Tutorial BASH n°3

Comandos para uso de expresiones regulares, filtros y datos de usuario

INDICE

1. Introducción 3
2. Metacaracteres. 4
2.1. Expresiones Regulares Básicas. 4
2.2. Expresiones Regulares Extendidas. 6
3. Filtros 8
3.1. Comando sort 8
3.2. Comando find 8
3.3. Comando grep 9
3.4. Comandos fgrep y egrep 10
3.5. Comando tr 11
3.6. Comando uniq 14
3.7. Comando dd 14
3.8. Comando cut 15
3.9. Comando wc 16
4. Comando id 17
5. Comando stat 18
6. Comando let 19
7. Variable IFS 20
8. Declaración de variables. 21

1. Introducción

Las **expresiones regulares** (**ER**), o llamados también patrones, son una forma de describir cadenas de caracteres. Las expresiones regulares permiten realizar búsquedas o sustituciones de gran complejidad de palabras o de conjuntos de un solo carácter, que cumplan ciertas condiciones.

Una expresión regular es un patrón que describe un conjunto de cadenas de caracteres. Por ejemplo, el patrón **aba*.txt** describe el conjunto de cadenas de caracteres que comienzan con aba, contienen cualquier otro grupo de caracteres, luego un punto, y finalmente la cadena txt. El símbolo ***** se interpreta como "0, 1 o más caracteres cualesquiera".

Las expresiones regulares se construyen como las expresiones aritméticas, usando operadores para combinar expresiones más pequeñas. Analizaremos esos operadores y las reglas de construcción de expresiones regulares, atendiendo siempre al **conjunto de cadenas que representa cada patrón**.

2. Metacaracteres.

La construcción de expresiones regulares depende de la asignación de significado especial a algunos caracteres. En el patrón **aba*.txt** el **carácter *** no vale por sí mismo, como el carácter asterisco, sino que indica un "**conjunto de caracteres cualesquiera**". Asimismo, el **carácter ?** no se interpreta como el signo de interrogación sino que representa "**un carácter cualquiera y uno solo**". Estos caracteres a los que se asigna significado especial se denominan "metacaracteres".

El conjunto de metacaracteres para expresiones regulares es el siguiente:

\ ^ $. [ ] { } | ( ) * + ?

Estos caracteres, en una expresión regular, son interpretados en su significado especial y no como los caracteres que normalmente representan. Una búsqueda que implique alguno de estos caracteres obligará a utilizar el barra de escape \, como se hace para evitar la interpretación por el shell de los metacaracteres del shell. En una expresión regular, el carácter ? representa "un carácter cualquiera"; si escribimos \?, estamos representando el carácter ? tal cual, sin significado adicional.

2.1. Expresiones Regulares Básicas.

Una expresión regular determina un conjunto de cadenas de caracteres. Un miembro de este conjunto de cadenas se dice que satisface la expresión regular.

Expresiones Regulares de un sólo carácter.

Las expresiones regulares se componen de un conjunto de expresiones regulares elementales que se satisfacen con un único carácter:

Expresión Regular	Representa
c	Expresión regular que enlaza con el **carácter ordinario c**
.	(punto) expresión regular que representa un **carácter cualquiera excepto nueva línea <eol>**
[abc]	Expresión regular que representa el carácter **a**, **b** o **c**
[^abc]	Expresión regular que representa un carácter que **no** sea **a**, **b** o **c**
[0-9][a-z][A-Z]	Expresiones regulares de un carácter que representan un carácter entre el 0 y el 9, u carácter entre la a y la z y un carácter entre la A y la Z. El signo - indica un intervalo de caracteres consecutivos.
\e	Expresión regular que sustituye alguno de estos caracteres (en lugar de la e): **. * [** cuando no están dentro de [ ] **^** al principio de la ER, o al principio dentro de [ ] **$** al final de una ER **/** usado para delimitar una ER

Los corchetes **[ ]** delimitan listas de caracteres individuales. Muchos metacaracteres pierden su significado si están dentro de listas: los caracteres especiales . *** [** valen por sí dentro de [ ]. Para incluir **un carácter]** en una lista, colocarlo al principio; para incluir un ^ colocarlo en cualquier lugar menos al principio; para incluir un - colocarlo al final.

