Tutorial de expresiones regulares

Prof. Enrique García Salcines

¿Qué es una expresión regular?

Una expresión regular es una serie de caracteres especiales que permiten describir un texto que queremos buscar. Por ejemplo, si quisiéramos buscar la palabra "linux" bastaría poner esa palabra en el programa que estemos usando. La propia palabra es una expresión regular. Hasta aquí parece muy simple, pero, ¿y si queremos buscar todos los números que hay en un determinado fichero? ¿O todas las lineas que empiezan por una letra mayúscula? En esos casos ya no se puede poner una simple palabra. La solución es usar una expresión regular.

Las expresiones regulares se utilizan en:

- En entornos UNIX, con comandos como grep, sed, awk...
- De manera intensiva, en lenguajes de programación como perl, python, ruby, php, .net
- En bases de datos.
- Ahorran mucho tiempo y el código es más robusto.

Expresiones regulares vs patrones de ficheros.

Antes de empezar a entrar en materia sobre las expresiones regulares, quiero aclarar un malentendido común sobre las expresiones regulares. Una expresión regular no es lo que ponemos como parámetro en los comandos como rm, cp, etc para hacer referencia a varios ficheros que hay en el disco duro. Eso sería un patrón de ficheros. Las expresiones regulares, aunque se parecen en que usan algunos caracteres comunes, son diferentes. Un patrón de fichero se lanza contra los ficheros que hay en el disco duro y devuelve los que encajan completamente con el patrón, mientras que una expresión regular se lanza contra un texto y devuelve las lineas que contienen el texto buscado. Por ejemplo, la expresión regular correspondiente al patrón *.* seria algo así como ^.*\..*$

Tipos de expresiones regulares.

No todos los programas utilizan las mismas expresiones regulares. Ni mucho menos. Existen varios tipos de expresiones regulares más o menos estándar, pero hay programas que cambian ligeramente la sintaxis, que incluyen sus propias extensiones o incluso que utilizan unos caracteres completamente diferentes. Por eso, cuando queráis usar expresiones regulares con algún programa que no conozcáis bien, lo primero es mirar el manual o la documentación del programa para ver cómo son las expresiones regulares que reconoce.

En primer lugar, existen dos tipos principales de expresiones regulares, que están recogidas en el estándar POSIX, que es el que usan las herramientas de Linux. Son las expresiones regulares básicas y las extendidas. Muchos de los comandos que trabajan con expresiones regulares, como grep o sed, permiten usar estos dos tipos. También están las expresiones regulares estilo PERL, y luego hay programas como vim o emacs que usan variantes de estas. Según lo que queramos hacer puede ser más adecuado usar unas u otras.

Ejemplos de expresiones regulares

- La expresión regular más simple sería la que busca una secuencia fija de caracteres literales.
- La cadena cumple la expresión regular si contiene esa secuencia.

o	l	a	`Ella me dijo hola.` ⇒ Empareja. `Ella me dijo mola.` ⇒ Empareja. `Ella me dijo adiós.` ⇒ No empareja.

- Puede que la expresión regular empareje a la cadena en más de un punto:

o	l	a	`Lola me dijo hola.` ⇒ Empareja 2 veces.

¿Por qué las necesito?

¿Para qué necesito aprender a utilizar las *regex*?:

- Direcciones de calles.
- Quiero actualizar su formato, de "100 NORTH MAIN ROAD" a "100 NORTH MAIN RD.", sobre un conjunto de muchas carreteras.

```
mdorado@mdoradoLaptop:~$ echo "100 NORTH MAIN ROAD" | sed -e 's/ROAD/RD\./'
100 NORTH MAIN RD.
mdorado@mdoradoLaptop:~$ cat carreteras.txt
100 NORTH MAIN ROAD
45 ST JAMES ROAD
100 NORTH BROAD ROAD
mdorado@mdoradoLaptop:~$ cat carreteras.txt | sed -e 's/ROAD/RD\./'
100 NORTH MAIN RD.
45 ST JAMES RD.
100 NORTH BRD. ROAD
```

¿Por qué las necesito?

- ¿Para qué necesito aprender a utilizar las *regex*?
  - A veces necesito hacer operaciones con cadenas con expresiones relativamente complejas.
  - P.Ej.: reemplazar "ROAD" por "RD." siempre que esté al final de la línea (carácter especial `$`).

```
mdorado@mdoradoLaptop:~$ cat carreteras.txt | sed -e 's/ROAD$/RD\./'
100 NORTH MAIN RD.
45 ST JAMES RD.
100 NORTH BROAD RD.
```

Caracteres especiales

- Las expresiones regulares se componen de caracteres normales (literales) y de caracteres especiales (o metacaracteres).
- "`[...]`": sirve para indicar una lista caracteres posibles:

b	[iur]	e	`Octubre me dijo bueno bien.` ⇒ Empareja 3 veces.

