Introducción a la programación en Perl, CGI y Javascript

Autor: José Vicente Núñez Zuleta (jose@ing.ula.ve, josevnz@yahoo.com)

Tabla de contenido

Tabla de contenido...*2*

Indice de tablas..**4**

Indice de ilustraciones..**6**

Objetivos de este trabajo..**9**

Introducción al Lenguaje Perl...**10**

Obtención en instalación de Perl 10

Perl básico..**11**

Hola mundo..11

Uso de variables y asideros de archivo[6]...11

Más sobre el uso de variables...12

Más sobre el uso de asideros..13

Arreglos, Arreglos asociativos y lazos..13

Arreglos y lazos con for..13

Arreglos asociativos y lazos con foreach..14

Lazos con while y uso de split()...14

Estructuras de decisión..15

Uso de patrones en búsquedas y reemplazos, usando expresiones regulares 17

Ejercicio 1: Uso de estructuras de decisión, patrones y estructuras de repetición 19

Solución:...19

Uso de rutinas en Perl...19

Referencias en Perl...21

Ejercicio 2: Uso de rutinas y referencias...22

Solución...23

Validación de programas en Perl...**25**

Seguridad en Perl...25

Depuración de programas en Perl..26

Perl en modo de advertencias (–w)..26

Perl en modo de depuración (–d)...27

Corregir sólo la sintaxis con –c...29

Ejercicio 3: Seguridad en Perl y depuración de programas................29

Solución...30

CGI (Common Gateway Interface)...31

Protocolos de comunicación...31

Modelo OSI y TCP/IP...31

HTTP..32

Modelo cliente – servidor...33

El cliente..33

El servidor..33

CGI...34

Funcionamiento de CGI...34

Generación dinámica de HTML...35

Ejercicio 4: Generación dinámica de HTML......................................36

Solución:...36

Métodos de comunicación..37

Método GET...38

Método POST..38

Variables de entorno adicionales...39

Codificación de los datos...39

Ejercicio 5: Variables de entorno..39

Solución...39

Formas en HTML...40

Elementos que contiene la etiqueta FORM...40

Uniendo todo: Manejo de los datos de una forma, seguridad..............43

Ejercicio 6: Uso de formas y CGI...46

Solución...47

Introducción a Javascript..51

Objetos en Javascript..52

Métodos de los objetos en Javascript. Programación orientada a eventos.............53

Hola mundo en Javascript..54

Variables en Javascript..55

Funciones en Javascript...55

Estructuras de repetición...56

Sintaxis y ejemplo de Do – while...56

Sintaxis y ejemplo de For...56

Sintaxis y ejemplo de For in...56

Sintaxis y ejemplo de while:...57

Estructuras de decisión...57

Cajas de alerta, confirmación y preguntas...58

Expresiones regulares y arreglos...59

Arreglos...59

Expresiones regulares..60

Formas y validación..61

Ejercicio 7: Validación de Formas con Javascript...............................64

Solución...64

Referencias bibliográficas 67

Autor: José Vicente Núñez Zuleta (jose@ing.ula.ve)..1

Tabla de contenido**...2**

Indice de tablas**...4**

Indice de ilustraciones**...6**

Objetivos de este trabajo**..9**

Introducción al Lenguaje Perl**..10**

Obtención en instalación de Perl 10

Perl básico**...11**

Hola mundo..11

Uso de variables y asideros de archivo[6]...11

Más sobre el uso de variables...12

Más sobre el uso de asideros..13

Arreglos, Arreglos asociativos y lazos..13

Arreglos y lazos con for..13

Arreglos asociativos y lazos con foreach...14

Lazos con while y uso de split()...14

Estructuras de decisión..15

Uso de patrones en búsquedas y reemplazos, usando expresiones regulares 17

Ejercicio 1: Uso de estructuras de decisión, patrones y estructuras de repetición 19

Solución:...19

Uso de rutinas en Perl...19

Referencias en Perl...21

Ejercicio 2: Uso de rutinas y referencias..22

Solución..23

Validación de programas en Perl..25

Seguridad en Perl...25

Depuración de programas en Perl..26

Perl en modo de advertencias (–w)..26

Perl en modo de depuración (–d)..27

Listando código fuente y el contenido de las variables.......................27

Ejecutando el código fuente...28

Puntos de quiebre, ejecución continua del programa..........................28

Busqueda con patrones...29

Corregir sólo la sintaxis con –c...29

Ejercicio 3: Seguridad en Perl y depuración de programas................29

Solución..30

CGI (Common Gateway Interface)...31

Protocolos de comunicación..31

Modelo OSI y TCP/IP..31

HTTP..32

Modelo cliente – servidor..33

El cliente..33

El servidor..33

CGI..34

Funcionamiento de CGI..34

Generación dinámica de HTML..35

Ejercicio 4: Generación dinámica de HTML....................................36

Solución:..36

Métodos de comunicación..37

Método GET....38

Método POST....38

Variables de entorno adicionales....39

Codificación de los datos....39

Ejercicio 5: Variables de entorno....39

Solución....39

Formas en HTML....40

Elementos que contiene la etiqueta FORM....40

Input: Recolecta información acerca del usuario....40

Option: Ocurre dentro del elemento select, el cual le permite al usuario escoger entre varias alternativas, y es usado para representar cada opción de select....41

Select: Le permite al usuario escoger entre múltiples alternativas....41

Textarea: Recolecta múltiples líneas de texto por el usuario. Al usuario se le presenta un panel cuadrado en el cual puede escribir. Se usa en pares....41

Uniendo todo: Manejo de los datos de una forma, seguridad....43

Ejercicio 6: Uso de formas y CGI....46

Solución....47

Introducción a Javascript....51

Objetos en Javascript....52

Métodos de los objetos en Javascript. Programación orientada a eventos....53

Hola mundo en Javascript....54

Variables en Javascript....55

Funciones en Javascript....55

Estructuras de repetición....56

Sintaxis y ejemplo de Do – while....56

Sintaxis y ejemplo de For....56

Sintaxis y ejemplo de For in....56

Sintaxis y ejemplo de while:....57

Estructuras de decisión...57

Cajas de alerta, confirmación y preguntas..58

Expresiones regulares y arreglos..59

Arreglos..59

Expresiones regulares...60

Formas y validación..61

Ejercicio 7: Validación de Formas con Javascript.................................64

Solución..64

Referencias bibliográficas 67

Indice de ilustraciones

Autor: José Vicente Núñez Zuleta (jose@ing.ula.ve)..1

Tabla de contenido...2

Indice de tablas..4

Indice de ilustraciones...6

Objetivos de este trabajo..9

Introducción al Lenguaje Perl..10

Obtención en instalación de Perl 10

Perl básico...11

Hola mundo...11

Uso de variables y asideros de archivo[6]..11

Más sobre el uso de variables...12

Más sobre el uso de asideros...13

Arreglos, Arreglos asociativos y lazos...13

Arreglos y lazos con for...13

Arreglos asociativos y lazos con foreach..14

Lazos con while y uso de split()...14

Estructuras de decisión..15

Uso de patrones en búsquedas y reemplazos, usando expresiones regulares 17

Ejercicio 1: Uso de estructuras de decisión, patrones y estructuras de repetición 19

Solución:...19

Uso de rutinas en Perl...19

Referencias en Perl...21

Ejercicio 2: Uso de rutinas y referencias..22

Solución...23

Validación de programas en Perl..25

Seguridad en Perl..25

Depuración de programas en Perl...26

Perl en modo de advertencias (–w)..26

Perl en modo de depuración (–d)..27

Listando código fuente y el contenido de las variables.......................................27

Ejecutando el código fuente...28

Puntos de quiebre, ejecución continua del programa..28

Busqueda con patrones..29

Corregir sólo la sintaxis con –c...29

Ejercicio 3: Seguridad en Perl y depuración de programas................29

Solución...30

CGI (Common Gateway Interface)...31

Protocolos de comunicación..31

Modelo OSI y TCP/IP...31

HTTP...32

Modelo cliente – servidor..33

El cliente...33

El servidor...33

CGI...34

Funcionamiento de CGI..34

Generación dinámica de HTML..35

Ejercicio 4: Generación dinámica de HTML......................................36

Solución:...36

Métodos de comunicación...37

Método GET..38

Método POST...38

Variables de entorno adicionales..39

Codificación de los datos..39

Ejercicio 5: Variables de entorno...39

Solución...39

Formas en HTML...40

Elementos que contiene la etiqueta FORM..40

Input: Recolecta información acerca del usuario...40

Option: Ocurre dentro del elemento select, el cual le permite al usuario escoger entre varias alternativas, y es usado para representar cada opción de select...41

Select: Le permite al usuario escoger entre múltiples alternativas........................41

Textarea: Recolecta múltiples líneas de texto por el usuario. Al usuario se le presenta un panel cuadrado en el cual puede escribir. Se usa en pares...41

Uniendo todo: Manejo de los datos de una forma, seguridad..............43

Ejercicio 6: Uso de formas y CGI...46

Solución...47

Introducción a Javascript..51

Objetos en Javascript...52

Métodos de los objetos en Javascript. Programación orientada a eventos.............53

Hola mundo en Javascript..54

Variables en Javascript...55

Funciones en Javascript..55

Estructuras de repetición..56

Sintaxis y ejemplo de Do – while..56

Sintaxis y ejemplo de For...56

Sintaxis y ejemplo de For in...56

Sintaxis y ejemplo de while:...57

Estructuras de decisión..57

Cajas de alerta, confirmación y preguntas..58

Expresiones regulares y arreglos..59

Arreglos..59

Expresiones regulares...60

Formas y validación...61

Ejercicio 7: Validación de Formas con Javascript...............................64

Solución..64

Referencias bibliográficas 67

Objetivos de este trabajo

Está trabajo teórico / práctico persigue los siguientes objetivos:

- Obtención de destrezas básicas en la programación y depuración de programas en Perl.
- Comprensión de que es y como funciona CGI, así como la elaboración y depuración de programas sencillos.
- Introducir a la programación con Javascript y la validación de formas en aplicaciones cliente – servidor.
- Introducir a la programación de scripts seguros (con y sin CGI).

Durante todo este trabajo se presenta código fuente como ejemplo para explicar los conceptos teóricos. Al final de cada sección se proponen ejercicios para que sean realizados por el lector.

La plataforma utilizada para la elaboración de los ejercicios fue *Linux Slackware 3.4*, por lo que algunos comandos podrían variar de plataforma en plataforma. El servidor Web utilizado para correr los ejemplos fue *Apache versión 1.3.0*. Los programas hechos en *Javascript* fueron probados con *Netscape Communicator 4.05*. La versión de *Perl* Utilizada fue *5.04*, la versión de *Javascript fue 1.2*.

Algunos conceptos fueron omitidos o tratados de manera breve por razones de espacio. Nada puede reemplazar la práctica y la investigación, por lo que se remite al lector interesado a la bibliografía al final de este documento.

Se supone que el lector tiene conocimientos básicos de programación en algún lenguaje estructurado, que tiene conocimientos básicos sobre *Unix* y *HTML* y que sabe utilizar un *browser* para navegar por internet.

Sí encuentra fallas o desea hacer algún comentario adicional puede escribir a:

Jose@ing.ula.ve

José Vicente Núñez Zuleta.

Perl es un lenguaje que permite la manipulación de archivos de texto y procesos. Perl provee una manera concisa y fácil para hacer las cosas las cuales pueden más difíciles en *C o en Shell*. Esas fueron las ideas que motivaron a su autor (Larry Wall)**[11]**.

En el principio, Perl fue pensado como un leguaje de reducción de datos, Un lenguaje que permitiera navegar con facilidad y de manera arbitraria por archivos de texto de manera eficiente; Sin embargo Perl ha evolucionado tanto que hoy en día se le considera como una herramienta de programación en Internet y Administración de sistemas y redes Unix.

Las siguientes son algunas de las razones por las cuales Perl es popular:

- Reduce el ciclo de programación. No tiene que compilar su aplicación, Perl es interpretado y por ello sus programas pueden ser corridos en muchas plataformas sin necesidad de ser recompilado.[1]
- Es portable, ya que hay un interpretador de Perl para cada variedad de Unix y Windows, por lo que los cambios que debe hacer a su aplicación son mínimos o nulos.
- Puede hacer mejor muchas cosas que serían más difíciles en otros lenguajes como C o Shell, como la manipulación de archivos de texto.
- La sintaxis de otros lenguajes como *Shell, Sed, AWK* o *C* es muy similar a la de Perl. Inclusive cuenta con herramientas para traducir código de Sed y AWK a Perl de manera automática.
- Es extensible. En Internet puede conseguir una enorme cantidad de módulos los cuales pueden ser incluidos en sus programas sin ninguna dificultad. Si lo desea, puede desarrollar sus propias extenciones.
- No cuesta nada. Perl esta protegido por una *licencia artística*, la cual permite su libre distribución.
- Es confiable y robusto. Programas como dnswalk, Majordomo y otros están hechos en Perl.

Sin embargo, pueden haber cosas del lenguaje que no le gusten como:

- Cualquiera puede ver el código fuente de su aplicación porque el código es interpretado y no compilado.
- Por ser interpretado y no compilado su velocidad puede ser inferior a la versión en C en algunos casos.

Teniendo esto en cuenta, usted puede decidir si Perl se adapta o no a sus necesidades.

Obtención en instalación de Perl

Puede conseguir la última versión de Perl[2] en su página Web **[8]**:

http://www.perl.com

Básicamente, lo que debe hacer para instalar Perl es lo siguiente:
- Descomprimalo con *gunzip xxx.tar.gz*
- Desempaquetelo en un directorio con *tar –xvf xxx.tar.gz*
- Ejecute el script *configure (por ejemplo configure –Dcc=gcc)*
- Escoja el sistema operativo sobre el cual correrá Perl

[1] Perl es rápido porque está hecho en C. De hecho, la diferencia de velocidad entre un script hecho en Perl y un programa en C no es muy grande.
[2] En este trabajo utilizamos la versión 5.004

- De aquí en adelante responderá muchas preguntas, de acuerdo a su sistema operativo (Crear las dependencias del programa con make depend, entre otros)[3]
- Luego corra los tests (el instalador le dirá como) y finalmente ejecute el comando make install.

Perl básico

Hola mundo

A continuación se muestra el programa *"hola mundo"*, escrito en Perl:[4]

```
1. #!/usr/bin/perl
2. print("Hola mundo cruel\n");
```

Guarde este pequeño fragmento de código como *hola_mundo.pl*, cambie su permisología a *ejecución sólo por el usuario*, y córralo:

$host> ***chmod u+x hola.pl***
$host> ***hola.pl***
Hola mundo

La línea 1 contiene lo que se conoce como una *"galleta mágica"* y le indica al programa en donde se encuentra el interpretador de Perl. La segunda línea le dice a Perl que imprima un mensaje por pantalla (Nótese el enorme parecido con la orden printf de C). Fíjese que cada comando en Perl termina en punto y coma ";".

Otra forma de correr ese script es eliminando la primera línea y diciéndole a Perl que corra el script directamente:

```
$host> perl hola.pl
Hola mundo
$host>
```

La línea de comandos de Perl es muy extensa, por lo que mencionaremos sólo lo necesario[5].

Uso de variables y asideros de archivo[6]

El programa anterior es bastante limitado, por lo que extenderemos su utilidad con el uso de variables:

```
1. #!/usr/bin/perl
2. print("Por favor introduzca su nombre\n");
3. $nombre=<STDIN>;
4. print("Hola $nombre, ese es un nombre bonito\n");
```

En la línea 3 hace una asignación a la variable *$nombre* de lo que se "*capture"* por el asidero de archivos estándar *STDIN* (Standard Input). Luego mostramos el contenido de esa variable en la línea 4. *STDIN*, *STDOUT* y *STDERR* son llamados *asideros* y vienen por omisión en cualquier sistema operativo, por lo que Perl provee acceso directo a *ellos. STDIN, STDOUT y STDERR* son *entrada, salida y error estándar* respectivamente. Note como al declarar a la variable $nombre no se le especifico ningún tipo.

Corramos el programa para ver que hace:

$host> **hola.pl**

---

[3] La instalación de Perl es algo larga, así que busque una tasa de café y ármese de paciencia

[4] No recuerdo quien me lo dijo, pero el programa "Hola mundo" era el programa que había sido portado a más lenguajes que cualquier otro.

[5] Escriba man perl si desea saber más.

[6] Se tradujo el término "filehandles" a asideros de archivo.

Escriba por favor su nombre
Jose Vicente
Hola Jose Vicente
, ese es un nombre bonito

¡Hay algo mal!, El programa está capturando el retorno de carro introducido después de obtener el nombre. Eso lo podemos solucionar con la rutina *chop(),* la cual elimina el último carácter de una variable:

```
1.  #!/usr/bin/perl
2.  print("Escriba por favor su nombre\n");
3.  $nombre=<STDIN>;
4.  chop($nombre); # Eliminamos el retorno de carro
5.  print("Hola $nombre, ese es un nombre bonito\n");
```

Vea como en la línea 4 introducimos un comentario, utilizando el carácter #.

Corra el programa de nuevo y note el cambio.

En Perl:

- No se utilizan tipos de datos en la declaración de variables
- Las variables pueden declararse a cualquier altura del código
- Los asideros permiten acceder a recursos del sistema como la salida y entrada por omisión.

