Programación en WWW usando CGI

Jesús Vegas
Departamento de Informática
Universidad de Valladolid
jvegas@infor.uva.es

Índice General

1 Introducción: Un Poco de Orden entre Tantas Siglas

El objetivo de este curso es explicar el modo en el que se puede utilizar el CGI [2] para interactuar con un documento HTML transmitido según el protocolo HTTP, a través de la WWW [1].

La forma en que se ha presentado el objetivo puede parecer más o menos precisa, pero resulta necesario aclarar ese montón de siglas que hemos utilizado.

La idea fundamental es que los navegadores [5], *browsers*, presentan documentos escritos en HTML que han obtenido de WWW. Estos documentos HTML habitualmente presentan información de forma estática, sin más posibilidad de interacción con ellos. Pero si incluyen formularios HTML, pueden servir para recoger información y luego procesarla.

Cuando la información es recogida por el navegador, es mandada al servidor HTTP especificado en el formulario, y el servidor arranca un programa, también especificado en el formulario, que procesa la información recogida en el formulario. Estos programas son conocidos como *scripts Common Gateway Interface*, CGI.

Aquí no se pretende explicar ni los protocolos de comunicación de WWW, ni de modo de crear documentos HTML, sólo introduciremos lo que necesitemos de todos ellos para hacer comprensible la explicación del CGI.

1.1 World Wide Web, WWW

WWW es un sistema hipermedia interactivo desarrollado sobre Internet. Definir Internet no está dentro de los objetivos de este curso y lo único que simplificando se puede decir sobre ella es que funciona. La idea de hipermedia es la de juntar texto, imágenes, audio y vídeo dentro de un mismo envoltorio llamado documento. WWW se asienta sobre el protocolo HTTP (Hyper Text Transfer Protocol) y sobre el lenguaje de definición de documentos hipermedia HTML (HyperText Markup Language).

El primer precedente del WWW se puede encontrar en un tratado escrito por *Vannevar Bush* titulado "*As We May Think*", de 1954, en el que básicamente animaba a los científicos a hacer más accesible a todo el mundo sus conocimientos y experiencia. En este tratado surgieron las nociones de hipertexto e hipermedia.

Desde entonces, muchos han sido los sistemas hipermedia que se han creado. Como ejemplo cabe destacar el *Augment* de Doug Englebart en los 70, y en los 90 el *WWW* de Tim Berners-Lee, TBL, creado en el Laboratorio Europeo de Partículas Físicas (CERN) en Ginebra, Suiza. TBL se basó en muchas fuentes para crear el WWW:

- TCP/IP,
- tipos MIME,
- SGML,
- arquitecturas Cliente-Servidor y Sistemas Abiertos.

Así, WWW es el sistema de difusión del conocimiento más importante que implementa estas nociones.

1.2 HyperText Transfer Protocol, HTTP

Aunque entender el modo en que funciona HTTP no es estrictamente necesario para desarrollar aplicaciones CGI, algunas nociones sobre lo que esconden esas siglas puede ayudar a desarrollarlas con más facilidad y confianza.

HTTP es el protocolo que permite que CGI pueda funcionar, y determina qué se puede o no mandar y recibir via CGI.

Entre las propiedades de HTTP se pueden destacar las siguientes:

- Un esquema de direccionamiento comprensible.
  Utiliza el *Universal Resource Identifier* (URI) para localizar sitios (URL) o nombres (URN) sobre los que hay que aplicar un método. La forma general de un URL es `servicio://host/fichero.ext` .

- Arquitectura Cliente–Servidor.
  HTTP se asienta en el paradigma solicitud/respuesta. La comunicación se asienta sobre TCP/IP. El puerto por defecto es el 80, pero se pueden utilizar otros.

- Es un protocolo sin conexión y sin estado.
  Después de que el servidor ha respondido la petición del cliente, se rompe la conexión entre ambos. Además no se guarda memoria del contexto de la conexión para siguientes conexiones.

- Está abierto a nuevos tipos de datos.
  HTTP utiliza tipos MIME (*Multipart Internet Mail Extension*) para la determinación del tipo de los datos que transporta. Cuando un servidor HTTP transmite información de vuelta a un cliente, incluye una cabecera que le indica al cliente sobre los tipos de datos que componen el documento. De la gestión de esos datos se encargan las utilidades que tenga el cliente (visor de imágenes, de vídeo, etc.)

Una transacción HTTP está compuesta por una cabecera, y opcionalmente, por una línea en blanco seguida de los datos. En la cabecera se especifica tanto la acción solicitada en el servidor, como los tipos de datos devueltos o un código de estado.

2 La Interacción entre el *Browser* y el Servidor

Durante una sesión normal de trabajo en WWW un cliente (navegador) solicita un documento de un servidor Web y una vez obtenido lo muestra al usuario que hizo la solicitud. Si este documento contiene un enlace a otro documento (en el mismo o en distinto servidor), y el usuario activa el enlace el cliente Web efectuará otra petición y mostrará el nuevo documento.

Durante la comunicación entre el cliente y el servidor HTTP en el que el cliente solicita el documento `doc1.html` al servidor se intercambian la siguiente transacción HTTP:

```
GET /doc1.html HTTP/1.0
Accept: www/source
Accept: text/html
Accept: image/gif
User-Agent:  Lynx/2.2  libwww/2.14
From: jvegas@infor.uva.es
/* esto es una linea en blanco */
```

La petición `GET` indica el fichero que el cliente solicita y la versión de HTTP. El cliente también muestra una lista de los tipos MIME que puede aceptar como retorno, además de identificar el *browser* que utiliza y su dirección de correo electrónico. Al final existe una línea en blanco que determina el final de la cabecera HTTP.