"`[^...]`": sirve para *negar* la ocurrencia de uno o más caracteres:

b	`[^ur]`	*e*	`Octubre me dijo bueno bien.` ⇒ Empareja 1 vez.

Caracteres especiales

- "`^`": empareja con el principio de una línea:

`^ O` `Octubre me dijo bueno` ⇒ Empareja 1 vez.

- "`$`": empareja con el final de una línea:

`e $` `Bueno, me dijo octubre` ⇒ Empareja 1 vez.

Caracteres especiales

- "`*`": empareja con cero, una o más ocurrencias del carácter anterior:

o	*l*	*a*	*	*s*	`Holaaaaaaas` ⇒ Empareja 1 vez. `Hols` ⇒ Empareja 1 vez.

- En caso de duda, el emparejamiento siempre es el de mayor longitud:

a	.	*	*e*	`Olas emocionantes`.

Caracteres especiales

- Los paréntesis `()` (o `\( \)`) permiten agrupar caracteres a la hora de aplicar los meta-caracteres:
  - `a*` empareja con `cadena vacía`, `a`, `aa`, `aaa`...
  - `abc*` empareja con `ab`, `abc`, `abcc`, `abccc`...
  - `(abc)*` empareja con `abc`, `abcabc`, `abcabcabc`...

Dos tipos de expresiones regulares:

- *Basic Regular Expressions* (BRE): propuesta inicial en el estándar POSIX.
- *Extended Regular Expressions* (ERE): ampliación con nuevos metacaracteres.

- Cada aplicación utiliza una u otra.

Resumen Caracteres especiales

Carácter	BRE	ERE	Significado
\	✓	✓	Interpreta de forma literal el siguiente carácter
.	✓	✓	Selecciona *un* carácter cualquiera
*	✓	✓	Selecciona *ninguna, una o varias* veces lo anterior
^	✓	✓	Principio de línea
$	✓	✓	Final de línea
[...]	✓	✓	Cualquiera de los caracteres que hay entre corchetes
\n	✓	✓	Utilizar la n-ésima selección almacenada
{n,m}	X	✓	Selecciona lo anterior entre n y m veces
+	X	✓	Selecciona *una o varias* veces lo anterior
?	X	✓	Selecciona *una o ninguna* vez lo anterior
|	X	✓	Selecciona lo anterior o lo posterior
(...)	X	✓	Selecciona la secuencia que hay entre paréntesis
\{n,m\}	✓	X	Selecciona lo anterior entre n y m veces
\(...\)	✓	X	Selecciona la secuencia que hay entre paréntesis
\\|	✓	X	Selecciona lo anterior o lo posterior

Rangos de caracteres

- `[aeiou]`: empareja con las letras `a`, `e`,`i`,`o` y `u`.
- `[1-9]` es equivalente a `[123456789]`.
- `[a-e]` es equivalente a `[abcde]`.
- `[1-9a-e]` es equivalente a `[123456789abcde]`.
- Los rangos típicos se pueden especificar de la siguiente forma:
  - `[[:alpha:]]`→ `[a-zA-Z]` alfabéticos
  - `[[:alnum:]]`→ `[a-zA-Z0-9]` alfanuméricos
  - `[[:lower:]]`→ `[a-z]` minúsculas
  - `[[:upper:]]`→ `[A-Z]` mayúsculas
  - `[R[:lower:]]`→ `[Ra-z]`.

Otros ejemplos

Expresión Regular	Equivalencia
a.b	axb aab abb aSb a#b ...
a..b	axxb aaab abbb a4$b ...
[abc]	a b c (cadenas de un caracter)
[aA]	a A (cadenas de un caracter)
[aA][bB]	ab Ab aB AB (cadenas de dos caracteres)
[0123456789]	0 1 2 3 4 5 6 7 8 9
[0-9]	0 1 2 3 4 5 6 7 8 9
[A-Za-z]	A B C ... Z a b c ... z
[0-9][0-9][0-9]	000 001 .. 009 010 .. 019 100 .. 999
[0-9]*	cadena_vacía 0 1 9 00 99 123 456 999 9999 ...
[0-9][0-9]*	0 1 9 00 99 123 456 999 9999 99999 99999999 ...
^.*$	cualquier línea completa

La repetición tiene preferencia sobre la concatenación; la concatenación tiene preferencia sobre la alternativa. Una expresión puede encerrarse entre paréntesis para que sea evaluada primero.

