Asignatura: Entornos de programación
Expresiones regulares
Herramientas Grep y AWK

En este tema se introducen las expresiones regulares y un par de utilidades que las usan de manera intensiva: 'grep' y 'awk'. Estas utilidades proceden del mundo UNIX, y están disponibles hoy día en todas las plataformas usadas habitualmente.

- **Grep**: Su nombre se toma de la expresión "***G****lobally search for* ***R****egular* ***E****xpression and* ***P****rint*", que describe una de las funciones disponibles en el editor "ed" de UNIX. Dicha función aparece documentada en el manual como `g/re/ p`. La función principal de la utilidad "grep" es extraer de un fichero de texto las líneas que cumplen con un patrón dado por una expresión regular.
- **AWK**: Su nombre se obtiene de las iniciales de los apellidos de sus autores: Alfred **A**ho, Peter **W**einberger y Brian **K**ernighan. El nombre se aplica a un lenguaje de *script* (AWK) y al procesador que lo interpreta (awk). Está orientado al proceso de ficheros de texto que contienen uno o varios campos de datos en cada línea. Usa expresiones regulares para seleccionar las líneas de texto deseadas, y las acciones a realizar se escriben como en lenguaje C.

1. Expresiones regulares

Las expresiones regulares permiten describir estructuras sintácticas, en particular las de los lenguajes regulares. Son, por tanto, patrones sintácticos a los que deben ajustarse los elementos que componen el vocabulario del lenguaje.

Dicho de una manera más sencilla, las expresiones regulares son patrones que permiten reconocer secuencias de símbolos con una estructura sintáctica determinada.

Hay diversas notaciones para representar expresiones regulares. Por ejemplo, la notación BNF y otras derivadas de ella, o los comodines (*wildcards*) usados para designar conjuntos de ficheros.

La notación que se describe aquí permite escribir patrones para secuencias de caracteres (*strings*), y corresponde a la que con ligeras variantes se usa en diversas utilidades para proceso de textos: "grep", "awk", "perl", la librería "regexp", etc. Los apartados siguientes describen los elementos principales. Para una descripción más completa se debe consultar el manual de referencia correspondiente.

1.1 Expresiones simples

Son patrones que se ajustan a un único símbolo (en general, a un único carácter):

Expresión	Significado
`x`	carácter `x`, si es carácter normal
`.`	cualquier carácter
`[aeiou]`	un carácter del conjunto

`[a-z]`	un carácter del rango
`[^aeiou0-9]`	complementa el conjunto
`\x`	carácter x, si x es un carácter especial
`^`	principio del texto, si va al comienzo
`$`	fin del texto, si va al final
`\<`	principio de palabra (*)
`\>`	fin de palabra (*)

(*) Patrón no estándar

1.2 Expresiones compuestas

Son patrones que combinan expresiones simples. Se ajustan a una secuencia de símbolos.

Expresión	Significado
`xy`	expresión `x` seguida de `y`
`x+`	una o más repeticiones de `x`
`x*`	cero o más repeticiones de `x`
`x?`	cero o una aparición de `x`
`una|otra`	una u otra expresión
`(x)`	expresión `x`

1.3 Ejemplos de expresiones regulares

- Un dígito numérico, decimal: `[0-9]`
- Una vocal: `[AEIOUaeiou]`
- Una letra, mayúscula o minúscula: `[A-Za-z]`
- Una palabra que empieza por letra y puede contener números: `[A-Za-z][A-Za-z0-9]*`
- Un número, con punto decimal, al comienzo del texto: `^[+\-0-9][0-9]*\.[0-9]*`
- Abreviatura del nombre de un mes: `(Ene|Feb|Mar|Abr|May|Jun|Jul|Ago|Sep|Oct|Nov|Dic)`

2. Herramienta Grep

Es un programa de utilidad que permite seleccionar las líneas de un texto que cumplen con un patrón determinado. La orden para invocarla es:

```
> grep patrón ficheros...
```

El patrón se escribe como una expresión regular. Los nombres de los ficheros de entrada pueden incluir comodines. El programa lee los ficheros y copia a la salida cada línea que se ajuste al patrón.

