SESIÓN N° 04

EXPRESIONES REGULARES

I
OBJETIVOS

- Explicar, manipular y trabajar con Expresiones Regulares como mecanismo de búsqueda, y modificaciones en archivos y flujos de salida.

II
TEMAS A TRATAR

- Wildcards.
- Comando *grep*
- Comando *set*

III
MARCO TEORICO

Una expresión regular define el conjunto de una o más cadenas de caracteres. Varios comandos de Linux utilizan expresiones regulares para buscar y reemplazar cadenas de texto, incluidos **ed**, **vi**, **grep, sed y ls**.

Las expresiones regulares son una herramienta para definir patrones de búsqueda y reemplazo. En ellas se definen lo que llamaremos átomos que serán las partes que buscaremos. En los patrones podemos hacer referencias a “un carácter cualquiera”, “el comienzo de línea”, “el final de línea” y demás. Para ello utilizamos caracteres reservados conocidos como wildcards.

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. Para practicar las expresiones regulares utilizaremos el comando "grep" y el sed.

Wildcards

. **caracter de correspondencia simple (Cualquier caracter)**

Ejemplo : „l . nu .“ podría resultar en lynux, linux o linus

***** **caracter de correspondencia múltiple. Cero o más apariciones del carácter previo. Se utiliza en combinación con ‘.’ y con los rangos.**

Ejemplo: f.*o (texto que empiezan con f y termina con o)

[rango] **Especifica un posible rango o conjunto de caracteres**

Ejemplo:

[a−d].* (texto que empieza con a,b,c o d)
[axf].* (texto que empieza con a,x,o f)

[^ rango] **Rango negado**

Ejemplo: [^0−9].* (texto que no empieza con un número)

^ **Ancla de comienzo de línea**

Ejemplo: ^j (línea que comienza con j)

$ **Ancla de fin de línea**

Ejemplo: j$ (línea que finaliza con j)

? **El signo de interrogación especifica cero o una ocurrencia del caracter precedente.**

Ejemplo: pesos? coincide con la cadena peso o con pesos.

+ **El símbolo más indica especifica una o más ocurrencias del caracter precedente.**

Ejemplo: la expresión [0−9]+ es equivalente a la expresión [0−9][0−9]*.

Más constructores de expresiones regulares

La tabla siguiente muestra algunos ejemplos más que representa líneas que calzan con el patrón especificado a través de la expresión regular:

Expresión regular	Descripción
/./	Apuntará a cualquier línea que contenga al menos un carácter
/../	Apuntará a cualquier línea que contenga al menos dos caracteres
/^#/	Apuntará a cualquier línea que comience con un '#'
/^$/	Apuntará a cualquier línea en blanco
/}$/	Apuntará a toda línea que termine con un '}' (sin espacios)
/} *$/	Apuntará a toda línea que termine con un '}' con cero o más espacios
/[abc]/	Apuntará a toda línea que contenga una 'a', 'b', o 'c' minúscula
/^[abc]/	Apuntará a cualquier línea que empiece con 'a', 'b', o 'c'

Clases de caracteres

Estas clases están disponibles para coincidir con algún patrón en particular.

Clase de carácter	Descripción
[:alnum:]	Alfanumérico [a-z A-Z 0-9]
[:alpha:]	Alfabético [a-z A-Z]
[:blank:]	Espacios o tabuladores
[:cntrl:]	Cualquier carácter de control
[:digit:]	Dígitos numéricos [0-9]
[:graph:]	Cualquier carácter visible (no espacios en blanco)
[:lower:]	Minúsculas [a-z]
[:print:]	Caracteres que no sean de control
[:punct:]	Caracteres de puntuación
[:space:]	Espacio en blanco
[:upper:]	Mayúsculas [A-Z]
[:xdigit:]	Dígitos hexadecimales [0-9 a-f A-F]

Comando grep

Muestra todas las líneas de un archivo dado que coinciden con cierto patrón.

```
grep [OPCIONES] PATRON [FILE1] [FILE2] [FILEn]...
```

