SAMICOM,Web&Solutions

Curso de capacitación para el diseño y desarrollo de aplicaciones web basadas en Linux

Instructor: Ing. Salvador Mondragón Avilés
Empresa: SAMICOM, Web&Solutions
Impartido a: Congreso Nacional de Software Libre (CONSOL)

Objetivo del curso:

Aprender a diseñar y desarrollar aplicaciones web con interacciones a bases de datos, cubriendo conceptos básicos de HTML, CGI (**C**ommon **G**ateway **I**nterface, por sus siglas en inglés), JavaScript, servidor web apache, base de datos Postgres y comandos de UNIX y sentencias SQL.

TEMARIO:

1. Justificación y utilidad de las aplicaciones Web
2. Linux como plataforma de desarrollo de aplicaciones web
3. Servidor web apache
4. Concepto y ejemplos de CGI
5. Formularios HTML
6. Validación de formularios
7. Recepción, formateo y almacenamiento de datos en el CGI
8. Presentación de datos recibidos
9. Base de datos Postgres
10. Conexión a la base a través de un CGI
11. Rutinas de sentencias SELECT, INSERT, UPDATE y DELETE
12. Integración de temas en una aplicación práctica
13. Bibliografía

Sur 105 No. 343
Col. Héroes de Churubusco C.P. 09090, Tel. 55815126
http://www.samicom.com.mx/site/
e-mail: servicioclientes@samicom.com.mx

Descripción de temas

1. **Justificación y utilidad de las aplicaciones Web**

Hoy en día las aplicaciones web están presentes prácticamente en la mayoría de las actividades de la sociedad y su única finalidad es la de hacer más rápidas y eficientes las tareas a las que se encuentran asignadas (o al menos ese es el propósito).

Es por lo anterior que encontramos una infinidad de aplicaciones en Internet que nos permiten realizar una sinúmero de servicios, que van desde la consulta en línea del correo electrónico hasta el pago de impuestos. Estas aplicaciones evitan que el usuario se traslade, por ejemplo, al banco para realizar en pago de servicios telefónicos, con sólo visitar la página web de la empresa telefónica puede consultar su saldo y también realizar el pago del mismo.

Lo único que limita a una aplicación web es nuestra imaginación, ya que prácticamente se puede pensar en realizar cualquier aplicación, sólo basta tener el interés y conocer las herramientas necesarias para lograrlo. Puesto que lo único que necesitamos para desarrollar una aplicación web es lo siguiente:

1. Un poco de conocimiento de HTML (lenguaje para las páginas web), es importante conocer este lenguaje ya que todas las respuestas y formularios de los CGI están 100% ligados a este lenguaje.
2. Un lenguaje de programación, que puede ser C, PERL, Bourne Shell, Vbscript, etc. El lenguaje que se use debe contar con la facilidades necesarias para realizar las tareas que se necesiten de él. Como programador se debe estar lo suficientemente familiarizado con el lenguaje elegido para trabajar con eficiencia.
3. Acceso a un servidor Web en funcionamiento, debido a que los CGI son una interfaz entre el cliente (navegador) y el servidor y no pueden ser probados si el servidor no ejecuta peticiones Web.

2. Linux como plataforma de desarrollo

Como uno de los requisitos para probar un CGI es tener acceso a un servidor Web, el sistema operativo Linux nos servirá para cumplir dicho requisito.

La idea de utilizar Linux como plataforma de desarrollo obedece al buen desempeño que éste tiene como servidor Web, sumado a su fácil instalación y configuración, además de que con Linux no se necesita al pago de ninguna licencia por su uso ya que forma parte del movimiento de software libre (*para mayor información del uso y desarrollo de software libre puedes visitar: http://www.fsf.org*). La distribución que utilizaremos para este curso es la de Linux Red Hat 7.1 (*http://www.redhat.com*), que como todas las distribuciones de Linux cuenta con un servidor Web, bases de datos, servidor de correo electrónico, DNS, entre otros servicios.

Otra ventaja que nos presenta Linux es que también corre en arquitecturas Intel, lo que nos permite instalarlo en un PC con un mínimo de recursos de hardware, permitiéndonos así tener en una pequeña máquina todo un servidor para el desarrollo de aplicaciones web.

Requisitos mínimos de instalación para Linux Red Hat 7.1

- PC Pentium 133 MHz
- 32 MB de RAM
- HD de 2.0 GB
- CD-ROM
- Tarjeta de red Ethernet de 10 MBPS

Manual de instalación:

Este curso no cumple con el objetivo de instalar Linux en una PC, pero se anexa la liga dierecta al manual de instalación de la versión 7.1 de Red Hat:

http://www.redhat.com/docs/manuals/linux/RHL-7.1-Manual/install-guide/

3. Servidor Web apache

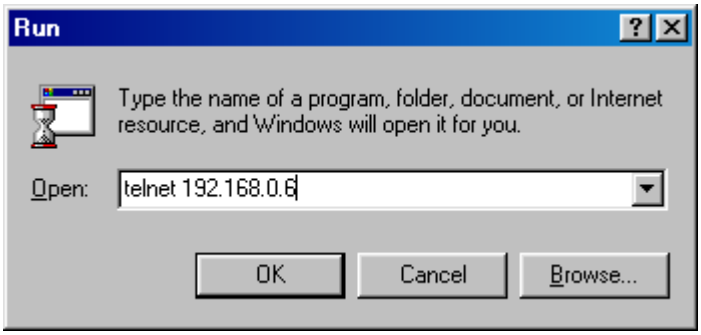
El servidor Web que utiliza Linux por default es apache (http://www.apache.org) ya que también es parte de un proyecto de software libre por lo que es libre de licencias y tiene todas las funcionalidades de software de servidores web como I Planet, Zeus, IIS, entre otros. Con el servidor Web apache podemos ejecutar aplicaciones programadas en distintos lenguajes como PERL, PHP, C, JAVA, entre otros. Otro servicio que ofrece apache como servidor web es el de "Servidor Virtual", este servicio permite tener hospedadas en un mismo servidor Linux distintas páginas web con sus respectivos dominios, es decir pueden estar en Linux páginas como: www.dominio**1**.com, www.dominio**2**.com, www.dominio**3**.com, ..., www.dominio**n**.com. Para el cliente que las visita puede

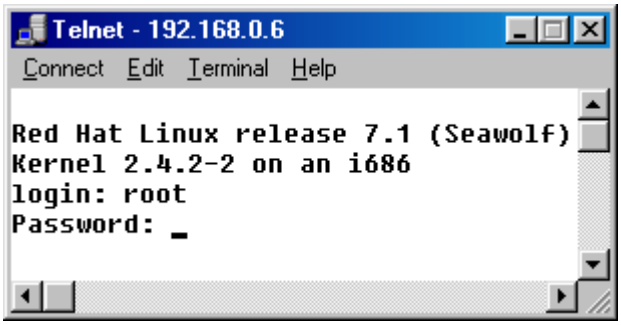
considerar que navega en distintos servidores, sin saber que está navegando en uno sólo.

Lo importante que hay que saber, como desarrollador, de apache es saber cómo ponerlo en servicio y saber dónde se deben de colocar las páginas web que contengan los formularios y dónde colocar los programas CGI para su ejecución, lo cual mencionaremos acontinuación:

Cómo inicio el servicio de apache:

En linux el servicio de apache se llama **httpd** *(hiper text transfer protocol daemon)* y para poder administrar este servicio debemos de entrar al servidor Linux como super usuario, es decir como **root**. Esto puede ser de manera local o remota (vía telnet o ssh, ver figura), ya sea desde una máquina con Windows o bien desde una máquina con Linux

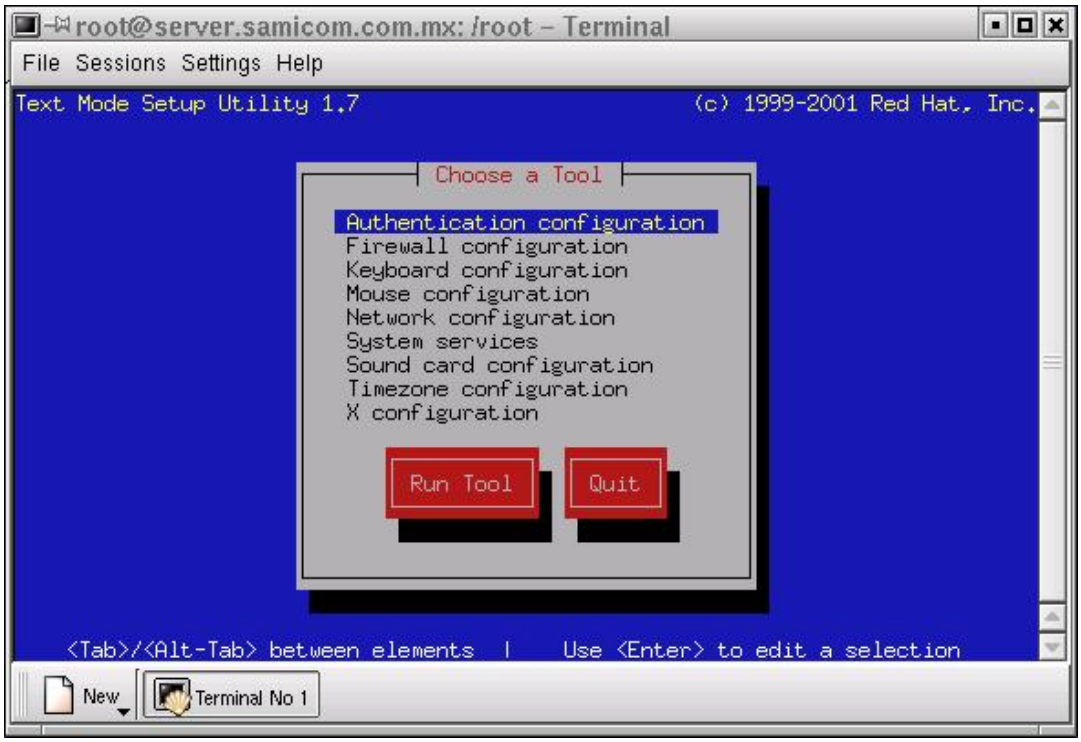




Una vez que se ha entrdado al sistema como superusuario, debe de aparecer un prompt parecido a este: **[root@server /root]#,** hay que verificar si el servicio **httpd** está dado de alta para que cada vez que se reinicie el servidor éste también se inicie. Esto se hace a través del comando **setup** y se ejecuta de la siguiente manera:

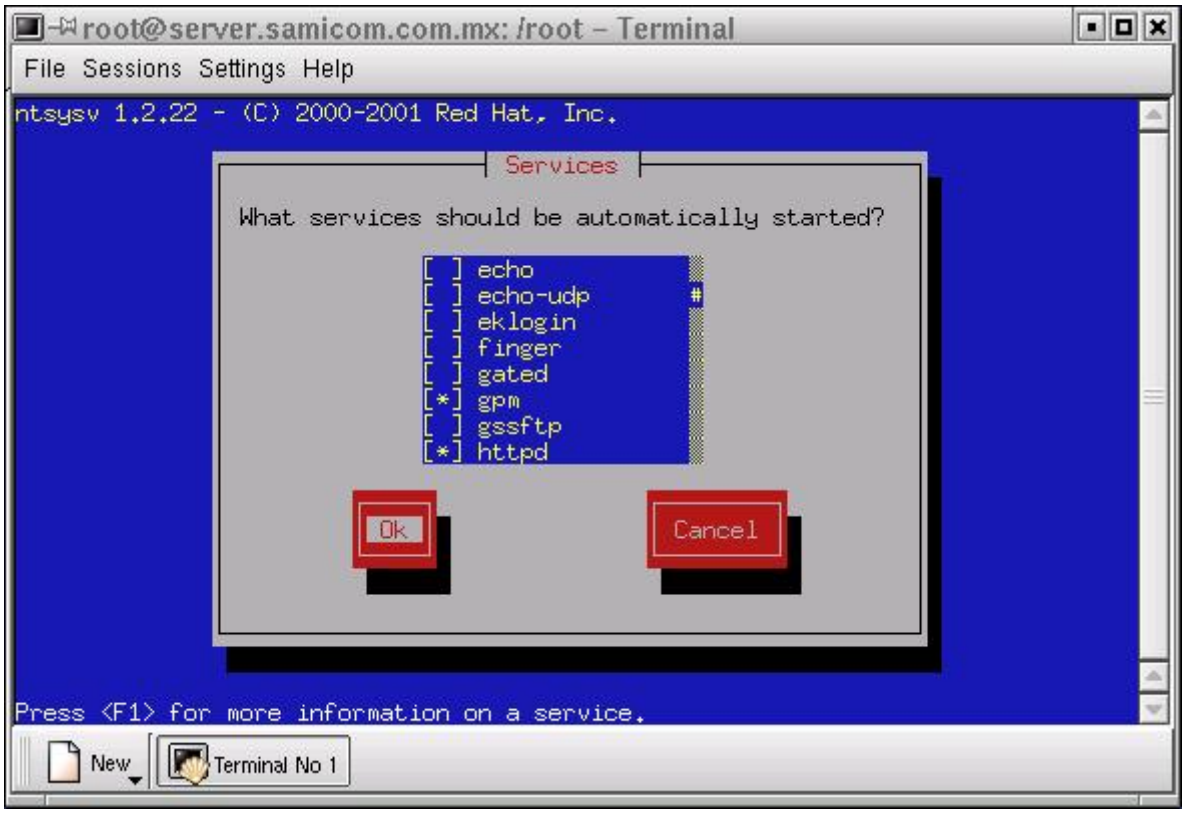
[root@server /root]#setup

Una vez ejecutado este comando, aparecerá una ventana como la siguiente:



Dentro de el menú que aparece, se debe de seleccionar la opción **System services** con la tecla de tab posicionarse en la opción **Run Tool** y teclear enter. Una vez que se hace lo anterior nos aprecerá una ventana donde se nos muestran los distintos servicios que se tiene habilitados para el arranque del sistema. Con la tecla de flecha hacia abajo hay que desplazarce hasta el servicio **httpd** y verificar si está habilitado, si está habilitado aparecerá un asterisco entre los corchetes, en caso contrario, se habilita con la tecla de barra espaciadora. Una vez habilitado con la tecla tab hay que posicionarse en la opcion OK y dar enter, con esto aseguramos que cada vez que se reinicie nuestro servidor se habilitará el servicio httpd. Una vez que damos OK, el sistema nos regresará a la ventana anterior y en ella debemos de posicionarnos en la opción Quit con la tecla de tab y dar enter para salirnos y regresar al prompt de root:

[root@server /root]#



Para evitar reiniciar el servidor para que se inicie el servicio de httpd se puede levantar el servicio de la iguiente manera:

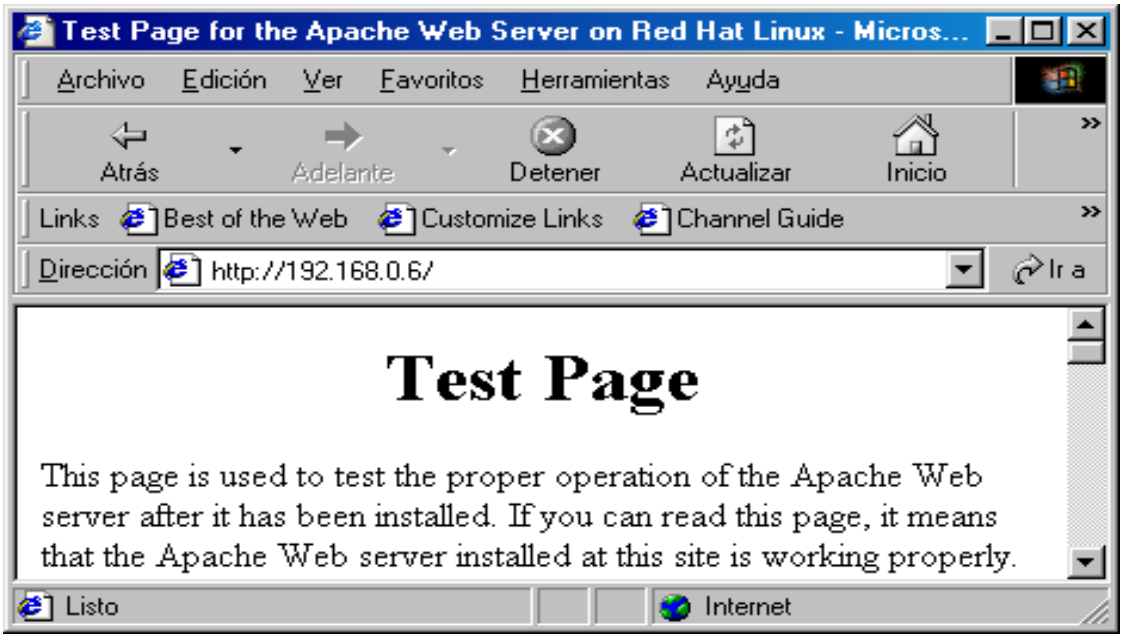
[root@server /root]#/etc/rc.d/init.d/httpd start

Si todo está bien nos aprecerá los siguiente:

Starting httpd: [OK]

[root@server /root]#

Para probar el buen funcionamiento del servidor Web, es posible hacerlo a través de un navegador y tecleando la dirección IP del servidor y si aparece una página como la siguiente toda está correcto y prácticamente estámos listos para empezar con la pruebas de nuestros primeros CGI:



Una vez que el servidor Web está funcionando correctamente estámos listos para iniciar con la prueba de los CGI, pero antes de iniciar con las pruebas es necesario saber algo más sobre el servidor Web apache y es lo siguiente:

Ubicación de archivos html y programas CGI dentro del servidor Web apache

Es claro que tanto la edicón de páginas html y programas CGI se pueden hacer en una máquina distinta a la del servidor, por lo que es necesario saber dónde deben de ser ubicados los archivos de las páginas Web y los archivos de los programas CGI dentro del servidor Web.

Archivos HTML

Dentro del servidor Web los archivos de las páginas Web deberán ser copiados al siguiente directorio:

/var/www/html/

El directorio /var/www/html/ es visible a través del navegador de los clientes que hacen peticiones a nuestro servidor web y todas las páginas web que coloquemos dentro de este directorio serán visbles para los clientes, siempre y cuando los archivos tengan al menos permisos de lectura. Todas las páginas de inicio deberan ser nombradas siempre como **index.html** o **index.htm** y a partir de éstas se derivará toda la navegación del sitio.

Archivos de programas CGI

Los archivos que contengan los programas CGI, deberán ser colocados en el siguiente directorio:

/var/www/cgi-bin/

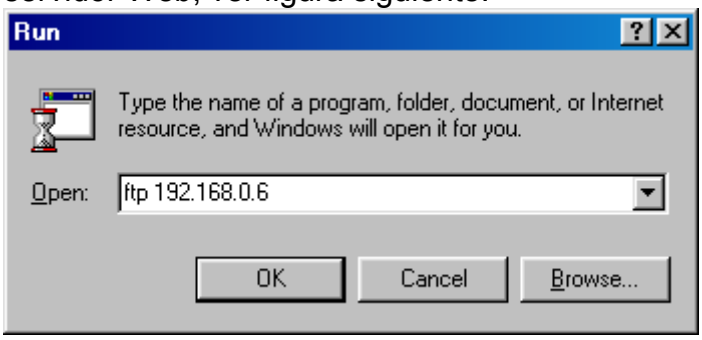
Una vez colocados los programas endicho directorio es necesario que éstos tengan permisos tanto de lectura como de ejecución, de lo contrario el cuando sean llamados el servidor Web no podrá ejecutarlos y nos enviará una página de error.

Hay que recordar que para cambiarle los permisos a cualquier archivo es necesario utilizar el comando **chmod** y con las siguiente combinación:

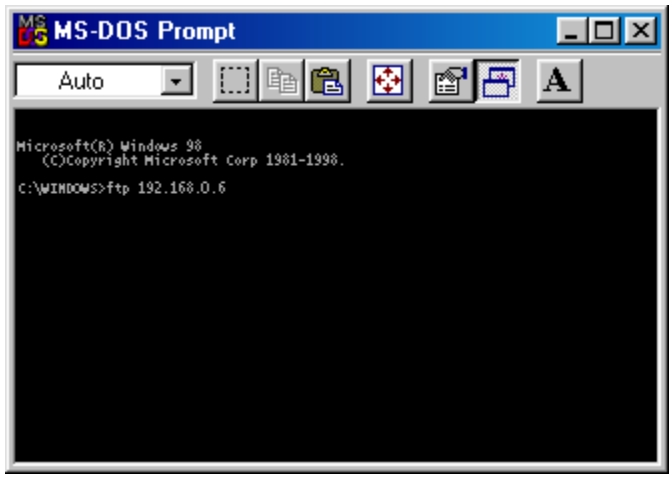
[root@server /root]#chmod 755 *.*

Con esta combinación de números garantizamos que el propietario de los archivos tendrá derecho a leerlos, editarlos y ejecutarlos y los demas usuarios tanto de grupo como finales podrán solamente leerlos y ejecutarlo, jamás editarlos.