Más sobre el uso de variables

En Perl no hace falta declarar el tipo de una variable como en C o Pascal; Otra diferencia es que Perl internamente utiliza sólo dos tipos de variables: Cadena de caracteres o números reales. Veamos por ejemplo, como el siguiente programa es correcto.

```
#!/usr/bin/perl
# Colocamos una cadena de caracteres
$variable="Hola, contengo una cadena";
print("El contenido de \$variable es :\t $variable\n");
# Colocamos ahora un numero real
$variable=3.141616;
print("El contenido de \$variable es :\t $variable\n");
# Colocamos ahora un entero
$variable=55;
print("El contenido de \$variable es :\t $variable\n");
```

Veamos su salida por pantalla:

```
leon[80] $host> variables.pl
El contenido de $variable es :   Hola, contengo una cadena
El contenido de $variable es :   3.141616
El contenido de $variable es :   55
```

Hemos introducido dos trucos nuevos: el uso del carácter "\" delante de otro (Le dice a Perl que no interprete el carácter, sino que lo proteja y "\t" (Deje una tabulación).

Veamos otras formas de asignación de variables:

```
$respuesta=666; # Un número entero
$pi=3.141516;   # Un número real
$cantidad=55e2; # Notación cientifica
$mascota='Godzila';     #Una cadena
$aviso="$mascota, cuando el tamaño cuenta\n";   # Interpolación de variables
$Directorio='ls';       # Guarda el resultado de la ejecución de un comando
```

Note que Perl distingue el uso de mayúsculas y minúsculas en la declaración de variables.

Más sobre el uso de asideros

La importancia real de los asideros está en que nos permiten redirigir la entrada y la salida de información en nuestro programa. Veamos brevemente las operaciones posibles con los asideros:

```
open(ASIDERO,"nombre del archivo");     # Abre el archivo para lectura
open(ASIDERO,">nombre del archivo");    # Abre el archivo para lectura, escritura
open(ASIDERO,">>nombre del archivo");   # Le agrega al archivo
open(ASIDERO,"| comando de salida");    # Envía salida a un filtro
open(ASIDERO,"nombre del archivo |");   # Recoge la salida de un filtro
```

En el siguiente ejemplo, los asideros se utilizarán para acceder a las facilidades del sistema:

```
1.  #!/usr/bin/perl
2.  # Veamos el contenido del archivo /etc/hosts
3.  # para ello se utiliza cat y eliminamos todos los
4.  # comentarios
5.  open(ARCHIVO,"cat /etc/hosts|grep -v '#' |");
6.  print <ARCHIVO>,"\n";
7.  # cerramos el archivo (No hace falta, perl lo hace solo)
8.  close(ARCHIVO);
```

Fíjese como en la línea 5 abrimos un asidero, utilizando un filtro compuesto por *cat* y *grep* para luego mostrar el resultado imprimiendo el contenido del asidero. Más adelante se explicará porque este programa es *inseguro*.

Arreglos, Arreglos asociativos y lazos

Perl cuenta con ciertas estructuras de repetición, las cuales le permiten procesar cierto grupo de instrucciones de manera repetitiva.

Arreglos y lazos con for

Un arreglo no es más que un conjunto de elementos almacenados en direcciones contiguas de memoria. Veamos como funcionan con el siguiente ejemplo:

```
1.  #!/usr/bin/perl
2.  @usuarios=('jose','luis','pepe'); #Nombres de los usuarios
3.  @peso_kg=(70.5,90,65.4);          #Peso
4.  $num=@usuarios;
5.  print("Usuarios a ser procesados: $num\n");
6.  for($i=0;$i<=$#usuarios;$i++) {
7.  print("Nombre:\t$usuarios[$i],\tPeso:\t$peso_kg[$i]\n");
8.  }
```

En las líneas 1 y dos se muestra como se puede inicializar a un arreglo (Denotado con el símbolo @). Obtenemos la cantidad de elementos que tiene almacenados con la instrucción que se muestra en la línea 3 y luego mostramos elemento por elemento en un lazo de repetición (utilizando un blucle *for*) el cual se mueve desde el primer elemento (con índice 0) hasta el último *($#usuarios*), incrementando el contador de uno en uno.

Arreglos asociativos y lazos con foreach

Este programa pudo ser implementado de otra forma. Perl cuenta con una estructura llamada *arreglo asociativo* o *hash*, la cual asocia una clave con un valor:

```
#!/usr/bin/perl
# Guardamos el nombre y peso de la persona en un solo sitio
%usuarios=(
'jose', 70.5,
'luis', 90,
'pepe', 65.4,
);
# Averiguamos cuantos elementos tiene el hash
$num=0;
# Repetimos el lazo por cada una de las claves del
# hash, incrementando el contador
foreach $aux (keys(%usuarios)) {
```

```
13. $num++;
14. }
15. print("Usuarios a ser procesados: $num\n");
16. foreach $aux (keys(%usuarios)) {
17. print("Nombre:\t$aux,\tPeso:\t$usuarios{$aux}\n");
18. }
```

Fíjese como la iniciación de un hash (líneas 3 a la 7) implica llenar dos valores: uno de ellos es la clave (en este caso el nombre) y el otro su valor asociado (el peso). Note también la forma de acceder a los valores del hash: Si no conocemos las claves, entonces las obtenemos por medio de la función *keys*(), luego vemos el contenido asociado usando $hash{$clave}.

El comando *foreach*[7] obtiene todas las claves del hash usuarios con ayuda de keys() y hace un recorrido (una por una) guardándolas en $aux.

Lazos con while y uso de split()

Los lazos con for y foreach son útiles, pero tienen limitaciones cuando se trata de utilizar una condición lógica complicada en la estructura de repetición. Es allí donde el uso de lazos con while se hace obligatorio. Ilustraremos esto con un programa que lee el contenido de una base de datos:

```
150.185.128.1 merlin SunOS
150.185.128.15 gandalf Solaris
150.185.128.2 arha SunOS
150.185.128.128 randu   Linux
150.185.128.12 melchor  SunOS
150.185.128.80 medussa Solaris
150.185.128.123 sorceles Linux
150.185.131.1 zeus Windows
```

Veamos el código que permite ver sus valores por pantalla:

```
1.  #!/usr/bin/perl
2.  print("Cargando la base de datos...");
3.  open(ARCHIVO,"maquinas.txt") || die "no pude abrir el archivo $!\n";
4.  print("listo!\n");
5.  while(<ARCHIVO>) {
6.  chop($_);       # Eliminamos el retorno de carro
7.  ($ip,$nombre,$so)=split(' ',$_);
8.  print("Nombre :$nombre <$ip,$so>\n");
9.  }
```

En la línea 2 se utiliza el operador lógico *//* ("or", más adelante se explicará con más detalle) para validar la si se ha podido abrir el archivo; Si falla la apertura del archivo, la función *die()* muestra un mensaje por pantalla y aborta la ejecución del script; Este es el uso más común de die().

En la línea 5 el programa entra en el lazo condicional de un bucle while. Este continuará repitiéndose indefinidamente mientras la condición encerrada entre paréntesis sea verdadera (En este caso, que no haya fin de archivo). Si la condición no se cumple, se sale del bucle, o nunca se entra. En este caso cada ciclo del bucle avanza el apuntador al archivo que al principio estaba en la primera línea a la siguiente, hasta que se llegue al fin del archivo. Como se abrió el archivo para lectura, cada línea es "leída del archivo" y es referenciada por <ARCHIVO>.

En la línea 6 se utiliza una variable especial llamada "*$_*". Significa *"la variable actual",* y en este caso se refiere a la línea que esta siendo leída en el bucle. Se necesita quitar el carácter de retorno de carro al final de la línea, por lo que se utiliza a la rutina *chop()*.

En la línea 7 se divide a la variable $_ en tres partes para ser almacenados en *$ip, $nombre y $so* respectivamente, con ayuda de la rutina *split(). Split* utiliza como separador de campos un espacio en blanco (hablamos de un separador de campos ya que cada palabra en la línea esta separada por un espacio).

[7] Si ha programado en Shell, este comando le debe resultar familiar.

En la línea 8 se muestra el contenido de la base de datos con la función *print()*.

Estructuras de decisión

Las estructuras de decisión tienen la siguiente forma en Perl:

If (condición) {
Código;
} elsif (condición) {
más código;
} else {
aún más código;
}

Supongamos que en el ejemplo anterior, se modifica a la base de datos para indique si la máquina es un servidor o no, el programa debe mostrar el nuevo campo por pantalla. Se cometio un error intencional en el archivo, colocando el tipo de *medussa* en minúsculas. Este tipo de error de entrada de datos es muy común, y se solucionará más adelante:

```
150.185.128.1 merlin SunOS Servidor
150.185.128.15 gandalf Solaris Cliente
150.185.128.2 arha SunOS Servidor
150.185.128.128 randu   Linux Cliente
150.185.128.12 melchor  SunOS Cliente
150.185.128.80 medussa Solaris cliente
150.185.128.123 sorceles Linux Servidor
150.185.131.1 zeus Windows Cliente
```

El siguiente código muestra el uso de las estructuras de decisión:

```
1.  #!/usr/bin/perl
2.  print("Cargando la base de datos...");
3.  open(ARCHIVO,"maquinas2.txt") || die "no pude abrir el archivo $!\n";
4.  print("listo!\n");
5.  while(<ARCHIVO>) {
6.  chop($_);       # Eliminamos el retorno de carro
7.  ($ip,$nombre,$so,$tipo)=split(' ',$_);
8.  if ($tipo eq "Servidor") {
9.  print("Servidor->\t");
10. } elsif ($tipo eq "Cliente") {
11. print("Cliente->\t");
12. } else {
13. print("Desconocido->\t");
14. }
15. print("Nombre :$nombre <$ip,$so>\n");
16. }
```

En la línea 8 se valida que la variable $tipo sea igual a "*Servidor*", utilizando el comparador de cadenas; Si no se cumple entonces comparamos para ver si es un "*Cliente*". Si no es ninguno de los dos tipos le asignamos "*Desconocido*":

```
leon[246] Si maestro while2.pl
Cargando la base de datos...listo!
Servidor->      Nombre :merlin <150.185.128.1,SunOS>
Cliente->       Nombre :gandalf <150.185.128.15,Solaris>
Servidor->      Nombre :arha <150.185.128.2,SunOS>
Cliente->       Nombre :randu <150.185.128.128,Linux>
Cliente->       Nombre :melchor <150.185.128.12,SunOS>
Desconocido->   Nombre :medussa <150.185.128.80,Solaris>
Servidor->      Nombre :sorceles <150.185.128.123,Linux>
Cliente->       Nombre :zeus <150.185.131.1,Windows>
```

La comparación de cadenas presenta el inconveniente de que la comparación es absoluta, es decir la cadena buscada tiene que coincidir de manera exacta con la cadena candidata, sino falla la.

Las condiciones lógicas de los *if – else* pueden ser tan complejas como se desee, y pueden estar anidadas si se quiere. Suponga por ejemplo que ninguna de las máquinas con Windows es servidora, entonces podría hacerce la siguiente modificación al programa en la línea 8, utilizando una comparación compuesta unida por una condición *"y":*

```
if (($tipo eq "Servidor") && ($so ne "Windows")) {
```

Se muestra a continuación un pequeño resumen de las posibles comparaciones lógicas que pueden hacerce bajo Perl:

<table>
<tr><th>Operador lógico</th><th>Significado</th><th>Resultado</th></tr>
<tr><td>$a && $b</td><td>And (Y)</td><td>Verdadero si a y b son verdaderos</td></tr>
<tr><td>$a || $b</td><td>Or (O)</td><td>$a si $a es verdadero, de lo contrario $b</td></tr>
<tr><td>! $a</td><td>Not (Negación)</td><td>Verdadero si $a es falso</td></tr>
</table>

• Tabla 1: Operadores lógicos posibles bajo Perl

Y estas son las comparaciones entre variables que se pueden hacer:

Prueba numérica	Prueba de cadena	Significado
==	Eq	Es igual a
!=	En	diferente
>	Gt	Más grande que
>=	Ge	Mayor o igual
<	Lt	Menor que
<=	Le	Menor o igual
< = >	Cmp	No es igual, con signo

• Tabla 2: Comparaciones lógicas posibles

Uso de patrones en búsquedas y reemplazos, usando expresiones regulares

Una de las características más atractivas de Perl es que permite identificar o cambiar porciones de texto de manera flexible, utilizando lo que se conocen como patrones. La sintaxis de un patrón es:

Patrón de búsqueda	Resultado
$variable =~ /patrón/	Retorna verdadero si patrón está dentro de $variable
$variable =~ s/patrón viejo/nuevo patrón/	Sustituye el patrón viejo por el patrón nuevo
$variable =~ tr/a–z/A–Z/	Convierte de minúsculas a mayúsculas el contenido de $variable.

• Tabla 3: Patrones de búsqueda y reemplazo más comunes

El programa que se mostró en el paso anterior tenía el inconveniente de que realizaba comparaciones absolutas; Para un usuario seria más cómodo poder escribir en mayúsculas o minúsculas en el archivo de datos y que el progerama se ocupara de corregir la apariencia del contenido; También sería comodo poder utilizar uno o más espacios o tabuladores para separar la información de cada máquina:

```
150.185.128.1           merlin SunOS Servidor
150.185.128.15 gandalf Solaris     Cliente
150.185.128.2 arha SunOS SERVIDOR
150.185.128.128 randu   Linux cLiENte
150.185.128.12 melchor  SunOS Cliente
150.185.128.80   medussa Solaris cliente
150.185.128.123 sorceles Linux Servidor
150.185.131.1   zeus Windows CliENTE
```

El siguiente código permite hacer eso utilizando caracteres especiales y expresiones regulares:

```
#!/usr/bin/perl
print("Cargando la base de datos...");
open(ARCHIVO,"maquinas3.txt") || die "no pude abrir el archivo $!\n";
```

```
4.  print("listo!\n");
5.  while(<ARCHIVO>) {
6.  chop($_);       # Eliminamos el retorno de carro
7.  ($ip,$nombre,$so,$tipo)=split('\s+',$_);
8.  $tipo =~ tr/a-z/A-Z/; #Convertimos el tipo a mayusculas
9.  $aux=$so; # Guardamos el viejo valor de $so
10. $so =~ tr/a-z/A-Z/; #Convertimos $so a mayusculas
11. if (($tipo eq "SERVIDOR") && ($so ne "WINDOWS")) {
12. print("Servidor->\t");
13. } elsif ($tipo eq "CLIENTE") {
14. print("Cliente->\t");
15. } else {
16. print("Desconocido->\t");
17. }
18. $so=$aux;
19. print("Nombre :$nombre <$ip,$so>\n");
20. }
```

En la línea 7 se sutituye el espacio en blanco (que es poco flexible), por la expresión ***\s+.*** Para entender lo que significa, se descompondrá la expresión:

- Proteja el carácter "s" poniéndole un \ adelante[8]
- El carácter s significa un carácter de espacio (espacio en blanco, tabulación).
- + significa uno o más

La conversión de minúsculas a mayúsculas se hace en las líneas 8 y 10. En las líneas 11 y 13 se hace una comparación con los valores, pero en mayúsculas, lo cual facilita la programación.

La siguiente tabla contiene caracteres para expresiones regulares:

Carácter especial	Significado
.	Encuentra cualquier carácter, excepto una nueva línea
[a–z0–9]	Encuentra cualquier carácter del conjunto
[^a–z–0–9]	Encuentra cualquier carácter que no este en el conjunto
\d	Encuentra cualquier dígito
\D	Encuentra cualquier cosa que no sea un dígito
\w	Encuentra cualquier carácter alfanumérico
\W	Encuentra cualquier carácter no alfanumérico
\s	Encuentra un carácter de espacio (espacio, tabulación, nueva línea)
\S	Encuentra un carácter que no sea de espacio (espacio, tabulación, nueva línea)
\n	Encuentra una nueva línea
\r	Encuentra un retorno
fi|fa|fo	Encuentra a fi o a fo o a fu en una cadena de caracteres
x?	Encuentra uno o cero x
x*	Encuentra cero o más x
x+	Encuentra una o más x
\b	Encuentra dentro de un bloque de palabra
\B	Encuentra fuera de un bloque de palabra
^	Encuentra al principio de la línea
$	Encuentra al final de la línea

- Tabla 4: Algunas expresiones regulares

Ejercicio 1: Uso de estructuras de decisión, patrones y estructuras de repetición

Haga un programa que muestre el siguiente menú por pantalla:

[8] Proteger en este caso significa que no lo interprete como un patrón de busqueda.

```
¡Bienvenido al la votación del mundial, Francia 98!:

Por favor, escriba el nombre del país por el cual desea votar:
1.  Brasil
2.  Colombia
3.  Alemania
4.  Argentina
5.  Salir: escriba "gol" o "salir"

Para salir debe escribir "gol" o "salir". Si el usuario escribe una opción que no
está especificada, el programa vuelve a mostrar el menú.
```

Al terminar el programa, debe mostrar la cantidad de votos obtenidos por cada País.

Solución:

```
#!/usr/bin/perl -w
$respuesta="";
%votacion=(
brasil=>0,
colombia=>0,
alemania=>0,
argentina=>0,
);
```

```
while (($respuesta ne "gol") && ($respuesta ne "salir")) {
if ($respuesta eq "brasil") {
$votacion{'brasil'}++;
} elsif ($respuesta eq "colombia") {
$votacion{'colombia'}++;
} elsif ($respuesta eq "alemania") {
$votacion{'alemania'}++;
} elsif ($respuesta eq "argentina") {
$votacion{'argentina'}++;
}
print("Bienvenido al la votación del mundial, Francia 98!:\n");
print("Por favor, escriba el nombre del país por el cual desea votar:\n");
printf("1)  Brasil\n2)  Colombia\n3)  Alemania\n4)  Argentina\n5)  Salir:  escriba
\"gol\" o \"salir\"\n");
$respuesta=<STDIN>;
chop($respuesta);
$respuesta =~ tr/A-Z/a-z/;
}
print("Resultados de la votacion por pais\n");
foreach $pais (sort keys(%votacion)) {
print("$pais: $votacion{$pais}\n");
}
```

Uso de rutinas en Perl

El uso de rutinas se hace indispensable en cualquier lenguaje de programación. Permiten mantener el tamaño del código dentro de límites aceptables, a la vez que facilitan la programación.