El servidor responde mandando la siguiente transacción HTTP:

```
HTTP/1.0  200 OK
Date: Friday, 03-Jul-98 12:00:00 GMT
Server: Apache/1.1.1
Content-type: text/html
Content-length: 230
/* esto es una linea en blanco */
<HTML><HEAD><TITLE> ........ </HTML>
```

En este mensaje el servidor utiliza la versión 1.0 de HTTP, y manda el código de estado 200 para indicar que la petición del cliente ha sido procesada satisfactoriamente. También se identifica como un servidor Apache. Indica al cliente que el contenido del documento es texto en formato HTML y que tiene una longitud de 230 bytes.

2.1 Ejecutando *scripts*

Un URL puede identificar un fichero que contiene un programa, en lugar de un documento HTML. De modo que este programa debe ser ejecutado cuando el usuario activa ese enlace.

La diferencia entre un fichero que contiene un documento HTML y un programa está en el modo y/o el lugar en que está almacenado en el servidor. Por razones de seguridad, la mayoría de los servidores Web no ejecutarán un fichero (aunque el fichero tenga los permisos de ejecución) a menos que esté colocado en un directorio determinado, o tenga una extensión apropiada. Incluso si un usuario puede crear y colocar su propia página Web, quizás no tenga permisos para escribir en ese directorio. Normalmente habrá que consultar al administrador del servidor Web, habitualmente conocido como *webmaster* para poder colocar los *scripts* en el directorio apropiado. El directorio que almacena los *scripts* CGI suele llamarse `cgi-bin`.

Cuando el servidor ejecuta el programa indicado en el URL, como resultado de la ejecución se produce un documento HTML que es mandado de retorno al cliente, y que es mostrado en el *browser* sin que se pueda apreciar la diferencia con un documento normal.

Estos programas llamados *scripts* CGI pueden estar escritos en cualquier lenguaje que se pueda ejecutar sobre la máquina que mantiene el servidor Web. Habitualmente se suelen escribir en C, TCL o Perl [7]. Los programas escritos en C son más rápidos, pero los *scripts* escritos en el `shell` o en Perl son más adecuados para el tipo de manipulación de textos que se suele realizar. En este curso, los ejemplos serán escritos en C, ya que es un lenguaje de programación universal, y esto justifica su uso, aunque no sea el más apropiado para tratar cadenas de caracteres. La mayoría de los *scripts* CGI y bibliotecas de herramientas CGI [3] están escritas en Perl, por lo que recomendamos su aprendizaje y manejo, aunque este punto queda fuera de este curso por razones de brevedad.

Como ejemplo podemos ver un *script* CGI (`cgi-bin/hola`) escrito en el lenguaje de comandos del *shell* cuyo resultado será un documento HTML con un mensaje de bienvenida.

```
#!/usr/bin/ksh
echo "Content-type: text/html"
echo ""
echo "<HTML>"
echo "<HEAD><TITLE>hola</TITLE></HEAD>"
echo "<BODY>hola.</BODY></HTML>"
echo "</HTML>"
```

Al que se invoca con el documento siguiente:

```
<HTML>
<HEAD>
<TITLE> Script CGI </TITLE>
```

```
</HEAD>
<BODY>
<P>Pulsa
<A HREF="http://descartes.dcs.fi.uva.es/cgi-bin/hola">aquí</A>
para recibir un saludo.</P>
</BODY>
</HTML>
```

2.2 Los *scripts* Gestionando Datos

Se puede dar un paso más en este proceso cuando el *script* CGI reciba datos del cliente, los gestione, y devuelva una respuesta adecuada al cliente. Esto es lo que necesitamos cuando queremos hacer accesible al mundo WWW una BD clásica. Así, el los *scripts* CGI hará de interfaz entre WWW y la BD.

Para ello, necesitaremos presentar un formulario al cliente, que una vez rellenado, envien la información recogida al servidor especificado en el formulario. El servidor arrancará el *script* CGI y le pasará los datos recogidos para que prepare un documento de respuesta apropiado, posiblemente consultado una BD de manera previa.

Los formularios en HTML se definen utilizando un conjunto de etiquetas HTML específico que vamos a presentar a continuación.

3 Formularios HTML

Los formularios añaden una dimensión nueva a las páginas HTML, ya que éstas pueden ser más o menos sofisticadas, pero sin los formularios son de sólo lectura.

Un formulario es una interfaz gráfica de usuario, GUI, que permite la inclusión de botones de diferentes tipos, areas de texto, listas de selección, etc. Cuando se escribe un elemento del formulario, cada uno de ellos tiene un par de atributos: `NAME` y `VALUE`, de modo que cuando el usuario pone información en ese elemento del formulario, esa información se asocia al nombre del elemento. Los elementos pueden tener valores iniciales.

Cuando el usuario presiona un botón indicando que el formulario debe ser enviado, *submitted*, la información en el formulario es mandada al servidor para ser procesada por un *script* CGI. Los datos se envian como pares `nombre=valor`, separados por *ampersands*, &, donde `nombre` es el dado en el atributo `NAME` y el `valor` es el dado en el atributo `VALUE` o el dado por el usuario. Cada par `nombre=valor` se codifica según el modo URL, esto es, los espacios en blanco se sustituyen por mases, +, y los caracteres no alfanuméricos son sustituidos por sus códigos hexadecimales, $\%hh$. En el siguiente ejemplo se puede ver el caso de la codificación URL de la información obtenida en un formulario en el que se han introducido el nombre y los apellidos del usuario *Pepe Pérez Pardo*:

```
nombre=Pepe&apellidos=P%e9rez+Pardo
```

La primera parte del procesamiento de la información del formulario por parte del *script* CGI será romper la secuencia de datos en por los *ampersands*, y luego cada par `nombre=valor` deberá ser decodificado. En C, lo más habitual es colocar cada par `nombre=valor` como elementos de un par de arrays paralelos de cadenas de caracteres.

