Utilidades para la gestión de ficheros y expresiones regulares

1. Búsqueda de patrones: GREP

El comando grep (Globally Regular Expressions Pattern) **busca patrones en ficheros**. Por defecto, devuelve **todas las líneas que contienen un patrón determinado** (cadena de texto) en uno o varios ficheros. Utilizando las opciones se puede variar mucho este comportamiento. Además, si no se le pasa ningún fichero como argumento, hace la búsqueda en su entrada estándar. Su sintaxis es:

grep [opciones] [expresión regular] [archivo]

Algunas de sus opciones de uso son:

-c → devuelve sólo la cantidad de líneas que contienen al patrón.

-i → ignora las diferencias entre mayúsculas y minúsculas.

-v → devuelve las líneas que no contienen el patrón.

-r → busca en un directorio de forma recursiva.

-n → imprime el número de cada línea que contiene al patrón.

-l → con múltiples ficheros sólo muestra los nombres de aquellos donde se encontró al patrón y no las líneas.

-H → con múltiples ficheros además de las líneas muestra el nombre del fichero donde se encontró el patrón.

```
paco@ubuntupaco:~$ grep linux /usr/share/doc
paco@ubuntupaco:~$ grep root /etc/passwd
paco@ubuntupaco:~# grep -n error /var/log/messages
paco@ubuntupaco:~$ grep -i pepe /etc/passwd
paco@ubuntupaco:~$ grep -c root /etc/group
```

A continuación vemos algunos ejemplos de uso de grep:

Para mostrar **todas las líneas** que contienen la cadena **tal** en una lista de archivos (donde * representa **todos** los archivos en el directorio actual):

```
paco@ubuntupaco:~$ grep tal *
```

Para mostrar **todas las líneas** que **no** contengan la cadena **tal**, se usa **-v**:

```
paco@ubuntupaco:~$ grep -v tal *
```

Para mostrar **sólo el nombre de los archivos** del ejemplo anterior, se usa **-l**:

```
paco@ubuntupaco:~$ grep -l tal *
```

Para mostrar **sólo el nombre de los archivos** que **no** contienen la cadena, se usa **-L**:

```
paco@ubuntupaco:~$ grep -L tal *
```

Para buscar **recursivamente**, no sólo en los archivos del directorio actual sino que también busca en los de sus subdirectorios, donde . (punto) representa el directorio actual, se usa **-r**:

```
paco@ubuntupaco:~$ grep -r tal .
```

Para buscar todas las líneas que comienzan por **Ahora** y terminan con **siempre** seguido de una cantidad arbitraria de espacio en blanco (el **carácter ^ representa el inicio de la línea**, así como **$ representa el final**):

```
paco@ubuntupaco:~$ grep '^Ahora.*siempre *$'
```

Para hacer que grep **lea de la entrada estándar** no se especifica archivo alguno. Por ejemplo, como **ps -ef** lista todos los procesos actualmente en ejecución, el siguiente comando imprime todos los procesos que **está ejecutando el usuario actual**:

```
paco@ubuntupaco:~$ ps -ef | grep $USER
paco@ubuntupaco:~$ ps -efa | grep $USER
```

2. Comparación de ficheros: DIFF

El comando diff nos permite **comparar ficheros de texto línea a línea** y nos indica en caso de que no sean iguales en que líneas cambian. Dado que en Linux se realizan muchas configuraciones del sistema modificando ficheros de texto, es una orden interesante para saber si algo se ha modificado.

Es invocado desde la línea de comando con los nombres de dos archivos: **fichero-desde fichero-hacia**. El resultado representa los **cambios requeridos para que el archivo original se convierta en nuevo archivo**.

diff [opciones] fichero1 fichero2

Si **fichero-desde** y **fichero-hacia** son **directorios**, entonces **diff se ejecutará sobre cada archivo** que exista en ambos directorios. La **opción -r** es para un uso recursivo, es decir, hará que cualesquiera subdirectorios emparejados comparen archivos entre directorios.

