Capítulo 5

Expresiones regulares

5.1. Expresiones regulares

Una expresión regular no es más que un conjunto de caracteres que definen un "patrón" de texto, con el que debe concordar.

Las expresiones regulares son utilizadas por utilidades del sistema, tales como los editores `ed`, `vi`, `ex`, y `emacs`, y los filtros `grep`, `egrep`, `sed` y `awk`, entre otros. La mayoría de estas órdenes buscan en un fichero o en la entrada para encontrar una cadena o secuencia de caracteres, que se ajusten a un patrón. Cuando se ha encontrado se dice que la expresión regular concuerda, equivale a, o coincide con la cadena. El uso de las expresiones regulares depende de la orden.

Una expresión regular no es más que una cadena en la que distinguimos dos tipos de caracteres:

Caracteres ordinarios Sólo concuerdan consigo mismos.

Caracteres especiales Tienen un significado especial para el programa que los interpreta.

Hay caracteres especiales que tienen un significado especial independientemente del lugar de la cadena donde aparezcan, estos son: `. * [ \`. Sin embargo hay otros que sólo tienen un significado especial cuando se sitúan en una posición determinada; el carácter `^` sólo tiene un significado especial al comienzo de una cadena y el carácter `$` sólo lo tiene cuando va al final.

Si queremos usar un carácter especial dentro de una cadena como carácter ordinario deberemos protegerlo.

A continuación veremos los distintos tipos de expresiones regulares que podemos encontrar y el significado de los caracteres especiales.

5.1.1. Expresiones regulares básicas

Las expresiones regulares básicas sólo hacen uso de los caracteres especiales que se describen en el cuadro 5.1.

Símbolo	Significado
`\`	Anula el significado especial del carácter que le sigue.
`.`	Concuerda con cualquier carácter simple.
`*`	Concuerda con cero o más ocurrencias de la expresión regular precedente.
`$`	Sólo tiene un significado especial cuando es el último carácter de la expresión regular. Concuerda con el final de la línea.
`^`	Sólo tiene significado especial cuando es el primer carácter de la expresión regular. Concuerda con el comienzo de la línea.
`[`*conjunto*`]`	Un conjunto de caracteres entre corchetes, concuerda con un carácter simple del *conjunto*. Dentro de un conjunto sólo tienen significado especial los caracteres `-`, `]`, `^`. La notación [*c1-c2*] indica el conjunto de caracteres ASCII comprendidos en el rango de *c1* a *c2*. La secuencia `[^`*conjunto*`]` concuerda con cualquier carácter simple que no esté en el *conjunto*.

Cuadro 5.1: Expresiones regulares básicas

Ejemplos:

1. La expresión regular `$9\.98` concuerda con `$9.98` ya que hemos anulado el significado especial del carácter punto. Observe, además, que el carácter `$` no tiene significado especial, ya que no va al final de la expresión regular.
2. `.ora` concuerda con `hora`, `ahora`, `espora`, no concuerda con `ora`.
3. `Pr*` concuerda con `P`, `Pr`, `Prr`, `Pato`, `Perro`.
4. `.*` concuerda con cualquier conjunto de caracteres.

5. `g.*d` concuerda con cualquier cadena que comience por `g` y termine en `d`.
6. `\.$` concuerda con cualquier línea que termine en punto.
7. `^[A-Z]` concuerda con cualquier línea que comience por letra mayúscula.
8. `^abc$` concuerda con las líneas que únicamente contienen `abc`.
9. `[AEIOUa-m]` concuerda con una vocal en mayúsculas o bien una letra comprendida en el rango de la `a` a la `m`.
10. `[^0-9]` concuerda con cualquier carácter que no sea un dígito.

5.1.2. Expresiones regulares extendidas

Las expresiones regulares extendidas pueden hacer uso de los caracteres especiales que forman parte de las expresiones regulares básicas más los que aparecen en el cuadro 5.2. Los paréntesis `( )` tienen la máxima prioridad, seguidos del grupo `*`, `?`, `+` y `|`.

Símbolo	Significado
`+`	Concuerda con una o más ocurrencias de la expresión regular precedente.
`?`	Concuerda con cero o una ocurrencia de la expresión regular precedente.
expr1 `|` *expr2*	Concuerda con cualquiera de las dos expresiones regulares separadas por `|`.
`(`*expr*`)`	Se utiliza para agrupar una expresión regular, por lo que `*`, `?`, `+`, `|`, podrán aplicarse a la expresión completa.

Cuadro 5.2: Expresiones regulares extendidas

Ejemplos:

1. La expresión `per+o` concuerda con `perro`, `perrro`, `perrrro`.
2. La expresión `aab?cc` concuerda con `aacc`, `aabcc`.
3. La expresión `hola|adios` concuerda con `hola` o con `adios`.
4. La expresión `a(bc)?` concuerda con `a`, o con `abc`.