3.1 La etiqueta FORM

Todo formulario HTML está delimitado por las etiquetas `<FORM>` y `</FORM>`, entre los que se deben encontrar los elementos que componen el formulario, así como cualquier otro elemento HTML que necesitemos: imágenes, enlaces, etc. En un documento HTML puede haber tantos formularios como se necesite, pero no pueden estar anidados.

La formá básica de definición de un formulario es la siguiente:

```
<FORM ACTION="http://www.sitio/el_script"
      METHOD="POST">

</FORM>
```

Los atributos de la etiqueta `FORM` son:

- `ACTION` especifica el URL del *script* CGI que debe tratar los datos recopilados por el formulario. Este es el enlace entre el formulario y el *script* CGI.

- `METHOD` determina el método a utilizar para acceder al URL determinado en el campo `ACTION`. Existen dos métodos, `GET` (el defecto) y `POST` (basados en los métodos HTTP). En los casos generales, se usará `GET` cuando tengamos formularios muy simples, y `POST` cuando los formularios tengan muchos campos, ya que `GET` podría no ser capaz de pasar todos los datos al servidor.

  – `METHOD="GET"`
  La información del formulario se añade al final del URL solicitado, detrá de un símbolo de interrogación ,'?'. Esto es lo que hacen la mayoría de los buscadores Web. El *script* CGI la recibirá codificada en la variable de ambiente `QUERY_STRING`. El principal problema de este método es que tiene restricciones de longitud debido a que muchos sistemas operativos limitan el nombre de un fichero a 256 caracteres, por ejemplo, y el servidor Web trata el URL, incluyendo la consulta, como si fuera un tipo de fichero. Otra desventaja es que el URL, incluyendo la consulta, es almacenado en los diarios de accesos, *access logs*, de los servidores, lo que puede ser poco recomendable si la información de la consulta se quiere mantener en secreto.

  – `METHOD="POST"`
  El *script* CGI recibe la información del formulario por su entrada estandar.

El servidor no pone la marca fin de fichero, EOF. En su lugar, y para conocer la longitud de los datos de entrada, se especifican los *bytes* en la variable `CONTENT_LENGTH`.

- `ENCTYPE` especifica el tipo de datos utilizado para codificar los pares nombre/valor durante el transporte. Por defecto es igual a `application/x-www-form-urlencoded` y no se suele especificar.

3.2 Elementos de Formulario

Dentro de un formulario podremos recoger información utilizando para ello los distintos elementos disponibles. La forma más usual es permitiendo que el usuario introduzca texto, aunque también podremos elegir una opción entre una lista, pulsando uno o más botones, etc. Todo ello se puede realizar combinando tres etiquetas HTML[1]: `INPUT`, `TEXTAREA` y `SELECT`.

3.2.1 La etiqueta INPUT

La forma general de la etiqueta `INPUT` es la siguiente:

```
<INPUT TYPE="tipo" NAME="nombre" VALUE="valor" SIZE="tamaño"
      MAXLENGTH="max_longitud" CHECKED>
```

Los atributos de `INPUT` son:

- `TYPE` determina el tipo de entrada que se va a utilizar y sus valores pueden ser uno de los siguientes:
  - `text`, muestra una caja donde introducir texto. Es el tipo por defecto de `INPUT`.
  - `password`, muestra una caja donde se tecleará texto cuyo eco será sustituido por asteríscos.
  - `checkbox`, un botón que sólo puede tener dos estados, {*on*, *off*}.
  - `radio`, un botón que sólo puede tener dos estados, {*on*, *off*}, pero que a diferencia de los `checkbox`, puede agruparse con otros botones de tipo `radio` con el mismo nombre y obtener así un comportamiento "*uno de muchos*".
  - `submit`, un botón que produce que el formulario actual sea enviado al URL especificado en el atributo `ACTION` del formulario.
  - `reset`, un botón que produce que todos los elementos del formulario pasen a su estado por defecto.

[1] Refiriéndonos a la versión 3.2 de HTML

- `hidden`, es una entrada que no se muestra en el *browser* y que sirve para pasar información desde el formulario al servidor que no se quiere que el usuario vea.

- `NAME` es el nombre simbólico al que se le va asociar el valor introducido en ese elemento por el usuario. Debe estar presente en todos los elementos excepto en `submit` y `reset`.

- `VALUE` tiene varios significados dependiendo del elemento que se trate. Para campos texto o `password` especificará el valor por defecto. En botones de tipo `radio` o `checkbox` especificará el valor que tendrá el botón cuando sea presionado (el defecto es *on*). Para los botones `submit` y `reset` el valor se puede utilizar para especificar la etiqueta del botón. En los campos ocultos especifica el valor que se pasa al servidor.

- `SIZE` es el tamaño físico del campo de entrada en elementos texto o `password` (por defecto es 20).

- `MAXLENGTH` es el número máximo de caracteres que se permiten en un elemento de tipo texto o `password` (ilimitado por defecto).

- `CHECKED` especifica si ese botón, `radio` o `checkbox` estará activado por defecto.

