Ejemplo 1. Comentarios:

El formato general de un programa o script de Perl:

- no hay una función principal como main() en C.
- los comentarios comienzan por # (excepto #!).

Se puede observar gran parte, aunque no toda, de la sintaxis de C; el operador de asignación, por ejemplo, el cual, igual que en C, se puede emplear como una expresión.

El operador ** es de FORTRAN.

No existen los prototipos o declaraciones de funciones como en C.

Indistintamente las definiciones de funciones pueden aparacer al comienzo o al final del script.

El valor devuelto por una función es el que se especifica con **return**, o bien, el valor de la ultima expresión.

En cada llamada a una función, los argumentos son asignados a una lista (en Perl una lista es lo mismo que un vector) propia de Perl llamada @_ (las variables de tipo vector comienzan por el signo @) y que es local para la funcion. A dichos argumentos se accede, o bien, como a los elementos de un vector mediante un índice **$_[0]**, **$_[1]**, ... o con el operador shift que devuelve el valor del primer elemento del vector con menor índice (parecido al operador shift de la shell de UNIX.)

Hay que notar el cambio de prefijo cuando se hace referencia al vector completo (@) y cuando se accede a los elementos del vector ($), ya que estos últimos son tratados como variables escalares.

Con la palabra reservada **my** se declaran las variables locales.

Las variables escalares (aquellas que comienzan por $) pueden ser números (enteros o en coma flotante, aunque Perl opera internamente solo con números en coma flotante de doble precisión), referencias (punteros a tipos de datos o funciones) o cadena de caracteres (strings). No es necesario definir las variables antes de emplearlas ya que su tipo se deduce de su contexto u operadores aplicados (por ello, en Perl no son necesarios las conversiones de tipo, como en C el type-cast.)

El nombre de cualquier tipo de variable debe comenzar por una letra y puede tener una longitud máxima de 255 caracteres.

El operador de lista para la asignación funciona de la siguiente forma: Cada elemento de la parte izquierda del operador le es asignado el valor del correspondiente elemento de la parte derecha.
Si a su derecha hay mas elementos que a su izquierda, los elementos sobrantes son ignorados; de lo contrario, si a su izquierda hay mas elementos que en la parte derecha, a los elementos sobrantes se les asigna el valor especial de indefinido (**undef**).

En principio, la utilización de paréntesis en las llamadas a las funciones **return** y en **print** es arbitraria, esto es, se pueden poner u omitir dependiendo de la claridad deseada.

Ejemplo 1

```
#!/usr/bin/perl

# factorial($n) calcula el factorial de $n
sub factorial {
    my ($n) = shift;
    return $n if $n <= 2;
    return $n * factorial($n - 1);
}

# fibonacci($n) calcula el $n-esimo termino de la
# secuencia de Fibonacci
sub fibonacci {
    my ($n, $s) = (shift, sqrt(5));
    return (((0.5 + 0.5*$s) ** $n) - ((0.5 - 0.5*$s) ** $n)) / $s;
}

print factorial(6), "\n";
print fibonacci(20), "\n";
```

Ejemplo 2. Comentarios:

La función **open** abre el fichero /etc/passwd para lectura y le asigna un handle, PW, para poder operar sobre el fichero.
Si se abre un fichero para escritura se debe especificar con un signo de mayor > en el nombre del fichero (**">/etc/passwd"**).

La funcion **die** es semejante a la funcion exit() de C.
Imprime la cadena de caracteres que se le pasa como argumento por STDERR y sale del programa devolviendo un código de error. Este código de error se guarda en una variable especial de Perl llamada **$!**.

Al igual que en la shell de UNIX, el operador lógico OR se puede representar con ||. De esta forma, se ejecutará la función **die** únicamente sí la función **open** falla.

El operador de entrada de línea utilizado como condición del **while**, asigna sucesivamente a la variable especial de Perl **$_** (que no es **@_**), cada línea leída de PW en cada iteración del bucle hasta llegar al final del fichero.