Dentro de los conjuntos de caracteres individuales, se reconocen las siguientes categorías:

[:alnum:]	alfanuméricos
[:alpha:]	alfabéticos
[:cntrl:]	de control
[:digit:]	dígitos
[:graph:]	gráficos
[:lower:]	minúsculas
[:print:]	imprimibles
[:punct:]	de puntuación
[:space:]	espacios
[:upper:]	mayúsculas
[:xdigit:]	dígitos hexadecimales

Por ejemplo, [[:alnum:]] significa [0-9A-Za-z], pero esta última expresión depende de la secuencia de codificación ASCII, en cambio la primera es portable, no pierde su significado bajo distintas codificaciones. En los nombres de categorías, los corchetes forman parte del nombre de la categoría, no pueden ser omitidos.

Construcción de Expresiones Regulares.

Una Expresión Regular se construye con uno o más operadores que indican, cada uno, el carácter a buscar. Los operadores más comunes y aceptados son los siguientes:

Operador	**Significado**
c	un carácter no especial concuerda consigo mismo
\c	elimina significado especial de un carácter c; el \ escapa el significado especial
^	Representa el comienzo de la cadena (cadena nula al principio de línea). Por ejemplo ^[a-z] representa todas los párrafos que comiencen en minúscula. Si va dentro de los corchetes, sirve para negar.
$	Representa el final de la cadena (cadena nula al final de línea)
.	(punto) un carácter individual cualquiera
[...]	uno cualquiera de los caracteres ...; acepta intervalos del tipo a-z, 0-9, A-Z (lista)
[^...]	un carácter distinto de ... ; acepta intervalos del tipo a-z, 0-9, A-Z
r*	0, 1 o más ocurrencias de la expresión regular r (repetición)
r1r2	la expresión regular r1 seguida de la expresión regular r2 (concatenación)

Ejemplos de Expresiones Regulares Básicas.

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

2.2. Expresiones Regulares Extendidas.

Algunos comandos, como **egrep** o **grep -E**, aceptan **Expresiones Regulares Extendidas**, que comprenden las Expresiones Regulares Básicas más algunos operadores que permiten construcciones más complejas. Los operadores incorporados son los siguientes:

Operador	**Significado**
r+	1 o más ocurrencias de la ER r
r?	0 o una ocurrencia de la ER r, y no más
r{n}	n ocurrencias de la ER r
r{n,}	n o más ocurrencias de la ER r
r{,m}	0 o a lo sumo m ocurrencias de la ER r
r{n,m}	n o más ocurrencias de la ER r, pero a lo sumo m
r1|r2	la ER r1 o la ER r2 (alternativa)
(r)	ER anidada
"r"	evita que los caracteres de la ER r sean interpretados por el shell

La repetición tiene precedencia sobre la concatenación; la concatenación tiene precedencia sobre la alternativa. Una expresión puede encerrarse entre paréntesis para ser evaluada primero.

Ejemplos de Expresiones Regulares Extendidas:

ER Extendidas	Representa
[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 ...

Ejemplos de expresiones regulares

*****	Representa el carácter *
.	Representa cualquier carácter excepto el <eol>
[a-f]	Representa un carácter cualquiera entre la a y la f
[A-Z]	Cualquier letra mayúscula
[^a-d]	Cualquier carácter que no sea una letra entra la a y la d
[a-z]*	cualquier palabra escrita con minúsculas
[a-z][A-Z]	cualquier palabra de dos letras, de las cuales la primera el minúscula y la segunda es mayúscula
^\(.*\)\1\1	Al comienzo de la línea, un campo formado por un carácter cualquiera que se repite las veces que sea, volviendo a aparecer dos veces más antes de que acabe la línea.
^[a-z]*	selecciona las líneas que sólo contengan letras minúsculas
^[a-z]\{3\}[^a-z]\{3\}	selecciona las líneas que comienzan con tres minúsculas, terminan con tres caracteres cualesquiera pero que no son minúsculas, y no tiene ningún otro carácter entre medio

3. Filtros

Se da el nombre de filtros a un grupo de comandos que leen alguna entrada, realizan una transformación y escriben una salida. Además de los que veremos aquí, incluye comandos tales como **head**, **tail**, **wc** y **cut**.

