Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Tutorial de Expresiones Regulares (84888 lectures)

Per **Daniel Rodriguez**, ***DaniRC*** (http://www.ibiza-beach.com/)

Creat el 30/07/2001 12:25 modificat el 15/02/2002 12:45

Había escrito otro articulo sobre los conceptos avanzados de las Expresiones Regulares[1] Pero seguía faltando un tutorial que las acercara un poco más a todos nosotros.
SUPER Actualizado: Mas de 6 páginas nuevas, mas de 50 ejemplos nuevos, tutoriales completos de GREP, SED, AWT, y mucho más! Todo gracias a la colaboración especial de un lector.
Nota: Todos los textos usados en el articulo son propiedad de sus respectivos autores y este artículo unicamente pretende ser una via para su divulgación.

Tutorial de Expresiones Regulares

No hace mucho publique un artículo sobre **la parte complicada de las expresiones regulares**, pero no caí en la cuenta de que no habia publicado nadie en Bulma un **tutorial para no iniciados**. Así que me he puesto manos a la obra. Esta vez no es una traducción, pero poco le falta, despues de todo no es que exista mucha documentación en castellano al respecto, ¿verdad?.

Introducción

Las *expresiones regulares vienen a ser una forma sofisticada de hacer un bucar&reemplazar*. En el mundo windows no tienen mucho sentido, despues de todo alli casi todo va a base de clicks. Pero en el mundo Unix/Linux, en el que casi todo son ficheros de texto, son casi una **herramienta imprescindible**. No tan solo de cara al administrador, sino tambien de cara a cualquier otro programador que puede ver como las expresiones regulares le salvan la vida en mas de una ocasión.

Particularmente llevo un tiempecillo dedicado a la programación y diseño web. No es de extrañar que un cliente que insistió en que su e-mail estuviera en cada página con un mailto: cambie de mail. Tampoco es de extrañar que hallas escrito una palabra mal 2 veces en 50 páginas distintas ... Y tampoco es especialmente raro que tengas que ir con cuidado para que la informacion que sacas no esté en el tag html adecuado. Por ejemplo ... hay clientes muy raritos que quieren que todo lo que hasta ahora era cursiva ... se vuelve negrita, pero solo si el contenido de la frase usa la palabra "clave".

¿**Os imaginais este problema en MSWord**? Busca una *frase que contenga la palabra "clave"*, ahora *mira si esta entre tag's de cursiva * y ahora *reemplaza por *

Pues para estas cosas se inventaron las expresiones regulares ;)

Mientras espero con vosotros ese dia en que los clientes no me compliquen la vida... es un consuelo saber que existen las **¡EXPRESIONES REGULARES!**

Nota: Todo lo que explico esta basado en las expresiones regulares de **PERL**. **Sed** por ejemplo no tiene porque funcionar exactamente igual. Pero la idea es basicamente la misma.

Presentando los carácteres especiales

```
[ ] cochetes
() parentesis
{} llaves
- guión
+ más
* asterisco
. Punto
^ circumflejo
$ dolar
? interrogante cerrado
| tuberia unix
\ barra invertida
     (se usa para tratar de forma normal un caracter especial)
/ barra del 7
```

Mención aparte para / puesto que es el simbolo que se usa para indicar la búsqueda. El resto son todo modificadores y se pueden usar sin restricciones.

Definiendo Rangos

```
/[a-z]/ letras minusculas
/[A-Z]/ letras mayusculas
/[0-9]/ numeros
/[,'¿!¡;:\.\?]/ caracteres de puntuacion
                -la barra invertida hace que
                no se consideren como comando
                ni en punto ni el interrogante
/[A-Za-z]/      letras del alfabeto (del ingles claro ;)
/[A-Za-z0-9]/   todos los caracteres alfanumericos habituales
                -sin los de puntuacion, claro-
/[^a-z]/        El simbolo ^ es el de negación. Esto es decir
                TODO MENOS las letras minusculas.
/[^0-9]/        Todo menos los numeros.
```

Para definir otros rangos, no dudeis en usar el operador de rangos "-" por ejemplo de la h a la m [h-m] ¿vale?

Coincidencia total

La expresion regular suele funcionar con que solo coincida un caracter de los muchos listados. Pero en ocasiones queremos que la coincidencia sea total.

Imaginemos que nos hemos dejado todas las tildes de palabras acabadas en ion. Se nos ocurre usar /[ion]/ como patron de busqueda.

Pero esto daria concordancia positiva con Sol -por la o- con noche -por la n- ... etc.

Para que la concordancia sea de cada caracter y en el orden adecuado, necesitamos el operador Punto "."
Por ejemplo [.ion] solo concordaria con cosas como camion o salsa lionesa

Si bueno, la "ion" de l**ion**esa no esta a final de palabra ... y eso tambien hemos de retocarlo, pero todo a su tiempo.

Aunque se os podria ocurrir que algo del tipo [.ion[^a-zA-Z]] tal vez funcionase ... ya veremos ;)

Por cierto que el punto "." no concuerda con \n \r \f \0 (newline, return, line feed, null respectivamente)

Agrupando, referenciando y extrayendo

Agrupación

En ocasiones nos interesa usar el resultado de una parte del patrón en otro punto del patrón de concordancia.

Tomemos por ejemplo una agenda en la que alguien puso su telefono en medio de su nombre.(Pepe 978878787 Gonzalez) Queremos extraer el telefono de esa frase y guardarlo para introducirlo en otro sitio. O para usarlo como patron de busqueda para otra cosa.

Lo primero que hay que hacer es agrupar. Agrupar permite que el resultado del patron se almacene en una especie de registro para su uso posterior. El **operador de agrupamiento son los parentesis ( )**

Los registros se llaman 1, 2, 3, ... 9 segun el orden del agrupamiento en el patrón.

```
Ejemplo (del libro de perl): /Esto es ([0-9]) prueba/
si el texto fuente es: Esto es 1 prueba.
El valor 1 seria almacenado en el registro $1
```

Referencias

Tenemos una serie de elementos agrupados, a los cuales se les ha asignado un valor. Ahora queremos usar ese valor en algun otro punto del patron de concordancia. Pues para eso existen las referencias.

\1 representa la referencia al grupo 1

```
Ejemplo (del libro de perl): /([0-9]) \1 ([0-9])/
si el texto es : 3 3 5
El resultado es: $1 --primer grupo-- vale 3
\1 --hace referencia al resultado del primer grupo-- vale 3
$2 --segundo grupo-- vale 5
```

En este caso hubiese habido concordancia, no asi si el texto hubiese sido 3 5 3

Extracción

La extracción es simplemente usar el **$**1 en los mensajes que se generan despues de haber usado la expresion regular.

Expresiones opcionales y alternativas.

Si os habeis fijado hasta ahora todo lo que hemos visto han sido condiciones mas bien del tipo AND (Y)

Si hay un numero Y hay un texto Y no hay otro numero ... blablabla. Siempre Y, es evidente que nos faltan las condiciones de tipo opcional y alternativo.