3.2.2 La etiqueta `TEXTAREA`

La introducción de un texto, con un número de filas y columnas determinado se puede conseguir con la utilización de la etiqueta `TEXTAREA` cuya forma general es la siguiente:

```
<TEXTAREA NAME="nombre"
          ROWS="filas" COLS="columnas">
texto por defecto
</TEXTAREA>
```

Con ello, se presenta una caja donde el usuario puede introducir varias líneas de texto. La caja de texto tendrá barras de desplazamiento y unas dimensiones de `ROWS`$\times$`COLS`. El texto por defecto (optativo) aparecerá entre las etiquetas de apertura y cierre del area de texto.

3.2.3 La etiqueta `SELECT`

Este elemento se utiliza para presentar al usuario varias opciones y que elija entre ellas una o más. La forma general es:

```
<SELECT NAME="nombre" SIZE="num" MULTIPLE>
<OPTION VALUE="valor_asociado_A" SELECTED>Opción A
```

```
<OPTION VALUE="valor_asociado_B" >Opción B
<OPTION VALUE="valor_asociado_C" >Opción C
</SELECT>
```

Los atributos de `SELECT` permiten especificar el nombre lógico al que se asociará la opción elegida de entre las posibles. Sólo serán visibles las que se especifiquen en `SIZE`. Si es 1, el elemento selección tendrá la apariencia de un menú desplegable (cuidado con el número de opciones, pues podría desbordar la pantalla). Normalmente, el elemento selección permite seleccionar sólo una de sus opciones, pero con `MULTIPLE` podremos hacer que la selección afecte a varios de sus elementos.

Las opciones se especifícan con la etiqueta `OPTION` que puede aparecer con un atributo `VALUE` que indica el valor que se asociará al elemento de selección en caso de que esa opción sea la elegida. Este atributo puede obviarse, en cuyo caso el valor devuelto será la descripción de la opción misma. Se puede especificar con `SELECTED` la opción seleccionada por defecto.

3.3 Diferencias en Formularios entre HTML 3.2 y 4

Existen varios problemas y deficiencias en la versión 3.2 de HTML que la versión 4 ha tratado de solucionar, entre ellos los siguientes:

- No existía la posibilidad de teclas atajo para acceder a botones, desplegar menús, o disparar una acción. El atributo `accesskey` provee el acceso directo a los campos de formulario desde el teclado.

- Con el atributo `disabled` se puede hacer que un elemento del formulario se vuelva insensible a las acciones del usuario, de modo que no reciba el foco, ni sea tenido en cuenta cuando se avanza por tabuladores, y su valor no sea enviado con el formulario.

- Se pueden declarar campos de sólo lectura con el atributo `readonly`.

- Las etiquetas de los botones de tipo `radio` y `checkbox` se vuelven sensibles.

- No se podía comprobar el valor de los campos antes de mandarlos al servidor, aunque ahora, con el atributo `onchange-INPUT` y otros, junto con los lenguajes como *javascript* se puede hacer dicha validación.

- Los botones pueden ahora estar etiquetados con figuras o texto formateado.

4 Un Formulario Ejemplo

Como en todo tutorial de este tipo de materias no nos podríamos salvar del consiguiente ejemplo en que se ilustre, de manera más o menos afortunada, las posibilidades de los formularios.

Así, en la Figura 1 se puede ver un ejemplo en el que se ilustra el uso de los formularios en HTML, con elementos de introducción de texto, diferentes maneras de permitir la selección entre posibilidades, y modos de tener el control sobre el formulario.

Formularios en HTML

Este es un pequeño muestrario de las posibilidades de interacción entre clientes y servidores Web a través de los formularios y los programas cgi-bin.

El único resultado es el eco de la información introducida en el formulario.

Introducción de textos		
Introduzca su nombre:		<INPUT TYPE="text" NAME="nombre">
Introduzca su edad:		<INPUT TYPE="password" NAME="edad">
Introduzca su dirección:		<TEXTAREA NAME="direccion" ROWS="2">>Aquí.</TEXTAREA>
Selección		
¿Estación preferida?	Primavera Verano Otoño Invierno	<INPUT TYPE="checkbox" NAME="primavera"> Primavera <INPUT TYPE="checkbox" NAME="verano"> Verano <INPUT TYPE="checkbox" NAME="otoño"> Otoño <INPUT TYPE="checkbox" NAME="invierno"> Invierno
¿Estación preferida?	Primavera Verano Otoño Invierno	<INPUT TYPE="radio" NAME="estacion" VALUE="0"> Primavera <INPUT TYPE="radio" NAME="estacion" VALUE="1" CHECKED> Verano <INPUT TYPE="radio" NAME="estacion" VALUE="2"> Otoño <INPUT TYPE="radio" NAME="estacion" VALUE="3"> Invierno
¿Estación preferida?		<SELECT NAME="select" SIZE="2"> <OPTION VALUE="0"> Primavera <OPTION VALUE="1" SELECTED> Verano <OPTION VALUE="2"> Otoño <OPTION VALUE="3"> Invierno </SELECT>
Control del formulario		
El mensaje oculto		<INPUT TYPE="hidden" NAME="secreto" VALUE="a voces">
Enviar		<INPUT TYPE="submit" VALUE="Enviar">
Limpiar		<INPUT TYPE="reset" VALUE="Limpiar">

Figura 1: Un ejemplo de formulario.

Como en el ejemplo no es posible apreciar del todo las posibilidades de los formularios, la dirección Web donde se encuentra dicho ejemplo para que se pueda visitar es `http://descartes.dcs.fi.uva.es/~jvegas/cursos/cgi-bin/formulario.html`. El código HTML de este formulario de ejemplo se puede ver en el Apéndice A.