3.1. Comando sort

La comparación u ordenación puede ser por caracteres ASCII o por valor numérico. La ordenación ASCII es la más parecida a la alfabética; sigue el orden del juego de caracteres ASCII. En este ordenamiento, los caracteres idiomáticos (vocales acentuadas, ñ) no se encuentran en el orden alfabético tradicional. En la ordenación numérica se respeta la ordenación por valor numérico de la cadena de caracteres: 101 va después de 21; en ordenamiento ASCII sería al revés.

sort arch1	ordena según el código ASCII.
sort -n arch2.num	ordena numéricamente.

Si no se indican campos de ordenación, la comparación se hace sobre toda la línea. Si se indican campos, la comparación se hace considerando la cadena de caracteres iniciada en el primer carácter del primer campo hasta el último carácter del último campo.

sort -t: -k1,3 f1.txt	Ordena por campos separados por ":", tomando en cuenta para la comparación los caracteres desde el primero del campo 1 hasta el último del campo 3.
sort -t: -k1.3,3.5 f1.txt	ordena por campos tomando en cuenta desde el 3er. carácter del campo 1 hasta el 5to. Carácter del campo 3.
sort -nr f2.num	ordena en orden numérico descendente.
sort -k3 f3.txt	ordena alfabéticamente, usando como cadena de comparación la comprendida desde el primer carácter del 3er. campo hasta el fin de línea. Como no se indica separador, los campos se definen por blancos (espacio o tabulador).

3.2. Comando find

El comando **find explora** una **rama** de **directorios buscando archivos** que cumplan determinados criterios. El comando find en GNU es extremadamente potente, permitiendo criterios de búsqueda tales como:

- el nombre contiene cierta cadena de caracteres o aparea con algún patrón,
- son enlaces a ciertos archivos;
- fueron usados por última vez en un cierto período de tiempo;
- tienen un tamaño comprendido dentro de cierto intervalo;
- son de cierto tipo (regular, directorio, enlace simbólico, etc.);
- pertenecen a cierto usuario o grupo;
- tienen ciertos permisos de acceso;
- contienen texto que aparea con cierto patrón.

Una vez ubicados los archivos, find puede realizar diversas acciones sobre ellos:

- ver o editar;
- guardar sus nombres en otro archivo;
- eliminarlos o renombrarlos;
- cambiar sus permisos de acceso;
- clasificarlos por grupos.

find /var -name *.log –print	busca en el directorio /var los archivos terminados en .log, imprime sus nombres en la salida.
find /tmp -size +200k –print	busca archivos mayores de 200k. En los argumentos numéricos, +N es mayor que N, -N es menor que N, N es exactamente igual a N.
find /var/spool/mail -atime +30 –print	busca archivos no accedidos hace más de 30 días. La opción -atime se refiere a tiempo transcurrido desde última lectura, -mtime desde última modificación de estado o permisos, -ctime de contenido.
find /var/tmp -empty -exec rm {} \;	busca archivos vacíos y los borra.
find /home -nouser –ls	busca archivos en los cuales en lugar del nombre de usuario dueño aparece un número (UID). Esta situación se da cuando la cuenta de usuario ha sido borrada pero han permanecido los archivos creados por ese usuario.

3.3. Comando grep

El comando **grep** (**Global Regular Expression** and **Print**) permite buscar las líneas que contienen una cadena de caracteres especificada mediante una expresión regular.

Este comando rastrea el fichero por fichero imprimiendo aquellas líneas que contienen el conjunto de caracteres buscado. Si el conjunto de caracteres a buscar está compuesto por dos o más palabras separadas por un espacio, se colocará el conjunto de caracteres entre apóstrofes (‘).

La sintaxis es:

grep patrón fichero1 fichero2 …

donde el patrón a buscar es una expresión regular a buscar y fichero1, fichero2 … son el conjunto de conjunto de archivos donde se va a buscar.

De manera general, la el formato de grep es:

grep [-opcion] expresión_regular [referencia]

Las opciones principales son:

c	la único que hace es escribir el número de las líneas que satisfacen la condición
i	no se distingue mayúsculas y minúsculas
l	se escriben los nombre de los ficheros que contienen las líneas buscadas
n	cada línea es precedida por su número en el fichero
s	no se vuelcan los mensajes que indican que un fichero no se puede abrir
v	se muestran sólo las líneas que no satisfacen el criterio de selección

Veamos a continuación una serie de ejemplos:

Suponiendo que tenemos un *fichero de nombre días*, donde cada línea contiene cada día de la semana, la ejecución del comando grep sobre ese archivo daría las siguientes salidas.

grep martes días muestra la línea donde está martes

grep tes días muestra las líneas del archivo dias que contienen la cadena "tes".