Donde:

PATRON es una expresión regular
[FILE1] [FILE2] [FILEn]... son los archivos donde buscar**.**

Ejemplo: Para buscar el texto "hello world" dentro del archivo main.c

```
$ grep 'hello world' main.c
$ grep 'hello world' < main.c
$ cat main.c | grep 'hello world'
```

(las tres líneas hacen lo mismo)

Comando *sed*

Significa Stream Editor, es decir, un editor de flujos. El comando sed nos va a permitir editar flujos de datos pasados a través de una tubería (pipe) por ejemplo. Es muy útil para hacer modificaciones en flujos de datos o ficheros de texto. Estas modificaciones pueden ser añadir o borrar una línea, o un conjunto de ellas, buscar y reemplazar patrones.

La sintaxis de sed normalmente es:

```
sed s/patrón/reemplazo/opción
```

pero también acepta la siguiente sintaxis:

```
sed s:patrón:reemplazo:opción
```

A continuación veremos algunos ejemplos en los que se utilizan expresiones regulares con el comando *sed,* el cual nos permite, de una forma cómoda, borrar líneas, registros o sustituir cadenas de caracteres dentro de las líneas.

Pruebe cada uno de los siguientes ejemplos y verifique los resultados obtenidos.

Eliminación de líneas

Para borrar una línea hacemos lo siguiente:

```
sed 'n°_de_línea' fichero
```

Ejemplos:

Eliminar la primera línea del archivo llamado datos:

```
$ sed '1d' datos
```

```
$ sed -e '1d' /etc/services | more
```

Indicando un intervalo de líneas a borrar.

```
$ sed '3,5d' datos
```

```
$ sed -e '1,10d' /etc/services | more
```

También podemos indicar que queremos borrar desde una determinada línea en adelante:

```
$ sed '3,$d' datos
```

Uso de expresiones regulares para indicar las direcciones de las líneas:

Borrar las líneas en blanco del archivo llamado datos

```
$ sed '/^$/d' datos
```

Lo siguiente elimina todas las líneas que comienzan con '#' (líneas de comentario) del archivo **services**:

```
$ sed -e '/^#/d' /etc/services | more
```

A la hora de borrar, también podemos especificar una cadena, de tal forma que el comando borrará todas las líneas que contengan esa cadena. El siguiente ejemplo borra todas las líneas en blanco del archivo *datos*, y lo guarda en el archivo *ficherodestino*.

```
$ cat datos | sed '/^[ ]*$/d' > ficherodestino
```

Otro de los usos interesantes es borrar los espacios al comienzo de cada línea:

```
$ sed 's/^ *//g' datos
```

O borrar los espacios al final de cada línea:

```
$ sed 's/ *$//g' datos
```

Sustitución de cadenas

Otro de los usos más interesantes de sed es sustituir cadenas.

Ejemplos:

Podemos sustituir una cadena por otra de la siguiente manera:

```
$ sed 's/cadena1/cadena2/' datos
```

Al ejecutar el comando anterior, se sustituye la primera cadena que encuentra por la segunda. Pero, si lo que queremos es sustituir todas las cadenas que encuentre, en cada una de las líneas, añadimos el parámetro g:

```
$ sed 's/cadena1/cadena2/g' datos
```

Lo siguiente sacará el contenido de miarchivo.txt a stdout, en donde la cadena *foo* es reemplazada por *bar* de forma global.

```
$ sed -e 's/foo/bar/g' miarchivo.txt
```

Por otra parte, también podemos hacer que sustituya la cadena1 por la cadena2 en un número de línea concreto:

```
$ sed '5 s/USUARIO/usuario/g' datos
```

Con cadenas de texto normales la cosa es sencilla, pero al que más y al que menos le resulta complicado cuando lo que hay que sustituir son caracteres especiales como el tabulador: \t o el caracter de nueva línea: \n. Pero veamos como tampoco es complicado: Imaginemos que tenemos un fichero con campos en los que el separador es el tabulador y queremos sustuir este caracter separador por otro caracter separador, como por ejemplo el punto y coma (;). Lo haremos de la siguiente manera:

```
$ sed 's/\t/;/g' fichero
```