En la primera invocación a **split** se emplea el operador de lista.
Cuando no se especifica una variable como argumento a **split** esta función se aplica por defecto a la variable **$_**.
En este caso la expresión regular /:/ es utilizada como separador de campo.
Como que en Perl una lista es equivalente a un vector, a esta se le puede aplicar la referencia múltiple que ofrece Perl.
El resultado es una lista del primer, tercer y quinto campo de cada linea del fichero /password. Los valores de esta lista son asignados respectivamente, a las variables **$user**, **$uid** y **$gcos**.

En la segunda invocación a **split**, se emplea como separador de campo la expresión regular /,/ y esta vez la función se aplica sobre la variable **$gcos**.
Como que la variable **$real** se encuentra entre paréntesis, se aplica también el operador de lista.
split devuelve una lista con al menos un valor pero a la variable **$real** solo le será asignado el primero de ellos, esto es, el nombre real del usuario.
Recuérdese, que si a la izquierda del operador de lista hay menos elementos que a la derecha, los elementos sobrantes de la derecha serán ignorados.

La función **write** forma parte de una construcción llamada **format**, que sirve para generar reportes con formato.
Primero se define el **format** y después, durante la ejecución se cargan los datos y se escriben en el **format** con la función **write**.
La carga de los datos se realiza asignando los valores a las variables que forman parte del **format**.
El nombre de un **format** hace referencia a un file handle que en este caso es STDOUT. La función **write** emplea por defecto STDOUT, por ello se omite el file handle en la invocación de **write**.
Cuando el nombre del **format** lleva el sufijo **_TOP**, quiere decir que se esta definiendo el formato de la cabecera.
En este caso, se describe el titulo de cada columna.
Cada definción de un **format** acaba en una línea que contiene únicamente un punto al comienzo de la misma.
En la siguiente definición de **format**, se define el formato del contenido: la posición, la alineación, el tipo, y la longitud de los datos.
En este caso, el valor de **$user**, por ejemplo, aparecerá en primera posición, alineado a la izquierda, con una longitud de 8 caracteres (se incluye el simbolo @) y será interpretado como de tipo carácter lo cual se especifica con el símbolo < ó >.
En el caso de valores de tipo numérico se emplearía el símbolo #.

Ejemplo 2

```
#!/usr/bin/perl

open(PW,"/etc/passwd") || die "How did you get logged in?";
while (<PW>) {
    ($user,$uid,$gcos) = (split /:/)[0,2,4];
    ($real) = split /,/,$gcos;
    write;
}

format STDOUT_TOP =
Username User ID Real Name
======== ======= =========
.

format STDOUT =
@<<<<<<< @>>>>>> @<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
$user, $uid, $real
.
```

Ejemplo de salida:

```
Username    User ID     Real Name
========    =======     =========
root              0     Super-User
daemon            1
bin               2
sys               3
adm               4     Admin
lp               71     Line Printer Admin
uucp              5     uucp Admin
nuucp             9     uucp Admin
listen           37     Network Admin
nobody        60001     Nobody
noaccess      60002     No Access User
nobody4       65534     SunOS 4.x Nobody
```

Ejemplo 3a. Comentarios:

En un sistema Windows se tiene un directorio con casi 60 artículos de una revista. Cada articulo es un fichero en formato HTML y todos los artículos tienen el mismo formato. En cada articulo hay publicidad (banners), imágenes y enlaces que no son interesantes.
Hay que editar cada fichero y borrar una sección (conjunto de líneas sucesivas.) Este conjunto de líneas se caracteriza porque en su primera y ultima línea se encuentran determinadas expresiones regulares.

Los scripts de Perl se ejecutan desde una ventana de DOS con perl <script> o con doble-clic si se han asociado los scripts con el interprete de Perl.

En Windows no es necesario decirle al interprete de comandos el programa que se va emplear para ejecutar el script. Por ello no se incluye la línea **#!**.