Opcionales

el simbolo que se emplea para indicar que una parte del patron es **opcional es el interrogante. ?**

```
/[0-9]? Ejemplo/
Esto concuerda tanto con textos del tipo
1 ejemplo, como con
ejemplo
```

Alternativas

No confundamos alternativas con opciones, la alternativa es el equivalente a la OR no lo era asi el operador opcional. Veamos la diferencia. Por cierto, el simbolo de la **alternativa es |**

```
/[0-9]? (EJEMPLO|ejemplo)/
a: 1 ejemplo
b: 1 EJEMPLO
```

```
c: 1
d: EJEMPLO
```

a,b,d concuerdan con el patron. Sin embargo c no concuerda con el patrón! Porque ejemplo o EJEMPLO son alternavitas válidas, pero tiene que haber uno de los 2. No ocurre lo mismo con la opcionalidad, el 1 puede estar o no estar. Esa es la principal diferencia.

Contando expresiones

A estas alturas te estas preguntando qué mas se puede hacer con una expresión. Pues la verdad es que ya queda poca cosa, pero lo de contarlas! vamos ... eso es imprescindible.

El operador para esta ocasión son las llaves {m,n}

Donde m, n son 2 enteros positivos con n mayor o igual que m. Esto se lee asi ... la expresion debe cumplirse al menos m veces y no mas de n veces.

Podemos usar {n}o bien {n,} para indicar que queremos que se repita exactamente n veces o que se repita n veces o más respectivamente.

```
De tal forma que el patrón /go{1,4}l/
a: gool
b: gooooooooool
c: gooool
concordaria con a y c pero no asi con c
```

Expresiones repetidas

Tenemos el **operador * asterisco** que representa que el patrón indicado debe aparecer **0 o mas veces** y el **operador + suma** que representa que el patrón indicado **debe aparecer 1 o más veces (pero almenos 1)**

Mucho ojo con el operador asterisco!!! leeros el articulo relacionado sobre conceptos avanzados de las expresiones regulares antes de usarlo o no me hago responsable de los resultados.

Comodines y abreviaturas

- \w para indicar word (alfanumericos y _)
- \W para indicar lo opuesto al word.
- \s para concordar con los caracteres los espacios y otros caracteres en blanco (\t \n \r y espacio)
- \S para lo contrario a \s
- \d para concordar con un digito
- \D para lo contrario al \d
- \A para empezar a mirar por el principio del string
- \Z para empezar a mirar por el final de string
- \b concuerda con las "word boundaries" los limites de palabra
- \B lo opuesto a \b

El operador ^ dice al compilador que empiece por el principio de la linea y $ indica que empiece por el final de la linea. Estos operadores cumplen este efecto cuando estan fuera de la expresion regular, evidentemente. Porque dentro de las barras tienen otras funciones ya comentadas anteriormente.

Y con esto creo que acabamos el listado de nuestras herramientas de busqueda. Queda sin embargo pendiente de explicar lo referente al reemplazo, con lo que por fin daremos solución al problema que plantee en la primera página.

Buscar y Reemplazar

El manual dice lo siguiente

```
[m]/PATRON/PATRON REEMPLAZO/[g][i][m][o][s][x]

Donde mviene a ser el ambito de la busqueda.
^ inicio de linea. $ fin de linea.
% para todo el documento.
1,15 para las lineas de 1 a 15 ...

PATRON es el patron de busqueda del que hemos
estado hablando todo el rato.

PATRON REEMPLAZO es lo mismo que el patron de
busqueda solo que este se usa para el reemplazo

g (global) reemplaza TODAS las ocurrencias
en el texto.

i (insensitive) es para evitarnos problemas
con la capitalizacion. A = a

o (interpola variables solo una vez)
        No me mireis asi que yo tampoco se lo que es eso XD

m (multiples lineas) Acepta strings de varias lineas

s (single line) solo mira el string de una linea.

x (extensiones) permite usar extensiones de expresion regular.
Insisto, yo tampoco se como se usa eso XD
```

Ejemplos en varios entornos.

Tal como prometi algunos ejemplos prácticos. Lo primero, advertiros que es una parte que ire cambiando con el tiempo. He descubierto que no todas las expresiones regulares soportan la misma notación. Si bueno ... deberia haberlo sabido, pero no lo sabia. Asi que hasta que me ponga las pilas con el sed ... voy a tocar un poco el tema del **PERL**

Lo primero es lo primero:
Tenemos el PERL en nuestra maquina muerto de risa asi que vamos a cambiar eso ;−)

editad un fichero con el nombre "ejemplo.pl" −por ejemplo−
En la primera linea poneis:

```
!#/usr/bin/perl
```

Si el perl no esta ahi, buscadlo y poned el path correspondiente en general */usr/local/bin/perl*
Ahora podriamos hacer el tipico **print "Hello Wordl\n";** tras lo cual cerramos el fichero e intentamos ejecutarlo.
Necesitaremos hacer **chmod u+x ejemplo.pl** para dar permiso de ejecucion al usuario en curso sobre el fichero.
Ahora tecleando **./ejemplo1.pl** ... deberia saliros en pantalla "Hello World". Lo que indicaria que el script funciona. De **no ser asi** os sugiero que repaseis los pasos y que os asegureis de que el perl existe en el directorio que habeis pasado en el path del principio del fichero.

Ejemplo 1 en PERL

```
#!/usr/bin/perl5
$texto="hola clave buena";
if ($texto=~ /<i>([a-z\s]{1,})/) {
        $frase=$1;
```

```
        if ($1 =~ /clave/) {
                $texto = "$frase";
                print "$texto\n";
        }
}
```

Se trata de buscar trozos de texto en **italica**. Si el texto en italica contiene la palabra **clave** entonces se cambia la italica por la negrita. La italica se identifica por y por . Apuesto a que todo esto se pude escribir en una sola linea, pero iba a ser ilegible, asi que lo he separado en 2 trozos para fines didacticos.
Primero buscamos los tags de italica. Lo que quede por enmedio, es decir ... caracteres en minusculas y espacios en blanco hasta el fin del tag de la italica se almacenan en $1.
Segundo comprobamos que en lo que habia escrito entre las italicas esta efectivamente el texto "clave"
Tercero ... procedemos a la sustitución.

Bien ahora probad esto en vuestras casas e id jugando con estos nuevos conceptos, poco a poco iremos ampliando el muestrario. Acepto colaboraciones ;−) a ver si anima Guillem.

SED es un editor de textos no interactivo diseñado para funcionar en Unix y ayudar con los scripts.

Editando Texto

Sed actua como un filtro, y esta orientado a lineas. La forma que tiene de trabajar suele ser un patrón y unas ordenes que aplicar a una serie de lineas.

Patrones orientados a linea

Un ejemplo simple del funcionamiento del Sed es *"Borrar las 10 primeras lineas de un documento"*

```
sed −e '1,10d'
```

El −e ejecuta el comando. En este sentido es realmente muy parecido al vi. Por ejemplo, sabiendo que $ indica fin del documento. Si queremos borrar *todo menos las primeras 10 lineas* haremos:

```
sed −e '11,$d'
```

Lo que no puede hacerse es usar el sed con comandos relativos y complicados como por ejemplo:

```
sed −e '$−10,$d'
```

En este caso el sed no comprendera que intentamos hacer.