Otros ejemplos de ejecución son:

grep unix01 /etc/passwd

grep unix0[1-9] /etc/passwd

ls -l /usr | grep '^d'	lista sólo los subdirectorios del directorio /usr cuyas líneas empiecen con"d".
ls -l / | grep '.....rw'	lista sólo los archivos que otros pueden leer y escribir en el directorio principal.
grep '^[^:]*::' /etc/passwd	busca usuarios sin contraseña; caracteres al principio de línea que no sean ":", y luego "::" (el segundo lugar, que es el de la contraseña, está vacío).
grep '^[^:]*:*:' /etc/passwd	busca usuarios que no pueden entrar al sistema; tienen un * en el lugar de la contraseña; \ escapa el significado del segundo *, que vale como carácter a buscar.
grep ‘^d’ text	busca las líneas que empiecen por d en el fichero text.
grep ‘^[^d]’ text	busca las líneas que no comienzan por d en el fichero text
grep –v ‘^C’ fich1 > fich2	quita las líneas de fich1 que comienzan por C y lo copia en fich2

3.4. Comandos fgrep y egrep

Hay dos versiones de grep que optimizan la búsqueda en casos particulares:

fgrep (**fixed grep**, o **fast grep**) acepta solamente una cadena de caracteres, y no una expresión regular, aunque permite buscar varias de estas cadenas simultáneamente.

egrep (**extended grep**), que acepta expresiones regulares extendidas con los operadores + ? | y paréntesis.

El comando **fgrep no interpreta metacaracteres**, pero puede buscar muy eficientemente muchas palabras en paralelo, por lo que se usa mucho en búsquedas bibliográficas; **egrep** acepta **expresiones más complejas**, pero es más lento; grep es un buen compromiso entre ambos.

Por ejemplo:

fgrep martes días busca la cadena martes en el archivo dias.

En **fgrep** y **egrep** puede indicarse la opción **-f buscar.exp**, donde *buscar.exp* es un archivo que contiene la expresión a buscar: cadenas simples para fgrep, expresiones regulares para egrep, separadas por nueva línea; las expresiones se buscan en paralelo, es decir que la salida serán todas las líneas que contengan una cualquiera de las expresiones a buscar.

Para comprobar la funcionalidad de fgrep, vamos a crear un archivo buscar.fgrep que contenga las cadenas "*tes*" y "*jue*", una por línea.

Ejemplo;

fgrep -f buscar.fgrep días extrae del archivo dias las líneas que contienen estas cadenas.

El comando grep soporta **fgrep** y **egrep** como opciones **-F** y **-E**, respectivamente.

- La opción **–E** hace que se interprete el patrón como un expresión regular extendida.
- Las opción **–F** hace que se interprete el patrón como una lista de cadenas fijas, separadas por retornos de carro (nueva línea).
- La opción **–f** obtiene los patrones de búsqueda del fichero que se le pasa con expresiones regulares.

Por ejemplo, las tres ejecuciones siguientes, obtienen el mismo efecto que el comando indicado anteriormente.

grep -F -f buscar.fgrep dias

egrep "tes|jue" dias

grep -E "tes|jue" días

La ejecución de egrep siguiente, busca cadenas comenzadas opcionalmente por un dígito y los caracteres ab, todo el paréntesis 0 o más veces, y hasta encontrar la cadena 1234.

egrep "([0-9]+ab)*1234" archivo

Escribir **grep -E** es similar a **egrep**, aunque no idéntico; egrep es compatible con el comando histórico egrep; grep -E acepta expresiones regulares extendidas y es la versión moderna del comando en GNU. fgrep es idéntico a grep -F.

3.5. Comando tr

El comando tr dado un flujo de datos (entrada estándar) nos permite modificarlos sustituyendo y/o borrando caracteres. La sustitución es carácter a carácter.