La utilidad grep procede de UNIX. Dispone de parámetros opcionales para controlar su funcionamiento. Para conocer las posibles opciones se puede invocar como:

```
> grep --help
> grep -h
> grep
```

Ejemplo: extraer todas las líneas de código C/C++ que empiecen con un comentario (`//` o `/*`) desde la primera posición.

```
> grep ^/[/\*] *.h *.c *.cpp
```

3. Lenguaje AWK

AWK es un lenguaje para procesar ficheros de texto, línea a línea. Resulta apropiado para extraer datos individuales, realizar recuentos, modificar el formato de los datos, generar resúmenes, etc. La potencia del lenguaje reside en el uso extensivo de expresiones regulares para seleccionar los fragmentos de información apropiados, y la posibilidad de combinar los estilos de programación declarativo e imperativo.

3.1 Elementos básicos

- Un programa consiste en una colección de cláusulas o reglas, cada una de las cuales sigue el esquema:

```
patrón { acción }
```

- Cada cláusula indica que si se cumple el patrón se debe realizar la correspondiente acción.
- El patrón puede omitirse. Es este caso es como si hubiera un patrón que se cumple siempre.
- La acción puede omitirse. En este caso es como si hubiera una acción `{print}` (imprimir línea).
- La ejecución de un programa AWK consiste en leer los ficheros de entrada línea por línea y aplicar a cada línea la colección de cláusulas, por su orden.
- Los datos pueden ser valores numéricos o de texto, y se convierten automáticamente de un formato a otro cuando sea necesario.
- El patrón puede ser una expresión regular. Se cumple si la línea de datos se ajusta a dicho patrón.
- El patrón puede ser también una expresión. El patrón se cumple si el valor resultante (número o texto) no es cero o nulo (`0` o `""`)
- Las acciones se escriben como en lenguaje C
- No hay que declarar las variables (se crean al usarlas por primera vez, con valores nulos)
- Los valores literales de texto se escriben como en C, entre comillas dobles: `"texto"`. Pueden incluir secuencias de escape (`\n \t \\` ...).
- Las funciones se invocan como `función(argumentos)`.
- El texto de un programa puede incluir comentarios, empezando con el carácter `#` y hasta el final de la línea.

3.2 Ejemplo: recuento de notas

Para dar una idea rápida de cómo es un programa en AWK, a continuación se presenta como ejemplo el proceso de una lista de calificaciones de un examen para obtener un resumen estadístico simple. El significado de los elementos que aparecen en el programa se irá viendo en los siguientes apartados.

Fichero de datos - notas de un examen (DNI del alumno y nota numérica):

```
051422949  4.3
```

```
051943388  8
005428776  7.5
052970557
052375629  3
002550123  9.5
014301873
050100456  6
079309554  5.5
002915589
008928257  8.7
```

Programa - recuento de aprobados y suspensos:

```
$1 {alumnos++}

$2 >= 5 {aprobados++}

NF==1 {nopresentados++}

END {
  suspensos = alumnos - aprobados - nopresentados
  print "Aprobados:     ", aprobados
  print "Suspensos:     ", suspensos
  print "No presentados:", nopresentados
  print "Total alumnos: ", alumnos
}
```

Resultados:

```
Aprobados:      6
Suspensos:      1
No presentados: 4
Total alumnos:  11
```

3.3 Patrones

En AWK hay varias clases de patrones. Todos ellos funcionan como condiciones con un resultado booleano.

- Expresiones regulares. Se escriben como `/expresión/`, entre barras. La condición se cumple si la línea de datos de entrada se ajusta a esa expresión regular.
- Expresiones aritméticas (o de texto). Se escriben prácticamente igual que en lenguaje C. La condición se cumple si el resultado de evaluar la expresión es un valor de texto no nulo o un valor numérico distinto de cero.
- Patrones especiales. Algunos de ellos son:
  - La palabra clave `BEGIN`. La condición se cumple al principio del programa, antes de leer los datos de entrada.
  - La palabra clave `END`. La condición se cumple al final de todo el proceso, después de procesar todos los datos de entrada y justo antes de terminar

- Un par de patrones separados por una coma, de la forma `x,y`. Se denomina patrón compuesto. La condición se cumple para una secuencia correlativa de líneas de entrada desde una que cumpla el primer patrón `x` hasta la siguiente que cumpla el segundo patrón `y`, ambas inclusive.