Ejercicios:

1. Dado el siguiente fragmento de texto:

   > *Desde enero de 1993 ha estado vigente en España el índice de Precios de Consumo (IPC) base 92. Desde enero de 2002 la metodología del IPC ha sido completamente renovada y se ha introducido un nuevo sistema de cálculo, IPC-2001.*

   Indique las concordancias que existen con las expresiones regulares que se dan continuación:

   a) `[Ii][Pp].`
   b) `s[aeiou]*t`
   c) `\.$`
   d) `^[Dd].*\.$`
   e) `[a-z]*002`

5.2. La familia `grep`

La familia `grep` está formada por tres filtros: `grep`, `egrep`, y `fgrep`. En general, esta familia de programas selecciona aquellas líneas de la entrada, que contienen los caracteres que concuerdan con la cadena de caracteres o la expresión regular especificada, y las copia normalmente en la salida estándar.

Como ya sabemos, una expresión regular es una cadena que contiene caracteres especiales que son interpretados por los programas correspondientes.

La familia `grep` está formada por tres programas cuyos formatos son:

- `grep` [*opciones*] *expresión_regular* [*fichero ...*]
- `egrep` [*opciones*] *expresión_regular* [*fichero ...*]
- `fgrep` [*opciones*] *cadenas* [*fichero ...*]

Las diferencias fundamentales entre las distintas órdenes son:

`fgrep` Sólo admite como patrón de búsqueda cadenas literales.

`grep` Admite expresiones regulares básicas.

`egrep` Admite expresiones regulares básicas y extendidas.

Todas las órdenes anteriores admiten las opciones que se citan a continuación.

Opciones:

`-n` Las líneas seleccionadas van precedidas del número de línea que le corresponde dentro del fichero.

Ejemplo:
```
$ grep -n man fichero
```

`-v` Muestra las líneas que no concuerdan con la expresión regular o con la cadena.

Ejemplo:
```
$ grep -v /bin/sh /etc/passwd
```

`-c` Visualiza sólo el número de líneas que contienen o concuerdan con la expresión regular o la cadena especificada.

Ejemplo:
```
$ grep -c /bin/sh /etc/passwd
```

`-i` Ignora la distinción entre mayúsculas y minúsculas en la búsqueda.

Ejemplo:
```
$ grep -i Man fichero
```

`-w` Busca palabras completas.

`-f` *fichero* Toma la cadena a buscar o la expresión regular del *fichero* especificado. Permite buscar simultáneamente varias cadenas o expresiones regulares. Previamente tendremos que crearnos el fichero con las cadenas o expresiones regulares que queramos buscar.

Ejemplo:
```
$ egrep -f patrones /etc/group
```

`-l` Muestra sólo el nombre de los ficheros con líneas coincidentes con la expresión; no muestra las líneas.

Ejemplo:
```
$ grep -il division ~/cobol/*
```

`-L` Muestra sólo el nombre de aquellos ficheros en los que no encuentre ninguna concordancia con la expresión regular.

`-b` Antepone a cada línea que se muestra en pantalla el número de caracteres que la preceden en el fichero de entrada.

5.2.1. La orden `grep`

Su formato es:

`grep` [*opciones*] *expresión_regular* [*fichero ...*]

Busca en la entrada estándar o en el *fichero* indicado las líneas que contienen la cadena de caracteres que concuerda con la expresión regular especificada.

Admite expresiones regulares básicas. Se pueden utilizar los caracteres comodines en los nombres de ficheros.

Ejemplos:

1. `$ grep hola *`
   Busca la expresión `hola` *en todos los ficheros del directorio de trabajo.*
2. `$ grep "la pera en dulce" frutas`
3. `$ grep 'sh$' /etc/passwd`
   Busca en el fichero `/etc/passwd` *todas aquellas líneas que terminen en* `sh` *y las copia en la salida estándar.*
4. `$ grep ^From $MAIL | grep -v pepe`
   Del fichero especificado en la variable `MAIL` *extrae las líneas que empiecen por* `From`*; la salida es filtrada por* `grep` *y se seleccionan las líneas que no contengan* `pepe`*.*
5. `$ ls | grep -v '~$'`
   Listado de todos los ficheros que no terminen en `~`*.*
6. `$ ls -l | grep '^.......rw'`
   Lista todos los ficheros que tengan permiso de lectura y escritura para los otros.

5.2.2. La orden `egrep`

Busca aquellas líneas de la entrada que contengan una o varias de las expresiones regulares dadas. Es la orden más potente de la familia. Admite expresiones regulares básicas y extendidas. El algoritmo de búsqueda de `egrep` consume gran cantidad de memoria. Su formato es:

`egrep` [*opciones*] *expresion_regular* [*ficheros ...*]

Ejemplo:

```
$ egrep 'mail|ftp' /etc/passwd
```

Busca en el fichero `/etc/passwd` *todas las líneas que contengan las expresiones* `mail` *o* `ftp` *y las copia en la salida estándar.*

5.2.3. La orden `fgrep`

Busca en la entrada aquellas líneas que contienen las cadenas especificadas. No admite expresiones regulares, pero sí más de una cadena. El algoritmo de búsqueda de `fgrep` es más rápido que el de `grep`. El formato de `fgrep` es:

`fgrep` [*opciones*] *cadena(s)* [*fichero ...*]

Para indicarle más de una cadena hay que separarlas por un `<intro>`.