La sintaxis de este comando:

tr [opción] .. SET1 [SET2]

donde opción puede ser:

-d	Borra los caracteres indicados en SET1
-s	Elimina o reemplaza los caracteres repetidos indicados en SET1
-c	Todos los caracteres que no sean los indicados en SET1 y los convierte en SET2
-t	Trunca SET1 a la longitude de SET2

Sustitución

En su forma más simple, podemos sustituir un carácter por otro, por ejemplo, podemos transformar todas las vocales en mayúsculas.

$ echo murcielago | tr aeiou AEIOU

Daría como resultado ***mUrcIElAgO***

En esta ocasión hemos ejecutado el comando tr sin ningún argumento, 'aeiou' seria SET1 y 'AEIOU' correspondería a SET2.

La sustitución se realiza a pares, el primer carácter de SET1 será reemplazado por el primer carácter de SET2, el segundo con el segundo y así hasta el final.

Si no tiene la mism longituda SET1 y SET2, la salida cambia.

$ echo murcielago | tr aeiou AE

mErcEElAgE

$ echo murcielago | tr aeiou AE.

*m.**rc**.ElAg.*

$ echo murcielago | tr ae AEIOU

murciElAgo

Borrar

El borrado de caracteres es más fácil de usar. En este caso solo necesitamos SET1, para indicar que caracteres queremos borrar.

En el siguiente ejemplo borraremos las letras 'a' y 'e' de la palabra 'murcielago'.

$ echo murcielago | tr -d ae

Daría como resultado ***murcilgo***

Sustituir con negados

Si utilizamos la opción -c, el comando localizará todos los caracteres que no coincidan con el patrón dado en SET1 y los reemplazara por sus correspondientes en SET2.

$ echo 'naci el 13-12-1986' | tr -c '0123456789' '-'

--------13-12-1986-

El comando **tr** ha substituido por un guion cualquier carácter que no fuera un número. Pues eso ha hecho, el salto de línea aunque invisible también es tenido en cuenta como un carácter.

Secuencias válidas

El comando reconoce los siguientes caracteres no visibles.

\NNN	carácter con valor octal NNN (de uno a tres dígitos)
****	barra invertida
\a	pitido audible (BELL)
\b	espacio hacia atrás

\f	salto de página
\n	salto de línea
\r	retorno de carro
\t	tabulación horizontal
\v	tabulación vertical

El comando **ls** sin argumentos, nos devuelve una lista tabulada por columnas con el nombre de los archivos de un directorio determinado, con tr podemos modificar la salida del comando ls para obtener una lista con un elemento por línea.

$ ls |tr '\t' '\n'

```
gedit.banyut.1053685335
kde-banyutyHf1n2
keyring-GLhk1g
ksocket-banyut
orbit-banyut
orbit-root
plugtmp
pulse-banyut
screenlets
seahorse-7f0yeD
Tracker-banyut.8264
virtual-banyut.F7zFmY
```

Hemos substituido los tabuladores por saltos de línea, también podríamos cambiar las barras de un path.

$ echo '\banyut\tmp\' |tr '\\' '/'

/banyut/tmp/

Ejemplos de uso de tr

cat dias | tr a-z A-Z	convierte todo a mayúsculas.
cat dias | tr -d aeiou	borra todas las vocales del *archivo dias*.

Agregar al archivo *dias* líneas en blanco, varias seguidas, intercaladas entre los nombres de los días.

cat dias

cat dias | tr -s "\n*"	convierte varios caracteres nueva línea seguidos en una solo; elimina renglones en blanco.
cat nota | tr -c "[a-zA-Z0-9]" "_"	transforma todos los caracteres que no sean letras o números en subrayas.
cat nota | tr -cs "[a-zA-Z0-9]" "[\n*]"	transforma todos los caracteres que no sean letras o números en nueva línea, y comprime las nueva líneas repetidas en una sola; deja cada palabra sola en un renglón.
ls -l /usr | tr -s " " | cut -f3,4	comprime los blancos en la salida para poder cortar campos.

3.6. Comando uniq

El comando **uniq** excluye todos los renglones adyacentes duplicados menos uno; es decir, elimina renglones repetidos consecutivos. La sintaxis es la siguiente:

uniq [OPCION]... [ENTRADA [SALIDA]]

Este comando muestra una única línea para una entrada ordenada, eliminando líneas repetidas sucesivas. Opcionalmente, puede mostrar solo líneas que aparecen una vez, o sólo líneas que aparecen varias veces. La entrada debe estar ya ordenada; si no lo está, se puede usar sort -u para lograr un efecto similar.

