrones equivalentes a algunos de los que ya hemos visto

Expresión regular	Equivalencia	Significado
'a{0,N}'	'a*'	De cero a N coincidencias de "a" (N es opcional)
'a{1,N}'	'a+'	De una a N coincidencias de "a" (N es opcional)
'a{0,1}'	'a?'	Cero o una coincidencias de "a"

|

Este símbolo nos permite poder utilizar más de un patrón de búsqueda en una misma expresión regular. Útil cuando queremos buscar 2 patrones sobre el mismo flujo de datos (ver ejemplo 10)

()

Los paréntesis nos permiten crear agrupaciones de patrones. Es una técnica útil cuando queremos crear "subpatrones" dentro de nuestro patrón de búsqueda ver ejemplo 11|

Ejemplos

Ejemplo 1

Busca dentro del fichero passwd la palabra bash

```
$ grep bash passwd
bash:x:102:106::/home/bash:/bin/bash_shell
dash:x:104:110::/home/bash:/bin/false
super:x:1001:1001::/home/super:/bin/bash
```

Ejemplo 2

Busca dentro del fichero passwd la palabra bash al inicio de línea

```
$ grep ^bash passwd
bash:x:102:106::/home/bash:/bin/bash_shell
```

Ejemplo 3

Busca dentro del fichero passwd la palabra bash al final de línea

```
$ grep bash$ passwd
super:x:1001:1001::/home/super:/bin/bash
```

Ejemplo 4

Busca dentro del fichero passwd usuarios que no sean de sistema (es decir, con UID por encima de 1000)

```
$ grep :....: passwd
super:x:1001:1001::/home/super:/bin/bash
test:x:1002:1002::/home/test:/bin/bash
```

Ejemplo 5

Busca dentro del fichero passwd líneas que contengan los números 3, 4 ó 5.

```
$ grep [3-5] passwd
dash:x:104:110::/home/bash:/bin/false
proves:x:1003:1003:/usr/share:/bin/sh
```

Ejemplo 6

Busca los usuarios que tienen su directorio personal dentro de /home

```
$ grep home.* passwd
bash:x:102:106::/home/bash:/bin/bash_shell
dash:x:104:110::/home/bash:/bin/false
super:x:1001:1001::/home/super:/bin/bash
test:x:1002:1002::/home/test:/bin/sh
```

Ejemplo 7

Localiza los usuarios del fichero `users` que tienen contraseña

```
$ cat users
user;password
user1;password-01
user2;
user3;password-03
user4;password-04
user5;
usuario6;
$ grep -E ';.+' users
user;password
user1;password-01
user3;password-03
user4;password-04
```

Ejemplo 8

Localiza todos los usuarios que contengan en el login `user`, ya sea con un secuencial númerico o no

```
$ grep -E 'user[0-9]?' users
user;password
user1;password-01
user2;
user3;password-03
user4;password-04
user5;
```

——————————————————

Ejemplo 9

Localiza páginas web que contengan “www” en el fichero `urls`

```
$  cat urls
cursosonline.idfo.org
www.idfo.org
labs.pue.es
www.pue.es
www.lpi.org
wiki.lpi.org
learning.lpi.org
$ grep -E w{3} urls
www.idfo.org
```

```
www.pue.es
www.lpi.org
```

Ejemplo 10

Busca a los usuarios test y super en el fichero `passwd`

```
$ grep -E 'super|test' passwd
super:x:1001:1001::/home/super:/bin/bash
test:x:1002:1002::/home/test:/bin/sh
```

Ejemplo 11

Encontrar las direcciones IP del fichero `access.log` que han visitado tu página web:

```
$ grep -o -E '^([0-9]{1,3}\.){3}[0-9]{1,3}' /var/log/apache2/access.log
114.34.231.197
115.124.64.146
13.66.139.110
13.66.139.162
...
```