Otra forma de obtener las 10 primeras lineas es con el modificador −n y la p. La p para que imprima esa linea, y el −n si no recuerdo mal es para que no haga eco en pantalla.

```
sed −n −e '1,10p'
```

Usando mas de un comando a la vez

sed −e '1,4d 6,9d' o bien sed −e '1,4d' −e '6,9d'

Usando las expresiones regulares en los patrones

El ejemplo tipico es ese fichero larguisimo de log's en el que no sabes que lineas borrar, pero en que quieres *borrar todas las lineas que contengan la palabra debug*

sed −e '/debug/d' < log que es lo mismo que si hiceramos grep −v debug

Un ejemplo mas rebuscadillo

Ahora ya no nos basta con borrar las lineas que tienen debug, sino que queremos borrar todas las lineas de debug que no tengan la palabra foo en esa misma linea. Eso se suelve hacer asi:

```
grep 'foo' <log | grep −v debug
```

Pero podemos usar el sed para ello de esta forma:

```
sed −n −e '/debug/d' −e '/foo/p'
```

Ahora las lineas que tienen debug son impresas si tienen la palabra foo, asi que no se borran. Es una explicacion muy mala, pero bueno, estas cosas, nada como probarlas para acabar de entenderlas ¿vale?

Insertando texto

Se puede insertar texto con i y con a. El primero para insertar antes de la linea en curso y el segundo para insertar despues de la linea en curso. Por ejemplo 1i para insertar en la primera linea y $a para insertar al final del documento

```
sed −e '10i\ Texto a insertar'
```

Se puede reemplazar la linea en curso con la c

sed −e '10c\ Nuevo contenido de linea 10'

Sustitución de expresiones regulares

```
sed −e 's/patron/reemplazo/[modificadores]'
```

Notas Finales

Ante todo gracias al autor del minitutorial ingles en que me he basado −no recuerdo su nombre−, no soy un plagiador, solo que me cuesta mucho estructurar un tutorial, asi que le robo la estructura de contenidos a otra gente XD.

Venga, ahora a practicar y a leerse las man's

Ejemplos de un lector de Bulma:

El autor de estos fantásticos ejemplos dice que no sabe mucho y que quiere el anonimato XD ... pero si sabe mas que yo!! En fin, respetamos su voluntad pero publicamos los ejemplos. Espero que esto anime a mas gente a colaborar en este articulo :−)

lynx −source "http://www.servidor.com/pagina.jsp?cod_articulo=1&cantidad=10C307" |grep 'abcd' >> ZZ

Recupera el codigo de articulo en una lista del servidor (El primer registro en este caso). El nombre se encuentra en una línea que contiene "abcd", y se añade a un archivo sacándolo por grep

Lo mismo, pero con contador:

cuentatu=0
while [$cuentatu −lt 10]
do
cuentatu=`expr $cuentatu "+" 1`
lynx −source "http://www.servidor.com/pagina.jsp?cod_articulo=**$cuentatu**&cantidad=10C307" | grep 'zxy' | sed −e 's/ //g | sed −e 's/
/"";"/g | sed −e 's/email://g | sed −e 's/<[^>]*>//g >> ZZ
done

En este caso, se hace un contador para sacar los diez primeros datos, se incrementa de uno, se llama lynx para ver el código fuente (−source), se saca con grep la parte que contiene "zxy", que es donde están los datos que se buscan, seguidamente, se quitan los   con sed, y los
, la palabra "email:" y todos los demás tags html, para acabar en el archivo... (La variable está en negrita en el link) Siempre he usado while, pero es una cuestión de costumbres...

Montando un servidor php básico, es posible hacer muchas cosas, que sería larguísimo hacer on line, subiendo, bajando archivos por ftp, para una pequeña corrección.

Cosas así:

```
<?php
  for($i=1 ; $i<10 ; $i++)
  {
   $web = "http://www.server.com/asp/ficha.php?ref=$i";
   $archivo = "c:\\archiv~1\\www\\strip\\1\\$i.html";
   $abre = fopen( $web, "r" );
   $copia = fopen($archivo, "w");
   while (!feof($abre)) {
      $line = fgets($abre, 1024);
      fputs($copia,$line);
   }
   fclose($abre);
   fclose($copia);
    }
?>
<?php
// meter una página web en un array e imprimirlo
$fcontents = file ('c:\\archiv~1\\easyphp\\www\\strip\\1\\13857.html');
while (list ($line_num, $line) = each ($fcontents)) {
   echo "<b>Line $line_num:</b>; ", htmlspecialchars ($line), "<br>\n";
}
?>
```

Otros truquillos.

−Uso ls− alt | grep ^d para ver los directorios, no hay nada más rápido que yo sepa.
−cat correo | grep mailto Ese está claro

Extraido de www.ciberdroide.com/misc/novato/curso/regexp.html[2] Nos llega este otro ejemplo util para extraer cadenas determinadas de un texto.

```
echo "abc1234def" | sed "s/[0−9][0−9]*//"

abcdef
```

Otro uso de este mismo ejemplo sería el siguiente:
Tenemos unas cadenas feas de este tipo:
X Calle nosecuanto 28000Madrid Tel 91xxxxxx
en un archivo llamado sucio.txt

Queremos sacar el codigo postal en "limpio":
sed "s/[0−9][0−9][0−9][0−9][0−9]/\";\"\"/" < sucio.txt > limpio.txt
y el resultado será:
X Calle nosecuanto ";"28000";" Madrid Tel 91xxxxxx

Lo mismo puede hacerse con otros campos. Es la gracia del uso del operador &que vimos anteriormente.

y del extranjero nos llega:

Extraigo del LinuxPlanet.com esta PERLa de sabiduria ;−) ... es que lo mejor es tener estas cosas recopiladitas y en castellano en lugar de dispersas y en "gringo" ¿no?

Articulo de James Andrews[3]

http://www.linuxplanet.com/linuxplanet/tutorials/214/ [4]

Reemplazar una palabra por otra en un texto: Es un problema habitual y si no sabemos lo del −p −i nos volvemos locos porque lo unico que logramos es un eco del cambio ... o un cambio en un fichero temporal, pero no encima del propio fichero.

perl −print −inplace −execute

1. `perl −p −i −e 's/this/that/g' filename`

El filename puede ser cualquier cosa del tipo *.html o simplemente * para hacerlo sobre todo un directorio. Ese ejemplo reemplaza todas las apariciones de this por that

2. `perl −e 'for (@ARGV) { rename $_, lc($_) unless −e lc($_); }' *`

Renombra todos los ficheros en el directorio local a minusculas. Muy util cuando en nuestro servidor web los clientes se quejan de que sus paginas dejan de verse. Eso es debido a que W$ capitaliza los ficheros y luego ignora las capitalizaciones pero Linux no.