Expresion regular	Se unifica con...
[0-9]+	0 1 9 00 99 123 456 999 9999 99999 99999999 ..
[0-9]?	cadena_vacía 0 1 2 .. 9
^a|b	a b
(ab)*	cadena_vacía ab abab ababab ...
^[0-9]?b	b 0b 1b 2b .. 9b
([0-9]+ab)*	cadena_vacía 1234ab 9ab9ab9ab 9876543210ab 99ab99ab ...

Probando expresiones regulares, comando grep, sed

La sintaxis de las expresiones regulares no es nada trivial. Cuando tengamos que escribir una expresión regular complicada estaremos delante de una ristra de caracteres especiales imposibles de entender a primera vista, así que para aprender a usarlas es imprescindible contar con una forma de hacer todas las pruebas que queramos y ver los resultados fácilmente. Por eso voy a poner ahora varios comandos con los que podremos hacer las pruebas y experimentar todo lo que necesitemos hasta que tengamos las expresiones regulares dominadas.

1) Comando grep

El primero de ellos es el comando grep. Este es el comando que usaremos con más frecuencia para hacer búsquedas de texto dentro de uno o más archivos. La sintaxis es la siguiente:

grep [-E] 'REGEX' FICHERO
COMANDO | grep [-E] 'REGEX'

La primera forma sirve para buscar una expresión regular en un fichero. La segunda permite filtrar la salida de un comando a través de una expresión regular. Por defecto, grep usa expresiones regulares básicas. La opción -E es para usar expresiones regulares extendidas.

`grep` proviene del editor `ed` (editor de texto Unix), y en concreto, de su comando de búsqueda de expresiones regulares "global regular expression print".

`grep` utiliza las BRE, `egrep` utiliza las ERE (no obstante, podemos usar `grep -E` para que considere ERE o `ogrep`).

Consejo: antes de incluirlas en el *script*, probar las expresiones regulares en la conso- la con `grep` (se resaltan los emparejamientos con `grep --colour`, que suele estar activo por defecto).

Como muchos de los caracteres especiales de las regex son también especiales en bash, es una buena costumbre rodear la regex con comillas simples (' ') cuando estemos escribiendo un script → Siempre que la regex no contenga variables.

Veamos las opciones de grep:

-c En lugar de imprimir las líneas que coinciden, muestra el número de líneas que coinciden.

grep -c 'palabra' logfile.txt

-e nos permite especificar varios patrones de búsqueda.

grep -e 'perro' -e 'gato' /home/miusuario/documentos/

-r busca recursivamente dentro de todos los subdirectorios del directorio actual.

grep -r 'palabra' /home/miusuario/documentos/

-v nos muestra las líneas que no coinciden con el patrón buscado.

-i ignora la distinción entre mayúsculas y minúsculas.
-n Numera las líneas en la salida.

grep -n 'palabra' logfile.txt

-E nos permite usar expresiones regulares. Equivalente a usar egrep.
-o le indica a grep que nos muestre sólo la parte de la línea que coincide con el patrón.
-H nos imprime el nombre del archivo con cada coincidencia.
-w fuerza la búsqueda a coincidir con la palabra exacta

Fichero de ejemplo para probar el comando grep

Para hacer nuestras pruebas también necesitaremos un texto que nos sirva como ejemplo para hacer búsquedas en él. Podemos utilizar el siguiente texto:

```
- Lista de páginas wiki:

ArchLinux: https://wiki.archlinux.org/
Gentoo: https://wiki.gentoo.org/wiki/Main_Page
CentOS: http://wiki.centos.org/
Debian: https://wiki.debian.org/
Ubuntu: https://wiki.ubuntu.com/

- Fechas de lanzamiento:

Arch Linux: 11-03-2002
Gentoo: 31/03/2002
CentOs: 14-05-2004 03:32:38
Debian: 16/08/1993
Ubuntu: 20/10/2004

Desde Linux RegularExp.
```

Este es el texto que usaré para los ejemplos del resto del post, así que os recomiendo que lo copiéis en un fichero para tenerlo a mano desde la terminal. Podéis poner el nombre que queráis. Yo lo he llamando regex.

Probando las expresiones regulares con grep

Ahora ya tenemos todo lo necesario para empezar a probar las expresiones regulares. Vayamos poco a poco. Voy a poner varios ejemplos de búsquedas con expresiones regulares en los que iré explicando para qué sirve cada carácter. No son ejemplos muy buenos, pero como me va a quedar un post muy largo no quiero complicarlo más. Y eso que sólo voy a arañar la superficie de lo que se puede hacer con expresiones regulares.