Una rutina se define de la siguiente manera:

```
sub mirutina ([parámetro]) {
[return(algo)];
}
```

Parámetro es opcional y se usa para obligar al usuario de la rutina a utilizarla con la cantidad de parámetros adecuados. Si desea especificar los parámetros, escriba *$* para una variable tipo escalar, *@* para un arreglo, *%* para un hash y *&* para una referencia a una rutina.[9] Una rutina puede retornar o no algun valor.

Todos los parámetros que se pasan a una subrutina vienen contenidos en la variable especial *@_*. Si desea acceder a algún elemento en especial de ese arreglo puede hacerlo con *$_[índice],* donde indice va desde 0 hasta el número de parámetros que tenga el arreglo.

Si desea que los cambios efectuados en una variable permanezcan aún después de finalizada la rutina, deberá utilizar un parámetro por referencia (similar a los punteros en C); Para ello colóquele un \ delante al tipo de parámetro (Esto es obligatorio cuando se pasa

[9] Sin embargo, nuestra rutina puede funcionar perfectamente sin pasarle ningún parámetro.

más de un arreglo como parámetro a una función). En Perl todas las variables que no son de tipo hash o arreglo (es decir, de tipo escalar) son pasadas por referencia.

Además de especificar el formato de un parámetro, estos pueden ser declarados como locales a la función utilizando la palabra reservada *my* o *local* (La diferencia entre ambas es que *my* no permite que una subrutina llamada dentro de una rutina modifique una variable, mientras que *loca*l sí).

La sintaxis para invocar una rutina es la siguiente:

```
&mirutina;
```

El siguiente programa convierte la primera letra de una cadena a mayusculas y el resto a minusculas[10]:

```
1.  #!/usr/bin/perl

1.  # Esta rutina convierte el contenido de la cadena de minúsculas
2.  # a mayúsculas
3.  sub mayusculas($) {
4.  $_[0] =~ tr/a-z/A-Z/; # Modificamos el contenido del primer arg.
5.  }

1.  # Adivine lo que hace esta rutina :-)
2.  sub minusculas($) {
3.  $_[0] =~ tr/A-Z/a-z/;
4.  }

1.  # Esta rutina solo pone en mayúscula el primer carácter
2.  sub oracion($) {
3.  # Si el primer carácter es una letra
4.  if ($_[0] =~ /^[a-z]|^[A-Z]/ ) {
5.  #Extraigo el primer carácter de la cadena
6.  local($primer)=substr($_[0],0,1);
7.  #Extraigo el resto de la cadena
8.  local($resto)=substr($_[0],1,length($_[0]));
9.  #Convierto el resto de la cadena a minusculas
10. #y la primera letra a mayusculas
11. &minusculas($resto);
12. &mayusculas($primer);
13. #Uno las dos subcadenas
14. $_[0]=$primer.$resto;
15. }
16.
17. $palabra="hola";
18. print("Antes: $palabra\n");
19. &oracion($palabra);
20. print("Después: $palabra\n");
21. $palabra="1hola";
22. print("Antes: $palabra\n");
23. &oracion($palabra);
24. print("Después: $palabra\n");
25. $palabra="hOlA";
26. print("Antes: $palabra\n");
27. &oracion($palabra);
28. print("Después: $palabra\n");
29. $palabra="HolA";
30. print("Antes: $palabra\n");
31. &oracion($palabra);
32. print("Después: $palabra\n");
```

Fíjese como se declaró el tipo de parámetro que va a recibir cada una de las funciones en las líneas 4, 8, y 12 como un *escalar*. Como Perl siempre pasa los parámetros por referencia, podemos estar confiados que cualquier cambio que hagamos permanecerá aún después de terminar la ejecución de la rutina.

Las conversiones de mayúsculas a minúsculas (y viceversa) se hacen con *tr()* en la línea 5 (y 9); La detección de sí el primer carácter de la palabra es una letra se hacecon ayuda de una *expresión regular compuesto,* en la línea 14, la cual significa: "*encuentre al principio de la línea una letra que sea minúscula o al principio de la línea una letra que sea mayúscula".*

[10] Esto se puede hacer más fácilmente con las funciones lcfirst, ucfirst, uc y lc.

Finalmente se concatenan los resultados en la variable original con el operador de interpolación de Perl (".").

La siguiente es una salida de ejemplo:

```
leon[121] $host> rutinas.pl
Antes: hola
Despues: Hola
Antes: 1hola
Después: 1hola
Antes: hOlA
Después: Hola
Antes: HolA
Después: Hola
```

Referencias en Perl

Una referencia es un apuntador a algo; El concepto de referencias en Perl es muy similar al concepto de apuntador en C o Pascal. Cuando se dice algo, nos referimos a un escalar, un arreglo, un hash o incluso una subrutina.

La siguiente tabla muestra como usar una referencia en Perl:

Tipo	Asignación	Ejemplo de uso
Escalar	$apuntador=\$variable;	print $$apuntador,"\n";
Arreglo	$apuntador=\@arreglo;	Print $$apuntador[5],"\n"; foreach (@$apuntador)
Arreglo asociativo	$apuntador=\%hash;	Print $$apuntador[$clave},"\n"; foreach $clave (keys %$apuntador)

• Tabla 5: uso de las referencias en Perl

Si desea utilizar más de un arreglo o hash como parámetro de una función, entonces debe pasarlos por referencia. Esto se debe a que Perl concatena todos los argumentos que le son pasados en una sola variable (@_). Para evitar que esto pase, usted debe:

- Pasar una referencia en el argumento de una función
- Utilizar el apuntador dentro de la función, con la sintaxis indicada en la tabla anterior.

El siguiente programa toma un arreglo y un hash e imprime su contenido por pantalla:

```
#!/usr/bin/perl

sub refer (\@\%) {
my($arreglo,$hash)=@_;
print("imprimendo el arreglo\n");
$i=0;
foreach (@$arreglo) {
print("$i : $$arreglo[$i]\n");
$i++;
}
print("imprimendo el hash\n");
foreach $i (keys %$hash) {
print("$i : $$hash{$i}\n");
}
}

@a=(1,2,3,4,5,6);
%b=(
1=>'Uno',
2=>'Dos',
3=>'Tres',
4=>'Cuatro',
5=>'Cinco',
6=>'Seis',
);

&refer(\@a, \%b);
```

Los parámetros de la función esperan una referencia, por lo que le colocamos el operador \ en la línea 24.

Note como se pueden declarar los elementos de un hash, en las línea 17 a la 22, utilizando el operador => (En vez de *","*. Esto ayuda a clarificar el programa).

Ejercicio 2: Uso de rutinas y referencias

Suponga que le han encomendado la tarea de verificar la conectividad de un grupo de máquinas en su laboratorio lo más rápido posible. Use para ello el programa que muestra el contenido de la base de datos y modifíquelo para que este le haga un ping a cada una de las máquinas que se encuentran en el laboratorio; El programa debe interpretar la respuesta del programa ping y basándose en eso muestra un mensaje por pantalla.

También se desea que el archivo de entrada pueda tener comentarios (En una línea individual, marcados con #), líneas en blanco y que al ser invocado de la siguiente manera produzca los siguientes resultados:

Verifica_red.pl, dice solamente si la máquina con dirección ip xx.yy.zz.ww está viva.
Verifica_red.pl –i, dice si la máquina con la dirección ip xx.yy.zz.ww está viva y si es o no un servidor y su sistema operativo. Para procesar la línea de comandos utilice el arreglo especial **@ARGV**, en el cual se guarda la línea de comandos cuando se ejecuta un programa.

Se recomienda que utilice subrutinas para que el código sea más legible y fácil de mantener.

Solución

El formato del archivo propuesto es el siguiente:

```
# Maquinas de mi dominio
192.168.1.1             leon    Linux   Servidor
192.168.1.2             tigre   Linux/Windows95 Cliente
192.168.1.3             pantera Linux/Windows95 Cliente
192.168.1.4             puma    Linux/Windows95 Cliente

# Maquinas externas a mi red, pero de interés
150.185.128.1           merlin          SunOS           Servidor
150.185.128.15          gandalf         Solaris         Cliente
150.185.128.2           arha            SunOS           SERVIDOR
150.185.128.128         randu           Linux           cLiENte
150.185.128.12          melchor         SunOS           Cliente
150.185.128.80          medussa         Solaris         cliente
150.185.128.123         sorceles        Linux           Servidor
150.185.131.1           zeus            Windows         CliENTE
```

El siguiente es el código fuente que realiza el trabajo:

```
#!/usr/bin/perl

# Este script verifica el funcionamiento de
# un grupo de maquinas funcionando en una red
# usando la herramienta ping
#
# Hecho por José Vicente Nuñez Zuleta (jose@ing.ula.ve)

# *************************************************]
# Configuración del script
#
$pingdir="/bin";        #Directorio donde va ping
$maqdir="./datos.txt";  #Ubicación y nombre del archivo de datos
$band="-c 1";

# *************************************************]
# Fin de la configuración del script

# Rutina que carga el contenido del archivo a memoria
sub cargar_red(\%\%\%) {
        my($ip,$so,$ti)=@_;     #Las variables vienen en @_
        # Leemos los datos y los guardamos en un hash
        open(DATOS,$maqdir) || die "No puedo cargar los datos de la red $!\n";
```

```
        my($aux,$aux2,$aux3,$aux4);     #Variable auxiliar
        #Comenzamos la lectura de los datos
        my $c=0;
        while(<DATOS>) {
                #Validamos que no sea un comentario o línea vacía
                chop($_);
                if (($_ !~ /^\#/) && ($_ ne "")) {
                        $c++; #Otra maquina leida
                        ($aux,$aux2,$aux3,$aux4)=split('\s+',$_);
                        $$ip{$aux2}=$aux;
                        $$so{$aux2}=$aux3;
                        $$ti{$aux2}=$aux4;
                }
        }
return($c);
} #Fin Cargar

# Devuelve el tipo de maquina, dado el nombre
#sub tipo($\%) {
#       my($nom,$tip)=@_;
#       return($$tip{$nom});
#}
# Devuelve el so de maquina, dado el nombre
#sub tipo($\%) {
#       my($nom,$so)=@_;
#       return($$so{$nom});
#}
# Devuelve el ip de maquina, dado el nombre
#sub tipo($\%) {
#       my($nom,$ip)=@_;
#       return($$ip{$nom});
#}

# Verificamos la lista de argumentos
if ($#ARGV <= 0) {
        # Verificamos los argumentos del programa
        if (($ARGV[0] ne "-i") && ($#ARGV == 0)) {
                die "Error: Argumento incorrecto <$ARGV[0]>\a\n";
        }
        # Para poder pasar cada uno de los arreglos asociativos
        # debemos hacerlo por referencia.
        $cont=&cargar_red(\%dirip,\%sistop,\%tipo);
        # Comenzamos a verificar las maquinas
        foreach $maquina (keys %dirip) {
                $res=`$pingdir/ping $band $dirip{$maquina}`;
                print("$maquina <$dirip{$maquina}");
                if ($#ARGV == 0) {
                        print(",$sistop{$maquina},$tipo{$maquina}>, ")
                } else {
                        print(">, ");
                }
                print("Estado...");
                if ($res =~ /0 packets/) {
                        print("X\n");
                } else {
                        print("V\n");
                }
        }
} else {
        die "Error: cantidad errónea de argumentos $! \n\a";
}
```

Seguridad en Perl

Por razones de espacio sólo se cubrirán algunos aspectos sobre Perl y seguridad. El primero de ellos es la versión de Perl; Las versiones anteriores a la 5.0.4 tienen un bug de seguridad (buffer overflow) el cual permite que un usuario cualquiera se convierta en root bajo algunas plataformas (Puede conseguir el script en www.rootshell.com).

Se mostrarán a continuación algunos errores comúnes y su solución:

- ¿Recuerda el programa que trabajaba con asideros? El no especificar la ruta absoluta cuando ejecuta un comando externo puede comprometer su seguridad. La línea que llama al programa externo dice:

  ```
  open(ARCHIVO,"cat /etc/jose|grep -v '#' |");
  ```

  El programador supone que invoca al programa /**bin**/**cat**, confiando en que este sólo abre el archivo /**etc**/**hosts**. Pero ¿qué ocurre si un atacante cambia la variable de entorno ***$PATH*** y la reemplaza para que busque primero un script llamado ***cat,*** como el que se muestra a continuación:

  ```
  #!/usr/bin/perl
  open(ARCHIVO,"/etc/passwd");
  while(<ARCHIVO>) {
          print("$_");
  }
  ```

  Se muestra a continuación la forma de explotar el error, escrito en Perl:

  ```
  1.  #!/usr/bin/perl
  2.  # Obtenemos el valor de la variable de ambiente PATH
  3.  $var=$ENV{PATH};
  4.  # Veamos lo que contiene la variable
  5.  print("Contenido de \$PATH: $var\n");
  6.  # Lo que hace el atacante es mover a "." de primero, para poder
  7.  # cambiar al legitimo cat por otro
  8.  $var="\.:$var";
  9.  $ENV{PATH}=$var;
  10. print("Contenido de \$PATH: $ENV{PATH}\n");
  11. #Ahora ejecutamos al script con problemas
  12. system("asidero.pl");
  ```

  La clave de este script está en la línea 8, en donde se cambia el orden de búsqueda.

  Este error se puede solucionar fácilmente de dos maneras:

  - Coloque la ruta completa del comando ping. De esta manera, cualquier cambio en la ruta de búsqueda quedará sin efecto.
  - Antes de correr el script, coloque valores conocidos a la variable de ambiente $PATH (Esto puede ser un poco tedioso a veces), justo lo necesario para correr el script.
  - El siguiente es el script "seguro":
  -

    ```
    #!/usr/bin/perl
    # Rutas "seguras" de mis programas
    $catdir="/bin";
    $grepdir="/usr/bin";
    # Ahora se define una ruta segura, la cual sera valida solo
    # mientras se ejecuta el programa
    $ENV{PATH}="/bin:/usr/bin";

    # Veamos el contenido del archivo /etc/hosts
    # para ello se utiliza cat y eliminamos todos los
    # comentarios
    open(ARCHIVO,"$catdir/cat /etc/hosts|$grepdir/grep -v '#' |");
    ```

```
print <ARCHIVO>,"\n";
# cerramos el archivo (No hace falta, perl lo hace solo)
close(ARCHIVO);
```

- Otra forma de violentar un programa es abusando de los parámetros que recibe:

```
#!/usr/bin/perl
system("/bin/ls $ARGV[0]");
```

El programa muestra el contenido del directorio actual; de hecho puede recibir parámetros de ls como *–l*:

```
leon[56] $host> seg2.pl –l
```

Pero podemos abusar de él:

```
leon[58] $host> seg2.pl –l;cat /etc/passwd
```

Para remediarlo, fije la cantidad de parámetros con los que puede correr el programa de la siguiente manera:

```
system('/bin/ls','-l');
```

Si quiere seguir utilizando línea de comandos, entonces debe los datos que puede recibir el script.

Depuración de programas en Perl

Perl en modo de advertencias (–w)

El primer paso es decirle a Perl que corra el programa con mensajes de advertencia:

```
#!/usr/bin/perl -w
```

Validemos el *programa verifica_red.pl* para ver como funciona está característica:

```
leon[53] $host> ./verifica_red.pl
Name "main::cont" used only once: possible typo at ./verifica_red.pl line 65.
Use of uninitialized value at ./verifica_red.pl line 60.
```

La advertencia de la línea 65 nos indica que no estamos haciendo nada con esta variable, sólo la llenamos con la cantidad de máquinas leídas pero no hacemos más nada. Si se agrega en la línea 66 estas instrucciones:

```
print("Leidas $cont maquinas\n");
```

Se elimina la advertencia.

Perl se queja en la línea 60 sobre utilizar un valor no inicializado. Esto ocurre sí *ARGV[0]* no existe (cuando se invoca al programa sin argumentos; pruebe invocándolo con la opción –i). Solucione este problema anidando las expresiones if en la línea 60:

```
# Verificamos los argumentos del programa
        if ($#ARGV == 0) {
                if ($ARGV[0] ne "-i") {
                        die "Error: Argumento incorrecto <$ARGV[0]>\a\n";
                }
        }
```

Perl en modo de depuración (–d)

Si cambia la bandera *–w* por la bandera *–d* arrancará Perl en modo de depuración. Si bien hay depuradores más amigables en C, C++ o Pascal, esta herramienta se volverá indispensable cuando haga programas más grandes. Una vez realizado el cambio, arranque el script para ver como funciona:[11]

```
$host> verifica_redd.pl

Loading DB routines from perl5db.pl version 1
Emacs support available.

Enter h or 'h h' for help.

main::(verifica_redd.pl:12):    $pingdir="/bin";        #Directorio donde va pin
g
  DB<1>
```

Para salir del depurador escriba "q".

Listando código fuente y el contenido de las variables

El comando **"l"** (list) sirve para mostrar 10 líneas de código a la vez:

```
leon[72] $host> verifica_redd.pl

Loading DB routines from perl5db.pl version 1
Emacs support available.