Hasta el momento ya sabemos dónde colocar los archivos tanto de páginas Web como de CGI, pero ahora cómo colocarlos si estámos en una máquina remota, la solución es muy simple y es través del servicio de FTP el cual viene de manera nativa en las máquinas con sistema operativo Windows en cualquier versión ya sea 9x/Me/2000/XP y se puede hacer con sólo dar un click en el menú de inicio y después en ejecutar y teclear **ftp** y el **nombre del servidor** o bien la **dirección IP** del servidor Web, ver figura siguiente:

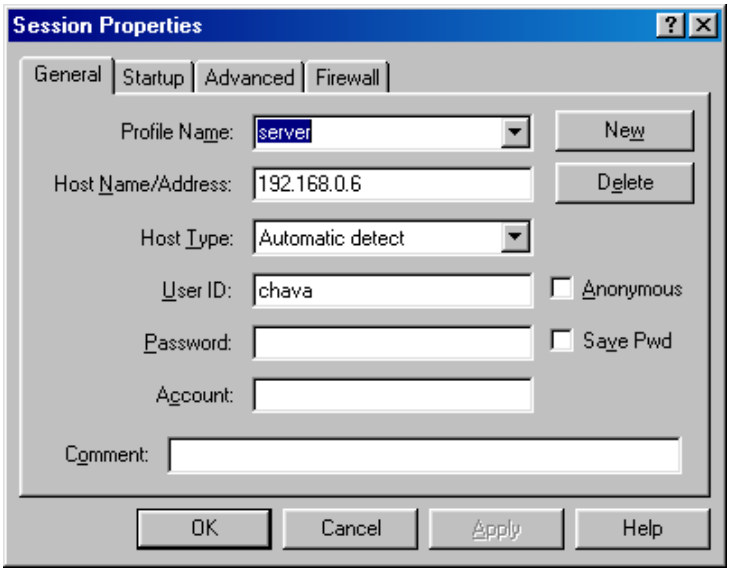


También se puede tener acceso a servicio de ftp desde la consola de MS-DOS y sólo hay que teclear **ftp** y el **nombre del servidor** o bien la **dirección IP** del servidor Web, ver figura siguiente:

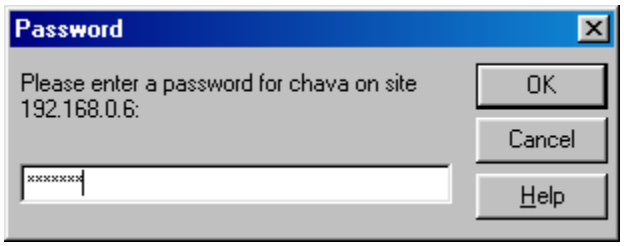


Otra manera de conectarse al servidor es con un programa FTP gráfico, el cual facilita ampliamente las cosas. Un programa de este tipo es el WS_FTP, el cual se puede descargar de manera gratuita del sitio de la empresa que lo creó http://www.ipswitch.com/

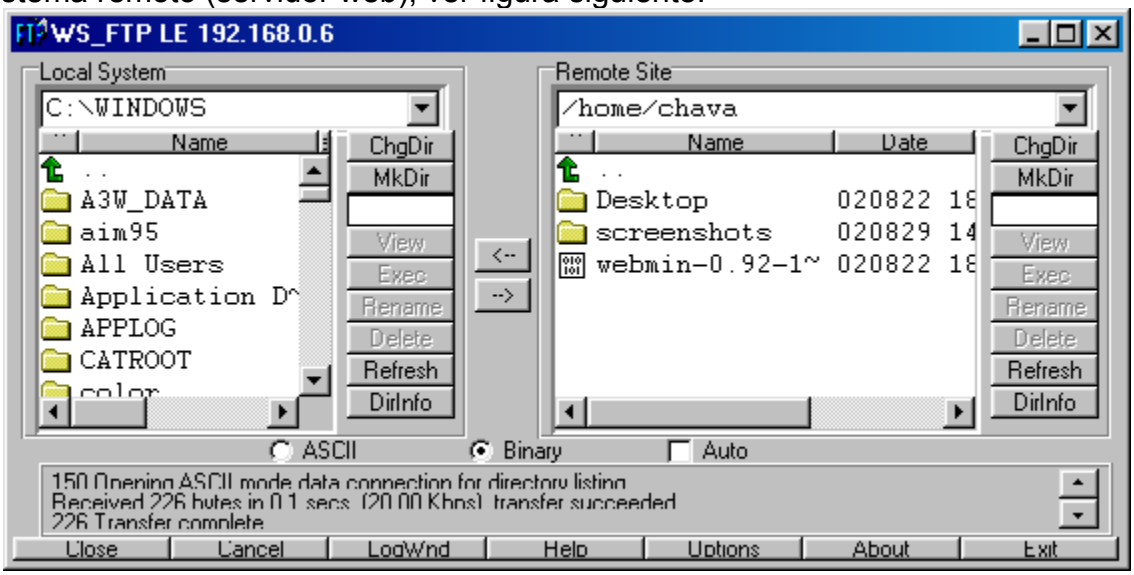
Una vez que se ejecuta el programa, aparece una ventana como la siguiente:



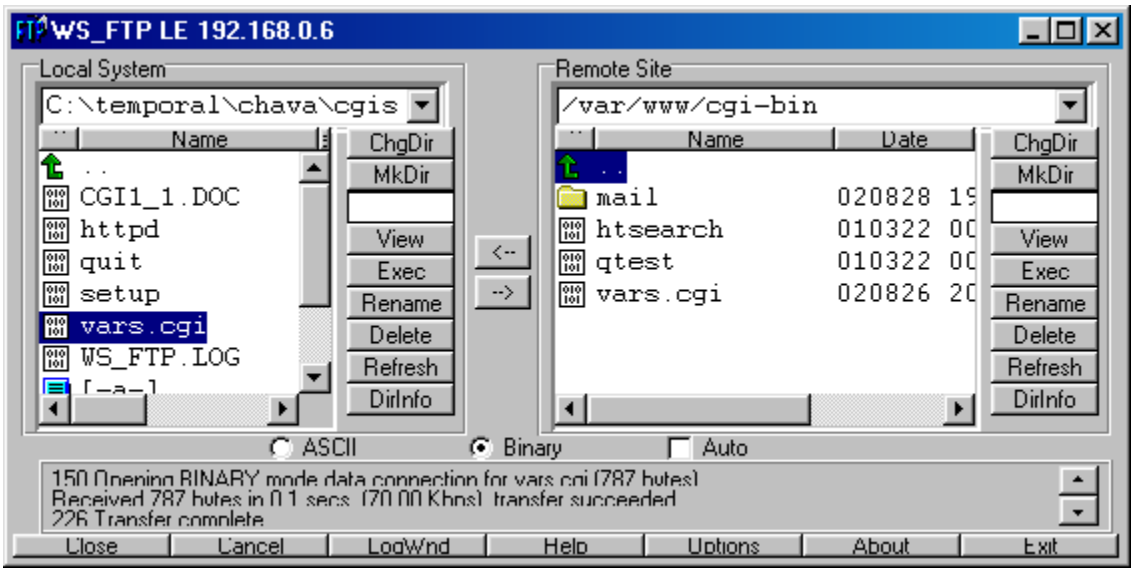
Donde se especifica el nombre o dirección IP del servidor un nombre de usuario, aquí es muy importante que el usuario que se conecta al servidor tenga los permisos necesarios para escribir en los directorios de las páginas Web y de los CGI del servidor, esto se deberá de acordar directamente con el administrador del servidor. Una vez que se proporcionó el usuario y el nombre o dirección IP del servidor, sólo hay que dar un click en el botón **OK** y nos aparecerá una ventana nueva dónde se pedira el password de acceso, ver figura siguiente:



En esta ventana sólo tenemos que teclear el password correcto y dar un click en el botón **OK** y si todo es correcto lograremos conectarnos al servidor y nos aparecerá una nueva ventana donde tendremos el sistema local (nuestra PC) y el sistema remoto (servidor web), ver figura siguiente:



Analizando la figura anterior, vemos que del lado izquierdo tenemos el contenido de nuestra PC y del lado derecho el contenido del directorio del usuario con el que hicimos la conexión al servidor, es decir el contenido del sistema remoto. Lo único que debe de hacerse dentro de esté programa es posicionarse dentro de los directorios correspondientes tanto en el servidor como en nuestra PC y hacemos esto ayudándonos de las las flechas verdes las cuales nos permiten movernos dentro de los sistemas tanto local como remoto y también dando clicks en los directorios a los que queremos entrar para poder llegar a los directorios donde se encuantran los archivos fuentes. Una vez que nos encontramos en los directorios deseados tanto del sistema local como del remoto, ya sólo tenemos que seleccionar el archivo del sistema local para ponerlo en el sistema remoto, una vz seleccionad sólo tenemos que dar un click en el botón que tiene el ícono de la flecha apuntando hacia la derecha, ver figura siguiente.