3.4 Operadores

Como se ha dicho, son similares a los del lenguaje C. Los más importantes son:

- Operadores aritméticos: `+ - * / % ^`
- Operadores de comparación: `== != > >= < <=`
- Operadores de asignación: `= += -= *= /=`
- Operadores de incremento o decremento, en sus formas prefija y postfija: `++ --`
- Operadores lógicos: `! && ||`
- Operador de ajuste de patrón: `~`
- La operación de concatenación usa un operador implícito: `"uno" "dos"` → `"unodos"`
- También existe la expresión condicional: `x ? y : z`

Los valores numéricos y de texto son convertidos automáticamente de un tipo a otro cuando sea necesario. Cuando se quiera forzar una conversión se puede escribir:

- `x + 0` → valor de `x` convertido a tipo numérico
- `x ""` → valor de `x` convertido a texto

3.5 Funciones predefinidas

Entre otras, las siguientes:

- Funciones matemáticas (numéricas): `sqrt()`, `sin()`, `cos()`, `exp()`, ...
- Funciones de texto: `length()`, `substr()`, `index()`, `toupper()`, `tolower()`, `gsub()`, `match()`, ...
- Otras funciones: `system()`, `getline`, `systime()`, ...

3.6 Campos en las líneas de entrada

Cada línea de texto de la entrada se descompone automáticamente en campos. Por defecto, los campos se separan por espacio en blanco. Los campos pueden referenciarse por separado, mediante el operador `$`.

- `$n` → n-simo campo
- `$0` → toda la línea
- `$expresión` → la referencia a un campo puede ser calculada

Ejemplo:

- `$0 = "ejemplo de línea de texto"`
- `$1 = "ejemplo"`

- `$2 = "de"`
- `$3 = "línea"`
- `$4 = "de"`
- `$5 = "texto"`
- `k = 3 → $k = "línea"`

3.7 Variables simples

Una variable simple puede contener un valor numérico o de texto. Las variables no se declaran, simo que empiezan a existir cuando se usan por primera vez. Inicialmente tienen valor nulo.

3.8 Variables simples predefinidas

Existen variables predefinidas que sirven de comunicación entre el código del usuario y el propio intérprete de AWK. Entre otras las siguientes:

- `NF` → número de campos
- `NR` → número de la línea (global, aunque haya varios ficheros de entrada)
- `FNR` → número de la línea (local al fichero)
- `FILENAME` → nombre del fichero actual

Las variables anteriores toman valor automáticamente con cada línea. Otras variables son asignadas por el usuario para controlar el funcionamiento del intérprete. Por ejemplo:

- `FS` → separador de campos de entrada (por defecto `" "`)
- `RS` → separador de líneas de entrada (por defecto `"\n"`)
- `OFS` → separador de campos en la salida (por defecto `" "`)
- `ORS` → separador de líneas en la salida (por defecto `"\n"`)

3.9 Acciones

Las acciones son sentencias o secuencias de sentencias que se escriben como en lenguaje C. El código de una sentencia termina implícitamente con el fin de línea o explícitamente con punto y coma (`;`). Algunas sentencias de uso frecuente son:

Sentencia	Significado
`var = expresión`	asignación de valor a una variable
`expresión`	evaluación de la expresión (la asignación anterior es realmente un caso particular de expresión, usando el operador de asignación)
`if (condición) acción`	acción condicional
`if (condición) acción else acción`	acciones alternativas

`while (condición) acción`	bucle WHILE
`for (k=ini; k<=fin; k++) acción-con-k`	bucle FOR (como en C)
`{ sentencia; sentencia ... }`	acción compuesta
`print expresión, expresión ...`	imprime valores separados por espacio en blanco
`print`	equivale a `print $0`
`printf( formato, expresión, expresión ... )`	impresión con formato (como en C)
`print ... > fichero`	redirige la salida al fichero
`getline variable < fichero`	lee explícitamente una línea de un fichero
`close( fichero )`	libera el fichero
`next`	termina el proceso de la línea de entrada actual
`exit`	termina la lectura de la entrada y da control a las acciones `END` (o bien termina definitivamente)

3.10 Vectores o tablas (*arrays*)

La única estructura de datos que existe en AWK son los vectores o tablas (en inglés *array*). También se les llama vectores asociativos. Tienen las siguientes características

- Son una colección de pares: (clave, información)
- Se usa la notación de *array* de C: `tabla[clave]` → `información`
- Las claves admiten valores de cualquier tipo (números o texto)
- Las tablas no se declaran. Los elementos empiezan a existir la primera vez que se hace referencia a cada uno
- Para eliminar un elemento no basta asignarle valor nulo, sino que hay que destruirlo explícitamente: `delete tabla[clave]`
- La tabla completa puede destruirse mediante: `delete tabla`
- Para consultar si existe un elemento dado se escribe: `clave in tabla`
- Hay una variante del bucle FOR que permite recorrer una tabla: `for (clave in tabla) acción-con-clave`