3. `perl −e 'for (@ARGV) { rename $_,$_.'l' unless −e lc($_); }' *`

`Añade una l a todas las extensiones de ficheros de un directorio, en plan .htm −> .html`

4.− Me la salto por usar librerias externas que no vienen a cuento ;−)

5. `perl −p −i −e 's/'`

Convierte ficheros tipo Unix a ficheros tipo DOS. Corrige los caracteres de escape y de fin de linea de nuestros ficheros de texto.

6,7,8,9. `Otro salto por no tener que ver con las expresiones regulares en si.`

10. `perl −l −e 'open(F,"/usr/dict/english"); $w=join("",sort split(//,$ARGV[0])); print grep {chop;join("",sort split(//,$_)) eq $w} ;' life`

Encuentra todos los anagramas de la palabra life.

Cosas del GREP y el FIND

$ which Despliega en qué directorio se encuentra un archivo.
Ejemplo: $ which ls

mount −t vfat /dev/fd0 /mnt/floppy
umount −t vfat /dev/fd0

Hacer ejecutable:
>chmod 755 script
o
>chmod a+x script

Ejecutarlo:
>./script

GREP
Usage: grep [OPTION]... PATTERN [FILE] ...
Search for PATTERN in each FILE or standard input.
Example: grep −i 'hello world' menu.h main.c

Pa sacar cosas:
grep 'http' < sucio.txt > limpio.txt
(Saca de sucio.txt todas las líneas −y solo ellas− que tengan la palabra http y las mete en limpio.txt)

Regexp selection and interpretation:
−E, −−extended−regexp PATTERN is an extended regular expression
−F, −−fixed−strings PATTERN is a set of newline−separated strings
−G, −−basic−regexp PATTERN is a basic regular expression
−e, −−regexp=PATTERN use PATTERN as a regular expression
−f, −−file=FILE obtain PATTERN from FILE
−i, −−ignore−case ignore case distinctions
−w, −−word−regexp force PATTERN to match only whole words
−x, −−line−regexp force PATTERN to match only whole lines
−z, −−null−data a data line ends in 0 byte, not newline

Miscellaneous:
−s, −−no−messages suppress error messages
−v, −−invert−match select non−matching lines
−V, −−version print version information and exit
−−help display this help and exit
−−mmap use memory−mapped input if possible

Output control:
−b, −−byte−offset print the byte offset with output lines
−n, −−line−number print line number with output lines
−H, −−with−filename print the filename for each match
−h, −−no−filename suppress the prefixing filename on output
−q, −−quiet, −−silent suppress all normal output
−−binary−files=TYPE assume that binary files are TYPE
TYPE is 'binary', 'text', or 'without−match'.
−a, −−text equivalent to −−binary−files=text
−I equivalent to −−binary−files=without−match
−d, −−directories=ACTION how to handle directories
ACTION is 'read', 'recurse', or 'skip'.
−r, −−recursive equivalent to −−directories=recurse.
−L, −−files−without−match only print FILE names containing no match
−l, −−files−with−matches only print FILE names containing matches
−c, −−count only print a count of matching lines per FILE
−Z, −−null print 0 byte after FILE name

Context control:
−B, −−before−context=NUM print NUM lines of leading context
−A, −−after−context=NUM print NUM lines of trailing context
−C, −−context[=NUM] print NUM (default 2) lines of output context
unless overridden by −A or −B
−NUM same as −−context=NUM
−U, −−binary do not strip CR characters at EOL (MSDOS)
−u, −−unix−byte−offsets report offsets as if CRs were not there (MSDOS)

`egrep' means `grep −E'. `fgrep' means `grep −F'.
With no FILE, or when FILE is −, read standard input. If less than

two FILEs given, assume −h. Exit status is 0 if match, 1 if no match,
and 2 if trouble.

Ej:

Algo mas complicado:

$ grep ^(f|b)*o(h)+. * # X* = X 0 o mas veces, X+ = X 0 o mas veces , ^X = X
al principio de linea,
. = cualquier caracter menos \n. X | Y = X o Y. Este ejemplo haceptaria
las palabras siguientes si
se encuentra en el margen izquierdo.

ohx
fbfbohhhhaavvddf
fffohhMAMAMIA

find sirbe para listar todos los archivos de un subarbol. Puede ejecutar
intrucciones por cada archivo:

$ find /usr/include −name "*.h" −exec grep open {} \; −print

por cada archivo de include busca open con grep e imprime el nombre de
archivo despues.

Como buscar texto en subdirectorios ?

Algunos ejemplos:

grep "string" `find . −type f`

find . −type f | xargs grep "string"

find . −type f | xargs fgrep "string" /dev/null

locate $PWD | grep "^$PWD" |xargs fgrep "string" /dev/null

find . \(−type f −name "*\.html" \) −exec grep −l "string" {} \;

Yo suelo usar el de find . −type f | xargs fgrep "string" /dev/null

11.1.7 Ejemplos
Sí, find tiene demasiadas opciones, lo sé. Pero, hay un montón de casos preparados que vale la pena recordar, porque son usados muy a menudo. Veamos algunos de ellos.

$ find . −name foo* −print

Encuentra todos los nombres de fichero que empiezan con foo. Si la cadena de caracteres está incluida en el nombre, probablemente tiene más sentido escribir algo como , en vez de ``foo".

$ find /usr/include −xtype f −exec grep foobar \
/dev/null {} \;

Es un grep ejecutado recursivamente que empieza del directorio /usr/include. En este caso, estamos interesados tanto en ficheros regulares como en enlaces simbólicos que apuntan a ficheros regulares, por tanto el test ``−xtype". Muchas veces es más simple evitar especificarlo, especialmente si estamos bastante seguros de cuáles ficheros binarios no contienen la cadena de caracteres deseada. (¿Y por qué el /dev/null en el comando? Es un truco para forzar al grep a escribir el fichero del nombre donde se ha encontrado un emparejamiento. El comando grep se aplica a cada fichero con una invocación diferente, y, por lo tanto no cree que sea necesario mostrar a la salida el nombre del fichero. Pero ahora hay dos ficheros, esto es: ¡el activo y /dev/null! Otra posibilidad podría ser redirigir la salida11.9 del comando a xargs y dejar llevar a cabo el grep. Yo lo intenté, e hice pedazos completamente mi sistema de ficheros (junto con estas notas que estoy intentando recuperar a mano :−( ).

```
$ find / −atime +1 −fstype ext2 −name core \
−exec rm {} \;
```

Es un trabajo clásico para la tabla de tareas preplaneadas. Borra todos los ficheros llamados core en el sistema de ficheros del tipo ext2 al cual no se ha accedido en las últimas 24 horas. Es posible que alguien quiera usar los ficheros de imagen de memoria11.10 para realizar un volcado post mortem, pero nadie podría recordar lo que estuvo haciendo después de 24 horas...

```
$ find /home −xdev −size +500k −ls > piggies
```

Es útil para ver quién tiene esos archivos que atascan el sistema de ficheros. Note el uso de ``−xdev"; como sólo estamos interesados en un sistema de ficheros, no es necesario descender otro sistema de ficheros montado bajo /home.

