Capítulo 10

Expresiones regulares

?	*El elemento precedente es opcional y debe coincidir al menos una vez.*
*****	*El elemento precedente debe coincidir cero o más veces.*
{n}	*El elemento precedente debe coincidir exactamente **n** veces.*
+	*El elemento precedente debe coincidir una o más veces.*
{,m}	*El elemento precedente es opcional y debe coincidir al menos **m** veces.*
{n,m}	*El elemento precedente debe coincidir al menos **n** veces pero no más de **m** veces.*
pablo	*La cadena pablo.*
^pablo	*La cadena pablo al principio de la linea.*
pablo$	*La cadena pablo al final de la linea.*
^pablo$	*La cadena pablo formando una única línea.*
niñ[oa]	*La cadena niño o niña.*
ni[^aeiou]o	*La tercera letra no es una vocal minúscula.*
ga.o	*La tercera letra es cualquier carácter.*
^....$	*Cualquier línea que contenga 4 caracteres.*
^\.	*Cualquier línea que comienza por punto.*
^[^.]	*Cualquier línea que no comienza por punto.*
ñños*	*niño, niños, niñoss, niñosss, etc*
“niño”	*niño entre comillas dobles.*
“*niño”*	*niño con o sin comillas dobles.*
[a-z][a-z]*	*una o más letras minúsculas.*
[a-z]+	*una o más letras minúsculas (sólo válido en algunas aplicaciones).*
[^0-9A-Z]	*cualquier carácter que no sea ni número ni letra mayúscula.*
[a-zA-Z]	*cualquier letra sea mayúscula o minúscula.*
[Ax5]	*cualquier carácter que sea A, x o 5.*
niño|niña|nada	*una de las tres palabras.*
(s|arb)usto	*la palabra susto o arbusto.*
ga?t[oa]	*gato, gata, gasto, gaita, etc.*
\<ga	*cualquier palabra que empiece por ga.*
ño\>	*cualquier palabra que termine por ño*
\<niño\>	*la palabra niño*
o\{2,\}	*dos o más oes en una misma fila.*
\t	*Representa un tabulador.*
\r	*Representa el "retorno de carro" o "regreso al inicio" o sea el lugar en que la línea vuelve a iniciar.*
\n	*Representa la "nueva línea" el carácter por medio del cual una línea da inicio.*
\a	*Representa una "campana" o "beep" que se produce al imprimir este carácter.*
\e	*Representa la tecla "Esc" o "Escape"*
\f	*Representa un salto de página.*
\v	*Representa un tabulador vertical.*
\x	*Se utiliza para representar caracteres ASCII o ANSI si conoce su código. De esta forma, si se busca el símbolo de derechos de autor*

	y la fuente en la que se busca utiliza el conjunto de caracteres Latin-1 es posible encontrarlo utilizando "\xA9".
\u	*Se utiliza para representar caracteres Unicode si se conoce su código. "\u00A2" representa el símbolo de centavos. No todos los motores de Expresiones Regulares soportan Unicode. El .Net Framework lo hace, pero el EditPad Pro no, por ejemplo.*
\d	*Representa un dígito del 0 al 9.*
\w	*Representa cualquier carácter alfanumérico.*
\s	*Representa un espacio en blanco.*
\D	*Representa cualquier carácter que no sea un dígito del 0 al 9.*
\W	*Representa cualquier carácter no alfanumérico.*
\S	*Representa cualquier carácter que no sea un espacio en blanco.*
\A	*Representa el inicio de la cadena. No un carácter sino una posición.*
\Z	*Representa el final de la cadena. No un carácter sino una posición.*
\b	*Marca el inicio y el final de una palabra.*
\B	*Marca la posición entre dos caracteres alfanuméricos o dos no-alfanuméricos.*

Buscar una expresion regular

El comando **grep** toma una expresión regular de la línea de comandos, lee la entrada estándar o una lista de archivos, e imprime las líneas que contengan coincidencias para la expresión regular.