En este formato de salida tradicional, **a** sustituye a la opción de **añadido** (added), **d** a **borrado** (deleted) y **c** a **cambiado** (changed). Los **números de línea** del archivo fichero-desde aparecen antes de **a/d/c** y los del archivo modificado después. Los **paréntesis angulares** aparecen al comienzo de las **líneas que son añadidas, borradas o cambiadas**.

- Las **líneas añadidas** se **incluyen** en el archivo fichero-desde para aparecer en el archivo nuevo.
- Las **líneas borradas** se **eliminan** del archivo fichero-desde para ser borradas en el archivo nuevo.

Por defecto, las líneas comunes a los dos archivos no se muestran. Las líneas que se han movido se muestran como añadidas en su nuevo lugar y como borradas en su antiguo lugar. Con el símbolo < se indica si la diferencia está en el primer archivo y con > si está en el segundo archivo.

```
paco@ubuntupaco:~$ cat a.txt
gato
perro
canario

paco@ubuntupaco:~$ cat b.txt
gato
armadillo
perro
canario

paco@ubuntupaco:~$ diff a.txt b.txt
1a2
> armadillo
```

Para obtener un fichero-hacia a partir de la salida de diff y el fichero fichero-desde: **patch fichero-desde salida-diff**.

3. Búsqueda de ficheros: FIND

El comando find es uno de los más poderosos en un sistema Linux. Permite buscar de forma recursiva en un directorio a todos los ficheros que cumplan ciertas condiciones. Las condiciones pueden estar relacionadas con el nombre de los ficheros, el tamaño, los permisos, el tipo, las fechas de acceso y modificación, etc.

find [ruta] [opciones]

Algunas opciones de uso:

-name <expresión> → especifica patrones para los nombres de los ficheros a buscar.

-iname <expresión> → igual que la anterior pero sin tener en cuenta mayúsculas y minúsculas.

-type <tipo> → indica el tipo de fichero a buscar. Puede ser **d para directorios** y **f para ficheros** regulares.

-size +/-<n> → indica el tamaño máximo y/o mínimo de los ficheros a buscar.

-perm [+|-]<modo> → especifica que los permisos sean como se indica en <modo>.

-exec <comando> → permite **definir un comando a ejecutarse** para cada resultado de la búsqueda. La cadena {} sustituye el nombre de los ficheros encontrados. El caracter ; indica la finalización del comando.

Tanto {} como ; tienen que ir entre comillas o **contrabarras** para evitar que sea sustituido por el shell. Algunos ejemplos de uso del comando:

Busca en **/etc** todos los ficheros con extensión **conf**.

```
paco@ubuntupaco:~$ find /etc -name '*.conf'
```

Busca los ficheros cuyo **tamaño esté entre 10 MB y 20 MB**. Por defecto, el tamaño se expresa en **bloques de 512 bytes**, pero si se precede este por un carácter **c** se referirá a **bytes**, **k** a **kilobytes**, **w** a **palabras de dos bytes** y **b** a **bloques**.

```
paco@ubuntupaco:~$ find / -size +10240k -size -20480k
```

Busca únicamente los **ficheros** con **todos los permisos para todos los usuarios** (también podría ser 777). Busca todos los ficheros ejecutables.

```
paco@ubuntupaco:~# find . -type f -perm 0777
paco@ubuntupaco:~# find / -perm /a=x
```

Busca todos los ficheros que se nombren core y los borra interactivamente.

```
paco@ubuntupaco:~# find / -name core -exec rm -i "{}" ";"
```

4. Ordenación de ficheros: SORT

Es una utilidad de la línea de comandos del sistema operativo que funciona tomando los archivos que figuran en su lista de argumentos y **ordena sus líneas**. La ordenación **se realiza sobre la base de una o más claves extraídas de cada línea de los archivos de entrada**.

De forma predeterminada, todos los datos de entrada se toman como clave de ordenación. Los espacios en blanco son tomados por defecto como separadores de campo.

sort [opciones] [archivo]

Algunas opciones de uso son:

-r → invierte el orden, es decir, ordena al revés.