Enter h or 'h h' for help.

main::(verifica_redd.pl:12):    $pingdir="/bin";        #Directorio donde va pin
g
  DB<1> l
12==>   $pingdir="/bin";        #Directorio donde va ping
13:     $maqdir="./datos.txt";  #Ubicacion y nombre del archivo de datos
14:     $band="-c 1";
15
16      # *************************************************]
17      # Fin de la configuracion del script
18
19      # Rutina que carga el contenido del archivo a memoria
20      sub cargar_red(\%\%\%) {
21:             my($ip,$so,$ti)=@_;     #Las variables vienen en @_
```

Puede mirar el código de una línea en especial con sólo indicar su número:

```
DB<2> l 5
5       # usando la herramienta ping
```

Puede listar un rango de líneas:

```
DB<3> l 5-15
5       # usando la herramienta ping
6       #
7       # Hecho por Jose Vicente Nunez Zuleta (jose@ing.ula.ve)
8
9       # *************************************************]
10      # Configuracion del script
11      #
12==>   $pingdir="/bin";        #Directorio donde va ping
13:     $maqdir="./datos.txt";  #Ubicacion y nombre del archivo de datos
14:     $band="-c 1";
15
```

La línea que está en negritas muestra en donde está nuestro apuntador de instrucción.

El comando **p** (print) le permite saber el valor de una variable:

```
DB<11> p $maqdir
./datos.txt
  DB<12>
```

[11] Se recomienda que escriba "man perldebug" para aprender acerca del depurador de Perl.

Puede hacer esto mismo utilizando el comando **"X":**

```
DB<12> X maqdir
$maqdir = './datos.txt'
  DB<13>
```

Ejecutando el código fuente

Puede correr el programa paso a paso con el comando **"s"** (step):

```
DB<4> s
main::(verifica_redd.pl:13):    $maqdir="./datos.txt";  #Ubicación y nombre del
archivo de datos
DB<4>
```

El comando **"n"** se comporta de manera similar, excepto que no entra dentro del código de una rutina cuando esta es llamada.

Puntos de quiebre, ejecución continua del programa

Se define un punto de quiebre con el comando "b". la ejecución del programa se detendrá en el momento en que el apuntador de instrucción llegue a la línea o condición indicada por el punto de quiebre. Por ejemplo, coloquemos un punto de quiebre en la línea en la línea 58 y corramos el programa:

```
DB<26> b 58
DB<27> c
main::(verifica_redd.pl:58):    if ($#ARGV <= 0) {
  DB<27>
```

Puede decirle al depurador que se detenga en una función en particular. Por ejemplo, detenga la ejecución en la rutina cargar_red:

```
DB<27> b cargar_red
DB<28> c
main::cargar_red(verifica_redd.pl:21):
21:             my($ip,$so,$ti)=@_;     #Las variables vienen en @_
DB<28>
```

Para ver todos los puntos de quiebra definidos, escriba **"L".** Si lo que desea es eliminarlos todos, escriba **"D";** Si escribe **"d numero de línea"** elimina un punto de quiebra en particular:

```
DB<28> L
verifica_redd.pl:
 21:            my($ip,$so,$ti)=@_;     #Las variables vienen en @_
   break if (1)
 58:    if ($#ARGV <= 0) {
   break if (1)
DB<28> d 58
DB<29> L
verifica_redd.pl:
 21:            my($ip,$so,$ti)=@_;     #Las variables vienen en @_
   break if (1)
  DB<29>
```

Busqueda con patrones

El depurador le permite buscar por patrones (utilizando inclusive expresiones regulares) con ayuda del operador / (búsqueda hacia delante) y **?** (búsqueda hacia atrás). Buque, por ejemplo, donde encuentra la palabra "ip":

```
DB<29> /ip
33:                             $$ip{$aux2}=$aux;
DB<30>
```

Busque ahora una línea en donde haya un comentario, utilizando una expresión regular:

```
DB<30> /^\#/
41:     # Devuelve el tipo de maquina, dado el nombre
DB<31>
```

Corregir sólo la sintaxis con –c

Si ejecuta *perl –c programa*, perl le indicará si la sintaxis está bien o no:

```
leon[67] $host> perl -c hola.pl
hola.pl syntax OK
```

Ejercicio 3: Seguridad en Perl y depuración de programas

Investigue que hace, depure y corrija el siguiente programa:

```
#!/usr/bin/perl
# Este programa tiene varios errores
# dejados de manera intencional
$inutil=9993.345;
@palabras=(
'hola',
'casa',
'mundo',
'prueba',
);
sub mayu(@) {
my(@arr)=@_;
for ($i=0;$i <$#arr;$i++) {
$arr[$i] =~ tr/a-z/A-Z/;
}
}
print("@palabras\n");
&mayu(@palabras);
print("@palabras\n");
$fecha=`date`;
print("Terminado a $fecha");
```

Solución

```
#!/usr/bin/perl

# Este programa tiene varios errores
# dejados de manera intencional
@palabras=(
                'hola',
                'casa',
                'mundo',
                'prueba',
);
sub mayu(\@) {
        my($arr)=@_;
        $lim=scalar(@$arr);
        for ($i=0;$i < $lim;$i++) {
                $$arr[$i] =~ tr/a-z/A-Z/;
        }
}
print("@palabras\n");
&mayu(\@palabras);
print("@palabras\n");
$fecha=`/bin/date`;
print("Terminado a $fecha");
```

CGI [1][3] es el "pegamento" que permite realizar transacciones bancarias por Internet o hacer búsquedas en sitios como www.yahoo.com. *CGI* provee un canal consistente, independiente del lenguaje de programación, para el acceso remoto a bases de datos o aplicaciones.

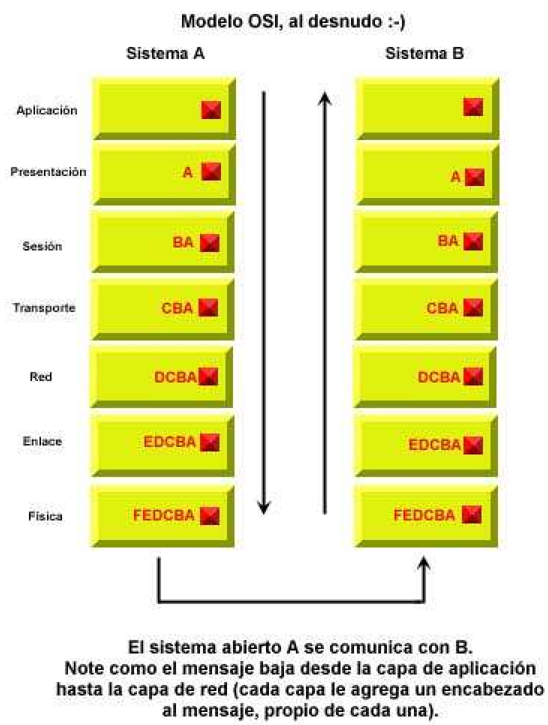
Antes de dar una definición formal sobre lo que es *CGI*, veamos algunos conceptos claves.

Protocolos de comunicación

Modelo OSI y TCP/IP

Para poder conectar dos aplicaciones que están separadas físicamente entre sí, se utiliza un lenguaje y reglas de comunicación común a ambas, llamado *protocolo de comunicación*. Existe una abstracción llamada el modelo ***OSI***, la cual permite a aplicaciones remotas comunicarse de manera confiable, a la vez de que su funcionamiento se mantiene independientemente del medio físico por el cual entran en contacto.

El modelo OSI[12] está compuesto de siete capas, cada una de las cuales es recorrida de arriba abajo, las cuales permiten la comunicación entre una aplicación y otra:



• Ilustración 1: Modelo OSI

Esto es lo que hace cada capa:

1. Capa de aplicación. Esta capa trata con los detalles específicos de la aplicación. El protocolo HTTP (El cual es el protocolo sobre el cual opera el WWW) está contenido

[12] OSI es un modelo de referencia, no es una aplicación en particular. Por ejemplo, TCP/IP se basa en este modelo de referencia.

aquí. Está capa es la responsable de darle al usuario las herramientas y medios para usar el programa.
2. Capa de presentación. La capa de presentación trata con la representación de los datos que los componentes de la capa de aplicación usan o refieren en sus comunicaciones. Por ejemplo, es en esta capa que un browser decide como presentar una imagen.
3. Capa de sesión. La capa de sesión provee mecanismos para organizar y sincronizar intercambios de datos con las capas superiores. Esto se debe a las diferencias en como una arquitectura almacena la información respecto a la otra.
4. Capa de transporte. Le asegura a las capas superiores que los datos serán transmitidos de manera confiable y con el menor costo posible. TCP (Uno de los protocolos sobre los cuales trabaja Internet) se encuentra aquí.
5. Capa de red. Se encarga de considerar el enrutamiento de la información. IP (Uno de los protocolos sobre los cuales trabaja Internet) se encuentra aquí.
6. Capa de enlace. Provee una transferencia punto a punto. También detecta errores provenientes de la capa física.
7. Capa física. Se encarga de los medios electrónicos y mecánicos que comienzan, mantienen y detienen las conexiones físicas entre dos entidades de enlace.

Como se dijo con anterioridad, el protocolo HTTP es el corazón del World Wide Web. HTTP surgió debido a las deficiencias de otros protocolos como FTP en el manejo de información de hipertexto. Al igual que FTP, HTTP corre en las capas superiores del modelo OSI.

HTTP

A continuación se explican brevemente algunas de las características del protocolo HTTP:

- Utiliza un esquema de direccionamiento completo: Cuando se ensambla un enlace en HTML, estos tienen la forma general de http://nombre–de–la–maquina:numero de puerto/ruta/archivo.html. Note que el nombre de la máquina conforma con las reglas de codificación de IP (aaa.bbb.ccc.ddd) o el esquema utilizado por DNS.[13]
- Es extensible y está abierto a la representación de tipos de datos: Cuando un cliente establece una conexión, se envía una cabecera que conforma con las especificaciones de correo electrónico estándares para Internet (RFC822)[14]. Normalmente, desde el punto de vista de la programación de "puentes", muchos clientes esperan más que un simple archivo de texto o HTML. Cuando el servidor transmite información de vuelta al cliente, incluye una cabecera con extensiones ***MIME*** *(Multipurpouse Internet Mail Extensión, Extensión Internet de correo electrónico de multipropósito)****,*** para decirle al cliente que tipo de datos sigue a la cabecera. El servidor no necesita tener la capacidad de interpretar el tipo de dato pedido, eso es la responsabilidad del cliente.
- No tiene estado: Sin estado significa que una vez que el cliente y el servidor han terminado sus diálogos, la conexión es terminada. Esto tiene consecuencias importantes y muestra una diferencia con respecto a otros protocolos como *FTP*. Sin estado también significa que no hay "memoria" entre las conexiones entre clientes. Esto significa que el servidor ve cada conexión de un cliente como una conexión nueva, aún si este vuelve a pedir el mismo documento.
- Es rápido.
- Existen implementaciones portables, y es un estándar abierto: No pertenece a un solo fabricante, corre en muchas plataformas y permite nuevas adiciones como protocolos de cifrado de datos como *SHTTP* o *Secure Socket Layers*.

Modelo cliente – servidor

Como muchas otras aplicaciones de red, el *World Wide Web* se ajusta al modelo cliente – servidor. Él termino cliente / servidor ha sido utilizado ampliamente durante muchos años para describir una gran variedad de sistemas.

[13] La forma más general es servicio://maquina/archivo.extensión
[14] RFC viene de Request For Comments (Petición de comentarios).

La comunicación entre el cliente y el servidor ocurre sobre la red, la cual en el caso del WWW es Internet.[15] El cliente y el servidor pueden estar corriendo en diferentes sistemas operativos, sobre diferentes arquitecturas, haciendo nuestro modelo independiente de la plataforma.

El cliente

En nuestro caso, el cliente es un browser como Netscape Communicator o Lynx. El browser lo que hace es pedirle un documento a un servidor (utilizando un localizador uniforme de recursos, URL o Uniform Resource Locators), para luego procesarlo (transformarlo para ser presentado por pantalla).

El servidor

El servidor se encarga de manejar archivos, responde a las peticiones del cliente y envía las respuestas a las preguntas realizadas. El servidor también provee un enlace a aplicaciones que no son clientes http, por medio de CGI.[16]

La siguiente ilustración muestra como es la interacción entre un par de clientes y un servidor:

• Ilustración 2: Comunicación entre un cliente y un servidor

Si usted ha hecho una página html y la ha puesto en un servidor, ha estado utilizando el primer tipo de interacción cliente – servidor. Ahora fíjese en el segundo tipo, el servidor no necesita saber como interpretar los datos que devuelve programa, el sólo los pasa al cliente.

CGI

CGI (Common Gateway Interface) **[10]** es el medio que tiene un servidor que habla HTTP para comunicarse con un programa. La idea es que cada cliente y programa servidor (independientemente del sistema operativo) se adhieran a los mismos mecanismos para el flujo de datos entre el cliente, el servidor y el programa que hace el puente.

[15] Aunque puede ser una Intranet sin ningún problema.
[16] Una aplicación CGI puede ser una base de datos, un programa de diagnóstico, un simulador, etc.

Cuando se habla de un cliente de un puente, se dice que puede ser un programa que actúa como intermediario entre el servidor HTTP y otro programa que puede ser ejecutado por medio de línea de comandos (Por ejemplo una base de datos relacional).

Cuandos e habla de una interfaz, se dice que es un mecanismo estándar que le permite a los programadores trabajar dentro de un entorno productivo sin tener que reinventar la rueda.

Funcionamiento de CGI

Una transacción con CGI efecuta los siguientes pasos:

1. El cliente envía una petición conformando con el formato estándar de un URL a el servidor (se incluye el tipo de servicio, ubicación del servicio). Además se envía un encabezado con datos provistos por el cliente
2. El servidor procesa la petición de llegada y decide que hacer luego, dependiendo del tipo de servicio solicitado. Si es un archivo html lo devuelve pero si es un programa CGI entonces:
   - El encabezado con datos es recibido por el cliente, si existe. Estos datos son pasados al programa como variables de entorno *($ENV{}* en Perl).
   - Los parámetros de ejecución del programa, si existen, son tomados por el programa. Estos datos pueden ser pasados por medio de variables de entorno *($ENV{QUERY_STRING})* o por Entrada estándar *(<STDIN>).* La forma en como el cliente envía los datos la decide el programador al hacer la interfaz.

1. El programa CGI devuelve una respuesta, como un documento HTML, al servidor. El programa CGI siempre debe devolver alguna respuesta. Los datos deben estar precedidos por un encabezado que conforme con las convenciones MIME, el tipo de dato devuelto es indicado en la cebecera.
2. La salida es devuelta al cliente por el servidor y se corta la comunicación.

Generación dinámica de HTML

Se supone que para esta parte que el administrador del sistema le ha dado permiso para ejecutar programas basados en CGI en el servidor. Para la parte práctica, se guardarán los programas en el directorio /**cgi–bin** del servidor.

Asegúrese que el directorio *public_html* y sus subdirectorios tienen permisología de lectura y ejecución para todo el mundo. Todos los script que haga deberán tener permiso de ejecución para todo el mundo:

```
$host> chmod a+xr cgi-bin/
```

A continuación se muestra el código básico para generar un documento dinámico con CGI, utilizando Perl **[2]**:

```
#!/usr/bin/perl
print "Content-type: text/html \n\n";
print("<!-- Esta pagina fue generada de manera dinamica -->\n");
print("<html>\n");
print("<head>\n");
print("<title>\"Hola mundo\"</title>\n");
print("</head>\n");
print("<body>\n");
print "<h1>Hola mundo. Este es mi primer CGI en Perl </h1> \n";
print("</body>\n");
print("</html>\n");
```

La línea 2 es muy importante, ya que allí se define el tipo de datos que va a recibir el browser (*text/html*) con el siguiente formato:

```
Content-type tipo/subtipo <retorno de carro> retorno de carro>
```

Los dos retornos de carro son muy importantes, sin ellos el programa no trabajará.
La siguiente tabla muestra algunas de los tipos de cabecera más usadas:

Tipo[17]	**Significado**
text/html	Documento en html
text/plain	Documento de texto
audio/wav	Audio en formato wav
video/mpeg	Películas en formato mpeg
image/jpg	Imágenes en formato JPG

• Tabla 6: Tipos MIME comunes

Se mostrará a continuación la salida esperada del programa:

• Ilustración 3: Salida de ejemplo de hola.cgi

Ejercicio 4: Generación dinámica de HTML

Convierta el programa "*verifica_red.pl*" a CGI para que este muestre su salida por pantalla por medio de una tabla. Para simplificar las cosas elimine la parte que procesa la línea de comandos.

Utilice colores para resaltar los resultados. El programa debe mostrar todos los datos que están en nuestra base de datos a medida que verifique cada máquina.

Solución:

```
#!/usr/bin/perl
# Este script verifica el funcionamiento de
```

[17] Visite http://www.cis.ohio–state.edu/htbin/rfc/rfc1521.html, para obtener una descripción completa de los tipos MIME.