Mas cosas y ejemplos sobre GREP y el SED

SED

sed [OPTION]... {script−only−if−no−other−script} [input−file]...

−n, −−quiet, −−silent
suppress automatic printing of pattern space
−e script, −−expression=script
add the script to the commands to be executed
−f script−file, −−file=script−file
add the contents of script−file to the commands to be executed
−−help display this help and exit
−V, −−version output version information and exit

If no −e, −−expression, −f, or −−file option is given, then the first
non−option argument is taken as the sed script to interpret. All
remaining arguments are names of input files; if no input files are
specified, then the standard input is read.

E−mail bug reports to: bug−gnu−utils@gnu.org .
Be sure to include the word ``sed" somewhere in the ``Subject:" field.

```
s/^ *//g
s/ *$//g
s/ */ /g
```

```
GETURL $GETURLFLAGS "$url" |
#$EGREP −i '/news/[0−9][0−9]*−[0−9][0−9]*−[0−9][0−9]*.*\.html' | $EGREP −i 'tag=(tp_pr|mn_hd)' |
$EGREP −v '].*>[ ]*links tr −d '\015' | # Remove carriage−return
sed '/tp_pr/a\
special feature' | # main titles only have one line
striphtml |
$EGREP −v '[ap]\.m\.[ ]PT' | # ignore time line
```

$EGREP −v '^[ ]*$' |
paste −d '|' − − | # concatenate two consecutive
lines sed 's:|: −− :' |
nl −s'. ' | # number lines, number separator
is '. ' umbruch −o 8

GREP
grep tcp | grep LISTEN
cat archivo | grep −v palabra (Lo enseña todo menos palabra) grep loquesea.*LOQUEQUIERAS
grep tcp | grep LISTEN
$ cat archivo | grep palabra
Para ver los logs "en directo" tenemos el comando tail, de forma que tail −f
/var/log/messages, por ejemplo nos muestra los mensajes del sistema segun
van sucediendo.

Y el otro truco es una opcion interesante del grep, el parametro −v que nos muestra
los datos que NO cumplen la cadena de busqueda
Por ejemplo para ver los accesos a la web que no vienen de nuestra maquina
cat /var/log/httpd/access_log | grep −v "192.168.0.1"

Otro comando útil en este tipo de programación shell es el tr. tr −s " ", por ejemplo, te sustituye las seqüencias de uno o más espacios por uno solo, esto es útil al combinarlo con cut −d " ", ya que a lo mejor entre dos campos de un log hay un número variable de espacios.

root
moul
xinit
enlightenment
Bi−Clic File y File Manager

kill −9 numero proceso
ps −a
xinit
enlightenment
xdm

http://searchenginewatch.com/resources/tutorials.html(5)
http://www.google.com/(6)
writes all keystrokes to [logfile] and sends them to
the shell to be executed
Allows you to create a shell script for a series
of commands as you do them
#
usage: logstrokes [file to log to]
when done, ctr−c to stop logging

comments / improvements: support@webmastersguide.com

history:
1.2 Fixed script so logfile could be entered as
either a full path or a relative path, without a switch
#
1.1 Initial release − required r switch

case $1 in
"")echo "usage: logstrokes [file to log to]"; exit 1;;
−*) shift;;

```
esac

case $1 in
/.*) logfile="$1";;
*) logfile="`pwd`/$1";;
esac
echo "logging to $logfile"

touch $logfile
chmod 755 $logfile

while :
do
echo −n "[`pwd`]# "
read args
echo "$args" >>$logfile
$args
done
```

Captura de web sites.

Antes que nada quiero aclarar que un proceso automático para hacer un mirror de un website es algo que por ahora me supera, y como no tengo una necesidad repetida de hacerlo, no profundizé en un spider mas o menos astuto y lo hice por pedacitos. Todo lo que sigue puede servir como una guia de procedimientos, es sólo orientativa. Si vos si tenés la necesidad del spider, me avisás y lo hacemos. Tené en cuenta que a diferencia de un directorio de ftp, que se parece más a un árbol de derivación (existe una estructura jerárquica, cada nodo se deriva de uno solo) como un directorio unix con ocasionales links, la estructura web es un grafo no dirigido y la lógica para recorrerlo integramente de forma eficiente es más compleja.

El otro dia un amigo me pidió que le sacara una copia a un site de Bach. Entre 1200 a 1800 htmls, no hay acceso por ftp, solo web. Un mail, please send me a zip, no, es antideportivo. Hay que hacer algo asi como un mirror por única vez, una instantanea del site

La estructura se puede ver en The J. S. Bach Home Page

Son varios index (por fecha, instrumento, nombre, combinados) que apuntan a un montón de archivos "bach[:digit:].html" y estos a otros que no recuerdo bien, pero algo asi como "p[:digit:].html

Webeando por ahi, encontré un simpatico script en Perl, wwwgrab, (los créditos están adentro) que me permitió con otro par de shell scripts hacerlo de un modo no del todo manual.

Primero te traés index.html y le extraés los links que sean de los index del site, a algunos de estos les corrés

```
# filtro para wwwgrab
grep "<A HREF" *.html | \ # extrae todos los links

cut −f 2 −d "" | \ # extrae el segundo campo segun delimitador (")
grep [0123456789] >> \ # extrae solo los archivos con numeros
archivos_a_bajar
```

Luego editá archivos_a_bajar y le sacás las irregularidades y

```
#!/bin/bash
URL=http://www.tile.net/tile/bach/
for file in $(cat archivos_a_bajar); do
wwwgrab $URL$file $file &# lo mando al background...
echo $URL$file # para monitorear
sleep s 1 # ... para que no se amontonen
done
```

Y repetís segun haga falta.

Más información en man grep, man cut, less wwwgrab

Cosas del AWT

AWK

por Javier Palacios Bermejo
Viene de http://www.linuxfocus.org/Castellano/September1999/article103.html

Sobre el Author:

Está realizando su tesis doctoral en Astronomía en una universidad española, donde está encargado de la administración del cluster . El trabajo diario se lleva a cabo con máquinas unix y, tras unos primeros intentos infructuosos, se consiguió instalar linux−slackware. Varias actualizaciones después sigue funcionando, mejor que algunos otros de los unix propietarios que se corren en las máquinas del cluster.

Contenidos:

Ejemplos con awk: Una breve introducción

Resumen:

Este artículo pretende ser una breve introducción al viejo comando/programa de unix awk. Aunque no tan popular como el shell o muchos otros lenguajes de scripting, es una herramienta muy potente cuando hay que tratar con información agrupada en tablas. No está enfocado como un tutorial, sino que se desarrollan algunos ejemplos de uso reales, para mostrar con cierto detalle sus capacidades.

