La programación de Perl para Unix

Guadalajara, Jalisco México
Octubre 1997

Autor: Candidato a M. en C. Mónico Briseño
correo electrónico: monico@redudg.udg.mx

Diseño gráfico: Rubén Gómez.

Presentación

Desde 1996, La Coordinación General de Sistemas de Información, de la Universidad de Guadalajara, se ha dado a la tarea de ofrecer material educativo a través de Internet.

En la página de Cultura y Entretenimiento, tiene una sección de tutoriales; enfocado hacia los lenguajes de programación. **Aquí se encuentra el primer material en español del lenguaje de programación Perl.**

Durante estos meses de estar navegando en Internet, diferentes personas de diferentes lugares del mundo nos han solicitado el material de programación en un formato distinto al de HTML. Siendo esta la primera propuesta para leerse fuera de línea que ofrezca al amable lector una fuente agradable y útil de éste popular lenguaje de programación para Internet y páginas Web.

Además invitamos a todos los interesados a que compartan sus experiencias de programación en Perl. Para ello, se tiene preparado un boletín de noticias a través del Web; en el cual se puedan intercambiar todo tipo de mensajes relacionado con este lenguaje de programación.
El boletín de noticias lo podrás encontrar en la siguiente dirección Internet:

http://www.cultura.udg.mx/tutoriales/perl/perl.html

Para no olvidar

Mónico Briseño
monico@redudg.udg.mx

Guadalajara, Jalisco
Octubre de 1997

Capítulo 1

Las características más relevantes en perl

Las características más relevantes de Perl

Perl (Practical Extraction and Report Language, por sus siglas en inglés), es un lenguaje intérprete. **Tom Christiansen y Nathan Tarkington** (1997) establecen que este lenguaje es de alto nivel. Considerado como un lenguaje escrito de forma ecléctica por **Larry Wall.** Perl se deriva del **lenguaje C** de programación, así como el sed, awk y el shell de Unix, además de otras herramientas y lenguajes.

Las facilidades para la manipulación de procesos, archivos, y texto hace que este lenguaje este particularmente bien situado en las tareas donde involucran el rápido desarrollo de programas, desarrollo de utilerías para el sistema operativo, herramientas de software, tareas relacionadas con la administración de sistemas, manejo de bases de datos, programación de gráficas, redes, y del **World Wide Web.** Estas fortalezas hacen que **Perl** sea un lenguaje de programación muy popular para los administradores de sistemas **UNIX** y los creadores de **"CGI scripts".** Aunque lo anterior, no es una limitante para que cualquier persona se involucre y use este lenguaje de programación.

Sin embargo, existen actividades en las cuales Perl, no es la solución más adecuada, por ejemplo: sistemas de tiempo real, desarrollo de bajo nivel del sistema operativo que trabajen con los dispositivos del sistema de cómputo, aplicaciones de memoria compartida de procesos o aplicaciones extremadamente largas.

La sintaxis del lenguaje perl

Al inicio de un programa de Perl, en Unix, debe iniciar con:

#!/usr/local/bin/perl
o donde se encuentren instalado.
Los programas de Perl, por convención, finalizan con la

extensión .pl

Cada línea de comando debe finalizar con punto y coma (;)

Cada línea de comentarios, sobre las líneas de programación deben iniciar con el símbolo: #

Los bloques de código de Perl, tales como los ciclos de control y las condiciones siempre deben encerrarse entre corchetes ({..}).

Tipos de datos en Perl

escalares
arreglos de escalares
arreglos asociativos, conocidos también como tablas "hash"

Por el momento, solo se hablará de las variables escalares; dejando para los siguientes temas el manejo de las listas o arreglos de escalares.

Escalar

El dato del tipo escalar es el dato básico de Perl. Un escalar puede ser un entero, punto flotante, o una cadena. Las variables escalares siempre tiene que iniciar con el símbolo prefijo $. Las variables en Perl no tiene que ser declaradas al inicio del programa, como ocurre con el lenguaje C. Además dichas variables se teclean y se evalúan en base al contexto del programa.

Por ejemplo:

```
$x=4; # un entero
$y="11"; # una cadena
$z= 4.5; #de punto flotante;
```


Creando programas en Perl

Para crear programas en Perl , debemos considerar el uso de comandos o instrucciones que le dicen al lenguaje que hacer. La instrucción print envía a un dispositivo de salida lo que viene a continuación. por ejemplo:

print “hola”;

muestra en pantalla: hola

$var=Pedro;
print $ var;

muestra en pantalla: Pedro

Ejercicio 1

Instrucciones:

a) teclear en un editor de texto, fielmente el código que aparece a continuación.

Programa 1.1

```
usr/local/bin/perl
#documentando el programa
#nombre del programa:uno.pl
#la instruccion print muestra en un dispositivo de salida lo que
#existe dentro de las comillas
print “Aprendiendo a programa en Perl”;
```

b) grabar el código con el nombre uno.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

Cuestionario

a) ¿Cuál es la salida del programa?
b) ¿Cuál sería el resultado si se omite la primera línea del programa?
c) ¿ Qué regla debe seguirse para crear comentarios dentro de un programa en Perl?
d) ¿Es necesario al extensión .pl en el programa a ejecutar?
e) ¿Cuál sería el resultado, si se omite el punto y coma (;) de la línea print?

Ejercicio 2

El uso de variables

En este programa se implementará el uso de variables.

Instrucciones:

a) teclear en un editor de texto, fielmente el código que aparece a continuación

Programa 1.2

```
#!/usr/local/bin/perl
#nombre del programa: dos.pl
#Asignacion de un valor de cadena a una variable
$nombre=Pablo;
#mostrar en pantalla el contenido de la variable nombre
print "Nombre del alumno:$ nombre";
```

b) grabar el código con el nombre dos.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

Cuestionario

a) ¿Cual sería el resultado del programa, si a la variable nombre se le quita el símbolo $?
b) ¿ Cual sería el resultado del programa, si colocamos comillas la palabra Pablo?

Ejercicio 3

Instrucciones:

a) Crear un programa que muestre en pantalla el siguiente

mensaje:

Las_variables_en_Perl_se_inicializan_con_$

b) El archivo se debe grabar como tres.pl
c) Cambiarle los derechos al programa, para hacerlo ejecutable.
d) Ejecutar el programa.
e) ¿Qué cambios deberan hacerse al programa anterior para que muestre el siguiente mensaje:

Las variables en Perl se inicializan con $

Ejercicio 4

Instrucciones:

Crear un programa que asigne a una variable el valor 11. A otra el valor 2.5. La suma de ambas debe ser almacenada en una tercera. El resultado debe mostrarse en pantalla.

Obteniendo información desde un dispositivo de entrada

La instrucción <STDIN> del lenguaje de programación Perl, permite obtener información desde un dispostivo de entrada, para nuestro caso será el teclado de la computadora.

Ejercicio 5

Instrucciones:

a) teclear en un editor de texto, fielmente el código que aparece a continuación:

Programa 1.5

```
#!/usr/local/bin/perl
#programa:cinco.pl
#obteniendo informacion con la instruccion <STDIN>
$valor=<STDIN>;
#imprime el contenido de la variable valor
print $ valor;
```

b) grabar el código con el nombre cinco.pl
c) salirse del editor de texto
d) cambiar derechos al programa cinco.pl
e) ejecutar el programa
f) Teclea en el programa:
capturando informacion[enter]

La instrucción <STDIN>, de Perl, le indica que se obtiene información desde un dispositivo de entrada.
La longitud de la información almacena se delimita por el símbolo “enter” que representa una nueva línea.
Su contenido se almacena en una variable, $ valor. Dicho contenido se muestra en pantalla.

Cuestionario

a) ¿Qué cambios tendrían que hacerse al programa cinco.pl, para que se capturen dos líneas de información?

Ejercicio 6

Instrucciones:

a) teclear fielmente, el código que se muestra a continuación:

Programa 1.6

```
#!/usr/local/ perl
#programa: seis.pl
print "Teclea un valor numerico entero y positivo ";
$var1 = <STDIN>;
chop $ var1;
print "Teclea otro valor numerico entero y positivo ";
$var2 = <STDIN>;
chop $ var2;
$var3=$ var1+$ var2;
print "El resultado de la suma de $ var1 y $ var2 es: $ var3 \n" ;
```

b) grabar el código con el nombre seis.pl
c) salirse del editor de texto
d) cambiar derechos al programa seis.pl
e) ejecutar el programa

La función chop, quita el último carácter de la línea que se introdujo vía teclado.
Que regulamente es el símbolo de retorno de carro, al momento de teclear [enter].