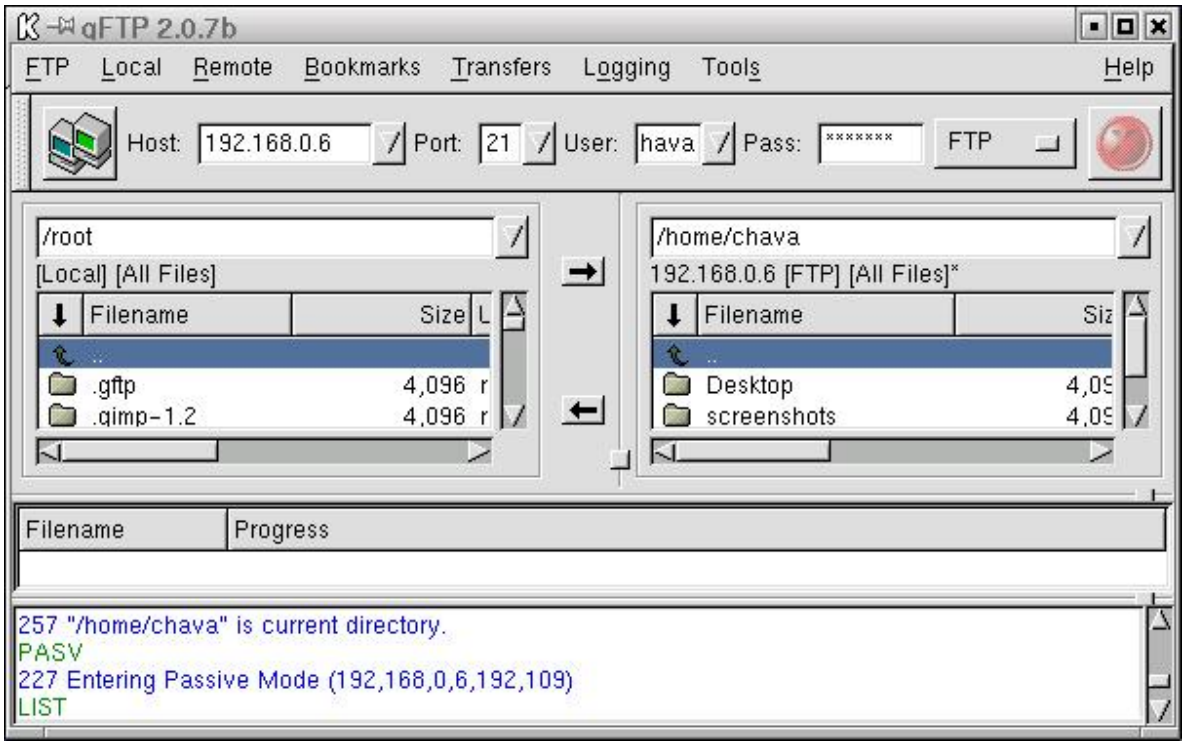
Si se desea enviar información del sistema remoto al local se hace el proceso contrario, se selecciona el o los archivos a transferir y se da un click en el botón con el ícono de flecha apuntando a la izquierda.

Dentro de este programa de FTP gráfico, se pueden crear directorios, borrar y renombrar archivos, cambiar permisos, entre otras cosas. Su utilización es realmente sencilla y la parte delicada de su buen funcionamiento radica en que el usuario con que se hace la conexión tenga los privilegios suficientes para poder transferir, borrar y renombrar los archivos así como cambiar permisos y crear directorios en el servidor Web, pero esto es tarea, como se mencionó anteriormente, del administrador del servidor.

Si se está trabajando en una máquina Linux, el proceso es el mismo, el FTP se puede hacer desde la consola de comandos o bien a través de un programa de FTP gráfico, que en Linux puede ser el programa GFTP y se invoca desde la consola de comandos de la siguiente manera:

[root@server /root]#gftp & (el símbolo & es una opción para que el proceso corra en background y deje libre la consola de comandos para que se puedan ejecutar otras tareas).

El manejo de GFTP es muy similar al de WS_FTP y si se ve la siguiente figura se darán cuenta de que es cierto.



Como se puede observar en la figura anterior, existen ambas ventanas que corresponden al sistema local y al sistema remoto, las mismas flechas para transferir los archivos, en fin su manejo no debe de ser ningún problema.

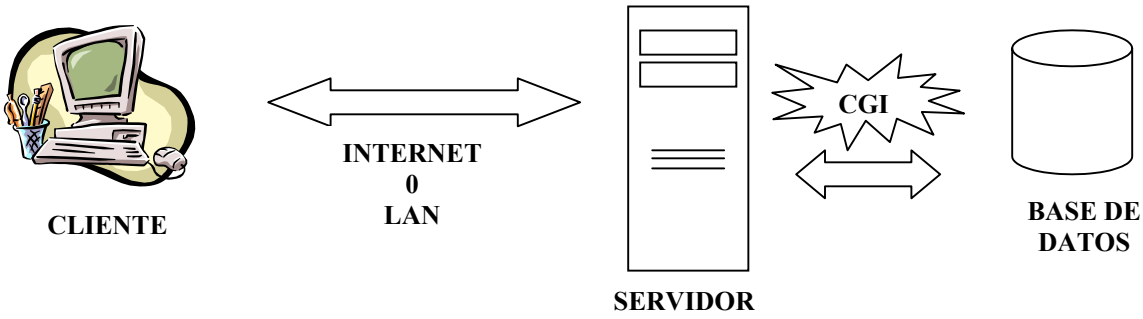
Ahora sí ya estámos listos para iniciar con el concepto de CGI y ver algunos jemplos básicos de los mismos.

4. Concepto y ejemplos de CGI

La Interfaz Común de Puerta de Enlace (CGI por sus siglas en inglés, Common Gateway Interface) en realidad no es un lenguaje o protocolo en el sentido más estricto de estor términos; en realidad sólo es un conjunto de variables y convenciones, de denominación común, para pasar información de ida y vuelta al servidor web. Cuando se le ve bajo esta perspectiva, CGI se vuelve mucho menos intimidante.

CGI es la interfaz entre el navegador Web y los demás recursos del servidor, por ejemplo, una base de datos.

Gráficamente podemos ver el concepto CGI de la siguiente manera:



En la figura anterior podemos ver los componentes básicos de una aplicación Web, donde el **cliente** es cualquier computadora conectada al servidor ya sea vía Internet o bien que el servidor se encuentre en la misma Intranet que el cliente y que tenga instalado un navegador de Internet, por ejemplo Internet Explorer o Netscape, por mencionar a los dos más conocidos.

El **servidor** es una computadora con un sistema operativo para redes y con un software de servidor Web instalado y configurado correctamente, en nuestro caso el sistema operativo es Linux Red Hat 7.1 y el servidor Web es Apache.

Los programas **CGI** nos ayudan a interconectar los recursos del servidor con los clientes, por ejemplo una base de datos, los servicios para enviar correos, una cámara Web, archivos de imágenes y audio, etc. Los programas CGI pueden estar hechos en cualquier lenguaje de programación, siempre y cuando el servidor donde van a ejecutarse soporte el lenguaje y nos presten todas la facilidades para realizar las tareas que necesitemos de ellos. Pueden estar escritos Bourne Shell, PERL, JAVA, VBScript, C, etc. En nuestro caso utilizaremos el lenguaje C para su desarrollo ya que es soportado por el servidor Linux Red Hat sin ning{un problema y su código puede ser fácilmente migrado a otros servidores con distinto sistema operativo y con un compilador para lenguaje C.

Los programas CGI

Un programa CGI tiene tres funciones básicas de entrada/salida, como sigue:

1. Recabar la entrada del servidor, en forma de variables estandarizadas, datos de formulario y datos de consulta.

2. Proporcionar datos de salida al cliente (navegador de Internet)
3. Proporcionar información de negociación del (el encabezado MIME para el servidor y el cliente)

Para explicar los puntos anteriores, analicemos el código de un programa CGImuy simple, pero muy descriptivo.

La forma más rápida de iniciarse en los programas CGI, como en cualquier otro ambiente de programación, es escribiendo un programa pequeño y sencillo y el más común de estos programas de inicio es el que escbibe la cadena “Hola Mundo”, lo que conlleva imprimir esta cadena en el navegador del cliente a manera de página Web, entoces vamos a escribir nuestro primer programa CGI en lenguaje C.

Código del CGI “Hola Mundo”

```
/*Nombre del archivo: hola.c*/

#include <stdio.h>  /*biblioteca de entrada salida*/
main ()
{
printf(“Content-type: text/html\n\n”); /* Encabezado de tipo mime*/
printf(“<HTML><HEAD><TITLE>CGI Hola Mundo</TITLE></HEAD>\n”); /* Título de la página*/
printf(“<BODY><p>Hola Mundo cómo estás?</BODY></HTML>”); /*se imprime cadena*/
}
```

Anatomía del programa CGI:

Independientemente del lenguaje en que se escriba el CGI, siempre hay que indicarle al navegador de Internet de qué tipo de datos se trata y esto se hace con la línea:

Content-type: MIME/type

El Content-type, especifíca el tipo de contenido que se va a enviar al cliente. La mayoría de las veces será “text/html” para los documentos HTML (páginas Web) y “text/plain” para documentos informativos. Sin esta línea el programa CGI simplemente no funcionaría ya que el navegador no sabría cómo interpretar la información recibida y enviaría el servidor un mensaje de error. El encabezado Content-type: text/html siempre deberá de ir seguido por dos líneas nuevas de quí que aparezca dos veces la sentencia de escape “\n”.

La siguiente línea:

<HTML><HEAD><TITLE>CGI Hola Mundo</TITLE></HEAD>

establece el título de la página Web que se muestra al usuario.

La línea:

<BODY><p>Hola Mundo cómo estás?</BODY></HTML>

imprime el cuerpo del documento HTML, como sólo se imprime una cadena de texto en esta misma línea empieza y termina el cuerpo de la página web.

Probando el primer programa CGI:

Una vez que tenemos escrito el código del programa CGI, debemos compilarlo, para esto hay que tener el código, bien en el servidor donde va a ejecutarse el CGI o en una máquina con el mismo sistema operativo, esto con el fin de que al compilarlo se genere un ejecutable capaz de ser interpretado por el servidor donde se aloje.