Ejemplo:

```
$ fgrep 'Susana
  Samuel' telefonos
```

La existencia de tres programas: `grep`, `egrep` y `fgrep`, con funciones tan parecidas, es debida a razones históricas. En la actualidad se tiende a integrar los tres programas en uno solo (`grep`) que puede ser llamado con distintas opciones para que se comporte como uno de los tres anteriores. En nuestro sistema disponemos de la orden `grep` de GNU; ésta admite tres opciones: `-G`, `-F` y `-E`, aparte de las citadas anteriormente. Si llamamos a `grep` con la opción `-G` se comportará como el `grep` descrito anteriormente; éste es el comportamiento predeterminado. Si lo llamamos con la opción `-F` se comportará como `fgrep`, y si lo llamamos con la opción `-E` se comportará como `egrep`. Además podemos utilizar las órdenes `fgrep` y `egrep`, que son enlaces al programa `grep`.

Ejercicios:

1. ¿Qué hacen las siguientes órdenes?

   a) `$ ypcat passwd | fgrep ".*"`

   b)
   ```
   $ ls -l | grep '^-' | cut -d" " -f1,1 | \
     grep 'rwx$' | wc -l
   ```

   c) `$ grep -cw '....' /usr/share/dict/words`

   d) `$ cat /usr/share/dict/words | egrep '^(l|r)+.*a$`

2. ¿Cuál será el resultado de ejecutar la siguiente orden? ¿Cuál es la primera operación que realiza el *shell* en esta línea de órdenes?

   ```
   sort -o datos > datos
   ```

3. ¿Qué diferencias hay entre estas dos órdenes?

   a) `sort numeros`

   b) `sort -n numeros`

4. Obtenga la información acerca de los usuarios del sistema ordenada según el nombre de *login*.

5. ¿Qué hace la siguiente orden?

   ```
   sort -u equipo1 equipo2 > equipo3
   ```

6. Muestre los números de identificación de los usuarios y los grupos que existen en el sistema.

7. Créese un fichero que contenga las siguientes líneas:

   ```
   8 Miguel
   12 Carlos Severo
   9 Roberto Astola
   6 Julia Fernández
   4 Adolfo Domínguez
   10 Pitita Crucecitas
   ```

   Ordene el fichero y muestre el resultado en la salida estándar. Ordénelo numéricamente. Compare los resultados.

8. Sobre el fichero anterior, escriba una orden que muestre sólo los nombres de las personas en mayúsculas.

9. ¿Cuántos usuarios del sistema tienen un nombre de usuario que empieza por `m`?

10. ¿Cómo comprobaría si el usuario `pepe` está conectado al sistema?

11. Obtenga una lista de todos los subdirectorios de su directorio de trabajo.

12. Obtenga una lista de los ficheros de su directorio de trabajo que tienen permiso de lectura para el grupo.

13. ¿Qué hacen estas órdenes?

    a) `$ egrep A[0-9]+Q numserie`

    b) `$ ypcat passwd | egrep '[Cc]onde|[Mm]ario'`

    c) `$ egrep -c 'DIVISION|SECTION' ~/cobol/*`

    d) `$ who | egrep 'tty?'`

14. Diga con qué concuerdan las siguientes expresiones regulares:

    a) `[a-z]+[0-9]`

    b) `[a-z]+[0-9]?`

    c) `[a-z]*[0-9]`

    d) `Tony|Antony`

    e) `(Tony|Antony) Quinn`

    f) `Tony|(Antony Quinn)`

    g) `(so?)+`

15. Combine dos ficheros, ordénelos y elimine las líneas duplicadas.

16. Créese un fichero con los siguientes valores:

```
-40
14.8
14,8
-12.6
0
0.00
-9
```

    Ordénelos primero sin emplear ninguna opción. Segundo con la opción `-n`. tercero con las opciones `-nu`. Compare los resultados.

17. ¿Qué hacen estas órdenes?

a) `$ who | grep "06"`

b) `$ ls -l | grep ^d`

c) `$ ypcat passwd | grep '^[^:]*::'`

d) `$ grep -c '......' palabras`

18. Obtenga en la salida estándar una lista de todos los usuarios del sistema cuyo nombre de *login* empiece por `n` y que utilicen el *shell* bash.

19. Escriba en una sola línea una orden que busque a partir de su directorio de entrada, todos los ficheros que terminen en `.c` y los imprima ordenados alfabéticamente.

20. A partir del fichero `/usr/share/dict/words`, créese otro llamado `WORDS` donde todas las letras minúsculas se conviertan en mayúsculas.