Práctica 4. Programación en shell. Introducción a awk.

1. Objetivos.

- Generar archivos de órdenes para automatizar trabajos.
- Utilizar *awk* como herramienta avanzada para procesar archivos.

2. Programación en *shell*.

Un archivo de texto que contenga un conjunto de líneas de órdenes Linux se conoce con el nombre de *script*. Cuando tiene permisos de ejecución activados, se ejecuta de igual manera que cualquier otro programa. Cuando no es así, es necesario llamar al intérprete de órdenes de la siguiente forma:

```
sh archivo_script
```

Donde sh es el intérprete de órdenes que se quiera utilizar (sh, ksh, csh, ...)

Introducción de comentarios y especificación del shell utilizado.

Todo el texto incluido detrás del carácter # (almohadilla) se considera un comentario.

Como excepción, en la primera línea del archivo se puede especificar el *shell* que se desea interprete las órdenes indicadas en el *script*:

```
Sintaxis:  #!ruta_del_shell
```

Así, para utilizar el C-shell:

```
#!/bin/csh
# Este script utiliza el C-shell
```

Empleo de variables.

Dentro de un *script* pueden utilizarse variables, siendo de especial interés las siguientes:

`$1, $2, ..., $9`	Son los diferentes parámetros de entrada posibles.
`$#`	Es el número total de parámetros introducidos desde la línea de órdenes.
`$?`	Toma valor 0 si la instrucción anterior se ejecutó correctamente, y valor distinto de 0 si no se ejecutó correctamente.

Así, por ejemplo, en:

```
programa parametro1 parametro2
```

⇒

$1 = parametro1
$2 = parametro2
$# = 2

Devolución de valores al shell: exit.

La orden exit finaliza la ejecución de un programa y devuelve un valor al *shell*.

Sintaxis: `exit [valor]`

Si no se le pasa ningún parámetro, el código de retorno es el de la última instrucción ejecutada. Si se le pasa como argumento un valor numérico entero, ese valor es el código de finalización del programa.

En general, un valor distinto de 0 representa que se ha finalizado debido a un error.

3. Procesamiento de archivos con *awk*.

Según Alfred V. Aho, Meter J. Weinberger y Brian W. Kernighan, creadores de *awk*, éste es un lenguaje de programación basado en emparejamiento de patrones (*pattern matching*) que puede aplicarse a una gran variedad de tareas de cálculo y manipulación de datos.

La función principal de *awk* consiste en buscar todas las líneas de un archivo que respondan a ciertos patrones y ejecutar una serie de acciones sobre ellas.

Cuando se ejecuta esta orden hay que especificar un programa *awk* que incorpore una serie de reglas, cada una de las cuales describe un patrón (que responde a una expresión regular) y una acción.

La sintaxis de una regla es la siguiente:

```
patrón {acción}
```

Si se omite el patrón, se ejecuta la acción para todas las líneas del archivo. Si se omite la acción, se envían a la salida por defecto todas las líneas en que se encuentre el patrón. No se pueden omitir simultáneamente el patrón y la acción.

Sintaxis:
```
awk  [-Fseparador] 'reglas' archivo
awk  [-Fseparador] -f fuentes archivo
```

Donde *separador* es el carácter separador de campos. Si no se especifica ningún separador, los campos se suponen separados por uno o varios espacios en blanco consecutivos. En caso de que el separador sea un carácter especial (como puede ser el carácter punto y coma, se debe eliminar su significado especial anteponiendo el carácter \ ).

El argumento *archivo* es el nombre del archivo en el que se encuentra la información y *fuentes* es un archivo de texto en el que se especifica el conjunto de reglas que se va a utilizar.

Ejemplo:

```
awk '/alumnos/ {print $0}' /etc/passwd
```

Vuelca en pantalla todas las líneas del archivo `/etc/passwd` que contienen la cadena *alumnos*. Este programa *awk* realiza la misma función que la orden

```
grep 'alumnos' /etc/passwd
```

De todas maneras, *awk* permite realizar operaciones mucho más sofisticadas que la orden *grep*.

Por ejemplo, sea *datos* un archivo de texto con el siguiente contenido:

```
Juan:22:informatica:juan@informatica.es
Pedro:21:telecomunicaciones:pedro@informatica.es
Sara:20:informatica:sara@informatica.es
Nerea:19:informatica:nerea@informatica.es
Jose:19:industriales:jose@informatica.es
Angel:20:telecomunicaciones:angel@informatica.es
Bernardo:21:industriales:bernardo@informatica.es
Luisa:18:informatica:luisa@informatica.es
```

Se puede interpretar como una tabla que utiliza el carácter dos puntos (`:`) para separar entre sí los campos de diferentes registros. El primer campo es el nombre de un alumno, el segundo su edad, el tercero los estudios que realiza y el cuarto su dirección de correo electrónico.