Las opciones de uniq son:

-f N	salta N campos antes de verificar unicidad.
-s N	salta N caracteres antes de verificar unicidad.
-c	indicar junto a cada línea el número de veces que aparece.
-i	ignorar mayúsculas y minúsculas al comparar.
-d	mostrar sólo líneas repetidas. -u mostrar sólo líneas únicas.

El siguiente ejemplo muestra las 5 palabras más frecuentes en el conjunto de archivos:

cat *.txt | tr -sc A-Za-z '\012' | \ sort | uniq -c | sort -n | tail 5

cat lista todos los archivos, **tr** comprime el renglón eliminando blancos, **sort** los ordena, **uniq** cuenta las ocurrencias y elimina repetidos, **sort -n** ordena por cantidad de ocurrencias y **tail** muestra las 5 ocurrencias más frecuentes.

3.7. Comando dd

El comando **dd** es un convertidor de datos: convierte de ASCII a EBCDIC y a la inversa, cambia los tamaños de los bloques de registros, hace ajuste de blancos y otras transformaciones usuales cuando se manejan transferencias entre sistemas operativos distintos o datos en bruto, sin formato, como en los respaldos en cinta.

La sintaxis es:

dd OPCION ...

Su función es copiar un archivo, de entrada estándar a salida estándar por defecto, opcionalmente cambiando tamaño de bloques de entrada salida y realizando diversas conversiones.

if=ARCHIVO	leer la entrada del archivo indicado.
of=ARCHIVO	dirigir la salida al archivo indicado.
ibs=BYTES	leer la entrada en bloques de BYTES bytes.
obs=BYTES	grabar la salida en bloques de BYTES bytes.
bs=BYTES	leer y grabar entrada y salida en bloques.
conv=CONVERSION[,CONVERSION]...	convertir según argumentos.

Las opciones numéricas admiten los multiplicadores b para 512, k para 1024.

Los argumentos de conversión se separan por comas sin espacios; incluyen:

ascii	convierte EBCDIC a ASCII.
ebcdic	convierte ASCII a EBCDIC.
ibm	convierte ASCII a EBCDIC alternativo.
block embloca	cada línea de entrada en 'cbs' bytes, reemplaza nueva línea por espacio, rellena con espacios.
unblock	reemplaza espacios finales de cada bloque con nueva línea.
lcase	convierte mayúsculas a minúsculas ucase convierte minúsculas a mayúsculas.
notrunc	no truncar el archivo de salida.

3.8. Comando cut

El comando cut se utiliza para poder cortar caracteres y campos, con la posibilidad de usar delimitadores y otras opciones, para extraer las partes seleccionadas de cada fichero en la salida estándar. La sintaxis de este comando es la siguiente:

cut opción … [fichero] …

Las opciones más importantes son:

-b	–bytes=LISTA muestra solamente estos bytes
-c	–characters=LISTA selecciona solamente estos caracteres
-d	–delimiter=DELIM usa DELIM en vez de caracteres de tabulación para delimitar los campos.
-f	–fields=LISTA selecciona solamente estos campos; también muestra cualquier línea que no tenga un carácter delimitador, a menos que se especifique la opción –s
–complement	complementa el conjunto de bytes, caracteres o campos seleccionados.
-s	–only-delimited no muestra las líneas que no contienen delimitadores

Por ejemplo, suponiendo que se tiene la siguiente línea de texto, y solamente se quiere la 3ª y la 6º línea:

foo,bar,baz,stuff,blah,oogabooga

Se podría probar el comando cut:

cut -d',' -f 3,6 y el resultado sería: baz,oogabooga

3.9. Comando wc

El nombre del comando wc proviene de word count, y como es de suponer, sirve para contar palabras. Pero no sólo palabras como se verá a continuación. Su sintaxis es como sigue:

wc [opción] [archivo …]

Si se omite el argumento archivo, wc tomará los datos (naturalmente) de la entrada estándar.

La lista de opciones más importantes es la siguiente:

-c Contar bytes.

-l Contar líneas.

-w Contar palabras.

