GREP, EXPRESIONES REGULARES, AWK Y SED

versión 1 - 12-9-2016

1 En primer lugar crea el fichero `prueba` copiando el siguiente código en la terminal:

```
cat > prueba << "FIN"
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu
abcdefghijklmnopqrstuvwxyz

1234567890
letras y   2435
a b c d z y x w
a b c  d z y x w
a b c   d z y x w
a b c    d z y x w
a b c     d z y x w
M
zzzzzzzzzz aaaaaaaaaaaaa
aaaaaaaaaa zzzzzzzzzzzzzz
ocho siete uno casa azul cielo mar
uno, dos, tres, cuatro, etc.
/bin/a/tmp/k
grep expreg awk y sed
FIN
```

2 Todos estos comandos (`grep`, `awk` y `sed`) trabajan línea a línea.

3 `grep` busca líneas que contengan un texto. Prueba estos comandos:

```
cat prueba
grep 1 prueba
grep d prueba
grep 'c d' prueba
grep 'cd' prueba
cat prueba| grep 1
grep U prueba
grep -i U prueba
grep -v U prueba
```

Observa que `ls` y `grep` interpretan el patrón de búsqueda de manera opuesta,

```
ls p
```

```
ls | grep p
```

`ls` busca el nombre `p` y `grep` busca un nombre que contenga una `p`.

4 Las expresiones regulares son conjuntos de caracteres y metacaracteres que permiten realizar búsquedas complejas. Hay dos sintaxis, la básica y la extendida. Ambas tienen la misma funcionalidad (en los programas GNU) y solo se diferencian en su sintaxis. Veremos la básica.

4.1 El punto `.` significa cualquier carácter (incluido el espacio):

```
grep 'c.d' prueba
```

4.2 El signo de cierre de interrogación `?` significa cero o una ocurrencia del carácter anterior (debe ser escapado con la contrabarra `\`):

```
grep 'c.\?d' prueba
```

El signo de suma `+` significa una o más ocurrencias del carácter anterior (debe ser escapado con la contrabarra `\`):

```
grep 'c.\+d' prueba
```

El asterisco `*` significa cero o más ocurrencias del carácter anterior:

```
grep 'c.*d' prueba
```

Fíjate bien en la diferencia entre `+`, `?` y `*`:

```
grep 'o\+' prueba
grep 'o\?' prueba
grep 'o*' prueba
```

Las llaves permiten indicar exactamente cuántas veces se repite el carácter anterior (deben ser escapadas con la contrabarra `\`):

```
grep 'c.\{2\}d' prueba
grep 'c.\{2,3\}d' prueba
grep 'c.\{2,\}d' prueba
```

4.3 Los corchetes permiten buscar la aparición de cualquiera de los caracteres que encierran:

```
grep 'letras' prueba
grep 'aetlrs' prueba
grep '[aetlrs]' prueba
```

y permiten buscar por rangos:

```
grep '[0-9]' prueba
grep '[a-z]' prueba
grep '[2-5]' prueba
grep '[6-9]' prueba
```

Si comienza por el acento circunflejo `^` busca cualquier carácter que no esté entre los corchetes:

```
grep '[M]' prueba
grep '[^M]' prueba
```

Cuidado no confundir:

```
grep '[6-9]' prueba
```

con

```
grep '[6,9]' prueba
```

En el segundo caso busca líneas que contengan un 6, una coma o un seis.

4.4 El acento circunflejo `^` también significa principio de línea (usado fuera de los corchetes) y el dólar `$` fin de línea:

```
grep '^1' prueba
grep 'w$' prueba
```

y juntos significan líneas vacías:

```
grep '^$' prueba
```

4.5 Ante comentamos que `ls` y `grep` buscan de forma opuesta:

```
ls p
ls | grep p
```

y para intercambiar su comportamientos sería:

```
ls *p*
ls | grep '^p$'
```

Como vemos `bash` interpreta el comodín de diferente manera que `grep` cuando se encuentra en una expresión regular. Para `bash` significa cualquier número de caracteres y en la expresión regular significa cualquier número de veces el carácter anterior.

4.6 Las expresiones regulares también se pueden usar en programas visuales e interactivos, como `OpenOffice`. Por ejemplo podemos usarlas para corregir las separaciones entre palabras que contengan más de un espacio usando la expresión “ {2,}”. No escribas las comillas y fíjate que comienza con un espacio.

5 `awk (gawk)`: busca patrones como `grep`, modifica ficheros como `tr` o `sed`, cuenta líneas como `wc`, une campos como `join` y `paste`, imprime los campos deseados como `cut` y además es un lenguaje de programación completo con variables, bucles y condicionales por lo que se pueden resolver múltiples problemas con él. Aquí veremos las opciones más básicas.

Su sintaxis es:

```
awk '/patrón/ {acción}' fichero
```

Por ejemplo, busca la línea que tenga el texto “`ocho`” y la imprime:

```
awk '/ocho/ {print $0}' prueba
```

La acción por defecto es imprimir:

```
awk '/ocho/' prueba
```

y si no indicamos un patrón de búsqueda aplica la acción a todas las líneas:

```
awk '{print $0}' prueba
```

`$0` significa toda la línea, `$1` el primer campo, `$2` el segundo, etc.:

```
awk '/ocho/ {print $2}' prueba
```

Podemos cambiar el separador de campo (que por defecto es el espacio) a, por ejemplo, la coma:

```
awk -F , '{print $2 $3}' prueba
```

y podemos usar las expresiones regulares como con `grep`:

```
awk '/[0-9]/' prueba
```

6 `sed` modifica ficheros de texto.