−−

La idea de escribir este texto surgió por la lectura de un par de artículos aparecidos en LinuxFocus y escritos por Guido Socher. Uno de ellos versaba sobre find y comandos relacionados y me hizo ver que al parecer no era el único que usaba todavía la línea de comandos, en lugar de bonitos GUI que consiguen que no sepas como se hacen las cosas (que es el camino por el que tiró Windows hace mucho). El otro de los artículos trataba sobre expresiones regulares que, aunque apenas mencionadas en este artículo, resulta muy conveniente conocer para sacar el mayor partido posible a awk y algún otro de los comandos sobre los que pensaba hablar inicialmente en este artículo (sed y grep principalmente). algunos otros comandos.

La pregunta clave es si es realmente útil este comando, y la respuesta es que sí. Le puede resultar útil a un usuario para procesar ficheros de texto, reformatearlos, etc... Para un administrador, awk es, simplemente, una utilidad casi imprescindible. Basta con pasear por /var/yp/Makefile, para darse cuenta de ello.

Introducción al awk
Supe de su existencia hace tanto que no lo recuerdo bien. Fue cuando un compañero tenía que trabajar con unos output impresionantes en un pequeño Cray y estuvo investigando muchas posibilidades de clasificación. La página man del awk en el Cray era de lo más exigua, pero el decía que parecía muy bueno para esa tarea aunque no había forma de hincarle el diente.
Mucho tiempo después, se volvió a cruzar en mi vida, mediante una especie de comentario casual (otro sitio, otro compañero), que lo usaba para extraer la primera columna de una tabla:

```
awk '{print $1}' fichero
```

Fácil, verdad ? Esta tarea tan simple requeriría una cierta dosis de programación en C o cualquier otro lenguaje compilado o interpretado.

Una vez aprendida la lección extrayendo una columna ya podemos hacer algunas cosillas como añadir una extensión a una serie de ficheros con secuencias como

```
ls −1 pattern | awk '{print "mv "$1" "$1".nuevo"}' | sh
```

Y más aún ...

renombrando el interior del nombre
ls −1 *viejo* | awk '{print "mv "$1" "$1}' | sed s/viejo/nuevo/2 | sh
(aunque en algunos casos fallará, como con fichero_viejo_y_viejo)
borrar solo ficheros (puede hacerse con rm nada más, pero qué pasa con alias como 'rm −r')
ls −l * | grep −v drwx | awk '{print "rm "$9}' | sh
Cuidado al probar esto en un directorio. ¡ Borramos ficheros !
borrar solo directorios
ls −l | grep '^d' | awk '{print "rm −r "$9}' | sh
o
ls −p | grep /$ | awk '{print "rm −r "$1}' | sh

Feedback de los lectores: Como Dan Debertin me hizo notar, algunos de los ejemplos anteriores se pueden realizar sin usar el comando grep, solo con las capacidades de matcheodel awk que se mencionan unas líneas más abajo.
ls −l *|grep −v drwx|awk '{print "rm "$9}'|sh

sería mas ilustrativo de la potencia de AWK en la forma:
ls −l|awk '$1!~/^drwx/{print $9}'|xargs rm

también,
ls −l|grep '^d'|awk '{print "rm −r "$9}'|sh

podría escribirse como
ls −l|awk '$1~/^d.*x/{print $9}'|xargs rm −r

Uso constantemente la siguiente línea para matar procesos:
(digamos que el proceso se llama 'sleep')
ps −ef|awk '$1~/'$LOGNAME'/$8~/sleep/$8!~/awk/{print $2}'|xargs kill −9

(hay que ajustarlo para la forma que adopte el comando ps en el sistema en
que trabajes. En algunas ocasiones será ps −aux, el número de campos
variará, etc.) Básicamente es "Si el dueño del proceso ($1) soy yo, y si
se llama ($8) "sleep", y no se llama "awk" (en ese caso el comando awk se
mataría a sí mismo), enviar el PID correspondiente ($2) al comando kill −9.".

¡Y sin usar grep!

Cuando, por ejemplo, se repiten los mismos cálculos una y otra vez, estas herramientas resultan una gran ayuda. Y, además, es mucho más divertido escribir un programa de awk que repetir manualmente lo mismo veinte veces.

Aunque nos referimos a él con ese nombre, el awk no es en realidad un comando, de igual forma que el gcc tampoco lo es. Awk es en realidad un lenguaje de programación, con una sintaxis con aspectos similares al C, y cuyo intérprete se invocacon la instrucción awk.

En cuanto a la sintaxis del comando, casi todo está dicho ya:

```
# gawk −−help
Usage: gawk [POSIX or GNU style options] −f progfile [−−] file ...
gawk [POSIX or GNU style options] [−−] 'program' file ...
POSIX options: GNU long options:
−f progfile −−file=progfile
−F fs −−field−separator=fs
−v var=val −−assign=var=val
−m[fr] val
−W compat −−compat
−W copyleft −−copyleft
−W copyright −−copyright
−W help −−help
−W lint −−lint
−W lint−old −−lint−old
−W posix −−posix
−W re−interval −−re−interval
−W source=program−text −−source=program−text
−W traditional −−traditional
−W usage −−usage
−W version −−version

Report bugs to bug−gnu−utils@prep.ai.mit.edu,
with a Cc: to arnold@gnu.ai.mit.edu
```

Baste destacar que, además de incluir los programas entre comillas sencillas (') en la línea de comandos, se pueden escribir en un fichero que invocamos con la opción −f, y que definiendo variables en la línea de comandos −v var=val, podemos dotar de cierta versatilidad a los programas que escribamos.

Awk es, básicamente, un lenguaje orientado al manejo de tablas, en el sentido de información susceptible de clasificarse en forma de campos y registros, al estilo de las bases de datos más tradicionales. Con la ventaja de que la definición del registro (e incluso del campo) es sumamente flexible.

Pero awk es mucho más potente. Está pensado para trabajar con registros de una línea, pero esa necesidad se puede relajar. Para profundizar un poco en algunos aspectos, vamos a echar un vistazo a algunos ejemplos ilustrativos (y reales).

Imprimir tablas un poco más bonitas

Es posible que alguna vez hayamos tenido que imprimir alguna tabla ASCII obtenida de alguna parte como, por ejemplo, las asociaciones de números ethernet, IP y nombres de hosts. Cuando las tablas son realmente grandes, su lectura se hace realmente difícil, y empezamos a echar de menos lo bien que se lee una tabla impresa con LaTeX o, al menos, formateada algo mejor. Si la tabla es sencilla (y/o sabemos usar bien el awk), no resulta demasiado difícil, aunque puede hacerse un poco tedioso:

```
BEGIN {
printf "preambulo LaTeX"
```

```
printf "\\begin{tabular}"
printf "{|c|c|...|c|}"
}

{
printf $1" &"
printf $2" &"
.
.
.
printf $n" \\\\ "
printf "\\hline"
}

END {
print "\\end{document}"
}
```

Ciertamente no es un programa lo que se dice genérico, pero estamos empezando ...
(los \ dobles son necesarios por tratarse del carácter de escape del shell)