Opciones :

-*c*	*Modificar la salida normal del programa, en lugar de imprimir por salida estándar las líneas coincidentes, imprime la cantidad de líneas que coincidieron en cada archivo.*
-e PATRÓN	*Usar `PATRÓN` como el patrón de búsqueda, muy útil para proteger aquellos patrones de búsqueda que comienzan con el signo «-».*
-f ARCHIVO	*Obtener los patrones del archivo `ARCHIVO`*
-H	*Imprimir el nombre del archivo con cada coincidencia.*
-r	*Buscar recursivamente dentro de todos los subdirectorios del directorio actual.*
-i	*Buscar mayúsculas o minúsculas.*
-w	*Buscar palabra completa.*
-v	*Buscar lo contrario.*
-l	*Mostrar solo los archivos que contenga la palabra buscada.*
-c	*Contar la cantidad de ocurrencia.*
-E	*Expresiones regulares extendidas en vez de usar **grep** podemos usar **egrep** que es lo mismo, la diferencia es que **egrep** son expresiones regulares extendidas y no tenemos que pasar la opción -**E** como en **grep**.*
-o	*Muestra solamente lo que queremos buscar en la linea.*

Ejemplo :

```
# grep 'root' /etc/passwd
# grep -i 'root' /etc/passwd
# grep -iw 'root' /etc/passwd
```

```
# grep -v 'root' /etc/passwd
# grep -l 'root' /etc/* | more
# grep -c 'root' /etc/* | more
```

Si queremos que tenga color la búsqueda realizamos la siguiente exportación :

```
# export GREP_OPTIONS='--color=auto' GREP_COLOR='100;8'
```

Modificar archivos, buscar, etc

El comando **sed** nos permite, de una forma cómoda, borrar líneas, registros o sustituir cadenas de caracteres dentro de las líneas, etc.

Opciones :

-*n*, --*quiet*, --*silent*	*Suprime la muestra automática del espacio de patrones.*
-e *guión*, --expresion=*guión*	*Agrega el guión a la lista de órdenes para ejecutar.*
-f *fichero-guión*, --file=*fichero-guión*	*Agrega el contenido del fichero guión a la lista de órdenes para ejecutar.*
--follow-symlinks	*Selecciona solamente una columna para mostrar.*
-i*[SUFIJO]*, --in-place*[=SUFIJO]*	*Edita ficheros en el lugar (crea un respaldo si se da una extensión).*
-l N, --line-length=*N*	*Especifica la longitud de corte de línea deseado para la orden `l'.*
--posix	*Desactiva todas las extensiones de GNU.*
-r, --regexp-extended	*Utiliza expresiones regulares extendidas en el guión.*
-s, --separate	*Considera los ficheros como separados en lugar de un solo flujo, largo y continuo.*
-u, --unbuffered	*Carga cantidades mínimas de datos de los ficheros de entrada y vacía los almacenamientos temporales de salida con más frecuencia.*
--help	*Ayuda.*
--version	*Version.*

Comandos :

d	*Borrar linea.*
i	*Insertar.*
s	*Sustituir.*
p	*Listar.*
\+	*Mas de una expresión.*
N	*Fuerza la lectura de la siguiente línea en la iteración actual.*
y	*Sustituir.*

Ejemplo :

```
# cp /etc/passwd .
# sed '1d' passwd | more
# sed '1,5d' passwd | more
```

```
# sed '1i Hola Mundo' passwd | more
# sed 's/root/pepe/' passwd | more
# sed -n '1,4p' passwd
# sed -n '/root/p' passwd

o

# grep 'root' passwd
```

Eliminar lineas en blanco

```
# sed '/^$/d' passwd
```

Ejemplo de obtener los mail de un archivo :

Explicaremos la expresión regular : '**\b[A-Z0-9._%-]+@[A-Z0-9.-]+\.[A-Z]{2,4}**' :

\b[A-Z0-9._%-]+	Buscamos un comienzo de palabra **\b** que contenga uno o más caracteres + dentro del conjunto **A a Z**, **0 a 9**,., **_**, **%** y -, que son los caracteres admitidos para un nombre de usuario.
@	Seguido del símbolo @.
[A-Z0-9.-]+	Seguido por uno o más + caracteres **A a Z**, **0 a 9**,. y -, que son los caracteres admitidos para un dominio.
\.	Seguido por un punto escapado con \, ya que el punto solo significa cualquier carácter.
[A-Z]{2,4}\b	Seguido por entre 2 y 4 caracteres **{2,4}** entre el conjunto **A-Z** y además éste tiene que ser el final de una palabra **\b.**

```
# echo “El mail de juan es juan@hotmail.com es programador” > mail.txt
# echo “El mail de pedro es pedro.al@hotmail.com ar es webmaster” >> mail.txt

# egrep -oi '\b[A-Z0-9._%-]+@[A-Z0-9.-]+\.[A-Z]{2,4}' mail.txt

        juan@hotmail.com
        pedro.al@hotmail.com
```