La línea $ var3=$ var1+$ var2; es una expresión en Perl, ya que contiene más de un operador. Pueden exisitir expresiones de mayor longitud. Pero, que dependen de la funcionalidad del programa mismo.

Cuestionario

a) ¿Qué pasaría si se quita el chop en ambas líneas del programa?

Ejercicio 7

Instrucciones:

Crear un programa que capture y muestre en pantalla el siguiente mensaje:

Los programas secuenciales son aquellos

ejecutan instrucciones sin considerar ninguna

estructura de selección o repetitiva.

solución de Problemas del capitulo uno

Programa 3

```
#!/usr/local/bin/perl
#nombre del programa: tres.pl
#manejo de variables
$var1=“Las_variables_en_Perl_se_inicializan_con_$ ”;
#mostrar la variable
print "$ var1$ \n";
```

Programa 4

```
#!/usr/local/bin/perl
#nombre del programa: cuatro.pl
#suma de variables
$numero1=11;
$numero2=2.5;
$suma=$ numero1+$ numero2;
#mostrar en pantalla la suma de las variables
print "La suma es: $ suma\n";
```

Programa 7

```
#!/usr/local/bin/perl
#programa: seis.pl
print "Teclea el primer parrafo: ";
$var1 = <STDIN>;
#chop $ var1;
print "Teclea el segundo parrafo ";
$var2 = <STDIN>;
#chop $ var2;
print "teclea el tercer parrafo ";
$var3=<STDIN>;
chop $ var3;
print "El resultado de la captura es:\n $ var1\n $ var2\n
$var3\n";
```

Capítulo 2

Las estructuras selectivas en **perl**

Las estructuras selectivas

Hasta ahora, se han desarrollado programas con una estructura lineal, es decir que una instrucción sigue a la otra en secuencia. Sin embargo, esta es muy rígida para crear programas más complejos, donde se involucren la toma de decisión en función del resultado obtenido. Para ello se pueden emplear las estructuras selectivas, las cuales evalúan una condición y en función del resultado de la misma se ejecuta una instrucción u otra. El ejemplo de una estructura selectiva es la definida simplemente como si, entonces. La forma más fácil de representarla es:

Si se cumple la condición {
secuencia de instrucciones
}

Que en la sintaxis del lenguaje Perl, es:

if (condición) {
secuencia de procesos
}

Ejercicio 1:

Instrucciones:

a) Teclear el código que se muestra a continuación

Programa 2.1

```
#!/usr/local/bin/perl
#programa: estruc1.pl
$var1 = <STDIN>;
chop($ var1);
if ($ var1) {
print ("El valor es distinto de cero");
}
```

b) grabar el código con el nombre estruc1.pl
c) salirse del editor de texto

d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

¿Qué hace el programa estruc1.pl ?

1 El programa espera un valor de entrada que se almacena en la variable $ var1
2 Con la función chop se quita el último caracter tecleado.
3 Si el valor de la variable $ var1 es diferente de cero ejecuta lo que esta encerrado entre “{}”; en este caso, presenta un mensaje en pantalla.
4 Si el contenido de la variable $ var1 es igual a cero, no ejecuta lo que se encuentra encuentra encerrado entre “{}” y concluye el programa.

Para una mejor implementación de las estructuras selectivas, es muy importante usar los operadores de comparación, facilitando la toma de decisiones en función de una condición. En el lenguaje Perl, se tiene dos tipos de operadores de comparación, el primero para números y el segundo para cadenas de caracteres.

Tabla A

Operadores de comparación de valores numéricos

Operador	Descripción
<	menor que
>	mayor que
==	igual que
<=	menor o igual que
>=	mayor o igual que
!=	no igual a

Tabla B

operadores de comparación de cadena de caracteres

Operador	Descripción
lt	menor que
gt	mayor que
eq	igual a
le	menor o igual que
ge	Mayor o igual que
ne	No igual a

Veamos con los siguientes ejemplos su aplicación dentro de las estructuras selectivas.

Ejercicio 2

Instrucciones

a) Teclear el código que se muestra a continuación:

Programa 2.2

```
#!/usr/local/bin/perl
#programa: estruc2.pl
print "Tecla un valor entero ";
$var1 = <STDIN>;
chop($ var1);
print "Teclea un segundo valor";
$var2=<STDIN>;
chop($ var2);
if ($ var1==$ var2) {
print ("Los valores son iguales\n");
}
```

b) grabar el código con el nombre estruc2.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable
e) ejecutar el programa

Variante de estructura selectival

Se capturan dos valores numéricos y se comparan entre sí, utilizando el operador "=="
Si son iguales se ejecuta el contenido entre "{}".
De lo contrario no entra y concluye.

Cuestionario

¿Qué pasaría si en vez de utilizar el operador "==", se emplea el "="?

Existe una variante en esta estructura selectiva que se muestra a continuación

Si se cumple la condición {
secuencia de procesos
}

de lo contrario {
secuencia de procesos
}

Que se representa con la sintaxis del lenguaje Perl, de la siguiente manera:

```
If (condición) {
 secuencia de procesos
}

else {
secuencia de procesos
}
```

Ejercicio 3

Instrucciones:

a) Teclear el código que se muestra a continuación:

Programa 2.3

```
#!/usr/local/bin/perl
#programa: estruc3.pl
print "Tecla un valor entero ";
$var1 = <STDIN>;
chop($ var1);
print "Teclea un segundo valor ";
$var2=<STDIN>;
chop($ var2);
if ($ var1==$ var2) {
print ("Los valores son iguales\n");
}
else {
print "Son diferentes\n";
```

b) grabar el código con el nombre estruc3.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable
e) ejecutar el programa

Actividad

Describe con tus propias palabras el funcionamiento del programa estruc3.pl

Ejercicio 4

Instrucciones:

a) Crear un programa que calcule la media aritmética de 5 números enteros y positivos.
b) grabarlo con el nombre estruc4.pl

Ejercicio 5

Instrucciones:

a)Crear un programa para sumar 5 números positivos introducidos por teclado.
b) grabarlo con el nombre estruc5.pl

Existe otra variante a la estructura de selección, donde es posible crear diferentes opciones de ejecución, en función de la condición. Veamos

```
Si se cumple la condición {
secuencia de procesos
}

Si se cumple la condición {
secuencia de procesos
}

.
.
de lo contrario {
secuencia de procesos
}
```

De acuerdo a la sintaxis en Perl, tenemos:

```
if (condicion) {
secuencia de procesos
}

elsif {
secuencia de procesos
}

elsif {
secuencia de procesos
}

.
.
.
else {
secuencia procesos
}
```

Ejercicio 6

Instrucciones:

a) Teclear el código que se muestra a continuación:

Programa 2.6

```
#!/usr/local/bin/perl
#programa: estruc6.pl
print "Tecla un valor entero ";
$var1 = <STDIN>;
chop($ var1);
print "Teclea un segundo valor ";
$var2=<STDIN>;
chop($ var2);
if ($ var1==$ var2) {
print ("Los valores son iguales\n");
}
elsif ($ var1==$ var2+1) {
print ("El primer numero es mayor\n");
}
elsif ($ var1+1==$ var2) {
print ("El segundo es mayor\n");
}
else {
print ("los dos numeros no son iguales\n");
}
```

b) grabar el código con el nombre estruc6.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable
e) ejecutar el programa

Ejercicio 7

Instrucciones:

a) teclear el código del siguiente programa:

Programa 2.7

```
#!/usr/local/bin//perl
#programa: estruc7.pl
$uno=lunes;
$dos=martes;
$tres=miercoles;
$cuatro=jueves;
$cinco=viernes;
$seis=sabado;
$siete=domingo;
print " Teclea el dia ";
$dia=<STDIN>;
chop($ dia);
if ($ dia eq $ uno) {
print "$ uno\n";
}
elsif ($ dia eq $ dos) {
print "$ dos\n";
}
elsif ($ dia eq $ tres) {
print "$ tres\n";
}
elsif ($ dia eq $ cuatro) {
print "$ cuatro\n";
}
elsif ($ dia eq $ cinco) {
print "$ cinco\n";
}
elsif ($ dia eq $ seis) {
print "$ seis\n";
}
elsif ($ dia eq $ siete) {
print "$ siete\n";
}
else {
print "fin del programa\n";
}
```

b) grabar el código con el nombre estruc7.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo

ejecutable
e) ejecutar el programa

¿Qué hace el programa estruc7.pl ?