La respuesta a este formulario es un simple eco que se puede ver en la Figura 2.

Variables de Ambiente

- QUERY_STRING =
- CONTENT_LENGTH = 67
- REQUEST_METHOD = POST
- HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
- REMOTE_HOST = descartes
- SCRIPT_NAME = /~jvegas/cgi-bin/echoPost.cgi
- SERVER_SOFTWARE = Apache/1.1.1
- SERVER_NAME = descartes.dcs.fi.uva.es
- SERVER_PORT = 80
- GATEWAY_INTERFACE = CGI/1.1

Resultado del cuestionario

Obtenidos los siguientes pares de nombre/valor:

- `nombre =`
- `edad =`
- `direccion = Aquí.`
- `estacion = 1`
- `select = 1`
- `secreto = a voces`

Figura 2: Eco de un formulario.

5 Leyendo los Datos CGI

Como ya hemos introducido anteriormente, existen dos métodos de enviar la información recogida por el formulario al servidor: `GET` y `POST`. Básicamente funcionan de la misma manera, y la diferencia estriba en el modo de leer la cadena de elementos `nombre=valor`.

Así, `GET` la obtiene de la variable de ambiente `QUERY_STRING`, mientras que `POST` lo hace de la entrada estándar.

Luego ambos debe decodificar la entrada (que esta codificada URL) y trabajar con la información así obtenida. Existen multitud de funciones en la red que nos ayudan a hacer todo esto, escritas en diferentes lenguajes, así que no es necesario que lo programemos nosotros. En las referencias bibliográficas se pueden encontrar varios ejemplos de esas funciones. Un ejemplo de ellas en el lenguaje C se puede ver en el Apéndice B.

Existe otro método por el que se complementa la información que el servidor Web pasa al *script* CGI y que hemos mencionado vagamente: las variables de entorno, que son parámetros que habitualmente toman la forma de variables de entorno del Sistema Operativo. De entre ellas se pueden destacar las siguientes:

- `AUTH_TYPE` el método de autentificación utilizado para validar al usuario, si el servidor soporta la autentificación de usuarios.
- `CONTENT_LENGTH` la longitud de la información mandada por el cliente.
- `CONTENT_TYPE` el tipo de información asociada a las consultas.
- `GATEWAY_INTERFACE` la revisión de la especificación CGI que soporta el servidor, con el formato CGI/revisión.
- `PATH_INFO` la información asociada al *path* virtual, tal cual es dada por el cliente, y que es decodificada por el servidor.
- `PATH_TRANSLATED` el *path* físico obtenido a partir del `PATH_INFO`.
- `QUERY_STRING` la información que sigue a la '`?`' en el URL.
- `REMOTE_ADDR` la dirección IP del *host* remoto que realiza la petición.
- `REMOTE_IDENT` el nombre de usuario remoto, si el servidor HTTP soporta la identificación RFC 931.
- `REMOTE_USER` el nombre de usuario autentificado si el *script* CGI está protegido y el servidor soporta autentificación de usuarios.
- `REQUEST_METHOD` el método mediante el cual se hizo la petición. Para HTTP será *GET*, *HEAD*, *POST*, etc.
- `SCRIPT_NAME` el *path* virtual del programa que se esta ejecutando.
- `SERVER_NAME` el nombre de *host* del servidor, alias DNS, o dirección IP.
- `SERVER_PORT` el número de puerto al que fue mandada la petición.

- `SERVER_PROTOCOL` el nombre y revisión del protocolo utilizado para esta petición.
- `SERVER_SOFTWARE` el nombre y revisión del software del servidor que responde a la petición.

Ambos métodos, `GET` y `POST` tienen en común que su salida estándar está conectada con el cliente que hizo la petición. De modo que para que el resultado de la ejecución de un *script* CGI sea recibido por el cliente que realizó la petición, el programa debe generar un documento HTML y escribirlo en su salida estándar. El servidor WWW se encargará de redirigirlo al cliente como respuesta a su petición.

6 Instalando un *Script* CGI

Resulta difícil indicar el modo de instalar un *script* CGI desde aquí, ya que muchas son las variables que afectan a este propósito, comenzando por la principal: el servidor Web; ya que cada servidor se configura de manera ligeramente distinta. Así, que lo primero a hacer es consultar al *webmaster* por el servidor, el camino donde instalar el *script*, cómo llamarlo, los permisos que debe tener, y si el *script* necesita escribir sobre algún fichero, qué permisos debe tener ese fichero.

El caso que vamos a describir se refiere al servidor Apache [4], que es un servidor Web muy popular en entornos UNIX. Apache tiene tres ficheros de configuración: `httpd.conf`, `srm.conf` y `access.conf`, normalmente instalados en algún sitio similar a `/usr/local/apache`.

El segundo de los ficheros es donde se especifíca cómo se deben gestionar las peticiones de los clientes. Si un *browser* solicita una página Web, es servidor devolverá el código fuente HTML de esa página, esto es, el contenido del fichero requerido. Pero si un *browser* solicita un *script* CGI, el servidor debe saber que no debe devolver el fichero tal cual, sino que debe ejecutar el *script* almacenado en el fichero requerido, y devolver al cliente el resultado. Así, el servidor debe conocer la diferencia entre un fichero HTML y otro CGI.