Obtener las ip de un archivo

```
# echo “ip del servidor 192.168.2.10” > mis-ip.txt
# echo “ip del router 10.1.1.1” >> mis-ip.txt
# echo “ip ana 192.168.2.100” >> mis-ip.txt
# echo “ip miguel 192.168.2.101” >> mis-ip.txt
# echo “ip pedro 192.168.2.102” >> mis-ip.txt

# egrep -oi '\b[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\b' mis-ip.txt

        192.168.2.10
        10.1.1.1
```

```
        192.168.2.100
        192.168.2.101
        192.168.2.102

o

# grep -oE '\b[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\b' mis-ip.txt
```

Listar los usuarios con password a partir del **/etc/shadow**

Resulta que queremos saber los usuarios del fichero ***/etc/shadow*** que pueden logearse al sistema (que no tienen un carácter ! O * en el campo password). La expresión regular a usar sería:

^[^:]+	Buscamos uno o más caracteres + que no sean : **[^:]** a principio de línea ^
:	Seguido de un carácter **:**
[^*!]+	Seguido por una secuencia de uno o más caracteres que no sean ! O *
:	Seguido de otro carácter **:**

```
# egrep '^[^:]+:[^*!]+:' /etc/shadow

      fulano:$1$PFC1F2pj$0KEflTjNmDQg9.rtFT/DK0:12793:0:99999:7:::
      mengano:$1$aAUa4js/$c2LEh.HeT0JvCG3B.34O.:12793:0:99999:7:::
```

Borra desde **root** hasta **nobody** inclusive.

```
# sed '/root/,/nobody/d' passwd
```

Si queremos obtener un listado sólo de los directorios (las filas que comienzan con la letra "d"). Podríamos utilizar el sed para borrar las entradas que no comienzan con esa letra:

```
# ls -l / | sed '/^d/!d'
```

Como vemos el símbolo de exclamación “**!”** detrás de la expresión regular niega la condición. Debería leerse como borrar todas las líneas que no comiencen con la letra '**d**'

Otra forma :

```
# ls -l / | sed '/^d/p'
```

Sustituir los valores 123 por 456:

```
# echo “123” | sed 'y/[123]/[456]/'

  456
```

Sacar los tabuladores de un archivo y reemplazarlo por un espacio :

```
# vi texto.txt
   Hola      Mundo  !     !!!!
```

```
# cat texto.txt | sed 's/\t\+/ /g'
```

Sacar los mas de un espacio de un archivo y reemplazarlo por un espacio :

```
# vi texto.txt
  Hola  Mundo  !     !!!!

# cat texto.txt | sed 's/ \+/ /g'
```

Procesando varias lineas

```
# vi agenda.txt

        Registro: 1
        Nombre: Anibal
        Telefono: 621-229

        Registro: 2
        Nombre: Hector
        Telefono: 562-245

        Registro: 3
        Nombre: Pablo
        Telefono: 622-354
```

Y queremos obtener lo siguiente :

```
1;Anibal;621-229
2;Hector;562-245
3;Pablo;622-354
```

Una solución es poder leer de a una línea a la vez. Este problema se resuelve con el comando **“N”**, que fuerza la lectura de la siguiente línea en la iteración actual.

```
# sed 'N;N;N;s/\n/;/g;s/^Registro: \(.*\);Nombre: \(.*\);Telefono: \(.*\);$/\1;\2;\3/'
agenda.txt
```

Son 3 líneas (N).

Que reemplace el retorno de carro **“\n”** por un **“;”**

```
buffer = "Registro: 1;Nombre: Anibal;Telefono: 621-229;"
```

Sacar las palabras **“Registro: ”**, **”Nombre: ”**, **“Telefono: ”** y el punto y coma de la linea.

Los paréntesis **\(.*\)** representa a cada valor **\1**, **\2** y **\3**.