Ejercicio 8

Instrucciones:

a) Crear un programa que convierta las calificaciones alfabéticas A, B, C, D y E a calificaciones numéricas 10, 9, 8, 7 y 6, respectivamente.
b) Grabarlo como estruc8.pl

Las estructuras repetitivas

Las computadoras están especialmente diseñadas para todas aquellas aplicaciones en las cuales una operación o conjunto de ellas deben repetirse muchas veces. Por ejemplo la lectura de las calificaciones de varias materias de un alumno para obtener su promedio.

Para implementar actividades como la anterior, los lenguajes de programación, incluyen estructuras repetitivas, donde la repetición de una secuencia de instrucciones se le denomina ciclo y se denomina iteración al hecho de repetir la ejecución de una secuencia de acciones.

En le lenguaje de programación Perl se tienen varias estructuras repetitivas, como son:

While, Unless, For, Do y el Foreach. Para este capítulo, se desarrollarán los cuatro primeros, dejando para el apartado de las listas y los arreglos asociativos la estructura repetitiva foreach.

la estructura WHILE

La sintaxis en lenguaje Perl es:

while (expresión) {

instrucciones

}

El formato de la estructura while es muy similar a la del if, pero trabaja de manera distinta.

Veamos un ejemplo:

Ejercicio 9

Instrucciones:

a) teclear el código del siguiente programa:

Programa 2.9

```
#!/usr/local/bin/perl
#programa:estruc9.pl
$valor=0;
$contar=1;
print (“Esta linea se imprime antes de iniciar el ciclo while.\n”);
while ($ valor==0) {
print (“El valor de contar es “, $ contar, “\n”);
if ($ contar ==3) {
$valor=1;
}
$contar=contar+1;
}
print (“Fin del ciclo while.\n”);
```

b) grabar el código con el nombre estruc9.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable
e) ejecutar el programa

¿Qué hace el programa estruc9.pl?

a) Inicializa las variables $ contar y $ valor con valores que serán utilizados a lo largo del programa.
b) Manda un mensaje a la pantalla.

c) en la línea While, el programa evalúa y “dice” mientras que el contenido de la variable $ valor sea igual a cero, ejecutar la secuencia de instrucciones que se encuentra dentro de los “{}”.
d) Manda un mensaje a pantalla
e) Compara el contenido de la variable $ contar con el número 3. Si es verdadera la condición, ejecuta la secuencia de instrucciones que se encuentra dentro de los “{}”.
f) Incrementa el contenido de la variable $ contar en 1.
g) Va a la línea donde está el While y repite la operación, hasta que el contenido de la variable $ valor sea distinto de 0.

Ejercicio 10

Instrucciones:

a) teclear el código del siguiente programa:

Program2.10

```
#!/usr/local/bin/perl
#programa:estruc10.pl
$suma=0;
$numero=2;
$valor=10;
while ($ numero<=$ valor) {

$suma=$ suma+$ numero;
$numero=$ numero+2;
}
print ("la suma de los numeros pares es:$ suma\n");
```

b) grabar el código con el nombre estruc10.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable
e) ejecutar el programa

Cuestionario

Describa con sus propias palabras ¿qué hace el programa estruc10.pl?

Ejercicio 11

Instrucciones:

Crear un programa que lea las calificaciones del grupo A, de la materia de computadoras 1 y contar el número de aprobados y la media aritmética del mismo.

la estructura repetitiva Until

La estructura repetiva until, permite llevar un control parecido al de while pero, trabaja hasta que la condición sea verdadera.

La representación del until en Perl es:

until (expresión) {
secuencia de instrucciones
}

Veamos un ejemplo

Ejercicio 12

Instrucciones:

a) Teclear el código que se muestra a continuación:

Programa 2.12

```
#!/usr/local/bin/perl
#programa:estruc12.pl
print (“Cual es la suma de 17 + 26\n”);
$resp_correcta=43;
$resp_entrada=<STDIN>;
until ($ resp_entrada == $ resp_correcta) {
```

Programa 2.12

```
print (“no es correcta. Intenta de nuevo\n”);
$resp_entrada=<STDIN>;
chop ($ resp_entrada);
}
print (“Lo has logrado\n”);
```

b) grabar el código con el nombre estruc12.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable
e) ejecutar el programa

¿Cómo funciona el programa estruc12.pl?

a) Muestra un mensaje en pantalla, preguntando por la suma de 17 + 26.
b)El usuario teclea el valor de la suma
c) La estrcutura repetitiva until compara y “dice” al menos que no sea verdadera , ejecuta la secuencia de instrucciones que estan dentro de “{}”. Espera un nuevo valor de la suma, si sigue siendo diferente, repite el ciclo.
d) Si es verdadero el valor de la suma, en este caso debe ser 43. Envía un mensaje a la pantalla, diciendo: lo has logrado.

Aqui podría surgir la duda, con respecto a la forma de estructurar el programa con el until. Sin embargo, funciona.

Cuestionario

¿Qué pasaría si se modifica el operador de comparación por el de ne ?

Ejercicio 13

Instrucciones:

a) Teclear el código que se muestra a continuación:

Programa 2.13

```
#!/usr/local/bin/perl
#programa:estruc13.pl
print ("Cual es el valor de N?\n");
$num=<STDIN>;
chop($ num);
$fac=$ num;
$valor=$ fac-1;
until ($ valor <= 0) {
$fac=$ fac*$ valor;
$valor=$ valor-1;
}
print ("el factorial del numero $ num es $ fac\n");
```

Ejercicio 14

Instrucciones:

a) Crear un programa que muestre en pantalla los números del 1 hasta el 100
b) Grabarlo con el nombre: estruc14.pl

La estructura repetitiva For

En muchas ocasiones se conoce de antemamo el número de veces que se desea ejecutar un proceso dentro de un ciclo repetitivo. Para ello se utiliza la estructura desde o para (for en inglés).

Dentro del lenguaje de programación Perl, la representación del for es:

For (expr1;expr2;expr3) {

instrucciones;

}

expr1 es el inicializador del ciclo. Es evaluado solo una vez, antes de iniciar el ciclo.
expr2 es la expresión de condición que termina el ciclo.
instrucciones son el conjunto de actividades que se ejecutaran cuando se cumpla la condición.
expr3 se ejecuta una vez por iteracción del ciclo y se ejecuta después de la última instrucción de las mismas.

Ejercicio 15

Instrucciones

a) Teclear fielmente el código que se muestra continuación:

Programa 2.15

```
#!/usr/local/bin/perl
#programa: estruc17.pl
for ($ count=1;$ count<=5;$ count++) {
print ("$ count\n");
}
```

Ejercicio 16

Instrucciones:

a) Teclear el código que se muestra a continuación

Programa 2.16

```
#!/usr/local/bin/perl
#programa:estruc16.pl
print ("Cual es el valor de N?\n");
$num=<STDIN>;
chop($ num);
$fac=$ num;
for ($ valor=1; $ valor < $ num; $ valor++) {
```

Programa 2.16

```
$fac=$ fac*$ valor;
}
print ("el factorial del numero $ num es $ fac\n");
```

Ejercicio 17

Instrucciones:

a) Crear un programa que obtenga la suma de los números pares comprendidos entre el 1 y el 100
b) Grabarlo con el nombre: estruc17.pl

Ejercicio 18

Instrucciones:

a) Crear un programa que muestre el valor máximo de una serie de 10 números
b) Grabarlo con el nombre: estruc18.pl

La estructura repetitiva Do

Esta estructura repetitiva evalúa la condición después de haber ejecutado por primera vez un ciclo. Mientras que el ciclo while, for lo hacen antes.