Lo más sencillo de todo es buscar una palabra concreta, por ejemplo, supongamos que queremos buscar todas las lineas que contengan la palabra "Linux". Esto es lo más fácil, ya que sólo tenemos que escribir:

grep 'Linux' regex

Y podremos ver el resultado:

```
ArchLinux: https://wiki.archlinux.org/
Arch Linux: 11-03-2002
Desde Linux RegularExp.
```

Estas son las tres lineas que contienen la palabra “Linux” la cual, si hemos usado el truco del color, aparecerá resaltada. Fijaros que reconoce la palabra que estamos buscando aunque forme parte de una palabra más larga como en “ArchLinux”. Sin embargo, no resalta la palabra “linux” que aparece en la URL “https://wiki.archlinux.org/”. Eso es porque ahí aparece con la “l” minúscula y la hemos buscado en mayúscula.

Con esta sencilla prueba podemos sacar la primera conclusión:

1. **Un carácter normal puesto en una expresión regular empareja consigo mismo.**

Lo que viene a decir que si pones la letra “a” buscará la letra “a”. Parece lógico, ¿verdad?

Supongamos ahora que queremos buscar la palabra “CentO” seguida de cualquier carácter, pero sólo un único carácter. Para esto podemos usar el **carácter “.”**, que es un comodín que empareja con un carácter cualquiera, pero sólo uno:

grep 'CentO.' regex

Y el resultado es:

```
CentOS: http://wiki.centos.org/
CentOs: 14-05-2004 03:32:38
```

Lo que significa que incluye la “S” de “CentOS” aunque en un caso es mayúscula y en otro minúscula. Si apareciera en ese lugar cualquier otro carácter también lo incluiría. Ya tenemos la segunda regla:

2. **El carácter “.” empareja con cualquier carácter.**

Ya no es tan trivial como parecía, pero con esto no podemos hacer mucho. Avancemos un poco más. Vamos a suponer que queremos encontrar las líneas en las que aparece el año 2002 y el 2004. Parecen dos búsquedas, pero se pueden hacer de una sola vez así:

grep '200[24]' regex

Lo que quiere decir que queremos buscar el número 200 seguido del 2 o el 4. Y el resultado es este:

```
Arch Linux: 11-03-2002
Gentoo: 31/03/2002
CentOs: 14-05-2004 03:32:38
Ubuntu: 20/10/2004
```

Lo que nos lleva a la tercera regla:

3. **Varios caracteres encerrados entre corchetes emparejan con cualquiera de los caracteres que hay dentro de los corchetes.**

Los corchetes dan más juego. también se pueden usar para excluir caracteres. Por ejemplo, supongamos que queremos buscar los sitios en los que aparece el carácter ":", pero que no vaya seguido de "/". El comando sería así:

grep ':[^/]' regex

Se trata simplemente de poner un "^" como primer carácter dentro del corchete. Se pueden poner a continuación todos los caracteres que se quieran. El resultado de este último comando es el siguiente:

```
ArchLinux: https://wiki.archlinux.org/
Gentoo: https://wiki.gentoo.org/wiki/Main_Page
CentOS: http://wiki.centos.org/
Debian: https://wiki.debian.org/
Ubuntu: https://wiki.ubuntu.com/
Arch Linux: 11-03-2002
Gentoo: 31/03/2002
CentOs: 14-05-2004 03:32:38
Debian: 16/08/1993
Ubuntu: 20/10/2004
```

Ahora aparecen resaltados los ":" que hay detrás de los nombres de las distros, pero no los que hay en las URL porque los de las URL llevan "/" a continuación.

4. **Poner el carácter "^" al principio de un corchete empareja con cualquier carácter excepto con los demás caracteres del corchete.**

Otra cosa que podemos hacer es especificar un rango de caracteres. Por ejemplo, para buscar cualquier número seguido de un "-" sería así:

grep '[0-9]-' regex

Con esto estamos especificando un carácter entre 0 y 9 y, a continuación, un signo menos. Veamos el resultado:

```
Arch Linux: 11-03-2002
CentOs: 14-05-2004 03:32:38
```

Se pueden especificar varios rangos dentro de los corchetes a incluso mezclar rangos con caracteres sueltos.

5. **Colocar dos caracteres separados por "-" dentro de los corchetes empareja con cualquier carácter dentro del rango.**

Vamos a ver ahora si podemos seleccionar la primera parte de las URL. La que pone "http" o "https". Sólo se diferencian en la "s" final, así que vamos a hacerlo de la siguiente manera:

grep -E 'https?' regex o también podemos poner egrep 'https?' regex

La interrogación sirve para hacer que el carácter que hay a su izquierda sea opcional. Pero ahora hemos añadido la opción -E al comando. Esto es porque la interrogación es una característica de las expresiones regulares extendidas. Hasta ahora estábamos usando expresiones regulares básicas, así que no hacía falta poner nada. Veamos el resultado:

```
ArchLinux: https://wiki.archlinux.org/
Gentoo: https://wiki.gentoo.org/wiki/Main_Page
CentOS: http://wiki.centos.org/
Debian: https://wiki.debian.org/
Ubuntu: https://wiki.ubuntu.com/
```