```
# un grupo de maquinas funcionando en una red
# usando la herramienta ping. Esta version utiliza
# CGI
#
# Hecho por Jose Vicente Nunez Zuleta (jose@ing.ula.ve)
```

```
# ****************************************************]
# Configuracion del script
#
$pingdir="/bin";    #Directorio donde va ping
$maqdir="./datos.txt";      #Ubicacion y nombre del archivo de datos
$band="-c 1";
```

```
# ****************************************************]
# Fin de la configuracion del script
```

```
# Rutina que carga el contenido del archivo a memoria
sub cargar_red(\%\%\%) {
my($ip,$so,$ti)=@_; #Las variables vienen en @_
# Leemos los datos y los guardamos en un hash
open(DATOS,$maqdir) || die "<P>No puedo cargar los datos de la red $!</body></html>\n";
my($aux,$aux2,$aux3,$aux4); #Variable auxiliar
#Comenzamos la lectura de los datos
my $c=0;
while(<DATOS>) {
#Validamos que no sea un comentario o linea vacia
chop($_);
if (($_ !~ /^\#/) && ($_ ne "")) {
$c++; #Otra maquina leida
($aux,$aux2,$aux3,$aux4)=split('\s+',$_);
$$ip{$aux2}=$aux;
$$so{$aux2}=$aux3;
$$ti{$aux2}=$aux4;
}
}
return($c);
} #Fin Cargar
```

```
print("Content-type: text/html \n\n"); #Encabezado del mensaje
print("<!-- Documento generado de manera dinamica -->\n");
print("<html><head><title>Verifica Red: 1.0</title></head>\n");
print("<body bgcolor=#000000 text=yellow>\n");
$cont=&cargar_red(\%dirip,\%sistop,\%tipo);
print("<font size=5 face=\"Arial,Helvetica,Verdana\">Leidas $cont maquinas</font>\n");
# Comenzamos a verificar las maquinas
print("<table border\=0>\n");
print("<tr bgcolor=blue>\n");
print("<th>Nombre<th>IP<th>Sistema operativo<th>Tipo<th>Estado\n");
foreach $maquina (keys %dirip) {
$res=`$pingdir/ping $band $dirip{$maquina}`;
print("<tr>\n");
print("<td>$maquina\n");
print("<td>$dirip{$maquina}\n");
print("<td>$sistop{$maquina}\n");
print("<td>$tipo{$maquina}\n");
if ($res =~ /0 packets/) {
print("<td bgcolor=red>\ \n");
} else {
print("<td bgcolor=green>\ \n");
}
}
print("</table>");
print("</body></html>");
```

Para probar el script, cambie su permisología con *chmod a+x verifica_red.pl* y coloque una instrucción similar a esta en su browser:

http://www.servidor.dominio/cgi-bin/verifica_red.cgi

Métodos de comunicación

Como se mencionó con anterioridad, un cliente puede pasar información al servidor utilizando dos métodos:

- Variables de entorno
- Entrada estándar

A continuación se explicará brevemente cada uno de ellos:

Método GET

El método *GET* trabaja con variables de entorno, colocando los datos en dos variables llamadas "*QUERY_STRING*" y "*PATH_INFO*". El contenido de estas variables es codificado por el servidor y es responsabilidad del programador decodificarlas de nuevo.

Sin embargo, esas no son las únicas variables que pasa el cliente al servidor. Las restantes variables envían información que puede ser utilizada de manera útil, como la validación del tipo de browser que utiliza un usuario. Ellas no dependen de la aplicación particular que maneja el cliente.

El siguiente código muestra el contenido de la cadena *QUERY_STRING* cuando es enviada por el usuario:

```
#!/usr/bin/perl
print ("Content type: text/plain\n\n");
print ("El contenido de la variable QUERY_STRING es: $ENV{QUERY_STRING}\n");
```

Corriendo el script como *http://www.servidor.dominio /cgi–bin/get.cgi?Esta es una prueba* , produce la siguiente salida:

```
El contenido de la variable QUERY_STRING es: Esta
```

Note como se pierde el resto de la cadena. Esto significa que debemos *codificar* nuestro mensaje para que pueda llegar completo, algo de lo que se encargan las *formas en html* de las cuales hablaremos más adelante. El contenido de la cadena *QUERY_STRING* viene justo después del signo de interrogación.

La gente de *NCSA*, los autores del browser Mosaic, no recomiendan el uso de GET ya que si el cliente envía una cadena muy larga, esta puede derribar al Shell que recibe la variable de ambiente (esto se conoce como "quedarse sin espacio de entorno").

Para evitar estas limitaciones, se usa entonces el método POST.

Método POST

El método POST no tiene las limitaciones del método GET ya que utiliza a STDIN para obtener los datos. Los datos también son codificados allí, de manera que es responsabilidad del programador decodificar los datos.

El siguiente código muestra como trabaja el método POST en Perl:

```
#!/usr/bin/perl
# read trata de leer CONTEN_LENGTH bytes de datos en $_ desde STDIN
$entrada=read(STDIN,$_,$ENV{CONTENT_LENGTH});
print ("Content type: text/plain\n\n");
print ("El usuarios escribio lo siguiente: $entrada\n");
```

La captura de datos se hace desde STDIN, con ayuda de la función read(). Para utilizar este método, debemos utilizar una forma de html.

Variables de entorno adicionales

El siguiente programa muestra el contenido de las variables de presentes al momento de ejecutar nuestra aplicación:

```
#!/usr/bin/perl
print("Content-type: text/html\n\n");
print("<html><head><title>Variables de entorno</title></head>\n");
print("<body>\n");
print("<ul>\n");
foreach $variable (keys %ENV) {
        print("<li>$variable = $ENV{$variable}\n");
}
print("</ul></body></html>\n");
```

La siguiente es una salida de ejemplo al correr este programa *con lynx servidor.dominio/cgi–bin/variables.cgi:*

Variables de entorno (p1 of 2)

```
* SERVER_SOFTWARE = Apache/1.3.0 (Unix)
* GATEWAY_INTERFACE = CGI/1.1
* DOCUMENT_ROOT = /usr/local/etc/httpd/htdocs
* REMOTE_ADDR = 150.185.146.42
* SERVER_PROTOCOL = HTTP/1.0
* REQUEST_METHOD = GET
* REMOTE_HOST = berlioz.ing.ula.ve
* QUERY_STRING =
* HTTP_USER_AGENT = Lynx 2.5 libwww-FM/2.14
* PATH =
  /usr/local/sbin:/usr/local/bin:/sbin:/usr/sbin:/bin:/usr/bin:/usr/
  X11/bin:/usr/andrew/bin:/usr/openwin/bin:/usr/games:.:/usr/lib/teT
  eX/bin
* HTTP_ACCEPT = audio/x-pn-realaudio, application/postscript,
  application/x-csh, application/x-perl, x-conference/x-cooltalk,
  audio/x-pn-realaudio, */*, application/x-wais-source,
  application/html, text/plain, text/html, www/mime
* REMOTE_PORT = 2918
* HTTP_ACCEPT_LANGUAGE = en
* SCRIPT_NAME = /cgi-bin/variables.cgi
* SERVER_NAME = www.felinos.ve
* REQUEST_URI = /cgi-bin/variables.cgi
* SERVER_PORT = 80
* HTTP_HOST = leon.felinos.ve
* SERVER_ADMIN = jose@ing.ula.ve
```

Codificación de los datos

El servidor codifica los datos de la siguiente manera:

- Los espacios son cambiados al signo +
- Ciertos caracteres de teclados son convertidos a su equivalente hexadecimal.
- Los campos entre las formas son concatenados a &

Ejercicio 5: Variables de entorno

Haga un pequeño programa en Perl que le diga al usuario remoto que browser está utilizando. Utilice para ello el contenido de la variable *HTTP_USER_AGENT*.

Solución

```
#!/usr/bin/perl
print("Content-type: text/html\n\n");
print("<HTML><HEAD><TITLE>Tu browser es</TITLE></HEAD><BODY>\n");
if ($ENV{HTTP_USER_AGENT}  =~ /Mozilla/) {
        print("<P>Estas utilizando Nestcape\n");
} elsif ($ENV{HTTP_USER_AGENT} =~ /Lynx/) {
        print("<P>Estas utilizando Lynx");
} else {
        print("<P>No puedo determinar tu browser\n");
}
print("</BODY></HTML>");
```

Formas en HTML

Para esta parte suponemos que el lector está familiarizado con el lenguaje básico de HTML. Si el lector está interesado puede consultar **[1],** en la sección de bibliografía al final de este trabajo.

FORM es usado en un documento HTML como cualquier etiqueta. Su estructura básica es:

```
<FORM Action = "URL" [Enctype] Method = "POST|GET"> Contenidos de la forma </FORM>
```

Aquí el elemento *Action = URL* corresponde al script o programa CGI que procesará la información enviada por el cliente. *Method* identifica la forma en que los contenidos de la forma son enviados por el programa que pregunta. El elemento *Enctype* identifica el método utilizado para codificar los pares de valor / datos (sino se especifica, el browser usa *application/x–www–form–urlencoded*[18]).

Una etiqueta FORM no puede ser anidada, pero puede haber tantas como sea en un documento.

Elementos que contiene la etiqueta FORM

Los siguientes son los elementos que integran la etiqueta FORM **[10]**:

Input: Recolecta información acerca del usuario

Atributos de Input

- *Align*: se usa sólo con un tipo de imagen. Los posibles valores son top, middle y bottom.
- *Checked*: Provoca que el estado inicial de un botón o caja de selección sea marcado.
- *Maxlength*: Coloca el tamaño máximo de caracteres que un usuario puede entrar en un campo de texto. El valor por omisión es ilimitado.
- *Name*: Especifica el nombre simbólico usado en la transferencia de salida de la forma.
- *Size*: Específica el ancho el campo mostrado al usuario.
- *Src*: Define el archivo fuente para la imagen cuando el atributo *Type* es colocado en *image.*
- *Text:* Es un área de entrada de datos de una sola línea. Si el usuario presiona entrar entonces los datos son enviados al script correspondiente.
- *Type:* Define que tipo de entrada es posible. Sus valores son:
  - *Checkbox:* Usado para recolectar datos que puedan tener múltiples valores a la vez.
  - *Hidden:* Especifica valores configurados por la forma, sin ayuda del usuario.
  - *Image:* Envía la forma una vez que el usuario presiona la imagen, con las respectivas coordenadas x/y más el nombre del objeto.
  - *Password:* El usuario entra texto aquí, pero este no se ve por pantalla.
  - *Radio:* Recolecta información donde uno y solo un valor puede ser seleccionado.
  - *Reset*: Reinicia la forma a sus valores por omisión. El atributo value muestra la cadena de caracteres que el usuario verá por pantalla.
  - *Submit:* Envía los valores de la forma. El atributo value muestra la cadena de caracteres que el usuario verá por pantalla.
  - *Text:* Es usada para una sola línea de texto. Utiliza los atributos *value* y *Maxlength*.

Value define el valor mostrado por el campo o el valor del campo cuando es seleccionado.

Option: Ocurre dentro del elemento *select*, el cual le permite al usuario escoger entre varias alternativas, y es usado para representar cada opción de *select.*

Atributos de option:

- Selected: indica sí la opción esta seleccionada
- Value: Si esta presente, este es el valor retornado por select, si no, el valor retornado es especificado por option.

[18] Consulte ftp://ftp.merit.edu/documents/rfc/rfc1590.txt para ver los tipos de codificación).

Select: Le permite al usuario escoger entre múltiples alternativas

Atributos de select:

- Multiple: Por omisión, el usuario solo puede seleccionar un ítem a la vez. El uso de este atributo permite seleccionar más de un atributo.
- Name: El nombre lógico que será enviado y está asociado al escoger select.
- Size: Especifica la cantidad de ítems visibles. Si no se especifica, la selección se comportará como un menú "pull–down".

Textarea: Recolecta múltiples líneas de texto por el usuario. Al usuario se le presenta un panel cuadrado en el cual puede escribir. Se usa en pares.

Atributos de textarea:

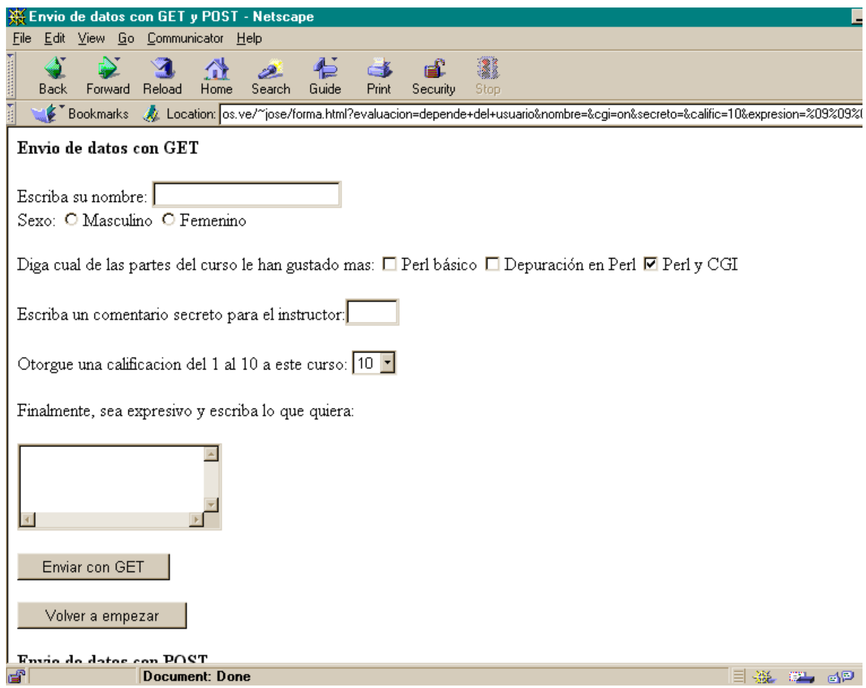
- Cols: El número de columnas que serán mostradas (el usuario puede usar la barra de desplazamiento para moverse a lo largo de más columnas si es necesario).
- Rows: Número de filas que serán mostradas.
- Name: Nombre lógico asociado al texto retornado.

El siguiente ejemplo muestra una par de formas que recogen datos. Estos son procesados por un script en CGI el cual sabe interpretar los datos provenientes del método POST y el método GET:

```
<!-- Ejemplo de una forma que trabaja con los dos metodos -->
<!-- Hecho por José Vicente Nuñez Zuleta (jose@ing.ula.ve) -->
<html>
<head>
<title>Envio de datos con GET y POST</title>
</head>
<body bgcolor=#ffffff>
<P><strong>Envio de datos con GET</strong>
<form action="http://www.felinos.ve/cgi-bin/res.cgi method=get">
<input type=hidden name="evaluacion" value="depende del usuario">
Escriba su nombre: <input type=text name="nombre" size=20><br>
Sexo:
<input type=radio name=sexo value=M>Masculino
<input type=radio name=sexo value=F>Femenino
<P>Diga cual de las partes del curso le han gustado mas:
<input type=checkbox name=perl-basico>Perl b&aacute;sico
<input type=checkbox name=perl-dep>Depuraci&oacute;n en Perl
<input type=checkbox name=cgi checked>Perl y CGI
<P>Escriba un comentario secreto para el instructor:<input type=password name=secreto size=5 maxlength=5><br>
<P>Otorgue una calificacion del 1 al 10 a este curso:
<SELECT name=calific>
<OPTION>9
<OPTION>8
<OPTION>7
<OPTION>6
<OPTION>5
<OPTION>4
<OPTION>3
<OPTION>2
<OPTION value="ups">1
<OPTION selected>10
</SELECT>
<P>Finalmente, sea expresivo y escriba lo que quiera:
<P><textarea cols=20 name=expresion rows=3></textarea>
<P><input type=submit value="Enviar con GET">
<P><input type=reset value="Volver a empezar">
</form>
<P><strong>Envio de datos con POST</strong>
<form action="http://www.servidor.dominio/cgi-bin/res.cgi method=post">
<input type=hidden name="evaluacion" value="depende del usuario">
Escriba su nombre: <input type=text name="nombre" size=20><br>
Sexo:
<input type=radio name=sexo value=M>Masculino
<input type=radio name=sexo value=F>Femenino
<P>Diga cual de las partes del curso le han gustado mas:
<input type=checkbox name=perl-basico>Perl b&aacute;sico
<input type=checkbox name=perl-dep>Depuraci&oacute;n en Perl
<input type=checkbox name=cgi checked>Perl y CGI
```

```
<P>Escriba un comentario secreto para el instructor:<input type=password
name=secreto size=5 maxlength=5><br>
<P>Otorgue una calificacion del 1 al 10 a este curso:
<SELECT name=calific>
<OPTION>9
<OPTION>8
<OPTION>7
<OPTION>6
<OPTION>5
<OPTION>4
<OPTION>3
<OPTION>2
<OPTION value="ups">1
<OPTION selected>10
</SELECT>
<P>Finalmente, sea expresivo y escriba lo que quiera:
<P><textarea cols=20 name=expresion rows=3></textarea>
<P><input type=submit value="Enviar con POST">
<P><input type=reset value="Volver a empezar">
</form>
</body>
</html>
<!-- Fin del documento -->
```

La apariencia del documento sería similar a la siguiente:



• Ilustración 4: Salida de ejemplo para un par de formas

Uniendo todo: Manejo de los datos de una forma, seguridad

Cuando haga programas en CGI prepárese para el peor escenario posible al recibir los datos de una forma. No sólo el usuario puede equivocarse escribiendo los datos, sino que un atacante malicioso puede tratar de violentar su sistema.

En la parte de seguridad en Perl, se vió como agregando más información a la línea de comandos de un programa podía violar la seguridad del sistema. Esto puede evitarse convirtiendo todos los caracteres que no pertenezcan a un grupo conocido en el carácter "_":[19] A continuación se muestra el código de un programa potencialmente inseguro:

```
#!/usr/bin/perl
```

[19] Se recomienda que consulte http://geek–girl.com/bugtraq/1997_4/0232.html.

```
1.  $result=`/bin/ping $ARGV[0] -c 1`;
2.  print("La maquina esta $result");
```

La línea 2 es potencialmente insegura ya que en vez de suministrar solamente una dirección o nombre de máquina, un atacante puede hacer algo como esto:

```
leon[91] $host> ping_inseguro.pl localhost;/bin/cat /etc/passwd
```

El siguiente ejemplo explota esa vulnerabilidad en el programa *test–env*, que viene con los servidores *NCSA* y otros, al utilizar los llamados *"metacaracteres"*, los cuales son caracteres especiales interpretados por el Shell, para obtener información no autorizada de una máquina (En este caso el contenido del directorio /cgi–bin):

```
berlioz[56] $host> lynx http://www.victima.com/cgi-bin/test-cgi?/usr/local/etc/httpd/cgi-bin/*
```

Con el siguiente resultado:

```
CGI/1.0 test script report:

argc is 1. argv is /usr/local/etc/httpd/cgi-bin/\*.