La barra invertida en la especificación de los directorios se debe poner doble, ya que de lo contrario serian interpretadas como caracteres de control.

La función **opendir** abre un directorio, para que, con la función **readdir** se pueda leer cada entrada del mismo, al igual que las líneas de un fichero de texto.

Por cada fichero cuya extensión sea htm, se crean los nombres de los ficheros de salida. Estos tendrán el mismo nombre pero otra extensión: el fichero cuya extensión acabe en ht1 contendrá él articulo con la sección borrada y aquel con extensión acabada en ht2 contendrá la sección borrada.

Después de abrir y crear los tres ficheros, el articulo y los ficheros de salida, se lee línea por línea del articulo y según el operador //...// se escribirá la línea actual en *.ht1 o en *.ht2.

Cuando hay una expresión regular dentro de dos barras inclinadas en el contexto de una condición, esta se compara por defecto con la variable **$_**.
El operador **=~** sirve para especificar una variable distinta a **$_** con la cual comparar la expresión regular.
En el contexto de una asignación, el operador **=~** permite especificar una variable distinta a la variable por defecto, **$_**, para asignarle un valor.

El operador //...// tendrá un valor verdadero cuando en la línea actual, contenida en **$_**, se encuentra la expresión regular de la izquierda del operador. Seguirá valiendo verdadero hasta llegar a una línea que contenga la expresión regular de la derecha, en cuyo caso seguirá valiendo verdadero pero a partir de esta ultima línea será falso hasta la siguiente línea que contenga la expresión regular de la izquierda y de esta forma se repita el ciclo descrito.
Nunca se buscarán las dos expresiones regulares en una misma línea; para ello se debe emplear el operador //..//.

Ejemplo 3a

```
$dir = "\\Usuario\\Perl\\Web Techniques";
opendir(DIR, $dir) or die "Can't open $dir: $!";
while( defined ($file = readdir DIR) ) {
    if ($file =~ /htm$/) {
       ($outfile1 = $file) =~ s/m$/1/;
       ($outfile2 = $file) =~ s/m$/2/;
       open(OUT1, "> $dir\\$outfile1") or die "Can't open $outfile1: $!";
       open(OUT2, "> $dir\\$outfile2") or die "Can't open $outfile2: $!";
       open(IN, "< $dir\\$file") or die "Can't open $file: $!";
       while (<IN>) {
           if (/<TABLE align/ ... /END -->/) {
                print OUT2 $_;
                next;
           }
           print OUT1 $_;
       }
       close(IN);
       close(OUT1);
       close(OUT2);
    }
}
closedir(DIR);
```

Ejemplo 3b. Comentarios

De los ficheros de salida del script anterior, *.ht2, que contienen la sección eliminada de un articulo, se quiere imprimir la primera y ultima línea de cada uno de ellos, precedidas por el nombre del fichero, en un único fichero llamado 2.out.

El fichero 2.out es abierto en un modo para poder añadir líneas al final del mismo.
La variable **@lines** representa un vector o lista y le es asignado el valor de lista devuelto por **<IN>**, esto es, cada elemento del vector contendrá una línea del fichero de entrada. En otras palabras, se ha cargado un fichero entero en memoria. Esto es posible, logicamente, mientras haya suficiente memoria.
En la primera función **print**, se emplea el operador de concatenación de cadena de caracteres para escribir el nombre del fichero seguido de la primera línea de cada fichero de entrada, la cual se encuentra en **$lines[0]**.
En la segunda llamada a **print**, se hace referencia al ultimo elemento del vector como **$lines[$#lines]**, que contiene la ultima línea de cada fichero *.ht2. El operador especial **$#** devuelve el ultimo índice valido del vector al cual se aplica.

Si el fichero fuera muy grande, mejor sería localizar la primera y ultima línea con la función **seek**.