La estructura del do en el lenguaje de Programación Perl, es:

do {

secuencia de instrucciones
} while o Until (condición)

Veamos un ejemplo

Ejercicio 19

Instrucciones:

a) Teclear el código que se muestra a continuación
valores sucesivos de la variable contar.
c) La estructura repetitiva se ejecuta hasta que el valor de la variable $ contar sea menor de 5. Concluyendo el programa.

Ejercicio 20

Actividad:

a)Utilizar la estructura do, en vez de while del ejercicio 11
b)Grabarlo como: estruc20.pl

Apéndice
Programas del curso del capitulo dos

Ejercicio 4

Crear un programa que calcule la media aritmética de 5 números enteros y positivos.

```
#!/usr/local/bin/perl
#programa: estruc4.pl
$valor=0;
print "Teclea el primer valor numerico a promediar :";
$num1=<STDIN>;
if($ valor ne 5) {
$suma=$ suma+$ num1;
$valor=$ valor+1;
}
print"Teclea el segundo valor numerico a promediar : ";
$num1=<STDIN>;
```

```
if($ valor ne 5) {
$suma=$ suma+$ num1;
$valor=$ valor+1;
}
print "Teclea el tercer valor numerico a promediar :";
$num1=<STDIN>;
if($ valor ne 5) {
$suma=$ suma+$ num1;
$valor=$ valor+1;
}
print "Teclea el cuarto valor numerico a promediar : ";
$num1=<STDIN>;
if($ valor ne 5) {
$suma=$ suma+$ num1;
$valor=$ valor+1;
}
print "Teclea el quinto valor numerico a promediar :";
$num1=<STDIN>;
if($ valor ne 5) {
$suma=$ suma+$ num1;
$valor=$ valor+1;
}
$media=$ suma/$ valor;
print "el contenido de suma es: $ suma\n";
print "el contenido de valor es: $ valor\n";
print "La media aritmetica es: $ media\n";
```

Ejercicio 5

Crear un programa que sume 5 números positivos introducidos por teclado

```
#!/usr/local/bin/perl
#programa: estruc5.pl
$valor=0;
print "Teclea el primer valor numerico a sumar : ";
$num1=<STDIN>;
if($ valor ne 5) {
$suma=$ suma+$ num1;
$valor=$ valor+1;
}
```

```
print "Teclea el segundo valor numerico a sumar : ";
$num1=<STDIN>;
if($ valor ne 5) {
$suma=$ suma+$ num1;
$valor=$ valor+1;
}
print "Teclea el tercer valor numerico a sumar :";
$num1=<STDIN>;
if($ valor ne 5) {
$suma=$ suma+$ num1;
$valor=$ valor+1;
}
print "Teclea el cuarto valor numerico a sumar : ";
$num1=<STDIN>;
print "Teclea el cuarto valor numerico a sumar : ";
$num1=<STDIN>;
if($ valor ne 5) {
$suma=$ suma+$ num1;
$valor=$ valor+1;
}
print "Teclea el quinto valor numerico a sumar :";
$num1=<STDIN>;
if($ valor ne 5) {
$suma=$ suma+$ num1;
$valor=$ valor+1;
}
print "el contenido de suma es: $ suma\n";
print "el contenido de valor es: $ valor\n";
```

Ejercicio 8

Crear un programa que convierta las calificaciones alfabéticas A, B, C, D y E a calificaciones numéricas 10, 9, 8, 7 y 6, respectivamente.

```
#!/usr/local/bin/perl
#programa: estruc8.pl
$letra1=A;
$letra2=B;
$letra3=C;
$letra4=D;
```

```
$letra5=E;
print "Teclea la letra de calificacion :";
$valor=<STDIN>;
chop($ valor);
if($ valor eq $ letra1) {
print "su calificacion es:10\n";
}
elsif($ valor eq $ letra2) {
print "su calificacion es: 9\n";
}
elsif($ valor eq $ letra3) {
print "su calificacion es: 8\n";
}
elsif($ valor eq $ letra4) {
print "su calificacion es: 7\n";
}
elsif($ valor eq $ letra5) {
print "su calificacion es: 6\n";
}
else {
print "No es letra para calificacion\n";
```

Ejercicio 11

```
#!/usr/local/bin/perl
#programa:estruc11.pl
print ("Cuantos alumnos son?\n");
$alumno=<STDIN>;
chop($ alumno);
while ($ valor ne $ alumno) {
print ("La calificacion del alumno: ");
$calif=<STDIN>;
chop($ calif);
$suma=$ suma+$ calif;
$valor=$ valor+1;
if ($ calif>=6) {
$aprob=$ aprob+1;
}
}
$media=$ suma/$ alumno;
```

```
print ("La cantidad de aprobados son: $ aprob\n");
print ("la media es:$ media\n");
```

Ejercicio 14

```
#!/usr/local/bin/perl
#programa:estruc14.pl
$num=1;
$valor=99;
until ($ valor <= 0) {
$num=$ num+1;
$valor=$ valor-1;
print ("el numero es: $ num\n");
}
```

Ejercicio 17

Instrucciones:

Crear un programa que obtenga la suma de los números pares comprendidos entre el 1 y el 100

```
#!/usr/local/bin/perl
#programa:estruc17.pl
$suma=2;
$numero=4;
for ($ valor=1; $ valor < 49; $ valor++) {
$suma=$ suma+$ numero;
$numero=$ numero+2;
}
print ("la suma de los numeros pares es:$ suma\n");
```

Ejercicio 18

Instrucciones:

Crear un programa que muestre el valor máximo de una serie de 10 números

```
#!/usr/local/bin/perl
#programa:estruc18.pl
print ("Valor: ");
$num=<STDIN>;
chop($ num);
$maximo=$ valor;
$n=1;
for ($ valor=1; $ valor < 10; $ valor++) {
print ("Valor: ");
$num=<STDIN>;
chop($ num);
if ($ num > $ maximo) {
$maximo=$ num;
}
}
print ("El numero mayor de la serie es:$ maximo\n");
```

Ejercicio 20

```
#!/usr/local/bin/perl
#programa:estruc20.pl
print ("Cuantos alumnos son?\n");
$alumno=<STDIN>;
chop($ alumno);
do {
print ("La calificacion del alumno: ");
$calif=<STDIN>;
chop($ calif);
$suma=$ suma+$ calif;
$valor=$ valor+1;
if ($ calif>=6) {
$aprob=$ aprob+1;
}
}while ($ valor ne $ alumno);

$media=$ suma/$ alumno;
print ("La cantidad de aprobados son: $ aprob\n");
print ("la media es:$ media\n");
```

Capítulo 3

Listas y Arreglos en**perl**

Listas y arreglos

En muchas situaciones se necesita procesar una colección de valores que están relacionados entre sí por algún método, por ejemplo: una lista de calificaciones, una serie de temperaturas medidas a lo largo de un mes, etc.

Un arreglo es una secuencia de posiciones de la memoria principal a las que se pueden leer directamente, que contiene datos de diferente tipo y pueden ser seleccionados individualmente mediante el uso de subíndices.

Un arreglo es un conjunto finito y ordenado de elementos. La propiedad “ordenado” significa que el elemento primero, segundo, tercero, ,..n-ésimo de un arreglo se puede identificar. Los arreglos se conocen como matrices (en matemáticas) y tablas (en cálculos financieros).

Las operaciones que se pueden efectuar en un arreglo son:

a) Asignación
b) Lectura/escritura
c) Acceso secuencial
d) Actualizar (añadir, borrar, insertar)
e) Ordenación
f) Búsqueda

Dentro del lenguaje de programación Perl, una lista es una secuencia de valores escalares dentro de un paréntesis. Veamos un ejemplo:

(4,5, “Perl”, 2)

La lista contiene cuatro elementos cada uno de los cuales son escalares.

Perl es capaz de almacenar las listas en una variable especial designada para tal propósito. Estas variables son conocidas como arreglos.

Veamos un ejemplo:

@arreglo=(4,5, “Perl”, 2)

Obsérvese que Perl utiliza el símbolo @ para el uso de arreglos.