PATH = /usr/local/bin:/usr/bin:/bin:.
SERVER_SOFTWARE = Apache/1.2b10 PHP/FI-2.0b11
SERVER_NAME = www.victima.com
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 80
REQUEST_METHOD = GET
HTTP_ACCEPT = audio/x-pn-realaudio, application/postscript, application/x-csh,
application/x-perl, x-conference/x-cooltalk, audio/x-pn-realaudio, */*, applica
tion/x-wais-source, application/html, text/plain, text/html, www/mime
PATH_INFO =
PATH_TRANSLATED =
SCRIPT_NAME = /cgi-bin/test-cgi
QUERY_STRING = /usr/local/etc/httpd/cgi-bin/analog /usr/local/etc/httpd/cgi-bin
/archie /usr/local/etc/httpd/cgi-bin/calendar /usr/local/etc/httpd/cgi-bin/came
```

A continuación se muestra el código que permite recolectar los datos de la forma que hicimos anteriormente, con las consideraciones de seguridad aplicadas:

```
#!/usr/bin/perl
#
# Hecho por Jose Vicente Nunez Zuleta (jose@ing.ula.ve)
#
# La siguiente rutina determina el tipo de metodo empleado
# en el envio de los datos
sub metodo_forma {
my($metodo)=$ENV{'REQUEST_METHOD'};
my($entrada);
#Validamos primero el tipo de codificacion
if (($ENV{'CONTENT_TYPE'} !~ /^application\/x-www-form-urlencoded/i) &&
($ENV{'CONTENT_TYPE'} ne '')) {
print("Content-type: text/html\r\n\r\n");
print("<html><head><title>Error en CGI</title></head>\n");
print("<body bgcolor=#FFFFFF>\n");
print("<H1>Error: Codificacion desconocida\n</H1>");
print("<H2>Soporta: application\/x-www-form-urlencoded\n</H2>");
print("</body></html>\n");
exit;
}
$metodo =~ tr/A-Z/a-z/; #Convierta la respuesta a minusculas
#El metodo es get y QUERY_STRING no esta vacia
if (($metodo eq "get") && ($ENV{QUERY_STRING} ne '')) {
return($ENV{QUERY_STRING});
#El metodo es POST y tiene longitud > 0
} elsif (($metodo eq "post") && ($ENV{'CONTENT_LENGTH'} > 0)) {
read(STDIN,$entrada,$ENV{'CONTENT_LENGTH'});
return($entrada);
} else { #Metodo desconocido o valor nulo
print("Content-type: text/html\r\n\r\n");
print("<html><head><title>Error en CGI</title></head>\n");
print("<body bgcolor=#FFFFFF>\n");
print("<H1>Error: Metodo desconocido o valor nulo\n</H1>");
print("<H2>Soporta: Method=GET\|POST\n</H2>");
print("</body></html>\n");
exit;
}
} #Fin metodo_forma()
```

```
# La siguiente rutina decodifica la entrada y la guarda en
# un arreglo asociativo
sub decodificar_forma($) {
my($BUENOS_CARACTERES)='-a-zA-Z0-9_.@ '; #Caracteres validos en los datos
```

```
5.  my($entrada)=@_;
6.  my(%pares,$clave,$valor);
7.  $entrada   =~   s/%([a-fA-F0-9][a-fA-F0-9])/pack("c",hex($1))/ge;   #Convierto   a
    alfanumerico los caracteres en hexadecimal
8.  @pares=split('&',$entrada); #Separo cada par y lo guardo en un arreglo
9.  foreach $par (@pares) { # proceso cada par en pares
10. ($clave,$valor)=split(/=/,$par); #Guarde clave,valor usando la busqueda anterior
11. $valor =~ s/\+/ /g; #Cambie + por espacio en toda la cadena
12. # Seguridad  en caracteres,segun aviso del Cern en
13. # http://geek-girl.com/bugtraq/1997_4/0232.html
14. $clave =~ s/[^$BUENOS_CARACTERES]/_/go;
15. $valor =~ s/[^$BUENOS_CARACTERES]/_/go;
16. #Finalmente construya el arreglo asociativo
17. $pares{$clave}=$valor;
18. }
19. return(%pares);
20. } # Fin Decodificar_forma()
```

```
1.  # Comienzo del programa principal
2.  $entrada=&metodo_forma();
3.  %datos=&decodificar_forma($entrada);
4.  $| = 1; #Vaciamos el buffer de escritura lo mas rapido posible
5.  print("Content-type: text/html\r\n\r\n");
6.  print("<html>\n");
7.  print("<head><title>Datos enviados por el usuario</title></head>\n");
8.  print("<body bgcolor=\#ffffff>\n");
9.  print("<pre>\n");
10. print("El contenido de la forma es:\n");
11. foreach $clave (keys(%datos)) {
12. print("$clave = $datos{$clave}\n");
13. }
14. print("Variables de entorno\n");
15. foreach $clave (keys(%ENV)) {
16. print("$clave = $ENV{$clave}\n");
17. }
18. print("</pre>\n");
19. print("</body>\n");
20. print("</html>");
```

En la línea 7 encontramos a la rutina *metodo_forma*, la cual se encarga básicamente de:

- Verificar que los datos estén codificados con *application\/x-www-form-urlencoded*. Este valor viene vacio cuando se usa el método GET.
- Averiguar si los datos fueron enviados con GET o POST y que tengan una longitud mayor que cero.

Una vez conocida la forma en como fueron enviados los datos, estos son guardados en una variable, listos para ser decodificados. De esto se encarga la rutina *decodificar_forma*,a cual hace lo siguiente:

- En la línea 44 se convierten los elementos que están en hexadecimal en la variable capturada con metodo_forma a su equivalente alfanumérico. Para ello se utiliza una expresión regular con reemplazo[20] y a la función unpack().[21]
- En la línea 45 se separa a los pares clave / valor y los guarda en un arreglo. Luego se procesa el arreglo, línea 46, y por cada par:
  - Se separan en clave y valor (línea 47);
  - A cada clave y valor les son sustituidos los signos + por espacios en blanco (línea 48).
  - Se eliminan los caracteres de cada clave y valor si no están en una lista de caracteres conocidos (líneas 51 y 52).
  - Se crea un arreglo asociativo con cada par y valor (línea 54).
- Al finalizar de procesar los pares se devuelve el arreglo asociativo (línea 56).

El código anterior produce una salida similar a esta:

```
El contenido de la forma es:
perl-basico = on
nombre = sfsdfsfsf
```

[20] En algunas de las expresiones regulares s/viejo/nuevo/ se utilizaron los modificadores g, e, i y o. El modificador hace una busqueda sin importar si es mayusculas o minusculas, g hace una busqueda en toda la variable, e evalua el resultado de la búsqueda y devuelve un valor, o se utiliza por eficiencia para aprovechar el hecho de que una expresión regular no cambia en una búsqueda. Escriba man perlfaq6 para mayor información.

[21] unpack(TEMPLATE, EXPR) toma una cadena, EXPR, que representa una estructura y la expande en un valor de arreglo. En este ejemplo TEMPLATE es igual a *c*, que significa *un valor de carácter con signo* y *$1* es *el primer valor del último carácter encontrado.*

```
cgi = on
calific = 2
secreto = sdfsf
expresion = ddsdfsfsdf
evaluacion = depende del usuario
sexo = F
Variables de entorno
SERVER_SOFTWARE = Apache/1.3.0 (Unix)
GATEWAY_INTERFACE = CGI/1.1
DOCUMENT_ROOT = /usr/local/etc/httpd/htdocs
REMOTE_ADDR = 192.168.1.3
SERVER_PROTOCOL = HTTP/1.0
REQUEST_METHOD = POST
REMOTE_HOST = pantera.felinos.ve
HTTP_REFERER                                                                                    =
http://www.servidor.dominio/~jose/forma.html?evaluacion=depende+del+usuario&nombre=&c
gi=on&secreto=&calific=10&expresion=%09%09%09
QUERY_STRING =
HTTP_USER_AGENT = Mozilla/4.01 [en] (Win95; I)
```

Note que sólo una de las formas puede hacer el envío, ya sea con GET o POST.

Ejercicio 6: Uso de formas y CGI

Modifique a *verifica_red.cgi* para que permita hacer un *ping* a un grupo de máquinas de manera dinámica,es decir, basandose en la base de datos que se utilizó en el ejemplo. El programa debe *mostrar todas las máquinas*, pero debe permitir hacer un ping a algunas o todas las máquinas (Utilice cajas de selección para ese fin). El programa también debe permitir hacer ping a una dirección arbitraria, usando una caja de texto. Si lo desea, valide la sintaxis de la dirección IP suministrada al programa *ping*. Si le parece muy complicado, entonces haga una forma con un número fijo de máquinas y envíe los datos al script que hace la verificación.

Sugerencia para el enfoque dinámico: Utilice dos programas, uno que lea la base datos y permita hacer la selección por pantalla con *HTML dinámico* y otro que haga el *ping* y devuelva los resultados al usuario.

Solución

Se muestra primero el código que lee la base de datos y muestra la forma por pantalla:

```
#!/usr/bin/perl

# Este Script carga la base de datos y la muestra por pantalla
# permitiendo luego llamar a verifica_red.cgi
#
# Hecho por Jose Vicente Nunez Zuleta (jose@ing.ula.ve)

# **************************************************]
# Configuracion del script
#
$maqdir="./datos.txt";      #Ubicacion y nombre del archivo de datos
$password="jose98"; #Password no encriptado, viaja por la red como texto
# **************************************************]
# Fin de la configuracion del script

# Rutina que carga el contenido del archivo a memoria
sub cargar_red(\%\%\%) {
my($ip,$so,$ti)=@_; #Las variables vienen en @_
# Leemos los datos y los guardamos en un hash
open(DATOS,$maqdir) || die "<html><head><title>Error en
CGI</title></head><body><P>Error: No puedo cargar los datos de la red
$!</body></html>\n";
my($aux,$aux2,$aux3,$aux4); #Variable auxiliar
#Comenzamos la lectura de los datos
my $c=0;
while(<DATOS>) {
#Validamos que no sea un comentario o linea vacia
chop($_);
if (($_ !~ /^\#/) && ($_ ne "")) {
$c++; #Otra maquina leida
($aux,$aux2,$aux3,$aux4)=split('\s+',$_);
$$ip{$aux2}=$aux;
$$so{$aux2}=$aux3;
$$ti{$aux2}=$aux4;
}
}
return($c);
} #Fin Cargar
```

```
1.  $| = 1; #Vaciamos el buffer de escritura lo mas rapido posible
2.  print("Content-type: text/html \n\n"); #Encabezado del mensaje
3.  print("<!-- Documento generado de manera dinamica -->\n");
4.  print("<html><head><title>Verifica Red: 1.0</title></head>\n");
5.  print("<body bgcolor=#000000 text=yellow vlink=red alink=yellow link=yellow>\n");
6.  $cont=&cargar_red(\%dirip,\%sistop,\%tipo);
7.  print("<font size=5 face=\"Arial,Helvetica,Verdana\">Leidas $cont
    maquinas</font>\n");
8.  # Presentamos el menu para probar las maquinas
9.  print("<H1>Verificar <em>una m&aacute\;quina</em> arbitraria</H1>\n");
10. print("<form action=\"http://www.felinos.ve/cgi-bin/verifica_red.cgi\"
    method=post>\n");
11. print("<P>Introduzca la direcci&oacute\;n IP o nombre que quiere verificar:
    <input type=text name=individual size=20 maxlength=20>\n");
12. print("<H1>Verificar <em>grupo</em> de m&aacute\;quinas </H1>\n");
13. print("<table border\=0>\n");
14. print("<tr bgcolor=blue>\n");
15. print("<th>Nombre<th>IP<th>Sistema operativo<th>Tipo<th>&#191Verificar?\n");
16. foreach $maquina (sort keys %dirip) {
17. print("<tr>\n");
18. print("<td>$maquina\n");
19. print("<td>$dirip{$maquina}\n");
20. print("<td>$sistop{$maquina}\n");
21. print("<td>$tipo{$maquina}\n");
22. print("<td align=center><input type=checkbox checked
    name=$maquina-$dirip{$maquina}>\n");
23. }
24. print("</table>");
25. print("<P>\n");
26. print("<BR><P><input type=submit value=Verificar> <input type=reset
    value=\"Volver a empezar\">\n");
27. print("</form>\n");
28. print("Sugerencias:<a
    href=\"mailto:jose\@ing.ula.ve\">jose\@ing.ula.ve\"</a>\n");
29. print("</body></html>");
```

El código de la línea 56 es importante, ya que allí se almacena el nombre de la máquina y su dirección ip como un solo valor para enviarlo al programa que hace la verificación de la red. Dejamos que *verifica_red.cgi* programa se encargue de revisar la sintaxis de las direcciones ip ya que eso lo hace más resistente a ataques.

Finalmente se muestra el código del programa que verifica la red:

```
1.  #!/usr/bin/perl
2.  # este programa recibe los datos enviados por carga_red.cgi
3.  # y verifica un grupo de maquinas
4.  # Hecho por Jose Vicente Nunez Zuleta (jose@ing.ula.ve)
5.  #
6.  # VARIABLES
7.  $pingdir="/bin";
8.  $banderas="-c 1";
9.  #
10. #
11. # La siguiente rutina determina el tipo de metodo empleado
12. # en el envio de los datos
13. sub metodo_forma {
14. my($metodo)=$ENV{'REQUEST_METHOD'};
15. my($entrada);
16. #Validamos primero el tipo de codificacion
17. if    (($ENV{'CONTENT_TYPE'}    !~    /^application\/x-www-form-urlencoded/i)    &&
    ($ENV{'CONTENT_TYPE'} ne '')) {
18. print("Content-type: text/html\r\n\r\n");
19. print("<html><head><title>Error en CGI</title></head>\n");
20. print("<body bgcolor=#FFFFFF>\n");
21. print("<H1>Error: Codificacion desconocida\n</H1>");
22. print("<H2>Soporta: application\/x-www-form-urlencoded\n</H2>");
23. print("</body></html>\n");
24. exit;
25. }
26. $metodo =~ tr/A-Z/a-z/; #Convierta la respuesta a minusculas
27. #El metodo es get y QUERY_STRING no esta vacia
28. if (($metodo eq "get") && ($ENV{QUERY_STRING} ne '')) {
29. return($ENV{QUERY_STRING});
30. #El metodo es POST y tiene longitud > 0
31. } elsif (($metodo eq "post") && ($ENV{'CONTENT_LENGTH'} > 0)) {
32. read(STDIN,$entrada,$ENV{'CONTENT_LENGTH'});
33. return($entrada);
34. } else { #Metodo desconocido o valor nulo
35. print("Content-type: text/html\r\n\r\n");
36. print("<html><head><title>Error en CGI</title></head>\n");
37. print("<body bgcolor=#FFFFFF>\n");
38. print("<H1>Error: Metodo desconocido o valor nulo\n</H1>");
39. print("<H2>Soporta: Method=GET\|POST\n</H2>");
40. print("</body></html>\n");
41. exit;
42. }
43. } #Fin metodo_forma()
```