O sea que ya tenemos una nueva regla:

6. **Un carácter seguido de "?" empareja con ese carácter o con ninguno (o sea, se repite cero o una vez el carácter de la izquierda). Esto sólo es válido para expresiones regulares extendidas.**

Ahora vamos a buscar dos palabras completamente diferentes. Vamos a ver cómo buscar las líneas que contengan tanto la palabra "Debian" como "Ubuntu".

grep -E 'Debian|Ubuntu' regex

Con la barra vertical podemos separar dos o más expresiones regulares diferentes y buscar las lineas que emparejen con cualquiera de ellas:

```
Debian: https://wiki.debian.org/
Ubuntu: https://wiki.ubuntu.com/
Debian: 16/08/1993
Ubuntu: 20/10/2004
```

7. **El carácter "|" sirve para separar varias expresiones regulares y empareja con cualquiera de ellas. También es específico de las expresiones regulares extendidas.**

Continuemos. Ahora vamos a buscar la palabra "Linux", pero sólo donde no esté pegada a otra palabra por la izquierda. Podemos hacerlo así:

grep '\<Linux' regex

Aquí el carácter importante es "<", pero es necesario escaparlo colocando "\" delante para que grep lo interprete como un carácter especial. El resultado es el siguiente:

```
Arch Linux: 11-03-2002
Desde Linux RegularExp.
```

También se puede usar "\>" para buscar palabras que no estén pegadas a otras por la derecha. Vamos con un ejemplo. Probemos este comando:

grep 'wiki\>' regex

El resultado que produce es este:

```
ArchLinux: https://wiki.archlinux.org/
Gentoo: https://wiki.gentoo.org/wiki/Main_Page
CentOS: http://wiki.centos.org/
Debian: https://wiki.debian.org/
Ubuntu: https://wiki.ubuntu.com/
```

Fijaros que no ha salido la primera línea "-Lista de páginas wikipedia" porque hay más caracteres a la derecha de wiki formado una palabra.

8. **Los caracteres "<" y ">" emparejan con el principio y el final de una palabra, respectivamente. Hay que escapar estos caracteres para que no se interpreten como caracteres literales.**

Vamos con cosas un poco más complicadas. El carácter "+" empareja con el carácter que aparece a su izquierda repetido al menos una vez. Este carácter sólo está disponible con las expresiones regulares extendidas. Con él podemos buscar, por ejemplo, secuencias de varios números seguidos que empiecen con ":".

```
grep -E ':[0-9]+' regex
```

Resultado:

```
CentOs: 14-05-2004 03:32:38
```

Queda resaltado también el número 38 porque también empieza con ":".

9. **El carácter "+" empareja con el carácter que aparece a su izquierda repetido al menos una vez (o sea, se repite una o más veces el carácter de la izquierda)**

También se puede controlar el número de repeticiones usando "{" y "}". La idea es colocar entre llaves un número que indica el número exacto de repeticiones que queremos. También se puede poner un rango. Vamos a ver ejemplos de los dos casos.

Primero vamos a buscar todas las secuencias de cuatro dígitos que haya:

```
grep '[0-9]\{4\}' regex
```

Fijaros en que hay que escapar las llaves si estamos usando expresiones regulares básicas, pero no si usamos las extendidas. Con extendidas sería así:

```
grep -E '[0-9]{4}' regex
```

Y el resultado en los dos casos sería este:

```
Arch Linux: 11-03-2002
Gentoo: 31/03/2002
CentOs: 14-05-2004 03:32:38
Debian: 16/08/1993
Ubuntu: 20/10/2004
```

10. **Los caracteres “{” y “}” con un número entre ellos emparejan con el carácter anterior repetido el número de veces indicado.**

Ahora el otro ejemplo con las llaves. Supongamos que queremos encontrar palabras que tengan entre 3 y 6 letras minúsculas. Podríamos hacer lo siguiente:

egrep '[a-z]{3,6}' regex

Y el resultado sería este:

```
- Lista de páginas wiki:
ArchLinux: https://wiki.archlinux.org/
Gentoo: https://wiki.gentoo.org/wiki/Main_Page
CentOS: http://wiki.centos.org/
Debian: https://wiki.debian.org/
Ubuntu: https://wiki.ubuntu.com/
- Fechas de lanzamiento:
Arch Linux: 11-03-2002
Gentoo: 31/03/2002
CentOs: 14-05-2004 03:32:38
Debian: 16/08/1993
Ubuntu: 20/10/2004
Desde Linux RegularExp.
```

Que, como veis, no se parece mucho a lo que queríamos. Eso es porque la expresión regular encuentra las letras dentro de otras palabras que son más largas. Probemos con esta otra versión:

egrep '\<[a-z]{3,6}\>' regex ó también se puede utilizar egrep '\b[a-z]{3,6}\b' regex

Resultado:

```
ArchLinux: https://wiki.archlinux.org/
Gentoo: https://wiki.gentoo.org/wiki/Main_Page
CentOS: http://wiki.centos.org/
Debian: https://wiki.debian.org/
Ubuntu: https://wiki.ubuntu.com/
```

Esto ya se parece más a lo queríamos. Lo que hemos hecho es exigir que la palabra empiece justo delante de la primera letra y termine justo detrás de la última.