Como ejemplo, se pueden contar las líneas del archivo **/etc/passwd** y de esta manera se sabrán rápidamente cuántos usuarios tiene definidos el sistema:

usuario@maquina:~/$ wc -l /etc/passwd

32 /etc/passwd

4. Comando id

El comando id muestra el ID real y efectivo del usuario y del grupo. La sintaxis del comando es la siguiente:

id [OPCION]... [USUARIO]

La salida del comando imprime información sobre el usuario y el grupo para el USUARIO especificado, si se omite el nombre del usuario, se muestra el resultado para el usuario actual.

Las opciones más importantes son las siguientes:

-g, --group	muestra solo el ID efectivo del grupo
-G, --groups	muestra todos los ID de todos los grupos
-u, --user	muestra solo el ID efectivo del usuario

Relacionado con este comando nos encontramos las variables de entorno:

$UID

que indica el número de usuario. Hay que recordar que el usuario **root** tiene un **UID**, **GID** con valor **0**.

5. Comando stat

El comando stat muestra información sobre el estado de un fichero o del sistema de ficheros.

Su sintaxis es la siguiente:

stat [OPCION]... FICHERO...

-f, --file-system	muestra el estado del fichero.
-c --format=FORMAT	utiliza el formato indicado, en su caso el formato por defecto.

Las secuencias de formato valido son las siguientes:

%a	Derechos de acceso en octal
%A	Derecho de acceso al fichero.
%d	Número de dispositivo en decimal
%F	Tipo de archivo
%g	ID del grupo de propietario
%G	Nombre del grupo del propietario
%n	Nombre del fichero
%s	Tamaño total en bytes
%u	ID del usuario propietario
%U	Nombre de usuario del propietario
%x	Tiempo del último acceso
%y	Tiempo de la última modificación

6. Comando let

La sintaxis del comando **let** es la siguiente

let arg [args …]

Este comando nos permite evaluar cada uno de sus argumentos como expresiones aritméticas. La evaluación la hace con valores enteros de longitud fija.

Los operadores, en orden de prioridad que admite, son los siguientes:

id++, id--	post-incremento, post-decremento de variable
++id, --id	pre-incremento, pre-decremento de variable
-, +	menos, más unario
!, ~	negación lógica y basada en bits
**	exponenciación
***, /, %**	multiplicación, división, residuo
+, -	suma, resta
<<, >>	desplazamientos de bits izquierdo y derecho
<=, >=, <, >	comparación
==, !=	equivalencia, inequivalencia
&	AND de bits
^	XOR de bits
|	OR de bits
&&	AND lógico
||	OR lógico
expr ? expr : expr	operador condicional
=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=	asignación

7. Variable IFS

Esta variable define el carácter separador que se utiliza para separar campos dentro de un fichero. Por ejemplo IFS es ":" dentro del fichero /etc/passwd, ya que este carácter es el que se utiliza para separar los diferentes campos de cada fila. Esta variable puede modificar igual que cualquier otra.

Podemos acceder a su contenido con echo $IFS, y podemos cambiar el valor separador por defecto de la siguiente forma:

antIFS=$IFS

IFS="-"

De forma que en la variable antIFS mantenemos el separador predeterminado, y puntualmente definimos "-" como nuevo separador.

8. Declaración de variables.

Cuando declara una variable dentro de un shell-script, por ejemplo si introducido la siguiente línea de código

X=343

se está creando una variable de nombre X que almacena el valor 343. Si queremos mostrar su contenido podemos ejecutar

echo $X

mostrándose 343

Esta variable es una variable global al shell, pero local a él. Es decir, la variable X puede ser accedida a través de todo el shell, pudiendo comprobar y modificar su contenido. Esta variable es, sin embargo global al shell donde se utiliza. Si queremos que sea accesible por otros Shell que queramos utilizar, podemos convertirla en global utilizando el comando **export**

export X=343

de esta forma, la variable X puede ser accedida y modificado desde el entorno BASH.

Si nos interesa definir variables locales a una función, se utiliza el comando local de la siguiente forma

```
function mi_funcion ()
{
  local variable_local=34
  echo $variable_local
}
```

La variable variable_local tiene como ámbito solo la función mi_funcion, no pudiendo ser accesible desde fuera de la función donde se declara y utiliza.