Troceando ficheros de output
SIMBAD es una base de datos de objetos astronómicos que, entre otras cosas, incluye las posiciones en el cielo de los mismos. En cierta ocasión, necesité hacer búsquedas para construir mapas alrededor de algunos objetos. Como el interface de dicha base de datos permite guardar los resultados en ficheros de texto, podía hacer dos cosas: generar un fichero para cada objeto o darle como input la lista completa, obteniendo un único y enorme log con la consulta. Como decidí hacer lo segundo, use awk para trocearlo. Obviamente, para ello tuve que aprovechar ciertas características del output.
cada solicitud generaba una línea de cabecera, con un formato del tipo
====> nombre : nlines <====
El primer campo me permitia saber cuando empezaba un objeto nuevo y el cuarto cuantas entradas correspondían al mismo (aunque ese dato no es imprescindible)
el caracter que separaba las columnas dentro de las listas de output era '|'. Eso requería dos líneas de código adicional para poder enviar al output sólo los campos de mi interés.

```
( $1 == "====>" ) {
NomObj = $2
TotObj = $4
if ( TotObj > 0 ) {
FS = "|"
for ( cont=0 ; cont getline
print $2 $4 $5 $3 >> NomObj
}
FS = " "
}
}
```

NOTA: Como en realidad no daba el nombre del objeto, era un poco más complicado, pero pretende ser un ejemplo ilustrativo.

Jugeteando con el spool del mail

```
BEGIN {
BEGIN_MSG = "From"
BEGIN_BDY = "Precedence:"
MAIN_KEY = "Subject:"
VALIDATION = "[RESUMEN MENSUAL]"
```

```
HEAD = "NO"; BODY = "NO"; PRINT="NO"
OUT_FILE = "Resumenes_Mensuales"
}

{

if ( $1 == BEGIN_MSG ) {
HEAD = "YES"; BODY = "NO"; PRINT="NO"
}

if ( $1 == MAIN_KEY ) {
if ( $2 == VALIDATION ) {
PRINT = "YES"
$1 = ""; $2 = ""
print "\n\n"$0"\n" > OUT_FILE
}
}

if ( $1 == BEGIN_BDY ) {
getline
if ( $0 == "" ) {
HEAD = "NO"; BODY = "YES"
} else {
HEAD = "NO"; BODY = "NO"; PRINT="NO"
}
}

if ( BODY == "YES" &PRINT == "YES" ) {
print $0 >> OUT_FILE
}
}
```

Tal vez administramos una lista de correo. Tal vez, de vez en cuando, se envían a la lista mensajes especiales (p.e. resúmenes mensuales) con algún formato determinado (p.e. un subject tipo '[RESUMEN MENSUAL] mes , dept'). Y de repente, se nos ocurre a fin de año recopilar todos los resúmenes, separándolos de los demás mensajes.
Esto podemos hacerlo usando el awk con el spool del mail y el programa que tenemos a la izquierda
Hacer que cada resumen vaya a un fichero requiere tres líneas adicionales, y hacer también que, por ejemplo, cada departamento vaya a un fichero diferente supone unos pocos caracteres más.

NOTA: Todo este ejemplo está basado en cómo creo yo que estan estructurados los mails en el spool. Realmente no se como lo hacen, aunque me funciona (de nuevo, en algunos casos fallará, como siempre).

Programas como éstos sólo necesitan 5 minutos pensando y 5 escribiendo (o más de 20 minutos sin pensar, mediante ensayo y error que es como resulta más divertido).
Si hay alguna forma de hacerlo en menos tiempo, quiero saberla.

He usado el awk para muchas otras cosas (como generación automática de páginas web con información obtenida de una base de datos) y se lo suficiente de programación como para estar seguro de que se pueden hacer con él cosas que ni siquiera se me han ocurrido.
Sólo hay que dejar volar la imaginación.

Un problema
El único problema del awk es que necesita información tabular perfecta, sin huecos: no puede trabajar con columnas de anchura fija, que son de lo mas común. Si el input del awk lo generamos nosotros mismos, no es muy problemático: elegimos algo realmente raro para separar los campos, lo fijamos luego con la variable FS y ya está. Pero si solo tenemos el input, esto puede ser más problemático. Por ejemplo, una tabla tipo
1234 HD 13324 22:40:54 ....

1235 HD12223 22:43:12 ....
no se podria tratar con el awk. Entradas como esta a veces son necesarias, aparte de ser bastante comunes. Aún así, rizando el rizo, si sólo tenemos una columna con esas características no todo está perdido (si alguien sabe manejarse con más de una en un caso general, soy todo oídos).
En una ocasión tuve que enfrentarme a una de esas tablas, similar a la descrita más arriba. La segunda columna era un nombre e incluía un número variable de espacios. Como suele pasar yo necesitaba ordenarla por una columna posterior a ella. Hice varios intentos con el sort +/−n.m que tenía el mismo problema de los espacios embebidos.

(y una solución)
Y me di cuenta de que la columna que yo quería ordenar era la última. Y de que awk sabe cuantos campos hay en el registro actual, por lo que bastaba ser capaz de acceder al último (unas veces $9, otras $11, pero siempre el NF). Total, que un par de pruebas, arrojaron el resultado deseado:

```
{
printf $NF
$NF = ""
printf " "$0"\n"
}
```

Y obtengo un output igual al input, pero con la última columna movida a la primera posición, y sorteo sin problemas.
Obviamente, el método es facilmente ampliable al tercer campo empezando por el final, o al que va despues de un campo de control que siempre tiene el mismo valor, porque es el que usamos al extraer nuestra subtabla de la base de datos original ...
Sólo deja volar tu imaginación de nuevo.

Profundizando en el awk

Trabajando sobre lineas matcheadas

Hasta ahora, casi todos los ejemplos expuestos procesan todas las lineas del fichero de entrada. Pero, como claramente explica la página de manual, es posible hacer que un cierto grupo de comandos procese tan sólo unas ciertas líneas por el simple método de incluir la condición antes de los comandos, al modo del segundo de los ejemplos anteriores. La condición que debe satisfacer la línea puede llegar a ser bastante flexible, desde una expresión regular, hasta un test sobre los contenidos de alguno de los campos, pudiendo agruparse condiciones en base a operadores lógicos.

awk como lenguaje de programación

Como todo lenguaje de programación, awk implementa todas las estructuras de control necesarias, así como un conjunto de operadores y funciones predefinidas, para manejar números y cadenas. Su sintaxis es en general muy parecida a la del C, aunque difiere de él en algunos aspectos.

Y, por supuesto, también es posible incluir funciones definidas por el usuario, usando la palabra function, y escribiendo los comandos como si se tratara de procesar una línea normal del fichero de entrada. E, igualmente, aparte de las variables escalares habituales, tambíen es capaz de manejar arrays de variables.

Incluyendo librerías

Como suele pasar con todos los lenguajes, hay una cierta serie de funciones que son bastante comunes, y llega un momento en que cortar y pegar no es la mejor forma de hacer las cosas. Para eso se inventaron las librerías. Y, al menos con la versión GNU de awk, es posible incluirlas dentro del programa awk. Pero eso es usar awk como una herramienta de trabajo mucho más seria de lo que se pretende mostrar en este artículo, aunque deja claro el nivel de complejidad que puede llegar a alcanzar el awk.