11. **Los caracteres “{” y “}” con dos números entre ellos separados por una coma emparejan con el carácter anterior repetido el número de veces en el rango indicado por los dos números.**

Veamos ahora un carácter que es primo de “+”. Se trata de “*” y su funcionamiento es muy parecido sólo que empareja con cualquier número de caracteres incluido cero. O sea que hace lo mismo que el “+” pero no exige que el carácter de su izquierda aparezca en el texto. Por ejemplo, probemos a buscar esas direcciones que empiezan en wiki y acaban en org:

grep 'wiki.*org' regex

Vamos a ver el resultado:

```
ArchLinux: https://wiki.archlinux.org/
Gentoo: https://wiki.gentoo.org/wiki/Main_Page
CentOS: http://wiki.centos.org/
Debian: https://wiki.debian.org/
```

12. **El carácter "*" empareja con el carácter que aparece a su izquierda cero o más veces**

Ahora el último carácter que vamos a ver. El carácter "\" sirve para escapar el carácter de su derecha de manera que pierda el significado especial que tiene. Por ejemplo: Supongamos que queremos localizar las líneas que terminen en punto. Lo primero que se nos podría ocurrir podría ser esto:

grep '.$' regex

El resultado no es el que buscamos:

```
- Lista de páginas wiki:
ArchLinux: https://wiki.archlinux.org/
Gentoo: https://wiki.gentoo.org/wiki/Main_Page
CentOS: http://wiki.centos.org/
Debian: https://wiki.debian.org/
Ubuntu: https://wiki.ubuntu.com/
- Fechas de lanzamiento:
Arch Linux: 11-03-2002
Gentoo: 31/03/2002
CentOs: 14-05-2004 03:32:38
Debian: 16/08/1993
Ubuntu: 20/10/2004
Desde Linux RegularExp.
```

Esto es porque el carácter "." empareja con cualquier cosa, así que esa expresión regular empareja con el último carácter de cada línea sea cual sea. La solución es esta:

grep '\.$' regex

Ahora el resultado sí que es el que queremos:

```
Desde Linux RegularExp.
```

Ejercicios

1. Encontrar todos los números con signo (con posibilidad o no de decimales)

2. Lo anterior pero en este caso 5 números decimales o más (sin signo)

3. Expresión regular para validad una dirección de email, sabiendo que la cuenta (antes del símbolo @) puede contener alfanuméricos en minúsculas, el carácter _, carácter -, carácter . para separar palabras y que el dominio (ej: .com) pueda contener de 2 a 4 caracteres en minúsculas.

4. Expresión regular para buscar una dirección IP

5. Expresión regular para buscar fechas con los formatos: dd/mm/yyyy ó dd-mm-yyyy

2) Comando sed

SED (Stream EDitor) es un editor de flujos y ficheros de forma no interactiva. Permite modificar el contenido de las diferentes líneas de un fichero en base a una serie de comandos o un fichero de comandos (-f fichero_comandos).

Sed recibe por stdin (o vía fichero) una serie de líneas para manipular, y aplica a cada una de ellas los comandos que le especifiquemos a todas ellas, a un rango de estas, o a las que cumplan alguna condición.

Formato de uso

Es parecido a grep pero permite cambiar las líneas que encuentra (en lugar de solo mostrarlas).

En realidad, es un editor de textos no interactivo, que recibe sus comandos como si fuesen un script.

Solo vamos a estudiar algunos de los comandos posibles.

Por defecto, todas las líneas se imprimen tras aplicar el comando.

`sed [-r] [-n] -e 'comando' [archivo]:`

- `-r`: uso de EREs en lugar de BREs.
- `-n`: modo silencioso, para imprimir una línea tienes que indicarlo explícitamente mediante el comando p (*print*)
- `-e 'comando'`: ejecutar el comando o comandos especificados.
- Sintaxis de comandos:
  `[direccionInicio[, direccionFin]][!]comando [argumentos]`:
  - Si la dirección es adecuada, entonces se ejecutan los comandos (con sus argumentos).
  - Las direcciones pueden ser expresiones regulares (*/regex/*) o números de línea (N), rango de números (N,M). Son opcionales.
  - Si no hay `direccionFin` solo se aplica sobre `direccionInicio` y si no se especifican se actúa sobre todas las líneas del flujo.
  - *`!`* emparejaría todas las direcciones distintas que la indicada.
- Comandos:
  - **i** => Insertar línea antes de la línea actual.
  - **a** => Insertar línea después de la línea actual.
  - **c** => Cambiar línea actual.