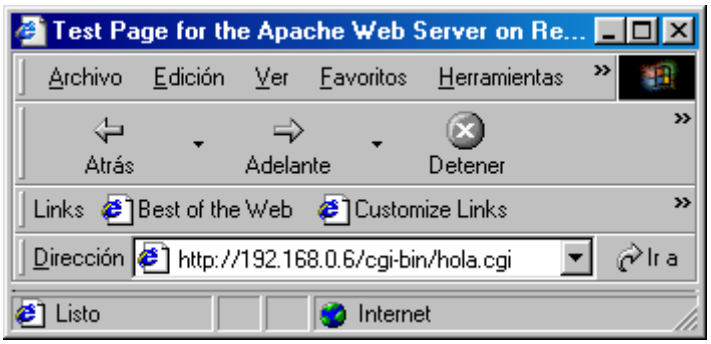
Una vez que se tiene el código en la máquina donde va a compilarse, es necesario abrir una consola de comandos y correr el siguiente comando:

[root@server /root]#make hola

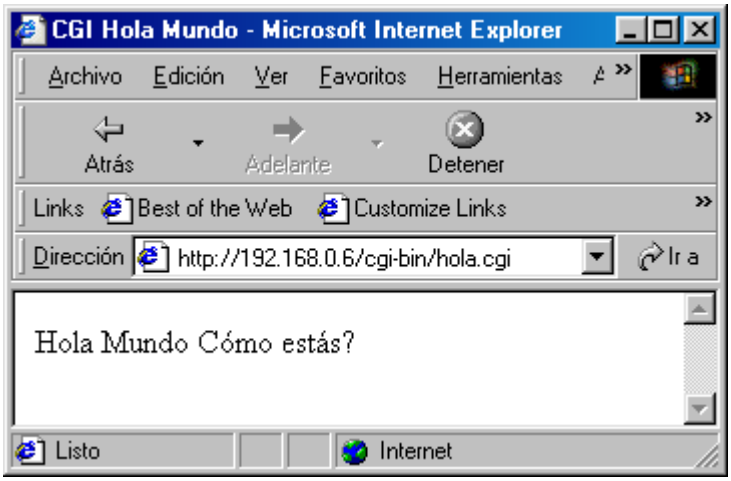
es necesario que nos encontremos dentro del directorio donde se encuentra el código a la hora de compilarlo, de lo contrario no se encontrará el codigo fuente y no podrá crearse el programa ejecutable. Si el código del programa hola.c está correctamente escrito, al finalizar la compilación se creará un archivo llamado **hola,** que es el archivo ejecutable que contiene al CGI, lo que prosigue es copiar el archivo **hola** al directorio cgi-bin del servidor Web con el nombre de **hola.cgi**, normalmente la extensión **.cgi** es utilizada para los ejecutables de los programas CGI, para copiar el archivo **hola** se puede correr el siguiente comando:

[root@server /root]#cp hola /var/www/cgi-bin/hola.cgi

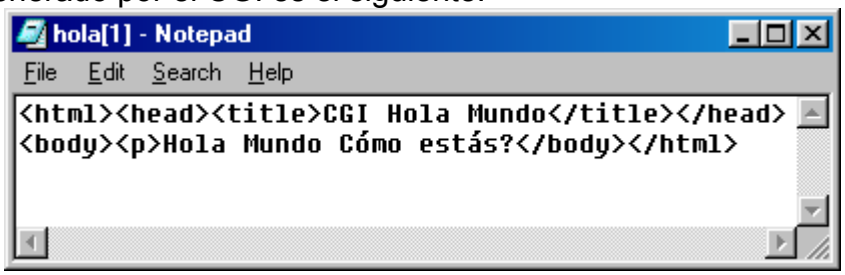
Ahora para probar su buen funcionamiento necesitamos abrir el navegador de Internet de nuestra preferencia y teclear la ruta del CGI en el servidor, por ejemplo, ver la siguiente figura:



Si el CGI no tiene ningún problema de compilación ni de programación, deberá mostrar la siguiente página Web:



El código generado por el CGI es el siguiente:



Para ver el código generado por el CGI basta con posicionarse con el cursor del mouse sobre la página Web y dar un clic con el botón derecho y seleccionar la opción **Ver código fuente** y aparecerá dicho código.

Las Variables CGI estándar

En los programas CGI, algunos datos de entrada se pasan en *variables de ambiente*, son datos importantes sobre información de los clientes, del servidor, de los mismos programas CGI, así como de otros factores. Cuando un programa CGI es llamado ya están establecidas numerosas variables para darnos información acerca del usuario, de la configuración del software de éste y del ambiente del servidor.

Vamos a ver a continuación la información de los clientes:

Acerca de los clientes:

Algunas de las variables CGI estándar contienen información acerca del usuario y su ambiente, las más útiles son: HTTP_USER_AGENT, HTTP_ACCEPT, REMOTE_ADDR y REMOTE_HOST. Veamos a continuación qué información nos proporciona cada una de estas variables:

HTTP_USER_AGENT:

Esta variable contiene el nombre y la versión del navegador del cliente, en el formato "*nombre/versión biblioteca/versión*". También contiene informe de cualquier proxy por el cual pueda estar pasando el cliente.

HTTP_ACCEPT:

Esta variable proporciona los formatos MIME que puede aceptar el navegador. El formato de esta variable es tipo/subtipo, tipo/subtipo, ...

REMOTE_ADDR:

Contiene la dirección IP en notación decimal punteada (xxx.xxx.xxx.xxx)

REMOTE_HOST:

Contiene el nombre en texto del host equivalente a la dirección numérica. Normalmente esta variable viene vacía debido a que para obtener el nombre del host hay que recurrir al servicio de DNS con resolución inversa que requiere de tiempo y consumo de ancho de banda de la red y por esto la mayoría de los servidores DNS deshabilitan este servicio.

Acerca del servidor

Las siguientes variables proporcionan información acerca del servidor y del software que éste ejecuta:

SERVER_SOFTWARE:

Contiene el nombre y la versión del software del servidor , en el formato *nombre/versión*.

SEVER_NAME:

Contiene el nombre del servidor

GATEWAY_INTERFACE:

Contiene las versiones de la especificación CGI que utiliza el servidor, en el formato *CGI/versión*.

Variables específicas de solicitud

Las siguientes variables son específicas de solicitud, ya que cambian según la solicitud específica que se envíe.

QUERY_STRING:

Es quizá la más importante de estas variables ya que forma parte del método más común de pasar información a un programa CGI.

Comúnmente, la solicitud de un programa CGI se ha ce mediante un signo de interrogación, "?", seguido de la información adicional en el URL, por ejemplo, si se envía la siguiente información a través del URL http://192.168.0.6/cgi-bin/vars.cgi?nombre=Salvador&apellido_paterno=Mondragon, todos los caracteres que van después del signo de interrogación serán colocados en la variable QUERY_STRING.

SCRIPT_NAME:

En esta variable se establece el nombre del archivo del programa CGI.

SERVER_PROTOCOL:

Esta variable contiene el nombre y número de revisión del protocolo del cual viene la solicitud. Está en el formato *protocolo/revisión*.

SERVER_PORT:

En esta variable se especifica el número de puerto al cual llegó la solicitud. Normalmente el puerto para atender peticiones de HTTP es número “80”.

CONTENT_TYPE:

Esta variable se llena para consultas que tienen información adjunta, como las solicitudes POST. Es el tipo de contenido MIME de los datos en la forma *tipo/subtipo*.

CONTENT_LENGHT:

Es el número de bytes de datos recibidos, sólo se llena esta variable cuando los datos son enviados al CGI por el método POST.

REQUEST_METHOD:

Esta variable contiene el nombre del método utilizado para la solicitud al programa CGI. Normalmente es POST o GET

Programa CGI de ejemplo de variables estándar.

A continuación se presenta el código de un programa CGI que imprime las variables estándar:

```
/*Nombre del archivo: vars.c*/
#include<stdio.h>
#include <stdlib.h>
main()
{
printf("Content-type: text/html\n\n");
printf("<html><head><title>variables CGI Estándar</title></head>\n");
printf("<body>\n");
printf("<p><b>Variables CGI Estándar</b>\n");
printf("<br><p><b>Acerca del Usuario</b>\n");
printf("<br>HTTP_USER_AGENT = %s\n", getenv("HTTP_USER_AGENT"));
printf("<br>HTTP_ACCEPT = %s\n", getenv("HTTP_ACCEPT"));
printf("<br>REMOTE_HOST = %s\n", getenv("REMOTE_HOST"));
printf("<br>REMOTE_ADDR = %s\n", getenv("REMOTE_ADDR"));
printf("<br><p><b>Acerca del Servidor</b>\n");
printf("<br>SERVER_NAME = %s\n", getenv("SERVER_NAME"));
```

```
printf("<br>SERVER_SOFTWARE = %s\n", getenv("SERVER_SOFTWARE"));
printf("<br>GATEWAY_INTERFACE = %s\n", getenv("GATEWAY_INTERFACE"));
printf("<br><p><b>Variables específicas de solicitud</b>\n");
printf("<br>QUERY_STRING = %s\n", getenv("QUERY_STRING"));
printf("<br>SCRIPT_NAME = %s\n", getenv("SCRIPT_NAME"));
printf("<br>SERVER_PROTOCOL = %s\n", getenv("SERVER_PROTOCOL"));
printf("<br>SERVER_PORT = %s\n", getenv("SERVER_PORT"));
printf("<br>CONTENT_TYPE = %s\n", getenv("CONTENT_TYPE"));
printf("<br>CONTENT_LENGTH = %s\n", getenv("CONTENT_LENGTH"));
printf("<br>REQUEST_METHOD = %s\n", getenv("REQUEST_METHOD"));
printf("<br>REMOTE_USER = %s\n", getenv("REMOTE_USER"));
printf("<br>REMOTE_IDENT = %s\n", getenv("REMOTE_IDENT"));
 printf("</body></html>\n");
} /**Termina main***/
```

Como puede verse en el código la función getenv() nos ayuda a recuperar las variables de ambiente y cuando alguna de éstas no existe getenv() nos devuelve un apuntador nulo.

Sólo resta compilar este código y ponerlo en el directorio cgi-bin del servidor , ejecutarlo desde un navegador y analizar sus resultados.