Si se quiere visualizar todas las líneas en las que aparecen estudiantes de informática, el empleo de la orden *grep* da problemas debido a que la palabra *informatica* aparece en todas las direcciones de correo-e.

Usar la orden *cut* seguida de *grep* no permite visualizar la línea completa, sino solamente la columna cortada.

En cambio, *awk* proporciona una solución sencilla al problema:

```
awk  -F: '$3~/informatica/ {print $0}' datos
```

Como efecto de la orden awk anterior, se recorren todas las líneas del archivo datos buscando aquellas que cumplan la condición:

`$3 ~ /informatica/` $\Rightarrow$ Condición cierta si el tercer campo (`$3`) contiene (`~`)el patrón `/informatica/`, es decir, si contiene los caracteres **`informatica`**.

Para cada línea que cumpla la condición anterior se ejecuta la acción `{print $0}`, que se encarga simplemente de enviar a la salida el contenido de la línea completa (`$0`).

Como resultado se visualizan todas las líneas en cuyo tercer campo aparezca la palabra informática, tal como se deseaba.

Campos y Registros.

Cada línea de un archivo es un registro. Se representa por la variable **$0**.

Se denominan campos a las cadenas de caracteres de un registro que están separadas por el separador de campos. Se representan por **$1**, **$2**, **$3**, ...

$NF es el último campo de un registro.

NF es el número de campos de un registro.

Ejemplos:

Si el resultado de la orden *who* es:

```
saf01      ttyq3        Mar 22 10:23  (n105160:0.0)
gsi01      ttyq4        Mar 20 19:08  (n105164:0.0)
ccs01      ttyq5        Mar 21 19:06  (n105161:0.0)
ccs01      ttyq8        Mar 21 19:50  (n105161:0.0)
```

Entonces, en el campo 1 se encuentra el nombre de usuario y en el 5 la hora a la que se conectó al sistema. Si se quieren visualizar esos dos campos solamente, se puede utilizar *awk*:

```
who|awk '{print $1, $5}'
```

El resultado es:

```
saf01 10:23
gsi01 19:08
ccs01 19:06
ccs01 19:50
```

Patrones awk.

Un patrón, dentro de la pareja patrón-acción, es una expresión condicional que determina si la acción se va a ejecutar o no. Los patrones son la representación de los elementos que se quieren buscar dentro de las líneas del archivo fuente.

Se pueden usar como patrones:

a. BEGIN y END.
b. Expresiones regulares.
c. Expresiones relacionales y aritméticas.
d. Cadenas que representen a números.

Los patrones BEGIN y END.

Son dos patrones especiales que, si se utilizan, deben aparecer en la primera línea y en la última del *script*, respectivamente.

La acción que acompaña a BEGIN se ejecuta siempre y antes de evaluar la primera línea del archivo. Se suele utilizar para inicializar variables y mostrar mensajes.

La acción que acompaña a END se ejecuta siempre y después de evaluar todas las líneas del archivo. Habitualmente se emplea para obtener y mostrar resultados finales tras operar con el archivo de datos.

La forma que tiene *awk* de trabajar con la información contenida en un archivo es:

1. Ejecutar la acción cuyo patrón es BEGIN.
2. Seleccionar la primera línea del archivo como línea actual.
3. Ejecutar todas las líneas del programa *awk* excepto las que tienen los patrones BEGIN y END utilizando como dato la línea actual.
4. Mientras no se encuentre el final del archivo, seleccionar la siguiente línea de dicho archivo como línea actual e ir nuevamente al paso 3.
5. Ejecutar la acción cuyo patrón es END.

Ejemplo:

```
awk 'BEGIN {print "Comenzando..."}
           {print $0}
     END   {print "...Procesamiento terminado"}' datos
```

Visualiza las cadenas de caracteres

```
Comenzando...
Juan:22:informatica:juan@informatica.es
Pedro:21:telecomunicaciones:pedro@informatica.es
Sara:20:informatica:sara@informatica.es
Nerea:19:informatica:nerea@informatica.es
Jose:19:industriales:jose@informatica.es
Angel:20:telecomunicaciones:angel@informatica.es
Bernardo:21:industriales:bernardo@informatica.es
Luisa:18:informatica:luisa@informatica.es
...Procesamiento terminado
```

Expresiones Regulares.