-f → trata las mayúsculas y minúsculas por igual.

-n → ordena de forma numérica.

Para ordenar el directorio actual por tamaño de archivo:

```
paco@ubuntupaco:~$ ls -s | sort -n
  0 GNUstep
  0 Mail
  4 ALT.txt
  4 Events
  4 Ted.lnk
  12 stats
  124 _backup
```

Para ordenar una lista en un archivo por orden alfabético:

```
paco@ubuntupaco:~$ cat agenda.txt
  Juan López      555-4321
  Rodolfo Ruiz    555-3214
  Ana Cohen       555-4321

paco@ubuntupaco:~$ sort agenda.txt
  Ana Cohen       555-4321
  Juan López      555-4321
  Rodolfo Ruiz    555-3214

paco@ubuntupaco:~$ du /bin/* | sort -n
  10      /bin/domainname
  10      /bin/run-parts
  42      /bin/cp
  675     /bin/bash
```

5. Eliminación de duplicidad: UNIQ

El comando uniq **elimina las líneas duplicadas de un fichero ordenado**, imprimiéndolo por la salida estándar o en otro fichero. De no especificarse un fichero toma la entrada estándar.

uniq [opciones] [fichero] [salida]

Sus opciones son:

-i → ignora si se trata de mayúsculas o minúsculas cuando compara las líneas.

-f → ignora un número de campos en una línea.

-s → salta un número de caracteres en una línea.

-w → especifica el número de caracteres o campos a comparar en una línea.

-u → muestra únicamente las líneas que no se repiten en el fichero original.

-d → no muestra las líneas que no están repetidas en el fichero de entrada.

-c → genera un informe de salida donde cada línea está precedida por un contador con el número de líneas que está repetida. Si se especifica esta opción, las **opciones -u y -d se ignoran**.

```
paco@ubuntupaco:~$ sort file | uniq -c | sort -n
```

6. Expresiones regulares

Las expresiones regulares son **patrones que afectan a una cadena de caracteres**. Son muy útiles para **seleccionar con gran precisión elementos de un fichero** y son ampliamente utilizadas en los scripts. Vamos a crear un fichero de texto llamado **expre** con el siguiente contenido:

```
cosas casa libros animal verdugo
LLUvia lanaa madre hermano nido
bicho limo asno vision alma
libro cosas tipo falso oso
bicha corbata talon barco tigre
ult
```

Ahora, vamos utilizar **grep** para **buscar la cadena cosas** dentro del fichero **expre**.

```
paco@ubuntupaco:~$ grep cosas expre
libro cosas tipo falso oso
cosas casa libros animal verdugo
```

Como el fichero **expre** tiene **cinco columnas**, cuando **grep** selecciona la cadena **cosas**, **nos muestra las filas completas en las que se encuentre**.

Ahora, vamos a hacer uso de expresiones regulares para lograr **selecciones más precisas**. La expresión "^cosas" hará que grep busque la **línea que comience por cosas**. Esto es consecuencia del metacaracter: ^. Las expresiones regulares son precisamente **caracteres especiales** (llamados **metacaracteres**) que influyen de una manera particular en la cadena de texto que referencian.

```
paco@ubuntupaco:~$ grep "^cosas" expre
cosas casa libros animal verdugo
```

Es importante **no escribir ningún carácter previo a la primera columna de palabras** al editar el fichero **expre**. No es correcto utilizar la barra espaciadora ni tabuladores, ya que grep los tomará por caracteres y en este caso, como ninguna línea empieza por "cosas" no mostrará resultado alguno.