Imágenes de resultados del CGI de variables estándar:

Con Internet Explorer:

Variables CGI Estándar

Acerca del Usuario
HTTP_USER_AGENT = Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, application/x-comet, */*
REMOTE_HOST = (null)
REMOTE_ADDR = 192.168.0.2

Acerca del Servidor
SERVER_NAME = server.samicom.com.mx
SERVER_SOFTWARE = Apache/1.3.19 (Unix) (Red-Hat/Linux) mod_ssl/2.8.1

OpenSSL/0.9.6 DAV/1.0.2 PHP/4.0.4pl1 mod_perl/1.24_01
GATEWAY_INTERFACE = CGI/1.1

Variables específicas de solicitud
QUERY_STRING =
SCRIPT_NAME = /cgi-bin/vars.cgi
SERVER_PROTOCOL = HTTP/1.1
SERVER_PORT = 80
CONTENT_TYPE = (null)
CONTENT_LENGTH = (null)
REQUEST_METHOD = GET
REMOTE_USER = (null)
REMOTE_IDENT = (null

Con Netscape:

Sur 105 No. 343
Col. Héroes de Churubusco C.P. 09090, Tel. 55815126
http://www.samicom.com.mx/site/
e-mail: servicioclientes@samicom.com.mx

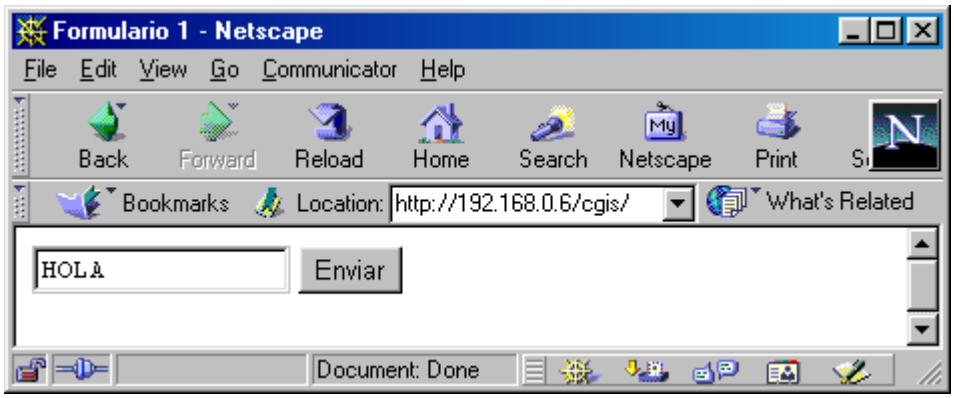
Variables CGI Estándar

Acerca del Usuario
HTTP_USER_AGENT = Mozilla/4.7 [en] (Win98; I)
HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
REMOTE_HOST = (null)
REMOTE_ADDR = 192.168.0.2

Acerca del Servidor
SERVER_NAME = server.samicom.com.mx
SERVER_SOFTWARE = Apache/1.3.19 (Unix) (Red-Hat/Linux) mod_ssl/2.8.1 OpenSSL/0.9.6 DAV/1.0.2 PHP/4.0.4pl1
mod_perl/1.24_01
GATEWAY_INTERFACE = CGI/1.1

Variables específicas de solicitud
QUERY_STRING =
SCRIPT_NAME = /cgi-bin/vars.cgi
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 80
CONTENT_TYPE = (null)
CONTENT_LENGTH = (null)
REQUEST_METHOD = GET
REMOTE_USER = (null)
REMOTE_IDENT = (null)

Llamando al CGI desde un formulario:



Resultado:

Variables CGI Estándar

Acerca del Usuario
HTTP_USER_AGENT = Mozilla/4.7 [en] (Win98; I)
HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
REMOTE_HOST = (null)
REMOTE_ADDR = 192.168.0.2

Acerca del Servidor
SERVER_NAME = server.samicom.com.mx
SERVER_SOFTWARE = Apache/1.3.19 (Unix) (Red-Hat/Linux) mod_ssl/2.8.1 OpenSSL/0.9.6 DAV/1.0.2 PHP/4.0.4pl1
mod_perl/1.24_01
GATEWAY_INTERFACE = CGI/1.1

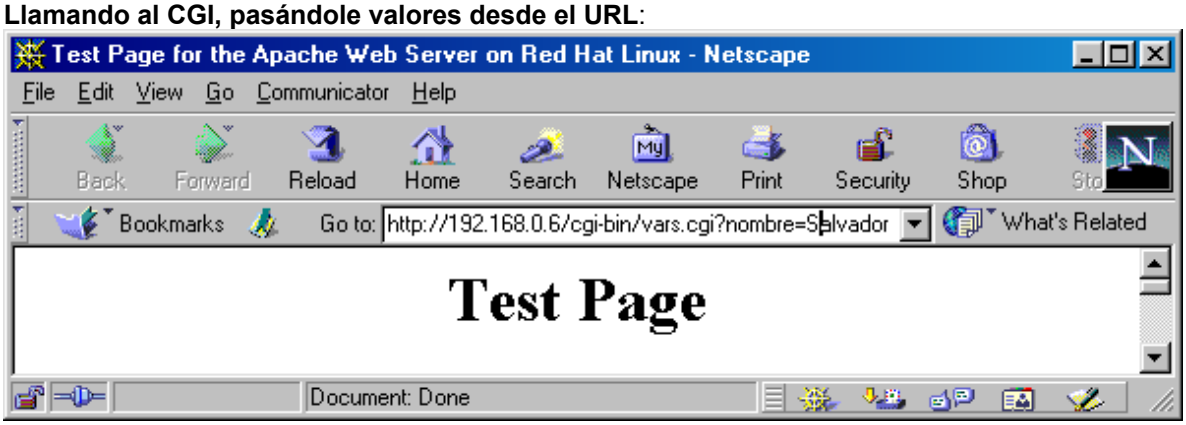
Variables específicas de solicitud
QUERY_STRING =
SCRIPT_NAME = /cgi-bin/vars.cgi
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 80
CONTENT_TYPE = application/x-www-form-urlencoded
CONTENT_LENGTH = 24
REQUEST_METHOD = POST
REMOTE_USER = (null)
REMOTE_IDENT = (null)

Código del formulario:

```
<HTML>
```

```
<HEAD>
<TITLE>Formulario 1</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<form name="variables" action="/cgi-bin/vars.cgi" method="POST">
<input type="TEXT" name="cadena" value="" size="15">
<input type="submit" name="boton" value="Enviar">
</form>
</BODY>
</HTML>
```

Llamando al CGI, pasándole valores desde el URL:



Resultados:

Variables CGI Estándar

Acerca del Usuario
HTTP_USER_AGENT = Mozilla/4.7 [en] (Win98; I)
HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
REMOTE_HOST = (null)
REMOTE_ADDR = 192.168.0.2

Acerca del Servidor
SERVER_NAME = server.samicom.com.mx
SERVER_SOFTWARE = Apache/1.3.19 (Unix) (Red-Hat/Linux) mod_ssl/2.8.1 OpenSSL/0.9.6 DAV/1.0.2 PHP/4.0.4pl1
mod_perl/1.24_01
GATEWAY_INTERFACE = CGI/1.1

Variables específicas de solicitud

```
QUERY_STRING = nombre=Salvador
SCRIPT_NAME = /cgi-bin/vars.cgi
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 80
CONTENT_TYPE = (null)
CONTENT_LENGTH = (null)
REQUEST_METHOD = GET
REMOTE_USER = (null)
REMOTE_IDENT = (null)
```

Hasta este punto hemos visto los conceptos básicos necesarios para que los datos entren y salgan de los programas CGI. Ya entendemos cómo obtener información del servidor y del cliente y también sabemos qué es lo que significan las variables estándar.

En el siguiente apartado vamos a ver el “procesamiento de formularios” cuyo objetivo básico es el saber cómo obtener información de los formularios HTML, que es la forma más común de pasarle datos a un programa CGI.

5. Formularios HTML

La característica básica de los *formularios HTML*, es la de que son la forma más común de llevar datos del usuario a un programa CGI. Ofrecen la posibilidad

de entrada del usuario mediante diversos tipos de elementos como cuadros de texto, botones, listas, entre otros.

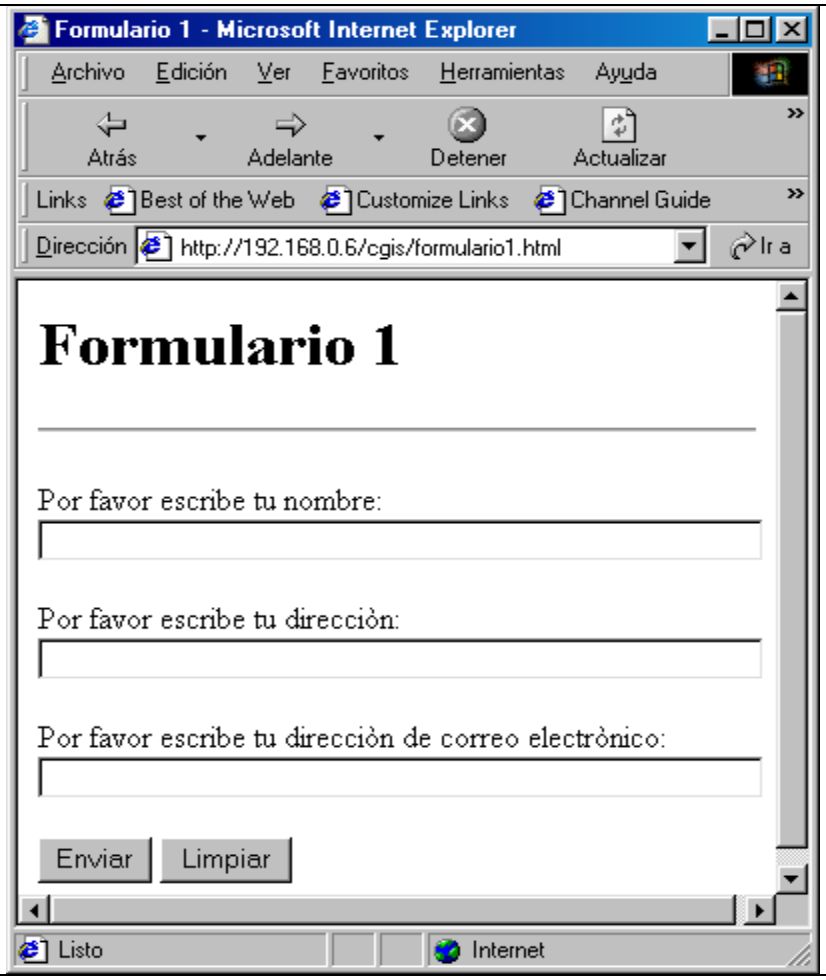
Los formularios pueden utilizarse para recabar información como nombres, direcciones, para realizar encuestas, sondeos, concursos y prácticamente cualquier cosa en la que se pueda pensar, por lo que nuestra imaginación es el único limite para desarrollar aplicaciones Web.

Ahora lo que necesitamos es un formulario muy básico que nos permita recabar los siguientes datos del usuario: nombre, dirección y correo electrónico y además que invoque al CGI que muestra las variables estándar (vars.cgi), aquí va el código HTML del primer formulario que utilizaremos.