Conclusiones

Ciertamente, puede no ser tan potente como numerosas herramientas que se pueden usar con la misma finalidad. Pero tiene la enorme ventaja de que, en un tiempo realmente corto, permite escribir programas que, aunque tal vez sean de un solo uso, están totalmente adaptados a nuestras necesidades, que en muchas ocasiones son sumamente sencillas.

awk es ideal para los propósitos con los que se diseño: leer ficheros línea por línea y procesar en base a los patterns y cadenas que encuentre en ellas.

Ficheros del sistema como el /etc/password y muchos otros, resultan sumamente fáciles de tratar mediante el awk, sin recurrir a nada más.

Y desde luego que awk no es el mejor. Hay varios lenguajes de scripting con capacidades mucho mayores. Pero awk sigue teniendo la ventaja de ser siempre accesible en cualquier instalación, por mínima que esta sea.

Información adicional
Este tipo de comandos tan básicos no suelen estar excesivamente documentados, pero siempre se puede encontrar algo buscando por ahí.

la sintaxis del awk no es igual en todos los *nix, pero siempre hay una forma de saber exactamente qué podemos hacer con el del nuestro particular: man awk;
Como no podía ser de otra forma, O'Reilly tiene un libro sobre el tema: Sed &Awk (Nutshell handbook) de Dale Dougherty.
Una búsqueda en Amazon, nos proporciona algunos otros títulos como Effective Awk Programming: A User's Guide, bastante orientado al gawk, y media docena más.

En general, todos los libros y manuales de unix mencionan estos comandos. Pero sólo algunos de ellos profundizan un poco y dan información útil. Lo mejor, hojear todos aquellos que pasen por nuestras manos, pues nunca se sabe donde podemos encontrar información valiosa.

Contactar con el equipo de LinuFocus

1999−06−05, generated by lfparser version 0.6

Enlaces a expresiones regulares y Unix.

Agradecimientos: Esta pagina y las 5 ateriores son fruto del trabajo recopilador y experimentador de un lector que ha preferido permancer anonimo. Gracias a él esta pagina se va a convertir en uno de los mejores, por no decir el mejor, recurso castellano sobre expresiones regulares y herramientas Unix para manejarlas.

Disfrutadlo, y no olvideis compartir con el resto de la comunidad lo que vayais aprendiendo ;−)

More Unix, Waterloo.[7]
Chicago: Referencias Unix[8]
Google: string comparisons unix script[9]
Waterloo Comparison Unix/Dos[10]
Index Fundamentals[11]
Unix Fundamentals[12]
Unix Commands[13]
Teaching Unix[14]
Comandos Misc[15]
Shell Scripts − case and for[16]
(if−then−else)[17]
Introducción a los scripts de shell[18]
How to write HTTP programs[19]
Programming and using TCP/IP[20]
Scripts[21]
Search engine Tutorial[5]
Dig[22]
Computers and Communication Standards[23]
www.all.net[24]
Unix Gurus[25]
Geek Girl[26]
www.w3.org/Protocols[27]
Cursos Unix[28]
Examples of shell scripts[29]
Más gurus: scripts[30]
Scripts Unix[31]
QNX[32]
Gethhtp por Java[33]
Comandos Unix en Castellano[34]
Manipulating Text[35]
Sed Tutorial[36]
Perl Regular Expression Simulator[37]
Gnome[38]
Config Hardware Linux[39]
KDE 7.1[40]
Resumen Comandos Unix[41]
Dragon Linux[42]
Comandos de Filtro[43]
KDE 6.2[44]
Expresiones Regulares[45]
Más Grep[46]
Expresiones REgulares[47]

Lista de enlaces de este artículo:

1. http://bulma.net/body.phtml?nIdNoticia=736
2. http://www.ciberdroide.com/misc/novato/curso/regexp.html
3. mailto:jandrews@mydesktop.com
4. http://www.linuxplanet.com/linuxplanet/tutorials/214/
5. http://searchenginewatch.com/resources/tutorials.html
6. http://www.google.com/search?num=30&hl=es&safe=o
7. http://www.ist.uwaterloo.ca/ec/unix/moreunix.htm
8. http://www.lib.uchicago.edu/TechInfo/Systems/IRSA/Unix/so urces.html
9. http://www.google.com/search?num=30&hl=es&safe=o
10. http://www.ist.uwaterloo.ca/ec/unix/comparison.html
11. http://www.sikh−history.com/computers/unix/
12. http://www.sikh−history.com/computers/unix/funda.html
13. http://www.sikh−history.com/computers/unix/commands.html
14. http://www.shu.ac.uk/schools/cms/teaching/ps/unix/
15. http://www.um.es/~eutsum/escuela/Apuntes_Informatica/Divulgacion/Informatica/Uni
16. http://www.shu.ac.uk/schools/cms/teaching/ps/unix/casefor.html
17. http://bragg.ivic.ve/Ivic/IvicCQ/curso/unixclass8.html
18. http://www.it.uc3m.es/ttao/Hw6/
19. http://www.hotswap.se/engineering/network/http.html
20. http://www.hotswap.se/engineering/network/introtcpip.html
21. http://www.psnw.com/~alcald/
22. http://www.library.ucg.ie/Connected/cgi−bin/dig.cgi
23. http://www.cmpcmm.com/cc/standards.html
24. http://www.all.net/
25. http://www.ugu.com/sui/ugu/show?ugu
26. http://www.geek−girl.com/unix.html
27. http://www.w3.org/Protocols/
28. http://www.library.ucg.ie/Connected/Course/Section3/15.htm
29. http://www.stud.arch.ethz.ch/~vasummer/doc/man/shell_script_sites.html
30. http://www.ugu.com/sui/ugu/show?SEARCH=scripts
31. http://oase−shareware.org/shell/scripts/
32. http://www.qnx.com/
33. http://vega.ijp.si/Gallery/HTML/g/geturl_s.htm
34. http://albergue.hypermart.net/comandos_unix.html
35. http://virtual.park.uga.edu/humcomp/perl/
36. http://virtual.park.uga.edu/humcomp/perl/sedtutorial.html
37. http://www.rutgers.edu/~sgro/perltutor/regexp.html
38. http://www.gnome.org/faqs/users−faq/starting.html
39. http://www.escomposlinux.org/hardware/
40. http://www.europe.redhat.com/documentation/rhl7.1/rhl−gsg−es−7.1/
41. http://www.ivia.es/~sto/alice/libro/comunix.htm
42. http://www.dragonlinux.net/readme.php
43. http://www.iie.edu.uy/~victor/unixbas/comando3.htm/
44. http://www.europe.redhat.com/documentation/rhl6.2/install−guide−es/
45. http://www.iie.edu.uy/~victor/unixbas/expreg.htm
46. http://bulma.net/body.phtml?nIdNoticia=675/
47. http://www.ciberdroide.com/misc/novato/curso/regexp.html#ejegrep−regexpr

---

E−mail del autor: danircJUBILANDOSEbulma.net
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=770