Una expresión regular (RE) representa a un conjunto de cadenas de caracteres y se interpretan de igual manera que en el caso de los filtros (ver práctica 2).

En el caso de awk, las expresiones regulares deben ir entre caracteres *slash* (**/**): /RE/

Formato de una expresión regular:

carácter Un carácter se representa a sí mismo.

[ ] Representa a uno cualquiera de los caracteres que encierran
```
awk '/[jJ]uan/' file1
```

| Representa dos posibles patrones válidos.
```
awk '/juan|pedro/' file1
```

^ Comienzo de una línea (o un campo).
```
awk '/^juan/'file1
```

$ Fin de línea (o de campo).
```
awk '/^juan|pedro$/'file1
```

. Un carácter cualquiera y solamente uno.
```
awk '/j..n/'file1
```

***** Cero o más apariciones del carácter que le precede.
```
awk '/ju*an/' file1
```

+ Una o más apariciones del carácter que le precede.
```
awk '/ju+an/' file1
```

? Cero o una aparición del carácter que le precede.
```
awk '/ju?an/' file1
```

**** Elimina la propiedad de ser un carácter especial al que le sigue
```
awk '/j\*an/' file1
```

Ejemplo:

```
awk '$1 ~ /^p/ {print $0}' /etc/passwd
```

Puesto que no se especifica otro separador, los campos se consideran separados entre sí por medio de uno o más espacios en blanco.

Entonces, se recorren todas las líneas del archivo `/etc/passwd` buscando aquellas que se ajusten al patrón dado o, lo que es lo mismo, que cumplan la condición:

`$1 ~ /^p/` $\Longrightarrow$ Condición cierta si el primer campo `($1)` contiene `(~)`el patrón `/^p/`, es decir, si comienza con el carácter p minúscula.

Si una línea cumple la condición anterior, entonces se ejecuta la acción `{print $0}`, que se encarga simplemente de enviar a la salida el contenido de la línea completa ($0).

En resumen, se muestran por pantalla todas aquellas líneas cuyo primer campo comienza por la letra p minúscula.

Expresiones relacionales y aritméticas.

Un patrón puede involucrar a los siguientes operadores relacionales:

<	Menor
<=	Menor o igual
==	Igual
!=	Distinto
>=	Mayor o igual
>	Mayor
~	Verifica la expresión regular
!~	No verifica la expresión regular

Sintaxis: `expresión operador_relacional expresión`

Las expresiones pueden llevar los operadores matemáticos siguientes:

+ - / * %

Si un operador es una cadena, se hace comparación de cadenas.

Si ambos son números, se hace comparación numérica.

Ejemplos:

`$1 ~ /f/`	Es cierto si `$1` contiene la letra *f*.
`$1 == "f"`	Es cierto si `$1` es igual que la cadena "*f*".
`$1 > $2`	Es cierto si `$1` es mayor que `$2`.
`$1 * $2 != 30`	Es cierto si el resultado de `$1*$2` es distinto del valor entero 30.

Combinación de patrones.

• Los patrones pueden combinarse con los siguientes operandos:

||	OR lógico
&&	AND lógico
!	NOT lógico

Así, la expresión

```
$1 >= 40 && $2 ~ /adulto/
```

es cierta para las líneas en las que el primer campo sea mayor o igual que *40* y, además, el segundo contenga la cadena *adulto*.

Acciones

Con las acciones se consigue manipular cadenas de caracteres y realizar algunas operaciones aritméticas.

Una acción no es otra cosa que una secuencia de sentencias separadas por el carácter punto y coma (;) o por el carácter de nueva línea, estando el conjunto de todas ellas encerrado entre llaves.

Las sentencias utilizadas pueden utilizar diferentes variables y operadores además de instrucciones de otros tipos para control de flujo o visualización de resultados.

Cuando no se especifica ninguna acción, por defecto se visualiza la línea completa.