Nombre del formulario: datos
Nombre del archivo: formulario.html

```
<head>
<title>Formulario 1</title>
</head>
<body><html>
<H1>Formulaio 1</H1>
<HR>
<FORM NAME="datos" METHOD="GET" ACTION="/cgi-bin/vars.cgi">
Por favor escribe tu nombre:<BR>
<INPUT TYPE="text" SIZE=50 NAME="Nombre" VALUE="">
<P>
Por favor escribe tu dirección:<BR>
<INPUT TYPE="text" SIZE=50 NAME="direccion" VALUE="">
Por favor escribe tu dirección de correo electrónico:<BR>
<INPUT TYPE="text" SIZE=50 NAME="e-mail" VALUE="">
<P>
<INPUT TYPE="submit" VALUE="Enviar">
<INPUT TYPE="reset" VALUE="Limpiar">
</FORM>
</body>
</html>
```

Vista como página Web:



Resultados:

Variables específicas de solicitud

QUERY_STRING
=Nombre=Salvador+Mondragon+Aviles&direccion=Sur+105+%23343&e-mail=salvador@samicom.com.mx
SCRIPT_NAME = /cgi-bin/vars.cgi
SERVER_PROTOCOL = HTTP/1.1
SERVER_PORT = 80
CONTENT_TYPE = (null)
CONTENT_LENGTH = (null)
REQUEST_METHOD = GET
REMOTE_USER = (null)
REMOTE_IDENT = (null)

De acuerdo a los datos resaltados en el código del formularios podemos comentar lo siguiente:

Etiquetas <FORM> y </FORM>

Estas etiquetas indican al navegador que dentro de ellas se encuentra un formulario, en algunos navegadores como Netscape la omisión de alguna de estas etiquetas provoca que el formulario no sea mostrado en la pantalla.

Todo formulario debe de contener un nombre de aquí el atributo de **NAME** donde entre comillas se da un nombre al formulario, esto es útil cuando después se utiliza JavaScript para validarlo y el nombre sirve como referencia en caso de haya más de un formulario en la página Web.

Otros dos argumentos muy importantes de un formulario son **ACTION** y **METHOD**, en el primero indicamos qué programa CGI (dándole la ruta de su ubicación) se va a encargar de manipular la información del formulario y en el segundo le indicamos al formulario la forma en que serán enviados los datos al servidor, existen dos formas **GET** y **POST**, en este ejemplo utilizamos GET, donde los datos se reciben la variable QUERY_STRING.

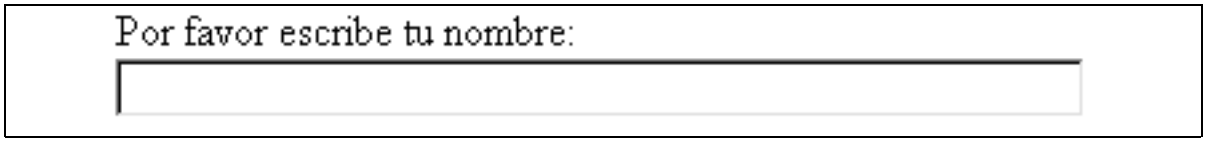
Etiqueta <INPUT>:

Mediante esta etiqueta son introducidos los datos al formulario y todas contienen atributos básicos como son tipo, nombre y valor (TYPE, NAME y VALUE), además de estos atributos pueden contener otros como tamaño, valores por default, entre otros.

Los tipos de entrada pueden ser por:

Caja de texto : Es la entrada más común para obtener datos por medio del teclado

<INPUT TYPE="**text**" SIZE=50 NAME="direccion" VALUE="">



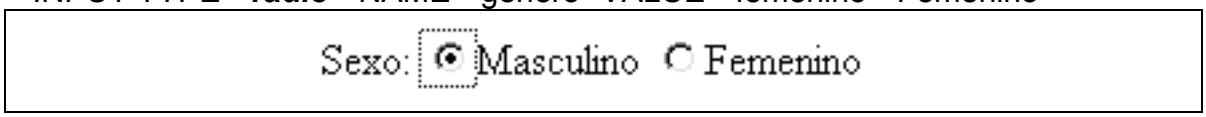
Tipo password, es igual al de texto, sólo que en este los datos escritos en esta caja no son visibles y aparecen normalmente asteriscos cuando escribimos en ellos:

<INPUT TYPE="**password**" SIZE=50 NAME="clave" VALUE="">

Tipo radio: este es un circulo vacío que al hacer click sobre él se rellena con un punto negro, nos sirve para hacer elecciones entre distintas opciones.

<INPUT TYPE="**radio**" NAME="genero" VALUE="masculino">Masculino

<INPUT TYPE="**radio**" NAME="genero" VALUE="femenino">Femenino



Tipo checkbox: este es un cuadrado vacío que al dar click sobre él se palomea, es útil para marcar más de una opción:

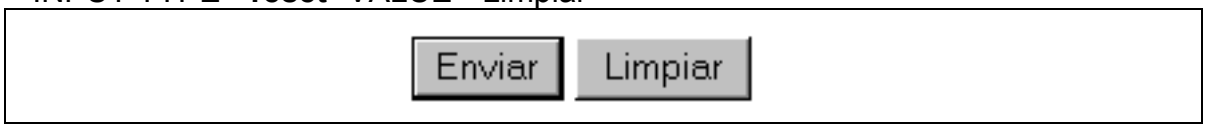
<INPUT TYPE="**checkbox**" NAME="libro1" VALUE="novela">Novelas
<INPUT TYPE="**checkbox**" NAME="libro2" VALUE="cuento">Cuentos
<INPUT TYPE="**checkbox**" NAME="libro3" VALUE="biografia">Biografías
<INPUT TYPE="**checkbox**" NAME="libro4" VALUE="comics">Comics

Tipo hidden: Este tipo como su nombre lo dice es oculto y sirve sólo al programador para ir pasando datos de un formulario a otro o bien datos del sistema o del navegador.

<INPUT TYPE="**hidden**" NAME="case" VALUE="A">

Tipo submit y reset: estos tipos aparecen como botones y el primero es para enviar los datos al CGI y el segundo para limpiar los datos que se hayan ingresado y volver a ingresar otros.

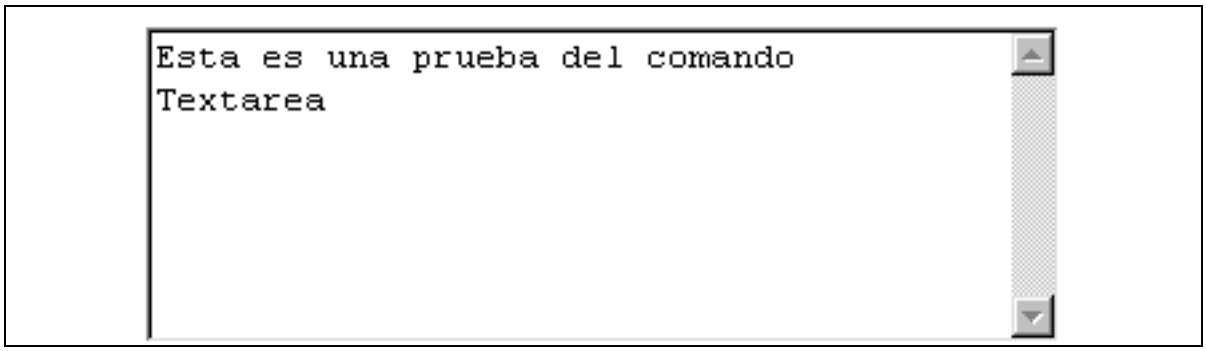
<INPUT TYPE="**submit**" value="Enviar">
<INPUT TYPE="**reset**" VALUE="Limpiar">



Existen otros comandos para ingresar información en un formulario, por ejemplo cajas de texto donde se puede escribir más de una línea, listas de donde se puede elegir algún valor en especial.

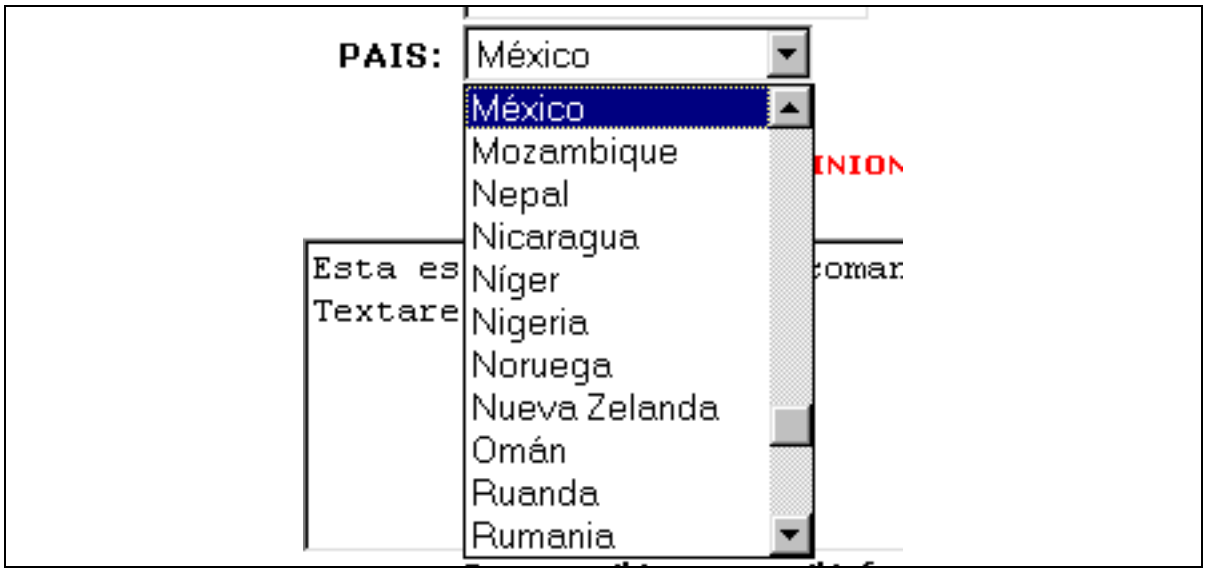
Comando TEXTAREA: se utiliza para introducir texto de más de una línea y sus atributos como en los anteriores comandos son nombre, valor y se le agregan unos extras que son los de columnas y filas los que determinan el tamaño del área:

<**TEXTAREA** NAME="opinion" rows="7" cols="40">
</**TEXTAREA**>



Comando SELECT: este comando se utiliza para incluir una lista extensible, tiene atributos de nombre, valor y opción, se puede elegir un valor por defecto agregando la opción selected.

```
<select name="pais">
<option value="">(Indicar)
<option value="México" selected>México
<option value="Mozambique" >Mozambique
<option value="Nepal" >Nepal
<option value="Nicaragua" >Nicaragua
<option value="Níger" >Níger
<option value="Nigeria" >Nigeria
<option value="Noruega" >Noruega
<option value="Nueva Zelanda" >Nueva Zelanda
<option value="Omán" >Omán
<option value="Ruanda" >Ruanda
<option value="Rumania" >Rumania
<option value ="Otro país" >Otro país
</select>
```



Básicamente con la combinación de los comandos INPUT (y sus distintos tipos), SELECT y TEXTAREA, podemos crear cualquier variedad de formularios desde los más sencillos hasta los más complicados y completos.