Queremos filtrar todos los **subdirectorios** del **directorio /usr**.

```
paco@ubuntupaco:~$ ls -la /usr | grep ^d
drwxr-xr-x 13 root root 368 Sep 15 00:56 .
drwxr-xr-x 21 root root 488 Sep 24 14:03 ..
drwxrwxrwx 123 root root 49648 Sep 15 20:13 lib
drwxrwxrwx 12 root root 304 Sep 7 00:57 local
drwxrwxrwx 2 root root 9984 Sep 8 00:03 sbin
...
```

El comando grep ha seleccionado únicamente aquellos **ficheros que comienzan por la letra d**. La primera columna, comenzando por la izquierda son los permisos y el primer carácter indica el tipo de fichero, en este caso, la **d significa directorio**. Con esto, podremos filtrar los ficheros de configuración del sistema en busca de la información que necesitemos para después hacer algo con ella.

En este caso, hemos seleccionado solamente aquella línea que empiece por una palabra de tres caracteres. Si la expresión regular fuese ^....$, entonces seleccionaríamos palabras de cuatro caracteres. Basta con escribir tantos puntos como caracteres tenga la palabra que queramos seleccionar.

```
paco@ubuntupaco:~$ grep "^...$" expre
ult
```

La orden grep ha seleccionado las líneas que contienen bicho o bicha. Otras posibilidades:

```
paco@ubuntupaco:~$ grep "^os[oa]" expre
bicho limo asno vision alma
bicha corbata talon barco tigre
```

Lo que grep ha hecho ahora es seleccionar la única línea que empieza por consonantes.

```
paco@ubuntupaco:~$ grep "[A-Z] [A-Z]*" expre
LLUvia lanaa madre hermano nido
```

Ahora vamos a buscar las líneas que terminan por el carácter: e.

```
paco@ubuntupaco:~$ grep 'e$' expre
bicha corbata talon barco tigre
```

Por último, probemos a buscar aquella línea donde se repite el carácter: a.

```
paco@ubuntupaco:~$ grep "a\{2,\}" expre
LLUvia lanaa madre hermano nido
```

Como no se trata de probar todas y cada una de las expresiones, he preparado el siguiente listado. Es buena idea practicar utilizando cada una de las expresiones y ver el resultado.

Expresión	Resultado
`bicho`	La cadena bicho
`^bicho`	La cadena bicho al comienzo de una línea.
`bicho$`	La cadena bicho al final de una línea.
`^bicho$`	La cadena bicho formando una única línea.
`bich[ao]`	Las cadenas bicha y bicho.
`bi[^aeiou]o`	La tercera letra no es una vocal minúscula.
`bi.o`	La tercera letra es cualquier carácter.
`^....$`	Cualquier línea que contenga cuatro caracteres cualesquiera.
`^\.`	Cualquier línea que comience por un punto.
`^[^.]`	Cualquier línea que no comience por un punto.
`bichos*`	bicho, bichos, bichoss, bichosss, etc.
`"*bicho"*`	bicho con o sin comillas dobles.
`[a-z][a-z]*`	Una o más letras minúsculas.
`[^0-9a-z]`	Cualquier carácter que no sea ni número ni letra minúscula.
`[A-Za-z]`	Cualquier letra, sea mayúscula o minúscula.
`[Ax5]`	Cualquier carácter que sea A, x o 5.
`bicho | bicha`	Una de las palabras bicho o bicha.
`(s | arb)usto`	Las palabras susto o arbusto.
`\<bi`	Cualquier palabra que comience por bi.
`bi\>`	Cualquier palabra que termine en bi.
`a\{2,\}`	Dos o más aes en una misma fila.2. Los filtros.

7. Filtros

El comando cut

Permite **cortar columnas o campos de un fichero de texto y enviarlos a la salida estándar**. Sus opciones:

-c → es para cortar columnas.

-f → para cortar campos.

-d → se utiliza para indicar que carácter hace de separación entre campos, es decir el delimitador.

Si no se indica nada, el **delimitador es el tabulador**. Para que cut sepa qué campos o columnas tiene que cortar, hay que indicarlo con una lista. Esta lista tiene tres formatos posibles:

1-2 Campos o columnas de 1 a 2, es decir, un rango.

1-	Campo o columna 1 y toda la línea.
1, 2	Campos o columnas 1 y 2.