Ejemplos:

Sea *file1* un archivo de texto cuyo contenido es:

```
root;desconectado;15;14:01
user;desconectado;13;15:45
user2;conectado;14;20:05
user3;desconectado;0;00:00
user;conectado;20;06:35
```

Entonces son equivalentes las siguientes dos instrucciones:

```
awk -F\; '$1 !~ /^user$/  { print $0; a = a + 1} END
                          { print "Total:",a}' file1
awk -F\; '$1 !~ /^user$/  { print $0
                             a = a +1 }
                      END {print "Total:",a}' file1
```

y permiten obtener como resultado:

```
root;desconectado;15;14:01
user2;conectado;14;20:05
user3;desconectado;0;00:00
Total; 3
```

Es decir, se visualizan las líneas en las que el primer campo no es *user* y, al final, se imprime una cadena con el total de líneas visualizadas.

Las siguientes instrucciones también son equivalentes:

```
awk '$1 !~ /^user$/' file1
awk '$1 !~ /^user$/ {print $0}' file1
```

y devuelven:

```
root;desconectado;15;14:01
user;desconectado;13;15:45
user2;conectado;14;20:05
user3;desconectado;0;00:00
user;conectado;20;06:35
```

El resultado consiste en la visualización de todas las líneas que no sean exactamente iguales a la cadena "*user*".

Variables y operadores.

Las variables utilizadas pueden ser numéricas o cadenas de caracteres en función del contexto. No hace falta declararlas ni asignarles un valor inicial, pues toman el valor 0 o el valor cadena vacía por defecto.

Para hacer referencia a una variable basta con utilizar su nombre.

Se pueden utilizar paréntesis y operadores para formar expresiones:

Operadores aritméticos:

+ - / * % ++ --

Operadores de asignación:

= += -= *= /= %=

La concatenación de cadenas se consigue separándolas con un espacio en blanco. Así, por ejemplo: `"cadena1" "=una cadena"` equivale a `"cadena1=una cadena".`

Las variables de *awk* son distintas de las que pueda haber en el *shell*. Si se quiere utilizar una variable definida en el *shell* dentro de *awk*, hay que indicarlo poniéndola entre caracteres comilla simple ('). Ejemplo: `$1 ~ /'$var'/`

Además, se pueden utilizar las variables `$0`, `$1`, `$2`, ... que contienen la línea completa y los diferentes campos de cada línea, respectivamente.

Visualización de resultados.

La acción print permite visualizar los resultados deseados.

Sintaxis: `print cadena`

Ejemplo:

```
awk  'BEGIN {print "Principio de la ejecucion..."}
      /^$/  {total++}
      END   {print "Hay" total "lineas vacias"}' datos
```

Muestra por pantalla un mensaje con el número de líneas vacías que hay en el archivo *datos*:

```
Principio de la ejecucion...
Hay 2 lineas vacias
```

4. Enunciado de la práctica.

a. Se dispone de un registro de las salidas de material de un almacén de componentes eléctricos. Dicho registro es el archivo de texto *almacen.txt* cuyo contenido es el siguiente:

```
Encargado:Modelo:Unidades:Fecha de Salida
Ernesto:R123:100:13_2_99
Andrea:R1C1:50:14_2_99
Luis:C35L:50:14_2_99
Ernesto:L1a0:75:15_2_99
Andrea:C250:25:18_2_99
Andrea:rL90:200:18_2_99
Ana:RA45:100:20_2_99
Ernesto:cr32:25:20_2_99
```

La primera letra del modelo indica si el componente es una resistencia (R), un condensador (C) o una bobina (L).

Elabora un *script* de nombre *totaliza*, que utilice *awk* para calcular el número total de componentes que han salido del almacén, el número de resistencias, el número de condensadores y el número de bobinas. Los resultados deben presentarse por pantalla con el siguiente formato:

```
Totalizando...

nR Resistencias
nC Condensadores
nB Bobinas
=====================
nT componentes totales
```

Donde nR, nC y nB corresponden a los valores totales de resistencias, condensadores y bobinas que han salido del almacén, respectivamente, mientras que nT es la suma de esos tres valores.

La sintaxis del script ha de ser la siguiente:

```
totaliza nombre_archivo_datos
```

Es decir, el nombre del archivo de datos se introduce como parámetro del *script*. Al final de la ejecución, se debe devolver como resultado al sistema operativo el valor 0 (orden *exit*).

b. Realiza un programa awk que muestre por pantalla las direcciones electrónicas de las personas del archivo datos que tengan más de de 19 años y estudien informática. Previamente debe mostrar el mensaje: “Lista de direcciones electrónicas:” y después de acabar: “Total de direcciones: #” (Donde # es el número de líneas que cumplan las condiciones exigidas). El contenido del archivo datos aparece en la sección 3.