Reemplazar 'enchantment' por 'entrapment', pero solo hasta la línea 10, inclusive.

```
$ sed -e '1,10s/enchantment/entrapment/g' miarchivo2.txt
```

Este ejemplo cambiará 'colinas' por 'montañas', pero únicamente en los bloques de texto que comiencen con una línea en blanco, y que terminen con una línea que comience con los tres caracteres 'FIN', inclusive.

```
$ sed -e '/^$/,/^FIN/s/colinas/montañas/g' miarchivo3.txt
```

Otra de las buenas cosas acerca del comando s/// es que tenemos muchas opciones cuando usamos estos / separadores. Si estamos realizando la sustitución de una cadena y la expresión regular o el reemplazo tiene muchas barras, podemos cambiar el separador especificando un carácter distinto después de la 's'. Por ejemplo, lo siguiente cambiará el contenido cambiando /usr/local por /usr:

```
$ sed -e 's:/usr/local:/usr:g' milista.txt
```

Nota: En este ejemplo, estamos usando los dos puntos como separador. Si alguna vez necesitamos especificar el carácter separador en nuestra expresión regular hay que ponerle antes una barra invertida.

Hasta ahora, únicamente hemos realizado una simple sustitución de cadenas. Aunque esto puede ser muy útil, podemos también buscar expresiones regulares. Por ejemplo, el siguiente comando sed encontrará una frase que comience con '<' y termine con '>', y que contenga cualquier número de caracteres entremedias. Esta frase se borrará (será reemplazada por una cadena sin contenido):

```
$ sed -e 's/<.*>//g' miarchivo.html
```

Coincidencia con más caracteres

La sintaxis de expresiones regulares '[ ]' tiene más opciones adicionales. Para especificar un rango de caracteres se puede usar '-', siempre y cuando no esté ni en la primera ni en la última posición, como se muestra a continuación.

```
'[a-x]*'
```

Apuntará a cero o más caracteres, siempre que cada uno de ellos sea una 'a', 'b','c'...'v','w','x'. Además,

Sustitución avanzada:

Nos hemos detenido con la realización de simples e incluso complejas sustituciones directas, pero *sed* puede hacer mucho más. Podemos referirnos a partes o a cadenas enteras con las que coincida la expresión regular y usar estas partes para construir la cadena de reemplazo. Como ejemplo, digamos que estamos respondiendo a un mensaje. El siguiente ejemplo pondrá el prefijo "Rafa dijo: " a cada frase:

```
$ sed -e 's/.*/Rafa dijo: &/' msjorig.txt
```

La salida será similar a esta:

```
Rafa dijo: Hola Jaime,
Rafa dijo:
Rafa dijo: ¡Seguro que me gusta este material acerca de sed!
Rafa dijo:
```

En este ejemplo, usamos el carácter '&' en la cadena de reemplazo, que le indica a sed que inserte la expresión regular completa con la que coincida. Por lo que todo lo que coincidió con '.*' (el mayor grupo de cero o más caracteres en la línea, o la línea completa) puede ser introducido en la cadena de reemplazo, incluso en múltiples ocasiones.

Uso de paréntesis con barras invertidas

Aún mejor que con '&', el comando s/// nos permite definir regiones en nuestra expresión regular, y podemos referirnos a estas regiones específicas en nuestra cadena de reemplazo.