Existen dos formas de hacer esto en el fichero `srm.conf`, cada una de ellas se corresponde con una directiva de configuración siguientes:

```
ScriptAlias  /cgi-bin/    /usr/local/apache/cgi-bin/
AddHandler   cgi-script   .cgi
```

La primera de ellas dice al servidor que cualquier fichero que se encuentre en el directorio `/usr/local/apache/cgi-bin/` es un *script* CGI que debe ser ejecutado cada vez que una petición se refiera a él.

La segunda línea indica al servidor que cualquier fichero, en cualquier directorio, con la extensión `.cgi` es un *script* CGI que debe ejecutarse.

Si su servidor no tiene la segunda directiva, y usted no tiene permisos para escribir en el directorio indicado por `ScriptAlias` entonces no podrá instalar *scripts* CGI en su servidor Web. Si su servidor no tiene ninguna de las dos directivas no podrá gestionar peticiones CGI.

7 Consideraciones de Seguridad

Los *scripts* CGI son, aun sin querer serlo, una puerta por donde pueden entrar los problemas debido a que dan acceso al *shell* de las máquinas donde está el servidor Web, permitiendo el acceso a información restringida, o posibilitando que el sistema sufra sabotajes del exterior. Por ello vamos a hacer alguna consideración sobre la seguridad en los servidores Web cuando se ejecutan *scripts* CGI. La mayoría de estos comentarios tienen sentido en entornos UNIX, y en otros sistemas operativos pueden carecer de valor.

Hay que intentar evitar que nuestros *scripts* se queden colgados por cualquier razón, y como respuesta a ese fallo se deje acceso al *shell* desde el que se estaban ejecutando, abriendo así las puertas a un acceso no controlado.

7.1 No Confiar en Nada

La idea principal es que nunca se debe confiar en nada, especialmente en los datos de entrada de los formularios. Las siguientes ideas son falsas:

- Si se ha puesto una lista de selección, la entrada será una de las opciones posibles.
- Si la longitud máxima de un campo de entrada está fijada, el *browser* mandará como mucho esos caracteres.
- Los campos de la variable `QUERY_STRING` se corresponderán con los de la página.
- El contenido de la variable `QUERY_STRING` será alguno de los válidos que pueden ser transmitidos por HTTP.

7.2 Los Nombres de los Ficheros

La mayoría de las ideas de esta sección parece obvia, pero conviene recordarlas.

Problamente los nombres de los ficheros codificados en los programas CGI son seguros, pero los nombres obtenidos de los formularios, de `PATH_INFO` por ejemplo, son sospechosos y hay que comprobarlos. Algunas veces es práctico mantener una lista de nombres válidos que eviten considerar, por ejemplo, como inválido cualquier nombre que comience por '`/`'.

Si se han de crear ficheros, nunca ha de permitirse que comiencen por '`.`' ni por '`;`'. También será una buena idea mantener una lista de caractéres válidos para formar los nombres de los ficheros.

La máscara `UMASK` por defecto de muchos servidores Web es `0`, por lo que cualquier fichero creado por un *script* CGI podrá ser escrito por cualquiera. La máscara `UMASK` debe ser puesta a `022` o a `077`.

7.3 Invocando Programas

Muchos *scripts* CGI invocan a otros programas, ya sean creados por el usuario o del sistema, lo que supone el principal agujero de seguridad en CGI.

El problema básico es que por ejemplo un *script* CGI llame a `grep` con una expresión regular de cualquiera de las maneras siguientes (en Perl o C):

```
system("grep $exp database");
sprintf(comando,"grep %s database",exp);system(comando);
```

tiene varios problemas, sobre todo si consideramos que `exp` puede valer "`root /etc/passwd; rm`", con lo que lee el fichero de *passwords* y además inutiliza el sistema.

La solución puede ser colocar comillas de la siguiente manera:

```
system("grep \"$exp\" database");
sprintf(comando,"grep \"%s\" database",exp);system(comando);
```

pero surge el problema que de las comillas se emparejen con otras dentro de `exp` con lo que se anule su efecto.

8 Referencias Biblográficas

Dada la naturaleza del tema central de este tutorial, parece lógico que este apartado de las referencias bibliográficas se alimente de referencias a documentos Web. No quiere decir esto que no haya buenos libros editados sobre el tema, sino que es más fácil y dinámico acceder a las referencias Web, además de que están actualizadas con más frecuencia y responden mejor a la evolución de la tecnología.

No hemos pretendido avasallar con un listado interminable de referencias sino mostrar las más interesantes. Además, siempre se pueden conseguir miles de documentos interesantes utilizando alguno de los buscadores Web.

1. *The World Wide Web Consortium.*
   `http://www.w3c.org/`
   Es la página del consorcio WWW y en ella se puede encontrar información técnica sobre cualquier tema relacionado con WWW, ya sea arquitectura de sistemas, protocolos, herramientas, etc. Es básico y el lugar de partida obligatorio.

2. *CGI: The Common Gateway Interface.*
`http://hoohoo.ncsa.uiuc.edu/cgi/`

3. *CGI: The Common Gateway Interface for Server-side Processing.*
`http://Stars.com/Authoring/CGI/`
Es un buen punto de partida para estudiar y poner en práctica la programación CGI.

4. *Apache Proyect.*
`http://www.apache.org/`
Es la página del servidor Web *Apache*. Es fácil de cargar, intalar, configurar y gratis.

5. *Browser Netscape.*
`http://home.netscape.com`
El navegador *Netscape* (gratis).

6. *Java Technology Home Page.*
`http://java.sun.com/`
La página central del lenguaje Java.

7. *The Perl Institute.*
`http://www.perl.org/`
La página del lenguaje Perl.