- **d** => Borrar línea actual.
- **p** => Imprimir línea actual en stdout.
- **q** = >Finalizar procesamiento del fichero.
- **s** => Sustituir una expresión por otra sobre las líneas seleccionadas.

Sintaxis Comando **s**:

s/patron/reemplazo/[banderas]

- patrón: expresión regular BRE.
- reemplazo: cadena con que reemplazarla.
- Bandera **n**: reemplazar solo la ocurrencia n-ésima.
- Bandera **g**: reemplazar todas las ocurrencias.
- Bandera **p**: forzar a imprimir la línea (solo tiene sentido si hemos utilizado -n).

Ejemplos

```
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt
Este es otro ejemplo de expresiones regulares
La segunda parte ya la veremos
,,,,adios,hola
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed '3p'
Este es otro ejemplo de expresiones regulares
La segunda parte ya la veremos
,,,,adios,hola
,,,,adios,hola
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed -n '3p'
,,,,adios,hola
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed -n '1,2p'
Este es otro ejemplo de expresiones regulares
La segunda parte ya la veremos
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed -n '1,2!p'
,,,,adios,hola
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed '/^L/d'
Este es otro ejemplo de expresiones regulares
,,,,adios,hola
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed '2,$d'
Este es otro ejemplo de expresiones regulares
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed '1,/s$/d'
,,,,adios,hola
```

```
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt
Este es otro ejemplo de expresiones regulares
La segunda parte ya la veremos
,,,,adios,hola
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed  -r 's/La/El/'
Este es otro ejemplo de expresiones regulares
El segunda parte ya la veremos
,,,,adios,hola
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed  -r 's/[Ll]a/El/'
Este es otro ejemplo de expresiones reguElres
El segunda parte ya la veremos
,,,,adios,hoEl
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed  -r 's/([Ll])a/era\1/'
Este es otro ejemplo de expresiones regueralres
eraL segunda parte ya la veremos
,,,,adios,hoeral
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed -r -n 's/(d[ea])/"\1"/p'
Este es otro ejemplo "de" expresiones regulares
La segun"da" parte ya la veremos
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo2.txt
Grado:Informatica
Informatica2:Grado2
i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo2.txt | sed -r -n 's/(.*):(.*)/\2:\1/p'
Informatica:Grado
Grado2:Informatica2
```

Otros ejemplos:

```
Sustituir apariciones de cadena1 por cadena2 en todo el fichero:
# sed 's/cadena1/cadena2/g' fichero

Sustituir apariciones de cadena1 por cadena2 en las líneas 1 a 10:
# comando | sed '1,10 s/cadena1/cadena2/g'

Eliminar las líneas 2 a 7 del fichero
# sed '2,7 d' fichero > fichero2

Buscar un determinado patrón en un fichero:
# sed -e '/cadena/!d' fichero

Buscar AAA o BBB o CCC en la misma línea:
# sed '/AAA/!d; /BBB/!d; /CCC/!d' fichero
```

Ejemplos de sustitución

```
Reemplazar cadenas:
# sed 's/^Host solaris8/Host solaris9/g' fichero

Reemplazar cadenas sólo en las líneas que contengan una cadena:
# sed '/cadena_a_buscar/ s/vieja/nueva/g' fichero

Reemplazar cadenas sólo en en determinadas líneas:
# sed '5,6 s/vieja/nueva/g' fichero

Reemplazar multiples cadenas (A o B):
# sed -r 's/cadena1|cadena2/cadena_nueva/g'
```

```
Sustituir líneas completas (c) que cumplan o no un patrón:
# echo -e "linea 1\nlinea 2" | sed '/1/ cPrueba'
  Prueba
  linea 2
```

Ejemplos de Inserción

```
Insertar 3 espacios en blanco al principio de cada línea:
# sed 's/^/   /' fichero

Añadir una línea antes o despues del final de un fichero ($=última línea):
# sed -e '$i Prueba' fichero > fichero2
# sed -e '$a Prueba' fichero > fichero2

Insertar una linea en blanco antes de cada linea que cumpla una regex:
# sed '/cadena/{x;p;x;}' fichero

Insertar una linea en blanco detras de cada linea que cumpla una regex:
# sed '/cadena/G' fichero

Insertar una linea en blanco antes y despues de cada linea que cumpla una regex:
# sed '/cadena/{x;p;x;G;}' fichero
```