Ejemplo, digamos que tenemos un archivo que contiene el siguiente texto:

```
foo bar oni
eeny meeny miny
larry curly moe
jimmy the weasel
```

Ahora, digamos que queremos escribir un archivo de comandos *sed* que reemplace "eeny meeny miny" con "Victor eeny-meeny Von miny", etc. Para hacerlo, primero deberíamos escribir una expresión regular con la que coincidan las tres cadenas, separadas por espacios:

```
'.* .* .*'
```

Ahora, definiremos regiones insertando paréntesis con barras invertidas alrededor de cada región de nuestro interés:

```
'\(.*\) \(.*\) \(.*\)'
```

Esta expresión regular funcionará exactamente igual que la anterior, excepto porque definirá tres regiones lógicas a las que podremos referirnos en nuestra cadena de reemplazo. He aquí el archivo de comandos final:

```
$ sed -e 's/\(.*\) \(.*\) \(.*\)/Victor \1-\2 Von \3/'
miarchivo.txt
```

Como puede verse, nos referimos a cada región delimitada por un paréntesis tecleando '\x', donde x es el número de región, comenzando por uno. El resultado será el siguiente:

```
Victor foo-bar Von oni
Victor eeny-meeny Von miny
Victor larry-curly Von moe
Victor jimmy-the Von weasel
```

IV
ACTIVIDADES

01. Para trabajar utilizaremos un archivo creado de la siguiente forma:
    ls /bin/ > b.txt (si el ls utiliza colores usar " \ls /bin/ > b.txt")
02. Buscar dentro de b.txt si esta listado el comando "mount" (utilizar grep)

~$

03. Buscar dentro de b.txt comandos que empiecen con "fs"

~$

04. Buscar dentro de b.txt comandos que terminen con "fs"

~$

05. Buscar dentro de b.txt comandos que tengan un caracter numérico mayor a 2

~$

06. Buscar dentro de b.txt comandos que tengan un caracter numérico menor o igual a 2

~$

07. Buscar dentro de b.txt comandos que empiecen con "s" y terminen con "n"

~$

08. Buscar dentro de b.txt comandos que empiecen con "f" o "p" , terminen con "k", "g" o "f" y tengan alguna "i" en el medio

~$

09. Utilizar "ps wax" para listar los procesos en ejecución y filtrar con grep las líneas que posean el texto "get" .

~$

10. Escriba una instrucción grep que obtenga los mismos resultados que la siguiente sentencia:

```
grep -v ^[^a] fichero
```

La sentencia grep que escriba, debe tener un patrón diferente del patrón usado.

~$

11. Contar el número de archivos de configuración (.conf) del directorio /etc.

~$

12. Cree un archivo llamado *nombres*, que contenga un nombre de una persona (nombre, apellido paterno y el materno, separados por espacios) por línea (mínimo 10 líneas). Utilice el archivo *nombres* y reemplazar todas las letras a y e minúsculas por sus correspondientes mayúsculas entre las líneas 3 a la 7. (utilice *sed*)

~$

13. En un archivo *apellidos* guarde solo los apellidos de las personas del archivo *nombres*, antes creado.

~$

14. En un archivo *listado* guarde todos los nombres de las personas del archivo *nombres*, cuyo separador entre cada nombre sea una "," y no el saldo de línea.

~$

15. Se tiene un archivo *log* en un servidor corporativo, donde queda registrada cada conexión. El formato del contenido del archivo log es:

```
IP: a.b.c.d P:p T:t
```

Siendo IP la dirección de la máquina que realiza la conexión, a, b, c y d números entre 0 y 255, p el número del puerto que recibe la conexión y t los segundos que dura la conexión.

Ejemplo del archivo log:

```
IP: 125.130.2.5     P:25    T:84
IP: 125.130.2.5     P:80    T:124
IP: 125.130.1.7     P:21    T:12
IP: 224.25.81.183   P:135   T:0
```

a) Cuente el número total de conexiones realizadas (utilice el comando wc)

~$

b) Genere un nuevo archivo llamado análisis.dat que contenga todas las conexiones recibidas por el puerto 25.

```
~$
```

c) Cuente el número de conexiones realizadas desde la máquina de la subred 125.130.

```
~$
```

d) Muestre por la consola todas las conexiones realizadas a través de los puertos por debajo del puerto 90.

```
~$
```