Los arreglos siempre inician con la posición cero. Veamos un ejemplo para ello:

Ejercicio 1

Instrucciones:

a) Teclear el código que se muestra a continuación:

Program 3.1

```
#!/usr/local/bin/perl
#programa:arreglo1.pl
@var=(1,2,3,4,5);
$var=@var[0];
print (“EL elemento 0 de la lista es:$ var”\n);
```

b) grabar el código con el nombre arreglo1.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

Cuestionario

¿Cómo funciona el programa arreglo1.pl?

a)Asigna al arreglo @var los valores: 1,2,3,4,5
b) Asigna el valor del elemento 0 a la variable $ var
c)Muestra en pantalla el elemento 0 de la variable $ var.

Se pueden asignar valores a los elementos de un arreglo en forma individual, veamos:

Ejercicio 2

Instrucciones:

a) teclear el código que se muestra a continuación:

Program 3.2

```
#!/usr/local/bin/perl
#programa:arreglo2.pl
@var=(1,2,3,4,5);
$var[2]=4;
$contador=1;
while (contador<=5) {
print (“El elemento del arreglo var es:$ var[$ contador-1]\n”);
$contador=$ contador+1;
}
```

b) grabar el código con el nombre arreglo2.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

¿Cómo funciona el programa arreglo2.pl?

a)Se crea un arreglo llamado @var con cinco elementos
b)Se modifica el contenido del elemento 2 del arreglo @var
c)Se emplea una variable para controlar la estructura repetitiva, iniciándolo con valor 1.
d)La estructura repetitiva, evalúa la condición, es verdadera, ejecuta la secuencia de instrucciones que se encuentran dentro de “{}”.
e)Dentro de la estructura repetitiva, se ejecuta la instrucción print. La cual muestra en pantalla los elementos del arreglo @var. Donde el elemento dos, tiene un valor distinto al original.
f) Se incrementa el valor de la variable $ contador en 1. Se repite la estructura repetitiva hasta que el valor sea mayor de 5. Concluye el programa.

Se puede usar el valor de una variable escalar como un subíndice.

Como se mostró en el ejercicio 2, se puede usar el valor de una variable escalar como un subíndice para el arreglo.

$indice=1;
$valor=$ arreglo[$ indice];

que significa

$valor=$ arreglo[1];

Ejercicio 3:

Instrucciones:

a) Teclear el código que se muestra a continuación

Programa 3.3

```
#!/usr/local/bin/perl
#programa: arreglo3.pl
@var=(1,2,3,4,5);
$contador=1;
while ($ contador<=5) {
print ("Elemento $ contador es $ var[$ contador-1]\n";
$contador =$ contador +1;
}
```

b) grabar el código con el nombre arreglo3.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

¿Qué hace el programa arreglo3.pl ?

1. El programa despliega en pantalla los cinco elementos del arreglo llamado @var. Para ello, emplea una estructura

repetitiva (while), con el subíndice, que viene de la variable $contador.
2. La variable $ contador, se incrementa en uno, cada vez que se ejecuta el ciclo
3. Cuando llega al valor 5, concluye el programa.

Ejercicio 4

Instrucciones:

a) Crear un programa que muestre en pantalla los elementos iniciales de un arreglo llamado var. Posteriormente asigne al elemento 9 y 10, los valores 12 y 20, respectivamente. Mostrando en pantalla sus valores finales.

Los elementos iniciales de dicho arreglo son:

(1,2,3,4,5,6,7,8,9,10)

b) Grabar el programa con el nombre arreglo4.pl

Conociendo la longitud de un arreglo

Como se ha visto en los ejercicios anteriores, se puede tener arreglos de diferentes longitudes. Como consecuencia, Perl provee una forma de determinar la longitud de un arreglo.

Veamos:

Ejercicio 5

a) Teclear el código que se muestra a continuación:

Programa3.5

```
#!/usr/local/bin/perl
#programa: arreglo5.pl
@var=(1,2,3,4,5);
$var=@var;
print ("La longitud del arreglo es:$ var\n");
```

b) grabar el código con el nombre arreglo5.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

¿Cómo funciona el programa arreglo5.pl?

a)Crea un arreglo de 5 elementos
b) se le asigna el arreglo @var a la variable $ var, donde Perl lo interpreta como la longitud del arreglo, en este caso 5.

De lo anterior, podemos modificar el ejercicio 3, sustituyendo en la estructura while el valor cinco por @var.

Ejercicio 6

Instrucciones:

a) Modificar el ejercicio 3, sustituyendo en la estructura while el valor 5 por @var.
b) Grabarlo como arreglo6.pl

Cuestionario

¿Funciona igual que el ejercicio 3?

La ventaja que nos ofrece utilizar la longitud del arreglo, es no tener que contabilizarlo antes de emplearlo. Simplemente usarlo.

Con un arreglo es posible almacenar información de los dispositivos de entrada, por ejemplo del teclado. La ventaja que ofrece su uso es asignar lo capturado en un arreglo. Donde cada elemento del arreglo es una línea de entrada. Mientras que el uso de variables escalares, solo almacena una sola línea.

Ejercicio 7

Instrucciones:

a) teclear el código que se muestra a continuación:

Programa 3.7

```
#!/usr/local/bin/perl
#programa: arreglo7.pl
@var=<STDIN>;
print (@var);
```

b) grabar el código con el nombre arreglo7.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.
f) Para concluirlo teclear simultáneamente ^D (fin de archivo).

¿Cómo funciona el programa arreglo7.pl?

a) Asigna al arreglo @var las líneas de texto que se capturan.
b) Muestra en pantalla su contenido.

Ordenando los elementos de un arreglo

Para ordenar alfabéticamente un arreglo se utiliza la función sort(). Veamos:

Ejercicio 8

Instrucciones:

a) Teclear el código que se muestra a continuación:

Programa 3.8

```
#!/usr/local/bin/perl
#programa: arreglo8.pl
@array=("prueba", "valor", "Trabajo",  "Salud");
@array=sort (@array);
print ("@array\n");
```

b) grabar el código con el nombre arreglo7.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

¿Cómo funciona el programa arreglo8.pl?

a) Inicializa el arreglo @array con una serie de elementos
b) Empleando la función sort(), se ordena alfabéticamente los elementos correspondientes al arreglo @array.
c) Muestra en pantalla el resultado.

Dentro de los arreglos es posible acomodar los elementos de un arreglo a la inversa de la función anterior.

Ejercicio 9

Instrucciones:

a) Teclear el código que se muestra a continuación:

Programa 3.9

```
#!/usr/local/bin/perl
#programa: arreglo9.pl
@array=("prueba", "valor", "Trabajo",  "Salud");
@array=reverse (@array);
print ("@array\n");
```

b) grabar el código con el nombre arreglo9.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo

ejecutable.
e) ejecutar el programa.
Cuestionario

¿Cuál será la salida del programa arreglo9.pl?

Utilizando la función chop() en los arreglos

Como se vio en el primer apartado del manual, la función chop quita el último carácter de una cadena de caracteres. Esta característica puede emplearse dentro de los arreglos.

Ejercicio 10

Instrucciones:

Teclear el código que se muestra a continuación:

Programa 3.10

```
#!/usr/local/bin/perl
#programa: arreglo10.pl
@array=("prueba", "valor", "Trabajo",  "Salud");
chop(@array);
print ("@array\n");
```

b) grabar el código con el nombre arreglo10.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.
Cuestionario

¿Cuál es la salida del programa arreglo10.pl?

La función chop, es muy útil para crear una sola cadena a partir de los elementos de un arreglo. Para ello se emplea la función join.

Ejercicio 11

Instrucciones:

a) Teclear el código que se muestra a continuación:

Programa 3.11

```
#!/usr/local/bin/perl
#programa: arreglo11.pl
@array=<STDIN>;
chop(@array);
$cadena=join(" ", @array);
pri       t$ cena");
```

b) grabar el código con el nombre arreglo11.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

¿Cómo funciona el programa arreglo11.pl?

a) desde un dispositivo de entrada se almacenan cadenas de caracteres separadas por el carácter “nueva línea”.
b) La función chop, quita el último carácter a cada una de las cadenas que se introducen. Que en este caso es el carácter de nueva línea.
c) la función join utiliza el primer elemento como el carácter para unir las cadenas en una. En este caso se unen con espacio en blanco “ “.
d) Muestra en pantalla la nueva cadena.