6.1 Su sintaxis es semejante a `awk`:

```
sed '/patrón/ acción' fichero
sed 'línea acción' fichero
```

y no hace falta dejar espacio:

```
sed '/patrón/acción' fichero
sed 'líneaacción' fichero
```

Aunque no haga nada siempre imprime todo el fichero:

```
sed '' prueba
```

Por eso se suele usar con la opción `n`, que elimina toda la salida excepto la que le digamos con el comando `p`, por ejemplo imprimimos solo la línea que tiene la palabra `ocho`:

```
sed -n '/ocho/p' prueba
```

o imprimimos la tercera línea:

```
sed -n '3p' prueba
```

Este comando imprime dos veces la tercera línea:

```
sed '3 p' prueba
```

y este borra la tercera línea:

```
sed '3 d' prueba
```

luego el comando `p` imprime y el `d` borra (delete).

Si no indicamos patrón ni línea la acción se realiza sobre todas las líneas:

```
sed -n 'p' prueba
sed 'd' prueba
```

6.2 Cuando queremos modificar un fichero redireccionamos la salida:

```
sed 's/awk/gawk/' prueba > prueba_mod
```

o usamos la opción que sobrescribe el fichero original:

```
sed –i 's/awk/gawk/' prueba
```

6.3 La coma permite indicar rangos entre líneas o patrones:

```
sed –n '2,5 p' prueba
sed –n '/zzz/, /ocho/ p' prueba
```

6.4 También podemos usar expresiones regulares y, por ejemplo, buscar líneas vacías:

```
sed –n '/^$/ p' prueba
```

o la palabra `uno` al principio de línea:

```
sed –n '/^uno/ p' prueba
```

o que no esté al principio de línea:

```
sed –n '/.\+uno/ p' prueba
```

6.5 El comando `s/texto1/texto2/` sustituye, en una línea, la primera aparición de `texto1` por `texto2` y `s/texto1/texot2/g` sustituye todas las apariciones. Por ejemplo, el siguiente comando sustituye en la primera línea la primera aparición de U por u:

```
sed '1 s/U/u/' prueba
```

y este sustituye todas la apariciones:

```
sed '1 s/U/u/g' prueba
```

y si no queremos distraernos con la salida ejecutamos dos acciones con la opción `e`:

```
sed –n –e '1 s/U/u/g' –e '1p' prueba
```

y evidentemente el orden importa:

```
sed –n –e '1p' –e '1 s/U/u/g' prueba
```

Ahora en la línea que tenga la palabra `ocho` sustuimos `siete` por `SIETE`:

```
sed '/ocho/ s/siete/SIETE/' prueba
```

Y si no indicamos la línea o el patrón `sed` realiza la sustitución en todas las líneas:

```
sed ' s/a/****************/' prueba
```

6.6 También podemos usar expresiones regulares como con `grep`. Por ejemplo buscamos las líneas que acaban con espacio:

```
sed –n '/ $/p' prueba
```

y los borramos:

```
sed –n 's/ $//g' prueba > prueba_mod
```

6.7 Buscamos las líneas que tienen más de un espacio entre `c` y `d`:

```
sed –n '/c \{0,5\}d/ p' prueba
```

y los eliminamos:

```
sed  's/c \{0,5\}d/c d/g' prueba
```

6.8 Podemos escoger el separador que queramos en el comando `s`, el carácter que sigue a la `s` es el separador para `sed`, por ejemplo los dos puntos:

```
sed '1 s:U:u:' prueba
```

Resulta útil en comandos liosos:

```
sed 's/\/bin\/a\/tmp\/k/\/sbin\/pmt\/w/' prueba
```

donde queremos cambiar `/bin/a/tmp/k` por `/sbin/pmt/w` y debemos escapar las barras de la ruta. En este caso se lee mejor usando otro separador y además no tenemos que escapar la barra:

```
sed 's:/bin/a/tmp/k:/sbin/pmt/w:' prueba
```

6.9 El signo et (`&`) significa el patrón buscado:

```
sed 's/ocho/& menos & es igual a cero/g' prueba
```

Los paréntesis (escapados) permiten crear átomos en el patrón de búsqueda y luego referenciarlos como `\1` para el primer átomo, `\2` para el segundo, etc.

Por ejemplo, para intercambiar `abc` y `def`:

```
sed 's/\(abc\)\(def\)/\2\1/' prueba
```

o para intercambiar `a/tmp` por `tmp/a`:

```
sed 's:\(a\)\/\(tmp\):\2\/\1:g' prueba
```

veamos el comando anterior con más detalle:

```
sed 's: \( a \) \/ \( tmp \) : \2 \/ \1 :g' prueba
```

Si queremos intercambiar `siete` y `cielo`:

```
sed 's/\(.*\)\(siete\)\(.*\)\(cielo\)\(.*\)/\1 \4 \3 \2 \5/g' prueba
```

como vemos creamos cinco átomos y luego los reordenamos conforme nos interesa.

7 La versatilidad de estos programas para manipular texto se ve multiplicada si los usamos en guiones de bash.