3.11 Vectores (*arrays*) predefinidos

Existen algunos vectores predefinidos que permiten acceder a información del ambiente de ejecución:

- `ARGV[0..ARGC-1]` → argumentos de llamada a "awk" (lista de ficheros, no las -opciones o -*switches*)

- `ENVIRON[nombre]` → valor de una variable de entorno

3.12 Funciones definidas por el usuario

Además de las funciones predefinidas el usuario puede definir sus propias funciones. La definición de una función tiene el mismo aspecto que una cláusula o regla del programa, usando un patrón especial que empieza con la palabra clave `function`.

- `function nombre( argumento, argumento, ...) { acción-con-argumentos }`
- no debe haber espacio en blanco entre el nombre y el parentesis
- el valor de la función se devuelve con la sentencia: `return valor`
- en el código de la acción, los nombres de variables que no sean argumentos se interpretan como variables globales
- para tener variables locales se usa el artificio de añadir argumentos adicionales, no usados en la llamada
  - `function nombre( arg1, arg2,      aux1, aux2 )`
  - se invoca como: `nombre( valor1, valor2 )`

3.13 Invocar un programa AWK

Los programas en AWK se ejecutan bajo intérprete. Por lo tanto para usar un programa hay que invocar el intérprete correspondiente, indicando cuál es el programa AWK a ejecutar y cuáles son los ficheros de datos de entrada. El formato de la orden es uno de los siguientes:

```
> awk  "programa"  fichero fichero ...
> awk  -f programa  fichero fichero ...
```

El primer caso sólo es útil para programas muy cortos. El texto del programa se escribe literalmente, entre comillas, en la propia orden, antes de los nombres de los ficheros de entrada. En el segundo caso el programa se almacena en un fichero de texto y el nombre de dicho fichero se indica con la opción `-f`. La orden puede incluir también una asignación de valores iniciales a ciertas variables del programa, con la opción `-v`:

```
> awk  ... -v variable=valor ...
```

El siguiente ejemplo usa "awk" para emular el funcionamiento de la herramienta "grep":

```
> awk  "/^[0-9]/" *.txt
```

Esta orden imprime (acción por defecto) cada línea que empiece (`^`) por un carácter numérico (`0-9`) en cualquier fichero de texto (`*.txt`) que haya en el directorio actual.

El nombre de la orden debe corresponderse con el intérprete concreto que se utilice. Hay distintos intérpretes para distintas variantes de AWK: `awk`, `nawk`, `mawk`, `gawk`, `tawk`, etc.

4. Ejemplo: vocabulario usado en un texto

Este programa imprime la lista de todas las palabras diferentes usadas en un texto, junto con la frecuencia de aparición de cada una. Para simplificar el programa se asume que no hay signos de puntuación, y las palabras están separadas por espacio en blanco o saltos de línea. Esto significa que cada palabra es un campo de la línea de entrada.

Código del programa (`vocabulario.awk`):

```
{
    for (k=1; k<=NF; k++) {
        cuenta[$k]++
    }
}

END {
    for (pal in cuenta) {
        print pal, cuenta[pal]
    }
}
```

Observaciones:

1. La primera cláusula del programa tiene el patrón omitido. Se aplica a todas las líneas del texto de entrada.
2. Las palabras en mayúsculas y en minúsculas se tratan como diferentes. Este es el comportamiento por defecto de AWK.
3. Los resultados no aparecen en orden alfabético (ni en ningún otro orden determinado). Las tablas suelen implementarse internamente como tablas "hash", y sus elementos se recorren en el orden que haya decidido quien construyó el intérprete.

Ejemplo de texto de entrada (`texto.txt`):

```
Este programa imprime la lista de todas las
palabras diferentes usadas en un texto, junto con
la frecuencia de aparición de cada una. Para
simplificar el programa se asume que no hay
signos de puntuación, y las palabras están
separadas por espacio en blanco o saltos de
línea. Esto significa que cada palabra es un
campo de la línea de entrada.
```

Orden de ejecución:

```
> gawk -f vocabulario.awk texto.txt
```

Resultados:

```
entrada. 1
Esto 1
no 1
```

```
frecuencia 1
palabra 1
el 1
con 1
palabras 2
todas 1
lista 1
imprime 1
programa 2
línea 1
Para 1
campo 1
que 2
... etc. ...
```