Existe la función opuesta a join, conocida como split. La cual divide en varias cadenas una sola, de acuerdo al primer elemento definido.

Ejercicio 12

Instrucciones:

a) Teclear el código que se muestra continuación:

Programa 3.12

```
#!/usr/local/bin/perl
#programa: arreglo12.pl
$cadena="Esto:es:una:prueba";
@array=split(":",$ cadena);
print (@array);
```

b) grabar el código con el nombre arreglo12.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

Cuestionario

¿Cómo funciona el programa arreglo12.pl?

a) Se crea una cadena, donde las palabras significativas se separan por ":", esto sirve como patrón en la función split.
b) La función split emplea el carácter ":" para separar en subcadenas la anterior.
c) muestra el resultado.

Ejercicio 13

Instrucciones

Crear un programa que cuente el número de palabras de un texto introducido vía teclado.

la estructura reptitiva Foreach

Una de las actividades comunes que se realizan dentro de una estructura repetitiva es aplicar alguna actividad sobre cada elemento almacenado en un arreglo. Por ejemplo, podemos utilizar la estructura repetitiva foreach para comprobar si existe un elemento de la lista que sea la cadena hola.

Ejercicio 14

Instrucciones:

a) Teclear el siguiente código:

Programa 3.14

```
#!/usr/local/bin/perl
#programa: arreglo14.pl
$contador=0;
@arreglo=(Hola, Adios, bueno);
foreach $ arreglo(@arreglo) {
if ($ arreglo eq "Hola") {
print ("Existe la palabra ‘Hola’\n");
}
}
```

b) grabar el código con el nombre arreglo14.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

¿Cómo funciona el programa arreglo14.pl?

a) Inicializa la variable contador a 0.
b) Inicializa el arreglo @arreglo con varios elementos
c) Ejecuta la estructura repetitiva foreach; desarrollando lo siguiente:

por cada elemento del arreglo @arreglo, asignado a la variable $ arreglo. Compáralo con la palabra Hola.
Si existe, manda un mensaje de confirmación a pantalla.

Ejercicio 4

Instrucciones:

a) Crear un programa que muestre en pantalla los elementos iniciales de un arreglo llamado var. Posteriormente asigne al elemento 9 y 10, los valores 12 y 20, respectivamente. Mostrando en pantalla sus valores finales.

Los elementos iniciales de dicho arreglo son:

(1,2,3,4,5,6,7,8,9,10)

Programa 3.4

```
#!/usr/local/bin/perl
#programa: arreglo4.pl
@var=(1,2,3,4,5,6,7,8,9,10);
print ("Los elementos originales son\n");
$contador=1;
while ($ contador<=10) {
print ("El elemento  $ contador es $ var[$ contador-1]\n");
$contador=$ contador+1;
}
$num1=12;
$num2=20;
$var[8]=$ num1;
$var[9]=$ num2;
print ("Los elementos finales son:\n");
$contador=1;
while ($ contador<=10) {
print ("Elemento $ contador es $ var[$ contador-1]\n");
$contador ++;
}
```

Solución de programas del capitulo 3

Programa 6

Instrucciones:

a) Modificar el ejercicio 3, sustituyendo en la estructura while el valor 5 por @var.

```
#programa: arreglo6.pl
@var=("Hola",2,3,4,5);
$contador=1;
while ($ contador<=@var) {
print ("Elemento $ contador es $ var[$ contador-1]\n");
$contador ++;
}
```

Programa 13

Instrucciones

Crear un programa que cuente el número de palabras de un texto introducido vía teclado.

```
#!/usr/local/bin/perl
#programa: arreglo13.pl
$contador=0;
$linea=<STDIN>;
while ($ linea ne "") {
chop($ linea);
@arreglo=split(/ /, $ linea);
$contador=contador+@arreglo;
$linea=<STDIN>;
}
print ("El total de palabras son: $ contador\n");
```

Capítulo 4

Manejo de archivos en **Perl**

Introducción

Manejo de Archivos en Perl

EL manejo de la información es crítica en todo proceso de cómputo. Cuando se trata de grandes cantidades de esta, es conveniente emplear sistemas de almacenamiento secundario. Los cuales pueden guardar de forma permanente los datos necesarios y poder emplearlos cuando se requieran.

Esta información almacenada tiene el nombre genérico de archivo. Un archivo en una computadora es una estructura diseñada para contener datos. Los archivos están organizados de tal modo que puedan ser recuperados fácilmente, actualizados o borrados y almacenados de nuevo en el archivo con todos los cambios previos.

La organización y gestión de archivos en perl

Una colección de archivos a los que puede accederse por un conjunto de programas y que contienen todos ellos datos relacionados, constituyen una base de datos.

Los conceptos carácter, campo, registro, archivo y base de datos son conceptos lógicos que se refieren al medio en que el usuario de computadoras ve los datos y se organizan. Las estructuras da datos se organizan de un modo jerárquico, de modo que el nivel más alto lo constituye la base de datos y el nivel más bajo el carácter.

Creando archivos de texto en Perl

Ejercicio 1

Instrucciones:

a) Teclear el código que se muestra a continuación:

Programa 4.1

```
#!/usr/local/bin/perl
#programa:archivo1.pl
print ("Teclear ^D para concluir la lectura del archivo\n");
if (open(DATOS, ">datos")) {
$linea=<STDIN>;
while ($ linea ne "") {
print DATOS ($ linea);
$linea=<STDIN>;
}
}
```

¿Cómo funciona el programa archivo1.pl?

a)Realiza una operación de lectura utilizando el símbolo “>“ dentro la opción open. Creando un archivo llamado datos.
b)Se introducen la información desde el teclado y es almacenado en la variable $ linea
c)Se utiliza una estructura repetitiva para introducir texto hasta que se teclee ^D.
d) Revisar el contenido del archivo llamado datos. Utilizando el comando more o cat para observar la información que contiene.

Abriendo un archivo de texto en perl

En el lenguaje de programación Perl, podemos abrir un archivo de la siguiente manera:

open(filevar, filename)

Donde filevar, representa el nombre del archivo a usar en Perl.
Mientras que filename, representa el lugar donde se encuentra almacenado el archivo. Si el archivo se encuentra en el directorio: /home/user/camilo/web
deberá incluirse dicha ruta en el apartado filename. Veamos:

open (NOTA, /home/user/camilo/web/datos)

Veamos un ejemplo:

Ejercicio 2

Instrucciones:

a) teclear el código que se muestra a continuación:

Programa 4.2

```
#!/usr/local/bin/perl
#programa:archivo2.pl
if (open(ARCHIVO, "datos")) {
$linea=<ARCHIVO>;
while ($ linea ne "") {
print $ linea;
$linea=<ARCHIVO>;
}
}
```

b) grabar el código con el nombre archivo2.pl
c) salirse del editor de texto
d) cambiarle los derechos al programa, para hacerlo ejecutable.
e) ejecutar el programa.

¿Cómo funciona el programa archivo2.pl?

a) El programa evalúa si existe el archivo llamado datos, con la estructura condicional if. Aquí se observa como se puede emplear una variable que maneje el archivo datos. En este caso se emplea la variable **ARCHIVO.**
Es recomendable utilizar siempre mayúsculas para tal actividad.
b) Si es afirmativo, el programa lee cada línea del archivo utilizando la sintaxis: **<ARCHIVO>**.
c) La variable **ARCHIVO** se le asigna a la variable $ linea.
d) Utilizando una estructura repetitiva compara la condición. Mientras que el contenido de la variable $ linea no sea igual a “espacio en blanco”. Ejecuta lo que está dentro de los “{}”.

Que en este caso es mostrar en pantalla su contenido.
Posteriormente, se asigna de nueva cuenta el contenido del ARCHIVO a la variable $ linea.
e) Repite el proceso hasta que no exista información alguna en el archivo llamado **ARCHIVO.**

¿Qué pasaría si por alguna razón no existiera el archivo uno en el programa archivo2.pl?

Para ello, renombre el archivo datos a temp.pl ejecute de nuevo el programa archivo1.pl

¿Qué pasa con el programa, se ejecuta correctamente? realiza lo que anteriormente había hecho?

¿Cómo podemos solucionar el problema?