Ejemplos de Selección/Visualización

```
Ver las primeras 10 líneas de un fichero:
# sed 10q

Ver las últimas 10 líneas de un fichero:
# sed -e :a -e '$q;N;11,$D;ba'

Ver un rango concreto de líneas de un fichero:
# cat -n fich2 | sed -n '2,3 p'
  2    linea 2
  3    linea 3
(Con cat -n, el comando cat agrega el número de línea).
(Con sed -n, no se imprime nada por pantalla, salvo 2,3p).

Ver un rango concreto de líneas de varios ficheros:
# sed '2,3 p' *
  linea 2 fichero 1
  linea 3 fichero 1
  linea 2 fichero 2
  linea 3 fichero 2
(-s = no tratar como flujo sino como ficheros separados)

Sólo mostrar la primera linea de un fichero:
# sed -n '1p' fichero

No mostrar la primera linea de un fichero:
# sed '1d' fichero

Mostrar la primera/ultima línea de un fichero:
# sed -n '1p' fichero
```

```
# sed -n '$p' fichero

Imprimir las líneas que no hagan match con una regexp (grep -v):
# sed '/regexp/!d' fichero
# sed -n '/regexp/p' fichero

Mostrar la línea que sigue inmediatamente a una regexp:
# sed -n '/regexp/{n;p;}' fichero

Mostrar desde una expresión regular hasta el final de fichero:
# sed -n '/regexp/,$p' fichero

Imprimir líneas de 60 caracteres o más:
# sed -n '/^.\{60\}/p' fichero

Imprimir líneas de 60 caracteres o menos:
# sed -n '/^.\{65\}/!p' fichero
# sed '/^.\{65\}/d' fichero
```

Ejemplos de Borrado

```
Eliminar un rango concreto de líneas de un fichero:
# sed '2,4 d' fichero

Eliminar todas las líneas de un fichero excepto un rango:
# sed '2,4 !d' fichero

Eliminar la última línea de un fichero
# sed '$d' fichero

Eliminar desde una línea concreta hasta el final del fichero:
# sed '2,$d' fichero

Eliminar las líneas que contentan una cadena:
# sed '/cadena/d' fichero
# sed '/^cadena/d' fichero
# sed '/^cadena$/d' fichero

Eliminar líneas en blanco (variación del anterior):
# comando | sed '/^$/d'
# sed '/^$/d' fichero

Eliminar múltiples líneas en blanco consecutivas dejando sólo 1:
# sed '/./,/^$/!d' fichero

Añadir una línea después de cada línea:
# echo -e "linea 1\nlinea 2" | sed 'aPrueba'
  linea 1
  Prueba
  linea 2
  Prueba
```

```
Eliminar espacios al principio de línea:
# sed 's/^ *//g' fichero

Eliminar todos los espacios que haya al final de cada línea:
# sed 's/*$//' fichero

Eliminar líneas en blanco y comentarios bash:
# comando | sed '/^$/ d'
# sed '/^$/d; /^#/d' fichero
```

Ejercicios

1. Utilizar sed para sustituir los delimitadores de columnas (comas y espacios sobrantes) por el carácter ; en el archivo ejercicio_sed.txt.

2. Utilizar expresiones regulares con `sed`, para transformar la salida del comando `df` al formato indicado abajo.

```
1  mdorado@mdoradoLaptop ~ $ ./espacioLibre.sh
2  El fichero de bloques /dev/sda2, montado en /, tiene usados 18218120 bloques de un total de
       49410864 (porcentaje de 39%).
3  El fichero de bloques udev, montado en /dev, tiene usados 0 bloques de un total de 10240 (
       porcentaje de 0%).
4  El fichero de bloques tmpfs, montado en /run, tiene usados 928 bloques de un total de 601488
        (porcentaje de 1%).
5  El fichero de bloques tmpfs, montado en /run/lock, tiene usados 0 bloques de un total de
       5120 (porcentaje de 0%).
6  El fichero de bloques tmpfs, montado en /run/shm, tiene usados 1560 bloques de un total de
       2025480 (porcentaje de 1%).
7  El fichero de bloques /dev/sdb1, montado en /boot/efi, tiene usados 42932 bloques de un
       total de 262144 (porcentaje de 17%).
8  El fichero de bloques /dev/sda3, montado en /home, tiene usados 50397976 bloques de un total
        de 65282844 (porcentaje de 82%).
9  El fichero de bloques /dev/sdb6, montado en /home2, tiene usados 282248360 bloques de un
       total de 372531364 (porcentaje de 80%).
10 El fichero de bloques none, montado en /sys/fs/cgroup, tiene usados 0 bloques de un total de
        4 (porcentaje de 0%).
```