Veamos a editar un fichero de texto llamado **amigos** con el siguiente contenido:

```
ANT:3680112
SEX:6789450
COM:3454327
VIM:4532278
DAO:5468903
```

Hemos cortado las **tres primeras letras de cada línea** del fichero **amigos**.

```
paco@ubuntupaco:~$ cut -c 1-3 amigos
ANT
SEX
COM
VIM
DAO
```

A continuación, vamos a cortar campos individuales. El carácter **delimitador** del fichero **amigos** es **:** y vamos a obtener el primer campo en el primer caso y el segundo campo en el segundo caso.

```
paco@ubuntupaco:~$ cut -d ':' -f 1 amigos
ANT
SEX
COM
VIM
DAO

paco@ubuntupaco:~$ cut -d ':' -f 2 amigos
3680112
6789450
3454327
4532278
5468903
```

Vamos a filtrar el fichero **passwd**, que se encuentra en **/etc** y vamos a seleccionar los **usuarios que utilizan el interprete de ordenes bash**.

```
paco@ubuntupaco:~$ cat /etc/passwd | grep bash | cut -d ':' -f 1,7
root:/bin/bash
bin:/bin/bash
daemon:/bin/bash
lp:/bin/bash
news:/bin/bash
uucp:/bin/bash
games:/bin/bash
man:/bin/bash
at:/bin/bash
ftp:/bin/bash
gdm:/bin/bash
nobody:/bin/bash
```

Hemos utilizado, en primer lugar **cat** para pasar el **fichero passwd por la salida estándar**, después mediante un pipeline (pipe o tubería) lo hemos **filtrado con grep seleccionando solamente aquellas líneas que contienen la cadena bash**. Finalmente, **nos quedamos con los campos 1 y 7**.

El comando tr

El **comando tr** es un **filtro que se emplea como traductor**, generalmente para convertir de minúsculas a mayúsculas y viceversa. Lo primero editamos un fichero de texto de nombre **gnu** que tenga el siguiente contenido:

```
Gnu/linux es un sistema operativo
QUE PRESENTA UN GRAN FUTURO
```

Y ejecutamos las siguientes acciones en la consola:

```
paco@ubuntupaco:~$ tr [A-Z] [a-z] < gnu
gnu/linux es un sistema operativo
que presenta un gran futuro
```

Hemos convertido todos los caracteres entre A y Z en sus homólogos entre a y z. Si queremos hacer lo contrario, escribimos:

```
paco@ubuntupaco:~$ tr [a-z] [A-Z] < gnu
GNU/LINUX ES UN SISTEMA OPERATIVO
QUE PRESENTA UN GRAN FUTURO
```

Ahora vamos a ver como sustituir caracteres. Todas las **mayúsculas** serán **sustituidas** por el **carácter g**.

```
paco@ubuntupaco:~$ tr [A-Z] g < gnu
gnu/linux es un sistema operativo
ggg ggggggggg gg gggg gggggg
```

Podemos modificar el **rango de caracteres implicados** si en lugar de utilizar [A-Z] utilizamos [A-M], es decir, solo se van a convertir en el carácter g las mayúsculas entre la A y la M.

```
paco@ubuntupaco:~$ tr [A-M] g < gnu
gnu/linux es un sistema operativo
QUg PRgSgNTg UN gRgN gUTURO
```

También podemos utilizar las **técnicas de sustitución para eliminar determinados caracteres**. Se utiliza la **opción -d**. Todas las mayúsculas han sido eliminadas. Además, **tr no afecta al fichero de entrada**. Lo toma, lo procesa y envía el resultado al monitor. El original no es modificado en ningún caso.

```
paco@ubuntupaco:~$ tr -d [A-Z] < gnu
nu/linux es un sistema operativo
```

El comando tee

Tiene como misión **bifurcar la salida de una orden enviando los resultados simultáneamente a la salida estándar (monitor) y a un fichero**. Su sintaxis es:

tee [-a] archivo(s)

La **opción -a** significa **append** o sea **añadir**. Con esta opción la salida a un fichero **no sobreescribe la información** sino que la añade al final del fichero de texto.