Una solución, podría ser colocar la función die, dentro del programa archivo2.pl
Esta función tiene la finalidad de concluir inmediatamente cualquier programa y enviar un mensaje a pantalla.

Entonces, debemos agregar en el programa archivo2.pl la función die. Veamos

Ejercicio 3

Instrucciones:

a) Editar el programa archivo3.pl
b) Agregar a la línea if (open..) lo siguiente: || die ("No puedo abrir el archivo datos"))

Deberá quedar de la siguiente manera:

Programa 4.3

```
if (open(ARCHIVO, "datos")|| die("No puedo abrir el archivo
datos")) {
$linea=<ARCHIVO>;
while ($ linea ne "") {
print $ linea;
$linea=<ARCHIVO>;
}
}
```

c) Grabar el programa con el mismo nombre.
d) Ejecutar el programa

Cuestionario

¿Qué muestra en pantalla?

Falta por mencionar que antes de la función die, se ha colocado el símbolo “||”, el cual representa un valor lógico para el lenguaje de programación. Que significa: si es verdadera la primera instrucción o la segunda. En caso que sea afirmativo para la primera, no ejecuta la segunda. Si no es verdadera la primera, ejecuta la segunda.

El uso de los arreglos con los archivos

En Perl es posible almacenar en una variable del tipo arreglo el contenido de un archivo. Veamos:

Ejercicio 4

Instrucciones:

a) Teclear el código que se muestra a continuación:

Programa 4.4

```
#!/usr/local/bin/perl
#programa:archivo3.pl
if (open(ARCHIVO, "datos")) {
@arreglo=<ARCHIVO>;
}
print(@arreglo);
```

Ejercicio 5

Instrucciones:

Crear un programa en Perl que almacene en un archivo de tipo texto los datos de un grupo de alumnos, el cual incluya su apellido y nombre.

Ejercicio 6

Instrucciones:

Crear un programa en Perl que muestre en pantalla ordenado alfabéticamente el archivo creado anteriormente.

Agregando nuevos alumnos a la lista de alumnos existente

Para agregar más alumnos a la lista de alumnos existente, teclear el código que se muestra a continuación:

Ejercicio 7

Instrucciones:

a) Teclear el código que se muestra a continuación:

Programa 4.7

```
#!/usr/local/bin/perl
if (open(OUTFILE, ">>alumnos") ||
die ("No puedo abrir el archivo alumnos")) {

$line=<STDIN>;
while ($ line ne "") {
print OUTFILE ($ line);
$line =<STDIN>;
```

b) ejecutarlo, agregando más alumnos a la lista.

¿Cómo funciona el ejercicio 7?

a) en la condición if, dentro del paréntesis se le agrega un símbolo “mayor que” al anterior. Con lo cual se le indica que el archivo se abrirá para agregar más elementos a este.
b) El resto del programa es igual al descrito en el ejercicio 5.

Ejercicios del capítulo Archivo

Programa 5

```
#!/usr/local/bin/perl
if (open(OUTFILE, ">alumnos") ||
die ("No puedo abrir el archivo alumnos")) {

$line=<STDIN>;
while ($ line ne "") {
print OUTFILE ($ line);
$line =<STDIN>;
}
}
```

Programa 6

```
#!/usr/local/bin/perl
if (open(OUTFILE, "alumnos")||die ("No puedo abrir el archivo
alumnos")) {
@line=<OUTFILE>;
print "Lista de alumnos\n";
@line=sort(@line);
print @line;
}
```

Proyecto de programación: Creación de un programa que administre un cajero automático

Proyecto de programación

Nombre del proyecto: Creación de un cajero automático

Requisitos de programación en Perl: conceptos básicos, estructuras de control, arreglos y archivos.

Planteamiento del proyecto: desarrollar e implementar con el lenguaje de programación Perl un cajero automático que permite tres opciones:

a) Entrega saldo de la cuenta
b)Saldo de la cuenta y retirar fondos
c) Solo retirar fondos

Los fondos a retirar son en billetes de $ 500, $ 1000 y $ 5000 con un tope de $ 25000, si tiene fondos.

Considerando como datos de entrada el número de cuenta del cliente, el saldo y el fondo a retirar.

Capítulo 5

Expresiones regulares en **Perl**

Expresiones regulares en Perl

Un patrón es una secuencia de caracteres que se buscan dentro de una cadena de caracteres. En Perl, un patrón se representa entre diagonales.

/Aprendiendo/

El patrón de referencia encontrará todas las ocurrencias que existan en una cadena de caracteres.

El operador de coincidencia

Perl define un operador especial que prueba si un patrón coincide dentro de una cadena de caracteres o no. Dicho operador se define por los caracteres: =~
Veamos un ejemplo:

Ejercicio 1

1) Capturar el código que se muestra a continuación

Programa 5.1

```
#!/usr/local/bin/perl
#programa: expre1.pl
print (“ Es divertido aprender Perl, si o no?\n”);
$question=<STDIN>;
if ($ question=~/si/) {
print (“Porque facilita mi trabajo de programacion\n”);
} else {
print (“Aparentemente no. Pero, es muy fácil de aprender\n”);
}
```

2) Grábalo con el nombre: expre1.pl
3) Cámbiale los derechos y ejecútalo

¿Cómo funciona el programa?

El programa es muy parecido a los que se han creado en los

anteriores capítulos, pero contiene una instrucción con la cual nos permite comparar un patrón de caracteres dentro de una cadena de caracteres. Dicha instrucción es:

```
if ($ question=~/si/) {
print (“Porque facilita mi trabajo de programacion\n”);
}
```

Donde el programa compara si la cadena de caracteres almacenada en la variable $ question que contiene un si. Si es afirmativo, ejecuta lo que se encuentra dentro de los corchetes “{}”. De lo contrario, ejecuta el else.

Como se puede observar, el operador “=~”, permite comparar caracteres, cadenas de caracteres que se encuentran dentro de una variable.

Ejercicio 2

Instrucciones: crear un programa que cuente las veces que aparece la palabra Internet del archivo muestra.txt

Ejercicio 3

Introducción: considera que es el responsable de una página Web en Internet. El protocolo httpd que administra dicha página se encuentra instalado en una máquina Unix. El protocolo genera dentro del subdirectorio logs, varios archivos, tales como el registro de los accesos a la página, los errores que genera, etcétera.
El propio protocolo genera un archivo de texto con esta información. Sin embargo, la forma como lo almacena es un poco legible para la persona que lo lea. Para solucionar este problema, se emplea un programa creado en Perl, empleando expresiones regulares para que nos muestre cuantas veces se

ingreso a documentos HTML, en que día de la semana, y mes del año. Para ello, tenemos listo un archivo llamado prueba.log que tiene parte del contenido del archivo llamado “access.log”. Entonces,

1) Revisar el contenido del archivo prueba.log
2) Observar el formato que contiene el archivo
3) Crear el programa en Perl que satisfaga los requerimientos antes mencionados
4) grabarlo con el nombre expre3.pl
5) Cambiarle los derechos y ejecutarlo

Caracteres especiales en los patrones

Perl puede manejar varios caracteres especiales dentro de los patrones, los cuales posibilitan el número de coincidencias en una cadena de caracteres. Estos caracteres especiales son lo que hace a los patrones útiles.

El caracter especial+

El caracter especial + significa uno o más de los caracteres precedentes. Por ejemplo, el patrón /de+f/ busca las coincidencias con cualquiera de los siguientes cadenas:

def
deef
deeef
deeeeef

El caracter especial[]

El caracter especial [] te permite definir patrones que coinciden con un grupo de alternativas. Por ejemplo el siguiente patrón encuentra las coincidencias: def o dEf.
/d[eE]f/

Ejercicio 4

Instrucciones: Teclear el código que se muestra a continuación:

Programa 5.4

```
#!/usr/local/bin/perl
#programa:expre4.pl
$contador=0;
$linea=<STDIN>;
while ($ linea ne “”) {
chop ($ linea);
@palabra=split(/[\t ]+/, $ linea);
$contador+=@palabra;
print $ contador;
$linea=<STDIN>;
}
print (“Numero total de palabras: $ contador\n”);
```

Los caracteres especiales * y ?