Ejemplo 3b

```
$dir = "\\Usuario\\Perl\\Web Techniques";
$outfile = "2.out";
open(OUT, ">> $dir\\$outfile") or die "Can't open $outfile: $!";
opendir(DIR, $dir) or die "Can't open $dir: $!";
while( defined ($file = readdir DIR) ) {
    if ($file =~ /ht2$/) {
        open(IN, "< $dir\\$file") or die "Can't open $file: $!";
        @lines = <IN>;
        print OUT "$file "."$lines[0]";
        print OUT "$file "."$lines[$#lines]";
        close(IN);
    }
}
close(OUT);
closedir(DIR);
```

Ejemplo 4. Comentarios:

La salida del comando who se convierte en la entrada para Perl.
El modificador **-n** sirve para que Perl ejecute un bucle con el código especificado después del modificador **-e**. Este código se ejecuta para cada línea de entrada, hallándose esta en la variable **$_**.
En lugar de escribir los modificadores por separado, estos pueden ser combinados como se muestra en este ejemplo.
Si se llama a la funcion **split** sin argumentos, **split** actúa sobre **$_** y considera el espacio en blanco como separador por defecto.

```
who | perl -ne '@F = split; print "$F[0]\n";'
```

El modificador **-a** indica a Perl que debe realizar un **split** implícito sobre la variable **$_** y el resultado asignarlo automáticamente al vector **@F**.
Solo cuando se utiliza el modificador **-a**, **@F** es considerado como una variable especial de Perl.

```
who | perl -ane 'print "$F[0]\n"'
```

Si se añade el modificador **-l** se consigue primero, que Perl elimine el ultimo carácter de la linea de entrada actual, \n, y despues, que lo vuelva a añadir al final del argumento de la funcion **print**.
Con el empleo de este modificador se ahorra escribir tres caracteres.

```
who | perl -lane 'print $F[0]'
```

Cambiando el modificador **-n** por **-p** indicará a Perl que, además de mantener el bucle implícito, imprima con **print** la variable **$_** como ultima instrucción del código ejecutado en cada iteración.

```
who | perl -lape '$_ = $F[0]'
```

La variable especial de Perl **$** contiene el ultimo carácter que imprime la función **print** (parecido a la variable ORS de Awk.) Por defecto, esta variable no contiene ningún valor, lo cual significa que en cada llamada a **print**, esta función imprime únicamente lo que se le especifica como argumento.
En este caso, a la variable **$** le es asignado el valor de otra variable especial de Perl. Concretamente el valor de **$/**, que representa el separador de entrada (en Awk sería RS), que por defecto es \n.
De esta forma, cuando se invoca **print**, automáticamente se imprimirá un carácter \n al final del string que forma parte del argumento de **print**.

Con la línea **use English** al comienzo de un script, se puede referir a la mayoría de las variables especiales de Perl con otro nombre más humano, como por ejemplo, **$ARG** en lugar de **$_**, **$ORS** o **$OUTPUT_RECORD_SEPARATOR** en lugar de **$**, etc...

En las sucesivas referencias al operador de entrada de línea, <>, este devuelve la siguiente línea del flujo de entrada que en este caso es la salida del comando who.

La función **chop** elimina el ultimo carácter del string que se le pasa como argumento.
Si no se especifica argumento en la llamada a **chop**, esta se aplica por defecto a la variable **$_**.
Esto ultimo también se aplica en la llamada a la función **print**.

```
#!/usr/bin/perl

$\ = $/;                # modificador -l
while (<>) {            # modificador -p
        chop;           # modificador -l
        @F = split;     # modificador -a
        $_ = $F[0];     # argumento del modificador -e
        print;          # modificador -p
}
```

Ejemplo 5

```
#*
#* querying a daytime server using Net::Telnet
#*

use Net::Telnet;

$host = $ARGV[0] ? $ARGV[0] : 'localhost';

$cn = new Net::Telnet(Host => $host, Port => 'daytime');

while (defined $ret = $cn->get) {
    $data .= $ret;
}
$cn->close;

print $data;
```