```
1.  # La siguiente rutina decodifica la entrada y la guarda en
2.  # un arreglo asociativo
3.  sub decodificar_forma($) {
4.  my($BUENOS_CARACTERES)='-a-zA-Z0-9_.@ '; #Caracteres validos en los datos
5.  my($entrada)=@_;
6.  my(%pares,$clave,$valor);
7.  $entrada =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("c",hex($1))/ge; #Convierto a
    alfanumerico los caracteres en hexadecimal
8.  @pares=split('&',$entrada); #Separo cada par y lo guardo en un arreglo
9.  foreach $par (@pares) { # proceso cada par en pares
10. ($clave,$valor)=split(/=/,$par); #Guarde clave,valor usando la busqueda anterior
11. $valor =~ s/\+/ /g; #Cambie + por espacio en toda la cadena
12. # Seguridad  en caracteres,segun aviso del Cern en
13. # http://geek-girl.com/bugtraq/1997_4/0232.html
14. $clave =~ s/[^$BUENOS_CARACTERES]/_/go;
15. $valor =~ s/[^$BUENOS_CARACTERES]/_/go;
16. #Finalmente construya el arreglo asociativo
17. $pares{$clave}=$valor;
18. }
19. return(%pares);
20. } # Fin Decodificar_forma()
21. # La siguiente rutina valida que el formato de la direccion IP sea
22. # www.xxx.yyy.zzz. Retorna una cadena vacia si la direccion
23. # contiene errores. Esta rutina no permite el uso de 127.0.0.1
24. sub valida_ip($) {
25. my($aux)=$_[0];
26. my($result)=$_[0];
27. #Validamos que solo hayan 4 octetos
28. $aux =~ s/[^\.]//g; #Dejo solo el caracter '.'
29. if (length($aux) != 3) {
30. return('');
31. }
32. my(@octetos);
33. ($octetos[0],$octetos[1],$octetos[2],$octetos[3])=split('\.',$result);
34. my($octeto);
35. #Validamos la sintaxis correcta de cada octeto
36. foreach $octeto (@octetos) {
37. if ( ($octeto =~ /\D|\W+/) || ($octeto <= 0) || ($octeto >= 255) ) {
38. return('');
39. }
40. }
41. return($aux);
42. } #Fin valida_ip()
43. # Comienzo del programa principal
44. $entrada=&metodo_forma();
45. %datos=&decodificar_forma($entrada);
46. $| = 1; #Vaciamos el buffer de escritura lo mas rapido posible
47. print("Content-type: text/html\r\n\r\n");
48. print("<html>\n");
49. print("<head><title>Resultados de Verifica Red 1.0</title></head>\n");
50. print("<body bgcolor=#000000 text=yellow vlink=red alink=yellow link=yellow>\n");
51. print("<font face=\"arial,helvetica,verdana\" size=5>Resultados de la
    inspecci&oacute;n:</font>\n");
52. print("<table border=0>\n");
53. print("<tr bgcolor=blue>\n");
54. print("<th>Nombre<th>IP<th>Estado\n");
55. #Validamos la direccion IP de la caja de texto
56. if (&valida_ip($datos{'individual'}) ne '') {
57. $maquinas{'No tiene nombre'}=$datos{'individual'};
58. }
59. #Validamos ahora la direccion de las demas maquinas
60. foreach $clave (keys %datos) {
61. if ($datos{$clave} =~ /on/) {
62. $datos{$clave} =~ s/on//ig; #Elimino a "= on"
63. ($maquina,$ip)=split('-',$clave); #Separo el par maquina-ip
64. if (&valida_ip($ip) ne '') {
65. $maquinas{$maquina}=$ip;
66. }
67. }
68. }
69. #Imprimimos los resultados
70. foreach $clave (keys %maquinas) {
71. print("<tr><td>$clave<td>$maquinas{$clave}");
72. $res=`$pingdir/ping $banderas $maquinas{$clave}`;
73. if ($res =~ /\b0% packet/) {
74. print("<td bgcolor=green>\ \n");
75. } else {
76. print("<td bgcolor=red>\ \n");
77. }
78. }
79. print("</table>\n");
80. print("<P><a href=\"http://www.servidor.dominio/cgi-bin/cargar_red.cgi\">Volver
    atras</a>\n");
81. print("</body>\n");
82. print("</html>");
```

Se utilizan la misma rutina para validar el formato de los datos que vienen por la red más la rutina *valida_ip* en la línea 67, la cual verifica solamente el formato de la dirección ip. Note que el programa ignora la prueba de la máquina si alguna de las conversiones falla.

En el siguiente punto se mostrará como se puede hacer parte de este procesamiento de datos en el cliente, utilizando el lenguaje Javascript.

Hasta este punto se ha mostrado solamente aplicaciones que corren en el servidor, utilizando CGI. Posiblemente se haya dado cuenta que este enfoque tiene varias deficiencias:

- El cliente queda inactivo una fracción de tiempo, esperando respuesta del servidor. Si el trabajo que hace el programa toma mucho tiempo en ejecutarse, se pierde el interés en utilizar al Web como herramienta.
- No se aprovecha el poder de procesamiento del cliente, todo el trabajo lo hace el servidor lo cual se traduce en una distribución desigual de trabajo.
- Hay que esperar a que los datos lleguen al servidor para que sean validados. Se considera un desperdicio de ancho de banda el enviar datos erróneos por la red al servidor.

La idea entonces es darle algo de poder de procesamiento al cliente, de manera que este realice algo de ese procesamiento, aligerando la carga del servidor y mejorando los tiempos de respuesta. Eso significa que el cliente pueda examinar los datos y le notifique al usuario acerca de cualquier error obvio.

Esto acerca aún más a los clientes HTML a la filosofía de cliente / servidor, a la vez que hace más interactivo el contenido del Web.

La primera idea de darle más poder a los clientes se vio plasmada con la versión 2 de *Netscape Navigator*, al utilizar un lenguaje llamado *Javascript*. Para ese entonces, el lanzamiento de Javascript coincidió con el del lenguaje *Java* por parte de Sun Microsystems.[22]

Aunque Java y Javascript tienen algunas similitudes sus propósitos principales los hacen muy distintos. La siguiente tabla muestra algunas de esas diferencias:

Java	Javascript
Compilado	Interpretado
Orientado a objetos. Soporta herencia.	Basado en objetos. No soporta herencia.
Applets como entidad separada entidad de HTML	Código insertado dentro de HTML
Tipos de datos definidos como entero, real	Tipos de datos no declarativos, como en Perl

• Tabla 7: Diferencias básicas entre Java y Javascript

Javascript no es el único lenguaje que trabaja de esta manera. *Microsoft* diseño un lenguaje basado en Visual Basic, llamado *VBScript*. Sin embargo, una de sus principales desventajas con respecto a Javascript es que está atado a correr dentro de Ambientes Windows mientras que Javascript es multi – plataforma. Otra ventaja de Javascript sobre VBScript es que Javascript está a punto de convertirse en un estándar internacional.

[22] Javascript soporta muchos de los constructores de Java.

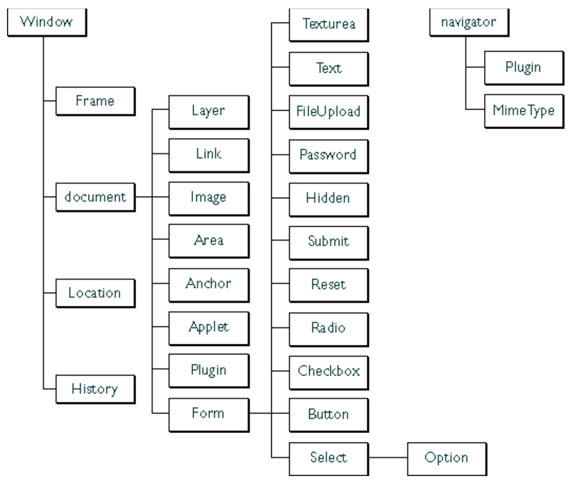
En los siguientes puntos se explicarán algunas características de Javascript con algunos ejemplos, sin embargo se recomienda al lector interesado que consulte la bibliografía recomendada **[4][6]** para profundizar sus conocimientos.

Objetos en Javascript

Javascript está basado en objetos, pero no está orientado a objetos. Se pueden tratar con los objetos que ya están creados o se pueden crear nuevos objetos, pero no se pueden crear nuevas clases ni hacer herencia.

Algunos de esos objetos están relacionados con el browser mientras que otros están relacionados con objetos genéricos, como una cadena.

Cada uno de los objetos que conforman el lenguaje Javascript forma una jerarquía similar a la siguiente:[23]



• Ilustración 5: Jerarquía de objetos en Javascript

La siguiente tabla muestra algunos objetos en Javascript:

Objeto	**Significado / Etiqueta en HTML**
Anchor	<a>
Button	<input type="button">
Checkbox	<input type="checkbox">
Date	Objeto genérico para el tratamiento de fechas
Document	La página que muestra el browser
Form	<form></form>
Frame	<frame></frame>
Hidden	<input type="hidden">
History	Contiene información acerca del URL visitado por el usuario.
Link	Es similar al objeto anchor (Ancla)
Location	Información acerca del URL actual
Math	Un objeto genérico que trata con funciones matemáticas
Navigator	Información acerca del navegador usado
Password	<input type ="password">
Radio	<input type="radio">

[23] Esta imagen fue tomada de http://developer.netscape.com/docs/manuals/communicator/jsguide4/ (Guía de desarrollo de Javascript de Netscape Communications Corp). El crédito le corresponde a sus respectivos autores

reset	<input type="reset">
Select	<select></select>
String	Objeto genérico para el manejo de cadenas
Submit	<input type=submit>
Text	<input type="text">
Textarea	<textarea></textarea>
Window	Información acerca de la ventana actual

• Tabla 8: Listado de algunos objetos en Javascript

A continuación se mostrará como puede trabajar con esos objetos, utilizando sus métodos, y como puede incorporar Javascript en sus documentos HTML.

Métodos de los objetos en Javascript. Programación orientada a eventos.

La programación orientada a eventos es distinta a la programación tradicional, en el sentido de que un objeto dentro del browser, como un botón en una forma o la misma ventana del browser, tiene asociado ciertas propiedades (valores asociados con los objetos) y un método (acciones asociadas al objeto) las cuales pueden cambiar por la interacción que hay entre el usuario y la página. Al producirse un cambio en el estado de un método, por ejemplo al presionar un botón en una forma, entonces se ejecuta el código asociado a ese método.

En un programa tradicional, este tiene un cuerpo o rutina principal y una serie de funciones que son llamadas de acuerdo a la ejecución del programa. En la programación orientada a eventos sólo se llama la rutina asociada a un evento, no existe algo así como una rutina central.

Por ejemplo, el objeto *<input type=submit value=Enviar>,* tiene como propiedades a *value,* el cual es el nombre que se mostrará en pantalla.

Para cada objeto están definidos un número fijo de métodos. Esos eventos son:

Evento	Significado
Blur	Un objeto select, text, o textarea pierde el foco en una forma
Change	El valor de un campo en select, text o textarea cambia
Click	Se hace clic en un objeto en una forma
Focus	Un campo recibe el foco de atención, ya sea utilizando el ratón o tabulación
Load	El browser terminó de cargar un documento HTML. Si el documento contiene Frames entonces la carga del documento termina cuando todos los Frames están cargados.
MouseOver	Cuando el puntero del ratón pasa sobre un objeto
Select	Cuando el usuario selecciona parte o todo el texto de un campo de tipo text o textarea
Submit	Cuando el usuario envía una forma
Unload	El usuario deja el documento

• Tabla 9: Eventos posibles para los objetos en Javascript

Más adelante se mostrará el uso de objetos según sus métodos.

Hola mundo en Javascript

A continuación se muestra el programa hola mundo en Javascript:

```
1.  <html>
2.  <head>
3.  <title>Hola mundo en Javascript :-)</title>
4.  </head>
5.  <body>
6.  <P>Esta es una demostraci&oacute;n de Javascript
7.  <script language="JavaScript">
8.  <!--
9.  document.write("<P>&#161 Hola mundo cruel!")
10. //-->
11. </script>
12. </body>
13. </html>
```

En la línea 7 se declara que el código que viene a continuación está hecho en Javascript. Para evitar problemas con browsers que no soportan a Javascript, ocultamos el código de Javascript con *el código de comentarios en HTML* (líneas 8 y 10. En la línea 9 se utiliza el método *write* del objeto *document* para mostrar un mensaje por pantalla. Note que este mensaje no aparecerá en un browser que no tenga soporte para Javascript. En la línea 11 cerramos la declaración de código de Javascript.

En Javascript se pueden utilizar la misma sintaxis para los comentarios que en C++, esto es:

- // Para comentarios cortos
- /* Comentarios que abarcan más de una línea */

Recuerde que Javascript trabaja con los métodos de los objetos. El siguiente código le indica al usuario la última vez que la página fue modificada:

```
1.  <html>
2.  <head>
3.  <title>Hola mundo en Javascript :-)</title>
4.  </head>
5.  <body>
6.  <P>Esta es una demostraci&oacute;n de Javascript
7.  <script language="JavaScript">
8.  <!--
9.  document.write("<P>&#161 Hola mundo cruel!<BR>")

1.  document.write("Ultima modificaci&oacute;n: "+document.lastModified)
2.  //-->
3.  </script>
4.  </body>
5.  </html>
```

En la línea 10 se utiliza otro truco: Se concatena la salida de un objeto con una cadena de caracteres (En Perl se hace con el operador punto, *'.'*).

La salida por pantalla es la siguiente:

```
Esta es una demostración de Javascript

¡ Hola mundo cruel!
Ultima modificación: 06/07/98 01:47:47
```

Variables en Javascript

En Javascript no se necesita especificar el tipo de una variable cuando se declara. Todas las variables se declaran utilizando la palabra reservada *var*:

```
var suma = 0
var v, w = "hola"
```

Javascript soporta los siguientes tipos de datos implícitos:[24]

- Números, como –32 o 555.432
- Una cadena de caracteres, como "Me gusta navegar en bote"
- Un valor lógico como *true* o *false*
- El valor especial *null*.

Funciones en Javascript

La sintaxis para declarar una función en Javascript es la siguiente:

```
function Nombre_funcion(argumento1,argumento2,...) {
        Código de la función
        return(algo)
}
```

Los argumentos son opcionales, así como la función puede no retornar nada.

La siguiente función calcula la multiplicación de dos números:

```
<html>
<head>
<title>Uso de funciones</title>
</head>
<body>
<script language="JavaScript">
<!--
function multiplic(num1,num2) {
return(num1*num2);
}
var x=10
var y=10
var res=multiplic(x,y)
document.write("Multiplicamos "+x+"*"+y+"="+res)
//-->
</script>
</body>
</html>
```

Estructuras de repetición

Las estructuras de repetición que utiliza Javascript son muy similares a las de otros lenguajes como Perl o C. A continuación se mostrará su sintaxis y un pequeño ejemplo de uso:

Sintaxis y ejemplo de Do – while

```
do
   sentencias
while (condición);
```

Por ejemplo:

```
var cont=0;
do
        Cont++;
while (cont <maximo);
```

Sintaxis y ejemplo de For

```
for ([expresión inicial;] [condición;] [expresión de incremento]) {
   sentencias
}
```

Por ejemplo:

```
for (var i = 0; i < 9; i++) {
   n += i
```

[24] Desde la versión 3.0 de Navigator, se puede utilizar el comando typeof para averiguar el tipo de una variable.

```
   función(n)
```

Sintaxis y ejemplo de For in

For in se utiliza para mostrar las propiedades de un objeto. Su sintaxis es:

```
for (variable in objeto) {
   sentencias
}
```

La siguiente rutina muestra las propiedades de un objeto:

```
function propiedades(objeto, nombre) {
   var resulta = ""
   for (var i in objeto) {
      resulta += nombre + "." + i + " = " + objeto[i] + "<BR>"
   }
   resulta += "<HR>"
   return resulta
}
```

Sintaxis y ejemplo de while:

```
while (condición) {
   sentencias
}
```

Por ejemplo:

```
While (i<=j) {
        document.write("Valor: "+i)
}
```

Todas estas estructuras de repetición pueden interrumpir su ejecución por medio de la sentencia break:

```
for (var i=0; i<j; i++) {
        if (i == valor) {
                break;
        }
}
```

Estructuras de decisión

La sintaxis de las sentencias *if – else* es la siguiente:

```
if (condición) {
   código}
[else {
   código}]
```

Las sentencia *if – else* trabaja con los siguientes operadores lógicos[25]:

<table>
<tr><th>Operador lógico</th><th>Significado</th><th>Resultado</th></tr>
<tr><td>a && b</td><td>And (Y)</td><td>Verdadero si a y b son verdaderos</td></tr>
<tr><td>a || b</td><td>Or (O)</td><td>$a sí $a es verdadero, de lo contrario $b</td></tr>
<tr><td>! a</td><td>Not (Negación)</td><td>Verdadero sí $a es falso</td></tr>
</table>

• Tabla 10: Operadores lógicos en Javascript

Como puede ver, su sintaxis es similar a la de Perl.

Los operadores de comparación son más sencillos que los de Perl, son iguales tanto para cadenas como para números:

Prueba	Significado

[25] No se mencionaron en este trabajo los operadores de bits ni los operadores de asignación para Perl o Javascript. Se espera que el lector consulte las referencias bibliográficas respectivas.

==	Es igual a
!=	diferente
>	Más grande que
>=	Mayor o igual
<	Menor que
<=	Menor o igual

• Tabla 11: Operadores de comparación en Javascript

Los siguientes operadores están relacionados con los objetos en Javascript. No necesitan estar asociados directamente a una estructura de decisión para poder ser utiilizados:

Operador	Significado
new	Permite crear una instancia de un objeto predefinido o elaborado por el usuario.
delete	Elimina la propiedad de un objeto o un elemento en un arreglo.
this	Palabra especial utilizada para referirse al objeto actual.

• Tabla 12: Operadores especiales en Javascript

Cuando la cantidad de opciones a evaluar es grande, se puede utilizar la sentencia switch:

```
switch (expresión){
   case etiqueta :
      sentencia;
      break;
   case etiqueta2 :
      sentencia;
      break;
   ...
   default : sentencia;
```

Cajas de alerta, confirmación y preguntas

Javascript tiene algunas herramientas que pueden facilitar la entrada de datos, o la presentación de información. Ellas son:

- Mensajes de alerta (*window.alert()*)
- Mensaje de confirmación (*window.confirm()*)
- Mensaje de pregunta (*window.prompt()*)

A continuación se presenta un ejemplo que utiliza estos tres métodos del objeto ventana, más el uso del objeto "*button*". La respuesta del usuario se obtiene utilizando el método onclick en cada uno de los botones (ese método se encarga de invocar a cada una de nuestras cajas de dialogo):

```
<html>
<head>
<title>Uso de funciones</title>
</head>
<body>
<script languaje=javascript>
<!--
var respuesta
//-->
</script>
<FORM method="POST">
<P><INPUT type="button" name="alerta" value="Boton de alerta" onclick="alert('Alerta Roja')"></P>
<P><INPUT type="button" name="pregunta" value="Boton de pregunta" onclick="respuesta=prompt('Alerta Roja'); alert('Usted escribio : '+respuesta) "></P>
<P><INPUT type="button" name="confirmacion" value="Confirmaci&oacute;n" onclick="respuesta=confirm('&#191 Desea continuar?'); alert('Usted decidio : '+respuesta) "></P>

</FORM>
</body>
</html>
```

Declaramos el inicio y final de una forma en las líneas 11 y 15. En la línea 12 tenemos una entrada de tipo botón, a la cual cuando se le hace clic invoca una caja de dialogo con un mensaje. En la línea 13 se utiliza otro botón e invocamos ahora una caja de confirmación la cual recoge nuestra respuesta, la cual es mostrada de inmediato por una caja de alerta.

En la línea 14 se utiliza una caja de decisión, capturamos el resultado de la escogencia del usuario y lo mostramos por pantalla con ayuda de una caja de alerta.

Expresiones regulares y arreglos

Al igual que Perl, Javascript dispone de expresiones regulares para encontrar y sustituir caracteres en una variable. También dispone del objeto arreglo, el cual se comporta de manera similar a los arreglos asociativos en Perl, ya que su índice puede ser numérico o no.

Arreglos

La forma de declarar un arreglo en Javascript es la siguiente:

```
var Arreglo = new Array()
```

El siguiente ejemplo muestra como imprimir el contenido de un par de arreglos:

```
<html>
<head>
<title>Arreglos en Javascript</title>
</head>
<body bgcolor=#ffffff>
<script>
<!--
var arreglo = new Array(5)
arreglo[0]='Hola'
arreglo[1]='esta es'
arreglo[2]='una'
arreglo[3]='prueba'
arreglo[4]='simple'
var sexo = new Array(3)
sexo['jose']='m'
sexo['sylvia']='f'
sexo['raul']='m'

for(var i=0;i<5;i++) {
document.write(arreglo[i]+" ")
}
document.writeln("<P>")
document.writeln("Sexo de las siguientes personas:<br>")
document.writeln("Jose "+sexo['jose']+"<br>")
document.writeln("Sylvia "+sexo['sylvia']+"<br>")
document.writeln("Raul "+sexo['raul']+"<br>")
//-->
</script>
<h1>uso de arreglos</h1>
<form name="datos">
</form>
</body>
</html>
```

Expresiones regulares

Javascript también cuenta con expresiones regulares[26], muy útiles para el tratamiento de caracteres en variables.

Javascript utiliza los mismos caracteres especiales que Perl para construir una expresión regular:

Expresión regular	Significado
.	Encuentra cualquier carácter, excepto una nueva línea

[26] Esta característica esta presente en Javascript 1.2, no en las versiones anteriores

<table>
<tr><td>[a–z0–9]</td><td>Encuentra cualquier carácter del conjunto</td></tr>
<tr><td>[^a–z–0–9]</td><td>Encuentra cualquier carácter que no este en el conjunto</td></tr>
<tr><td>\d</td><td>Encuentra cualquier dígito</td></tr>
<tr><td>\D</td><td>Encuentra cualquier cosa que no sea un dígito</td></tr>
<tr><td>\w</td><td>Encuentra cualquier carácter alfanumérico</td></tr>
<tr><td>\W</td><td>Encuentra cualquier carácter no alfanumérico</td></tr>
<tr><td>\s</td><td>Encuentra un carácter de espacio (espacio, tabulación, nueva línea)</td></tr>
<tr><td>\S</td><td>Encuentra un carácter que no sea de espacio (espacio, tabulación, nueva línea)</td></tr>
<tr><td>\n</td><td>Encuentra una nueva línea</td></tr>
<tr><td>\r</td><td>Encuentra un retorno</td></tr>
<tr><td>fi|fa|fo</td><td>Encuentra a fi o a fo o a fu en una cadena de caracteres</td></tr>
<tr><td>x?</td><td>Encuentra uno o cero x</td></tr>
<tr><td>x*</td><td>Encuentra cero o más x</td></tr>
<tr><td>x+</td><td>Encuentra una o más x</td></tr>
<tr><td>\b</td><td>Encuentra dentro de un bloque de palabra</td></tr>
<tr><td>\B</td><td>Encuentra fuera de un bloque de palabra</td></tr>
<tr><td>^</td><td>Encuentra al principio de la línea</td></tr>
<tr><td>\v</td><td>Encuentra una tabulación vertical</td></tr>
<tr><td>$</td><td>Encuentra al final de la línea</td></tr>
</table>

- Tabla 13: caracteres especiales para expresiones regulares en Javascript

También soporta el uso de los operadores *g*, búsqueda global, e *i*, sin importar mayusculas o minusculas.

La sintaxis es la siguiente:

```
result = /patrón/[g|i|gi]
```

El siguiente fragmento de código verifica si el nombre de una persona no tiene números o caracteres extraños al ser introducido por teclado. Sólo se explicará el método test, si el lector interesado desea saber que otras posibilidades existen en el uso de expresiones regulares, se recomienda que consulte **[4]:**

```
<html>
<head>
<title>Expresiones regulares en Javascript</title>
</head>
<body bgcolor=#ffffff>
<script>
<!--
function vnombre() {
/*  Busque uno o mas caracteres distintos a letras en
minusculas o mayusculas en toda la cadena */
validacion =/[0-9\W]/g
if (validacion.test(document.datos.nombre.value)) {
alert('El nombre contiene caracteres raros')
} else {
alert('El nombre esta bien')
}
}
//-->
</script>
<h1>Escriba el nombre del usuario</h1>
<form name="datos">
<input type="text" size=15 maxlength=15 name=nombre>
<input  type="button"  name="Validacion"  onclick="vnombre()"  value="Validar  el
nombre del usuario">
<input   type="button"   name="limpiar"   onclick="document.datos.nombre.value=''"
value="Volver a empezar">
</form>
</body>
</html>
```

Formas y validación

Este es uno de los usos más comunes para Javascript. La validación del contenido de una forma se basa casi siempre en los siguientes eventos:

- onBlur
- onSubmit

Se busca entonces que los datos sean coherentes y estén completos antes de enviarlos por la red al servidor.

Para que pueda entender como trabaja la validación examine el siguiente ejemplo. Suponga que le piden que almacene los siguientes datos sobre una persona:

- Nombre
- Edad
- Estado Civil
- Sexo
- Sueldo

Estos datos tienen las siguientes restricciones:

- El nombre no puede ser un número
- La edad debe ser mayor que 15 años y menor o igual que 95
- La persona debe tener un único estado civil, así como un único sexo
- El sueldo puede ser un número real, mayor que cero y menor que 5000$

La siguiente es una implementación de la solución utilizando la etiqueta FORM y Javascript:

```
<html>
<head>
<title>Datos personales</title>
<script>
<!--
//Validacion del nombre
function vnombre() {
var cadena=document.datos.nombre.value
if ((! isNaN(cadena)) && (cadena != ’’)) { //Validamos que el nombre no sea un numero y que tenga informacion
alert(’El nombre no puede ser un numero’)
document.datos.nombre.value=’’ //Borramos el error
document.datos.nombre.focus() //Colocamos el cursor en ese campo
}
}
// Validacion de la edad
function vedad() {
var cadena=document.datos.edad.value
var error=false
if (cadena != ’’) {
if (isNaN(cadena)) {
alert(’La edad debe se un numero’);
error=true
} else if ((cadena <= 15) || (cadena > 99)) {
alert(’La edad debe ir entre 16 y 99 anos’);
error=true
}
if (error == true) {
document.datos.edad.value=’’
document.datos.edad.focus()
}
}
}
//Validacion del sueldo
function vsueldo() {
var cadena=document.datos.sueldo.value
var error=false
if (cadena != ’’) {
if (isNaN(cadena)) {
alert(’Sueldo debe ser un numero’);
error=true
} else if ((cadena <= 0.0) || (cadena > 5000.0)) {
alert(’Sueldo debe ser mayor que 0.0 y menor o igual que 5000 ’);
error=true
}
if (error == true) {
document.datos.sueldo.value=’’
document.datos.sueldo.focus()
}
}
}
```

```
// Validacion antes del envio con submit
function vtodo() {
var error=false
if ((document.datos.sexo.value == ’’) || (document.datos.estado.value == ’’) || (document.datos.edad.value == ’’) || (document.datos.sueldo.value == ’’) ) {
error = true
}
if (error) {
alert(’Datos incompletos en la forma, todos los campos son requeridos’)
return false
}
}
//-->
</script>
</head>
<body bgcolor=#ffffff>
<h1>Recolecci&oacute;n de datos personales</h1>
<form name="datos" onSubmit="return vtodo()" action="http://www.servidor.dominio/cgi-bin/res.cgi" method="post">
<p>Escriba su nombre completo: <input type=text name=nombre size=20 maxlength=20 onBlur="vnombre()">
<p>Escriba su edad: <input name=edad type=text size=2 maxlength=2 onBlur="vedad()">
<P>Sexo:<input type=radio name=sexo value=m>M<input type=radio name=sexo value=f>F
<p>Indique su estado civil:
<p><input type=radio name=estado value=casado>Casado
<p><input type=radio name=estado  value=soltero checked>Soltero
<p><input type=radio name=estado value=divorciado>Divorciado
<p><input type=radio name=estado value=viudo>Viudo
<p>Escriba su sueldo: <input name=sueldo type=text size=6 maxlength=6 onBlur="vsueldo()">
<p>
<input type=submit value="Enviar datos" ><input type=reset value="Volver a empezar">
</form>
</body>
</html>
```

Note como el código en Perl que muestra los resultados por pantalla no realiza ningún tipo de validación:

```
#!/usr/bin/perl
# este programa recibe los datos enviados por carga red
# y verifica un grupo de maquinas
# Hecho por Jose Vicente Nunez Zuleta (jose@ing.ula.ve)
#
# La siguiente rutina determina el tipo de metodo empleado
# en el envio de los datos
sub metodo_forma {
my($metodo)=$ENV{’REQUEST_METHOD’};
my($entrada);
#Validamos primero el tipo de codificacion
if (($ENV{’CONTENT_TYPE’} !~ /^application\/x-www-form-urlencoded/i) && ($ENV{’CONTENT_TYPE’} ne ’’)) {
print("Content-type: text/html\r\n\r\n");
print("<html><head><title>Error en CGI</title></head>\n");
print("<body bgcolor=#FFFFFF>\n");
print("<H1>Error: Codificacion desconocida\n</H1>");
print("<H2>Soporta: application\/x-www-form-urlencoded\n</H2>");
print("</body></html>\n");
exit;
}
$metodo =~ tr/A-Z/a-z/; #Convierta la respuesta a minusculas
#El metodo es get y QUERY_STRING no esta vacia
if (($metodo eq "get") && ($ENV{QUERY_STRING} ne ’’)) {
return($ENV{QUERY_STRING});
#El metodo es POST y tiene longitud > 0
} elsif (($metodo eq "post") && ($ENV{’CONTENT_LENGTH’} > 0)) {
read(STDIN,$entrada,$ENV{’CONTENT_LENGTH’});
return($entrada);
} else { #Metodo desconocido o valor nulo
print("Content-type: text/html\r\n\r\n");
print("<html><head><title>Error en CGI</title></head>\n");
print("<body bgcolor=#FFFFFF>\n");
print("<H1>Error: Metodo desconocido o valor nulo\n</H1>");
print("<H2>Soporta: Method=GET\|POST\n</H2>");
print("</body></html>\n");
exit;
}
} #Fin metodo_forma()
# La siguiente rutina decodifica la entrada y la guarda en
# un arreglo asociativo
sub decodificar_forma($) {
my($BUENOS_CARACTERES)=’-a-zA-Z0-9_.@ ’; #Caracteres validos en los datos
my($entrada)=@_;
my(%pares,$clave,$valor);
```

```
$entrada =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("c",hex($1))/ge; #Convierto a alfanumerico los caracteres en hexadecimal
@pares=split('&',$entrada); #Separo cada par y lo guardo en un arreglo
foreach $par (@pares) { # proceso cada par en pares
($clave,$valor)=split(/=/,$par); #Guarde clave,valor usando la busqueda anterior
$valor =~ s/\+/ /g; #Cambie + por espacio en toda la cadena
# Seguridad  en caracteres,segun aviso del Cern en
# http://geek-girl.com/bugtraq/1997_4/0232.html
$clave =~ s/[^$BUENOS_CARACTERES]/_/go;
$valor =~ s/[^$BUENOS_CARACTERES]/_/go;
#Finalmente construya el arreglo asociativo
$pares{$clave}=$valor;
}
return(%pares);
} # Fin Decodificar_forma()
```

```
# Comienzo del programa principal
# Note como no se realiza validacion en este script, eso se
# lo confiamos al cliente (No es totalmente seguro)
$entrada=&metodo_forma();
%datos=&decodificar_forma($entrada);
$| = 1; #Vaciamos el buffer de escritura lo mas rapido posible
print("Content-type: text/html\r\n\r\n");
print("<html>\n");
print("<head><title>Datos del usuario</title></head>\n");
print("<body bgcolor=#ffffff\n");
print("<h1>Datos del usuario</h1>\n");
print("<ul>\n");
foreach $clave (keys %datos) {
print("<li><strong>$clave :</strong> $datos{$clave}\n");
}
print("<\ul>\n");
print("</body>\n");
print("</html>");
```

Ejercicio 7: Validación de Formas con Javascript

Utilice expresiones regulares para validar el contenido del campo nombre del ejemplo anterior. Se desea que en el nombre no aparezcan números ni caracteres de puntuación.[27]

Valide también al programa *cargar_red.cgi*. El programa no debe permitir el envio de datos si no se va a verificar a ninguna máquina:

Solución

Se muestra a continuación el código de cargar_red.cgi. Nótese como es necesario proteger ciertos caracteres para que estos puedan ser impresos desde Perl:

```
#!/usr/bin/perl

# Este Script carga la base de datos y la muestra por pantalla
# permitiendo luego llamar a verifica_red.cgi
#
# Hecho por Jose Vicente Nunez Zuleta (jose@ing.ula.ve)

# ***************************************************]
# Configuracion del script
#
$maqdir="./datos.txt";      #Ubicacion y nombre del archivo de datos
# ***************************************************]
# Fin de la configuracion del script

# Rutina que carga el contenido del archivo a memoria
sub cargar_red(\%\%\%) {
my($ip,$so,$ti)=@_; #Las variables vienen en @_
# Leemos los datos y los guardamos en un hash
open(DATOS,$maqdir) || die "<html><head><title>Error en CGI</title></head><body><P>Error: No puedo cargar los datos de la red $!</body></html>\n";
my($aux,$aux2,$aux3,$aux4); #Variable auxiliar
#Comenzamos la lectura de los datos
my $c=0;
while(<DATOS>) {
#Validamos que no sea un comentario o linea vacia
chop($_);
if (($_ !~ /^\#/) && ($_ ne "")) {
$c++; #Otra maquina leida
($aux,$aux2,$aux3,$aux4)=split('\s+',$_);
$$ip{$aux2}=$aux;
$$so{$aux2}=$aux3;
$$ti{$aux2}=$aux4;
```

[27] Sugerencia: Utilice la función implementada en el ejercicio de expresiones regulares.

```
}
}
return($c);
} #Fin Cargar
```

```
$| = 1; #Vaciamos el buffer de escritura lo mas rapido posible
print("Content-type: text/html \n\n"); #Encabezado del mensaje
print("<!-- Documento generado de manera dinamica -->\n");
print("<html><head><title>Verifica Red: 1.0</title>\n");
# Aqui colocamos el codigo de Javascript, escrito desde Perl
print("<script languaje=\"Javascript 1.2\">\n");
print("function vmaquinas() \{\n");
print("var vacio1=true\n");
print("var vacio2=true\n");
print("//Verificamos si hay una direccion en la caja de texto IP\n");
print("\tif (document.datos.individual.value \!= '')\{\n");
print("\t\tvacio1=false\n");
print("\t\}\n");
print("//Verificamos si el usuario selecciono alguna casilla \n");
print("\tfor (var i=0;i<document.datos.elements.length;i++) \{\n");
print("\t\tif (document.datos.elements[i].type == \"checkbox\") \{\n");
print("\t\t\tif (document.datos.elements[i].checked) \{\n");
print("\t\t\t\tvacio2=false\n");
print("\t\t\t}\n");
print("\t\t}\n");
print("\t}\n");
print("if ((vacio1 && vacio2)) \{\n");
print("alert('Debe escoger una maquina para poder trabajar');return(false)\}\n");
print("\}\n");
print("//-->\n");
print("</script>\n");
print("</head>\n");
print("<body bgcolor=#000000 text=yellow vlink=red alink=yellow link=yellow>\n");
$cont=&cargar_red(\%dirip,\%sistop,\%tipo);
print("<font size=5 face=\"Arial,Helvetica,Verdana\">Leidas $cont maquinas</font>\n");
# Presentamos el menu para probar las maquinas
print("<H1>Verificar <em>una m&aacute\;quina</em> arbitraria</H1>\n");
print("<form name=\"datos\" action=\"http://www.servidor.dominio/cgi-bin/verifica_red.cgi\" onSubmit=\" return vmaquinas()\" method=post>\n");
print("<P>Introduzca la direcci&oacute\;n IP que quiere verificar: <input type=text name=individual size=20 maxlength=20><P>\n");
print("<table border\=0>\n");
print("<tr bgcolor=blue>\n");
print("<th>Nombre<th>IP<th>Sistema operativo<th>Tipo<th>&#191Verificar?\n");
foreach $maquina (sort keys %dirip) {
print("<tr>\n");
print("<td>$maquina\n");
print("<td>$dirip{$maquina}\n");
print("<td>$sistop{$maquina}\n");
print("<td>$tipo{$maquina}\n");
print("<td align=center><input type=checkbox checked name=$maquina-$dirip{$maquina}>\n");
}
print("</table>");
print("<P>\n");
print("<BR><P><input type=submit value=Verificar> <input type=reset value=\"Volver a empezar\">\n");
print("</form>\n");
print("Sugerencias: <a href=\"mailto:jose\@ing.ula.ve\">jose\@ing.ula.ve\"</a>\n");
print("</body></html>");
```

Se recomienda que el lector interesado consulte la siguiente documentación para que pueda profundizar los conocimientos acerca de los puntos expuestos en este trabajo.

1. December John, Ginsburg Mark. "HTML 3.2 & CGI Professional Reference Edition". Sams.net Publishing. 1996. 1321 p.

1. Kamran Husain, Breedlove F. Robert. "Perl 5 Unleashed" Sams.net Publishing. 1996. 798 p.

1. Mohseni Piroz. "Web Database Primer Plus, Conect your Database to the World Wide Web Using HTML, CGI, AND Java". Waite Group Press. 1996. 482 p.

1. Netscape Communications Corporation. "Javascript Guide 1.2". DevEdge On–line Documentation. http://developer.netscape.com/docs/manuals/communicator/jsguide4/. 1997.

1. Netscape Communications Corporation. "Getting Started with the Javascript debugger". ". DevEdge On–line Documentation. http://developer.netscape.com/docs/manuals/jsdebug/index.htm.
2. Netscape Communications Corporation On–line Documentation. http://developer.netscape.com/docs/manuals/.

1. Perl Institute. http://www.perl.org/.

1. Perl Source. http://www.perl.com/.

1. Veer Vander Emily A. "Javascript for Beginners". http://www.builder.com/Programming/Javascript/.

1. World Wide Web Consortium. http://www.w3.org/.

1. Wall Larry, Schwartz L. Randal. "Programming Perl". O'Reilly & Associates, Inc. 1992. 463 p.