A Formulario Ejemplo

Este es el código HTML del formulario presentado como ejemplo.

```
<HTML>
<HEAD>
<TITLE>Formularios en HTML</TITLE>
</HEAD>
<BODY>
<H1>Formularios en HTML</H1>
<P> Este es un pequeño muestrario de las posibilidades de interacción
entre clientes y servidores Web a través de los formularios y los
programas cgi-bin. </P>
<P> El único resultado es el eco de la información introducida en el
formulario.</P>
<P>
<FORM METHOD="POST"
```

```
ACTION="http://descartes.dcs.fi.uva.es/~jvegas/cgi-bin/echoPost.cgi">
<TABLE BORDER=1 ALIGN=CENTER CELLSPACING=0 CELLPADDING=0>
<TR>
<TH>Introducción de textos</TH>
</TR>
<TR>
<TD>Introduzca su nombre:</TD>
<TD><INPUT TYPE="text" NAME="nombre"> </TD>
<TD>&lt;INPUT TYPE="text" NAME="nombre"&gt; </TD>
</TR>
<TR>
<TD>Introduzca su edad:</TD>
<TD><INPUT TYPE="password" NAME="edad"> </TD>
<TD>&lt;INPUT TYPE="password" NAME="edad"&gt; </TD>
</TR>
<TR>
<TD>Introduzca su dirección:</TD>
<TD><TEXTAREA NAME="direccion" ROWS="2">Aquí.</TEXTAREA></TD>
<TD>&lt;TEXTAREA NAME="direccion" ROWS="2"&gt;>Aquí.&lt;/TEXTAREA&gt;</TD>
</TR>
<TR>
<TH>Selección</TH>
</TR>
<TR>
<TD>>Estación preferida?</TD>
<TD>
<INPUT TYPE="checkbox" NAME="primavera"> Primavera <BR>
<INPUT TYPE="checkbox" NAME="verano"> Verano <BR>
<INPUT TYPE="checkbox" NAME="otoño"> Otoño <BR>
<INPUT TYPE="checkbox" NAME="invierno"> Invierno
</TD>
<TD>
&lt;INPUT TYPE="checkbox" NAME="primavera"&gt; Primavera <BR>
&lt;INPUT TYPE="checkbox" NAME="verano"&gt; Verano <BR>
&lt;INPUT TYPE="checkbox" NAME="otoño"&gt; Otoño <BR>
&lt;INPUT TYPE="checkbox" NAME="invierno"&gt; Invierno
</TD>
</TR>
<TR>
<TD>>Estación preferida?</TD>
<TD>
```

```
<INPUT TYPE="radio" NAME="estacion" VALUE="0"> Primavera <BR>
<INPUT TYPE="radio" NAME="estacion" VALUE="1" CHECKED> Verano <BR>
<INPUT TYPE="radio" NAME="estacion" VALUE="2"> Otoño <BR>
<INPUT TYPE="radio" NAME="estacion" VALUE="3"> Invierno
</TD>
<TD>
&lt;INPUT TYPE="radio" NAME="estacion" VALUE="0"&gt; Primavera <BR>
&lt;INPUT TYPE="radio" NAME="estacion" VALUE="1" CHECKED&gt; Verano <BR>
&lt;INPUT TYPE="radio" NAME="estacion" VALUE="2"&gt; Otoño <BR>
&lt;INPUT TYPE="radio" NAME="estacion" VALUE="3"&gt; Invierno
</TD>
</TR>
<TR>
<TD>>Estación preferida?</TD>
<TD>
<SELECT NAME="select" SIZE="2">
<OPTION VALUE="0"> Primavera <BR>
<OPTION VALUE="1" SELECTED> Verano <BR>
<OPTION VALUE="2"> Otoño <BR>
<OPTION VALUE="3"> Invierno <BR>
</SELECT>
</TD>
<TD>
&lt;SELECT NAME="select" SIZE="2"&gt;
&lt;OPTION VALUE="0"&gt; Primavera <BR>
&lt;OPTION VALUE="1" SELECTED&gt; Verano <BR>
&lt;OPTION VALUE="2"&gt; Otoño <BR>
&lt;OPTION VALUE="3"&gt; Invierno <BR>
&lt;/SELECT>
</TD>
</TR>
<TR>
</TR>
<TR>
<TH>Control del formulario</TH>
<TR>
<TD>El mensaje oculto</TD>
<TD><INPUT TYPE="hidden" NAME="secreto" VALUE="a voces"> </TD>
<TD>&lt;INPUT TYPE="hidden" NAME="secreto" VALUE="a voces"&gt; </TD>
</TR>
</TR>
```

```
<TD>Enviar</TD>
<TD><INPUT TYPE="submit" VALUE="Enviar"></TD>
<TD>&lt;INPUT TYPE="submit" VALUE="Enviar"&gt;</TD>
</TR>
<TR>
<TD>Limpiar</TD>
<TD><INPUT TYPE="reset" VALUE="Limpiar"></TD>
<TD>&lt;INPUT TYPE="reset" VALUE="Limpiar"&gt;</TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

B Código CGI Ejemplo

A continuación exponemos un ejemplo de funciones para leer y decodificar la información obtenida por un formulario.

Las funciones fueron obtenidas de la red hace un tiempo y hemos perdido la referencia, por lo que no podemos ponerla. A pesar de todo agradecemos su exfuerzo a los anónimos autores de las mismas.

El ejemplo realiza un eco de la información que le llega por un método `POST`.

B.1 El fichero `echoPost.c`