Con el caracter especial * se tiene coincidencias de cero o más ocurrencias del caracter al que precede. /de*f/ tenemos: df, deef.

Con el caracter especial ? se tiene coincidencias de cero o un caracter del caracter que le precede.

/de?f/

Secuencias de escape para los caracteres especiales

Si quieres que el patrón incluya un caracter que normalmente es tratado como caracter especial, coloca el caracter con una diagonal invertida \.

Ejercicio 5

Instrucciones: capturar el programa que se muestra a continuación:

Programa 5.7

```
#!/usr/local/bin/perl
#programa: expre5.pl
$wordcount=0;
(open (ARCHIVO, “prueba.log”));
$line=<ARCHIVO>;
while($ line ne “”) {
chop ($ line);
if ($ line=~/\/export\//) {
$wordcount=$ wordcount+1;
}
$line=<ARCHIVO>;
}
print (“directiva encontrada: $ wordcount\n”);
```

La coincidencia de cualquier letra o número

Se puede incluir dentro de un rango, veamos:

/a[0-9]c/

Lo mismo ocurre para las letras

/[a-z][A-Z]/

Sujeción de patrones en Perl

^ o \A coincidencia al inicio de una cadena solamente
$ o \Z coincidencia al final de la cadena solamente
\b Coincidencia solo al inicio de una palabra
\B Coincidencia dentro de una palabra.

Veamos: /^def$ / significa que solo la coincidencia def y no otra.

Ejercicio 6

Instrucciones:

Capturar el programa que se muestra a continuación

Programa 5.6

```
#!/usr/local/bin/perl
#programa: expre6.pl
$wordcount=0;
(open (ARCHIVO, “prueba.log”));
$line=<ARCHIVO>;
while($ line ne “”) {
chop ($ line);
if ($ line=~/^\[Wed/) {
$wordcount=$ wordcount+1;
}
$line=<ARCHIVO>;
}
print (“Palabras con Wed que aparecen: $ wordcount\n”);
```

Ejercicio 7

Instrucciones: teclear el programa que se muestra a continuación:

Programa 5.7

```
#!/usr/local/bin/perl
#programa: expre7.pl
$wordcount=0;
(open (ARCHIVO, “prueba.log”));
$line=<ARCHIVO>;
while($ line ne “”) {
chop ($ line);
if ($ line=~/\bWed/) {
$wordcount=$ wordcount+1;
}
```

Programa 5.7

```
$line=<ARCHIVO>;
}
print (“La palabra Wed encontrada: $ wordcount\n”);
```

Substitución de variables en los patrones

Si lo deseas, se puede usar el valor de una variable escalar en un patrón. Veamos un ejemplo:

Ejercicio 8

Instrucciones:

Programa 5.8

```
#!/usr/local/bin/perl
#programa: expre8.pl
$wordcount=0;
(open (ARCHIVO, “prueba.log”));
$line=<ARCHIVO>;
while($ line ne “”) {
chop ($ line);
$patron=”Wed”;
if ($ line=~/$ patron/) {
$wordcount=$ wordcount+1;
}
$line=<ARCHIVO>;
}
print (“Palabras con Wed encontradas:: $ wordcount\n”)
```

El operador de substitución

Perl permite substituir una parte de una cadena usando el operador de substitución. El cual tiene la siguiente sintaxis:

s/pattern/replacement/

veamos un ejemplo

Ejercicio 9

Instrucciones: teclear el código que se muestra a continuación:

Programa 5.9

```
#!/usr/local/bin/perl
#programa: expre9.pl
$wordcount=0;
(open (ARCHIVO, “trabajo.txt”));
$line=<ARCHIVO>;
while($ line ne “”) {
chop ($ line);
$line=~s/Internet/internet/;
print “$ line\n”;
$wordcount=$ wordcount+1;
$line=<ARCHIVO>;
}
```

Ejercicio 10

Instrucciones:

¿Qué cambio debería hacerse en el programa anterior, para que almacenara los cambios utilizados en operador de substitución?

Solución de problemas del capítulo 5

Ejercicio 2

```
#!/usr/local/bin/perl
#programa: expre2.pl
$wordcount=0;
(open (ARCHIVO, “muestra.txt”));
$line=<ARCHIVO>;
while ($ line ne “”) {
chop ($ line);
if ($ line=~/Internet/) {
$wordcount=$ wordcount+1;
}
$line=<ARCHIVO>;
}
print (“Cuantas veces aparecio la palabra Internet:
$wordcount\n”);
```

Ejercicio 4

```
#!/usr/local/bin/perl
#programa: expre4.pl
$wordcount=0;
(open (ARCHIVO, “prueba.log”));
$line=<ARCHIVO>;
while($ line ne “”) {
chop ($ line);
if ($ line=~/html/) {
$wordcount=$ wordcount+1;
}
$line=<ARCHIVO>;
}
print (“Cuantos documentos HTML fueron?: $ wordcount\n”);
```

Capítulo 6

subrutinas en perl

Subrutinas en Perl

Las subrutinas son programas que están diseñados para ejecutar alguna tarea específica. Estas se escriben una sola vez, pero pueden ser referenciadas en diferentes puntos de un programa de modo que se puede evitar la duplicación innecesaria del código.

Ejercicio 1

1) Capturar el código que se muestra a continuación

Programa 6.1

```
#!/usr/local/bin/perl
#programa:sub1.pl
print (“ Teclea 5 numeros enteros\n”);
&suma;
print $ valor;

sub suma {

$valor=0;
for ($ contador=1; $ contador<=5; $ contador ++) {
$num=<STDIN>;
$valor=$ valor+$ num;
}
$valor;
}
```

Cómo funciona el programa?

El programa ejecuta la suma de cinco números enteros empleando una subrutina. El caracter “&”, Perl interpreta que lo que viene a continuación es una subrutina. Por lo tanto ejecuta lo que está a partir de la palabra “sub”.
Al concluir su ejecución, el programa continua ejecutándose en la línea inmediata inferior de donde fue llamada la subrutina. Siendo para este caso la impresión del contenido de la variable ‘$ valor”.
Además es importante mencionar que en Perl siempre la última línea es valor que regresa al programa personal.

Variables locales en las subrutinas

El uso de variables locales en las subrutinas permite una mayor claridad de organización dentro de los programas creados. Perl, permite declarar variables locales de la siguiente forma:

my ($ var1, $ var2 ...)

además

local ($ var1, $ var2 ..)

Ejercicio 2

1) Captura el código que se muestra a continuación

Programa 6.2

```
#!/usr/local/bin/perl
#programa:sub2.pl
print (“ Teclea 5 numeros\n”);
&suma;
print $ valor;

sub suma {

my($ valor, $ contador, $ num);
$valor=0;
for ($ contador=1; $ contador<=5; $ contador ++) {
$num=<STDIN>;
$valor=$ valor+$ num;
}
$valor;
}
```

¿Cómo funciona el programa?

Pasando valores a la subrutina

La diferencia con el ejemplo 1, es fundamentalmente la declaración de variables locales empleando el término “my”, al inicio de la subrutina.

Se pueden crear programas más útiles y flexibles, pasando valores desde el programa principal hacia las subrutinas, a estos

se le conoce como argumentos.

Ejercicio 3

1) Capturar el código que se muestra a continuación

Programa 6.3

```
#!/usr/local/bin/perl
#programa: sub3.pl
print (“ Teclea 3 numeros\n”);
print (“Primer numero:\n”);
$num1=<STDIN>;
chop($ num1);
print (“Segundo numero:\n”);
$num2=<STDIN>;
chop($ num2);
print (“Tercer numero:\n”);
$num3=<STDIN>;
chop($ num3);
&total($ num1, $ num2, $ num3);


sub total {

local($ num1, $ num2, $ num3)=@_;
local ($ suma);
$suma=$ num1+$ num2+$ num3;
print (“La suma es: $ suma\n”);
}
```

¿Cómo funciona el programa?

Las variables $ num1, $ num2 y $ num3, pasan sus valores a la subrutina total. Dentro de esta, se definen las variables locales respectivas. Entonces se les asigna su contenido de la variable del sistema @ a éstas. Las variables del sistema @ se crean cuando una subrutina se llama con argumentos. Esta contiene los valores en el orden en que se almacenaron.