```
#include <stdio.h>
#include <stdlib.h>
#include "cgi_util.c"

#define MAX_ENTRIES 10000

typedef struct {
    char *name;
    char *val;
} entry;

main(int argc, char *argv[]) {
    entry entries[MAX_ENTRIES];
    register int x,m=0;
    int cl;
```

```
    printf("Content-type: text/html%c%c",10,10);

    if(strcmp(getenv("REQUEST_METHOD"),"POST")) {
        printf("Este script debe ser referenciado con el metodo POST.\n");
        exit(1);
    }
    if(strcmp(getenv("CONTENT_TYPE"),
                ªpplication/x-www-form-urlencoded")) {
        printf("Este script solo puede ser usado para decodificar
formularios . \n");
        exit(1);
    }

    printf("<h1>Variables de Ambiente</h1>%c",10);
    printf("<ul>%c",10);
    printf("<li> QUERY_STRING = %s",getenv("QUERY_STRING"));
    printf("<li> CONTENT_LENGTH = %s",getenv("CONTENT_LENGTH"));
    printf("<li> REQUEST_METHOD = %s",getenv("REQUEST_METHOD"));
    printf("<li> HTTP_ACCEPT = %s",getenv("HTTP_ACCEPT"));
    printf("<li> REMOTE_HOST = %s",getenv("REMOTE_HOST"));
    printf("<li> SCRIPT_NAME = %s",getenv("SCRIPT_NAME"));
    printf("<li> SERVER_SOFTWARE = %s",getenv("SERVER_SOFTWARE"));
    printf("<li> SERVER_NAME = %s",getenv("SERVER_NAME"));
    printf("<li> SERVER_PORT = %s",getenv("SERVER_PORT"));
    printf("<li> GATEWAY_INTERFACE = %s",getenv("GATEWAY_INTERFACE"));
    printf("</ul>%c",10);

    cl = atoi(getenv("CONTENT_LENGTH"));

    for(x=0;cl && (!feof(stdin));x++) {
        m=x;
        entries[x].val = fmakeword(stdin,'&',&cl);
        plustospace(entries[x].val);
        unescape_url(entries[x].val);
        entries[x].name = makeword(entries[x].val,'=');
    }

    printf("<H1>Resultado del cuestionario</H1>");
    printf("Obtenidos los siguientes pares de nombre/valor:<p>%c",10);
    printf("<ul>%c",10);
```

```
    for(x=0; x ≤ m; x++)
        printf("<li> <code>%s = %s</code>%c",entries[x].name,
               entries[x].val,10);
    printf("</ul>%c",10);
}
```

B.2 El fichero cgi_util.c

```
#include <stdio.h>

#define LF 10
#define CR 13

void getword(char *word, char *line, char stop) {
    int x = 0,y;

    for(x=0;((line[x]) && (line[x] ≠ stop));x++)
        word[x] = line[x];

    word[x] = '\ 0';
    if(line[x]) ++x;
    y=0;

    while(line[y++] = line[x++]);
}

char *makeword(char *line, char stop) {
    int x = 0,y;
    char *word = (char *) malloc(sizeof(char) * (strlen(line) + 1));

    for(x=0;((line[x]) && (line[x] ≠ stop));x++)
        word[x] = line[x];

    word[x] = '\0';
    if(line[x]) ++x;
    y=0;

    while(line[y++] = line[x++]);
    return word;
```

```
}

char *fmakeword(FILE *f, char stop, int *cl) {
    int wsize;
    char *word;
    int ll;

    wsize = 102400;
    ll=0;
    word = (char *) malloc(sizeof(char) * (wsize + 1));

    while(1) {
        word[ll] = (char)fgetc(f);
        if(ll==wsize) {
            word[ll+1] = '\0';
            wsize+=102400;
            word = (char *)realloc(word,sizeof(char)*(wsize+1));
        }
        --(*cl);
        if((word[ll] == stop) || (feof(f)) || (!(*cl))) {
            if(word[ll] != stop) ll++;
            word[ll] = '\0';
            return word;
        }
        ++ll;
    }
}

char x2c(char *what) {
    register char digit;

    digit = (what[0] >= 'A' ? ((what[0] & 0xdf) - 'A')+10 : (what[0] - '0'));
    digit *= 16;
    digit += (what[1] >= 'A' ? ((what[1] & 0xdf) - 'A')+10 : (what[1] - '0'));
    return(digit);
}

void unescape_url(char *url) {
    register int x,y;

    for(x=0,y=0;url[y];++x,++y) {
```

```
        if((url[x] = url[y]) == '%') {
            url[x] = x2c(&url[y+1]);
            y+=2;
        }
    }
    url[x] = '\0';
}

void plustospace(char *str) {
    register int x;

    for(x=0;str[x];x++) if(str[x] == '+') str[x] = ' ';
}

int rind(char *s, char c) {
    register int x;
    for(x=strlen(s) - 1;x != -1; x--)
        if(s[x] == c) return x;
    return -1;
}

int getline(char *s, int n, FILE *f) {
    register int i=0;

    while(1) {
        s[i] = (char)fgetc(f);

        if(s[i] == CR)
            s[i] = fgetc(f);

        if((s[i] == 0x4) || (s[i] == LF) || (i == (n-1))) {
            s[i] = '\0';
            return (feof(f) ? 1 : 0);
        }
        ++i;
    }
}

void send_fd(FILE *f, FILE *fd)
{
    int num_chars=0;
```

```
    char c;

    while (1) {
        c = fgetc(f);
        if(feof(f))
            return;
        fputc(c,fd);
    }
}
```