A continuación veremos un aspecto muy importante de la validación de los datos que se envían a través de los formularios.

6. Validación de formularios

Es muy importante realizar previamente la validación de los datos que se van a enviar a un CGI a través de un formulario, por ejemplo, verificar que todos los campos requeridos sean llenados, que solo se escriban cierto tipo de caracteres en un campo, que la cantidad de caracteres en un campo se limitada, etc.

Para la validación de los datos podemos actuar de dos maneras, una antes de que los datos envíen y la otra es validarlos dentro del CGI y enviar una respuesta a través de él indicando que ciertos valores no fueron enviados o bien fueron incorrectos. En este punto vamos a tratar la validación previa auxiliándonos de algunas funciones básicas de JavaScript y los ejemplos de validación por medio del CGI los veremos en puntos posteriores.

JavaScript es particularmente eficaz para la validación de formularios, sobre todo porque puede manipular elementos del formulario, por ejemplo, campos de texto y listas de selección. En su expresión más sencilla, la validación implica verificar la entrada para asegurar que los valores proporcionados se ajusten a lo que se espera de ellos y que estén dentro de límites aceptables

Como primer ejemplo vamos a ver una función que valide que todos los campos del formulario sean llenados. Para esto vamos a tomar el código del primer formulario que hicimos en el punto anterior y lo vamos a enriquecer con código de JavaScript.

Cómo asegurar que un campo de texto no está vacío:

Una cadena que tiene valor "" está vacía , así por medio de la propiedad **length** del campo de texto podemos verificar si es cero, lo cual indica que se trata de un campo vacío y de esta manera podemos advertir al usuario que el campo está vacío. Veamos el código siguiente:

```
<script lanjuage="JavaScript">
<!--
function ValidarForma() {
if (document.datos.Nombre.value.length==0) {
alert("El campo de Nombre es obligatorio.");
document.datos.Nombre.focus();
return false;}
if (document.datos.direccion.value.length==0) {
alert("El campo de Dirección es obligatorio.");
document.datos.direccion.focus();
return false;}
if (document.datos.mail.value.length==0) {
alert("El campo de e-mail es obligatorio.");
document.datos.mail.focus();
return false;}
}
//-->
</script>
```

Este código se encarga de validar que los campos de texto del formulario no estén vacío a la hora de ser enviados al CGI. Si el valor de alguno de los campos está vacío se envía a la pantalla un mensaje de alerta que le indica al usuario que el campo está vacío y que debe de escribirlo para poder procesar sus

datos y ubica el prompt en el campo vacío para que escriba el dato faltante, una vez que todos los campos ha sido llenados se envían los datos al CGI.

¿Dónde se coloca el código de JavaScript en la página web y cómo se invoca?

El Código de JavaScript se coloca antes de la etiqueta <BODY> y se invoca dentro de la etiqueta <FORM> de esta manera:

```
<head>
<title>Formulario 1</title>
</head>
<script lanjuage="JavaScript">
<!--
function ValidarForma() {
if (document.datos.Nombre.value.length==0) {
alert("El campo de Nombre es obligatorio.");
document.datos.Nombre.focus();
return false;}
if (document.datos.direccion.value.length==0) {
alert("El campo de Dirección es obligatorio.");
document.datos.direccion.focus();
return false;}
if (document.datos.mail.value.length==0) {
alert("El campo de e-mail es obligatorio.");
document.datos.mail.focus();
return false;}
}
//-->
</script>

<body><html>
....
<FORM NAME="datos" METHOD="GET" ACTION="/cgi-bin/vars.cgi"
onSubmit="return ValidarForma()">
…
</body>
</html>
```

Hay distintas maneras de invocar a las funciones en JavaScript y la que usamos en este caso es **OnSubmit** y significa que cuando se dé un click en el botón de *Enviar* se invocará a la función. También existe la opción **OnChange** y de esta manera se invoca a la función cuando se cambia de un campo a otro.

A continuación se mostrarán algunas funciones de JavaScript que pueden ser utilizadas para validar algunos datos.

Función para que todos los datos de un campo se envíen en minúsculas aún cuando hayan sido escritas en mayúsculas:

```
<script lanjuage="JavaScript">
<!--
function lower() {
var variable=document.forma.campo.value.toLowerCase();
document.forma.campo.value=variable;
}
//-->
</script>
```

La función que hace el caso contrario, es decir convertir las minúsculas a mayúsculas es prácticamente la misma función y sólo cambia la instrucción **toLowerCase()** a **toUpperCase()**;

Función que valida que un campo sólo tenga ciertos caracteres en especial y no permita caracteres especiales.

```
<script lanjuage="JavaScript">
<!--
function ValidarForma() {
var checa=document.forma.campo.value.toUpperCase();

	var validChars="ABCDEFGHIJKLMNÑOPQRSTUVWXYZ ÁÉÍÓÚ";
	var goodName = true;
	var tmpString;
	for (i=0; i<checa.length; i++)
		{
		tmpString = "" + checa.substring(i, i+1);
		if (validChars.indexOf(tmpString) == "-1")
			goodName = false;
		}
	if (goodName == false) {
			alert("Por favor verifique el dato  que desea enviar. Sólo se aceptan
letras.");
```

```
                    document.forma.campo.focus();

                     return false;
                    }
document.forma.campo.value=checa;

}
//-->
</script>
```

En el ejemplo anterior queremos que un campo sólo contenga letras y además que lo que escriban en ese campo lo envíe al CGI como mayúsculas. Con esta función se pueden validar distintas cosas, por ejemplo que un campo sea numérico solamente, lo que hay que hacer es cambiar las condiciones de la cadena de comparación y listo.

Estas son unas funciones básicas de JavaScript, pero son de mucha ayuda a la hora de validar campos de formularios. Si se desea saber más sobre JavaScript y todas las ventajas que nos ofrece sólo hay que buscar en la red algún tutorial o consultar un libro de fundamentos de JavaScript y encontrarán un sin número de funciones útiles para validar formularios y para hacer más interactivas sus aplicaciones Web.

A continuación veremos la manera de recibir los datos de los formularios a través de los métodos GUEST y POST.

7. Recepción, formateo y almacenamiento de datos en el CGI

El método GET:

Dentro de la etiqueta <FORM> de HTML podemos poner el atributo METHOD=GET, esto le dice al navegador que envíe los datos al CGI usando una solicitud *get*. Cuando el navegador envía los datos, éstos pasan como parte del URL. Cuando se utiliza este método los datos se reciben en la variable estándar QUERY_STRING y la variable REQUEST_METHOD contendrá la palabra GET.

La variable QUERY_STRING se formatea como flujo de pares *nombre=valor*, separados por el signo ampersand (&). La parte *nombre* viene directamente de la etiqueta INPUT en el formulario HTML.

Descifrar los datos del formulario es cuestión de dividir las partes nombre y valor de la cadena y reemplazar después cada uno de los indicadores que haya dentro de la cadena por los datos textuales.

Código del CGI que decodifica los datos del CGI.

```
#include<stdio.h>
#include <stdlib.h>

/* Esta es la estructura para las variables enviadas al CGI  */
struct {
  char name[128];
  char val[128];
} elements[16];

void splitword(char *out, char *in, char stop);
char x2c(char *x);
void unescape_url(char *url);




main()
{
/*Enviar primero el encabezado MIME*/
printf("Content-type: text/html\n\n");
char * qs; /* qs es para almacenar la variable QUERY_STRING */
int i;
/* se asigna la variable QS a qs; se aborta el programa si està vacìa*/
if((qs = getenv("QUERY_STRING")) == NULL)
  {
  printf("No hay información a decodificar.\n");
  exit(1);
  }

/* Separación de datos por nombre valor y sustitución de caracteres especiales */
```

```
for(i = 0; qs[0] != '\0'; i++)
  {
  /* se dividen los valores por '&'*/
  splitword(elements[i].val, qs, '&');
  /* se convierten los caracteres especiales */
  unescape_url(elements[i].val);
  /* se separan los datos por nombre valor */
  splitword(elements[i].name, elements[i].val, '=');
  }
/*Código HTML*/
printf("<html><head><title>variables CGI Estándar</title></head>\n");
  printf("<body>\n");
  printf("<p><b>Variables CGI Estándar</b>\n");
  printf("<br><p><b>Acerca del Usuario</b>\n");
  printf("<br>HTTP_USER_AGENT = %s\n", getenv("HTTP_USER_AGENT"));
  printf("<br>HTTP_ACCEPT = %s\n", getenv("HTTP_ACCEPT"));
  printf("<br>REMOTE_HOST = %s\n", getenv("REMOTE_HOST"));
printf("<br>REMOTE_ADDR = %s\n", getenv("REMOTE_ADDR"));
  printf("<br><p><b>Acerca del Servidor</b>\n");
  printf("<br>SERVER_NAME = %s\n", getenv("SERVER_NAME"));
 printf("<br>SERVER_SOFTWARE = %s\n", getenv("SERVER_SOFTWARE"));
 printf("<br>GATEWAY_INTERFACE = %s\n",
getenv("GATEWAY_INTERFACE"));
 printf("<br><p><b>Variables específicas de solicitud</b>\n");
 printf("<br>QUERY_STRING = %s\n", getenv("QUERY_STRING"));
 printf("<br>SCRIPT_NAME = %s\n", getenv("SCRIPT_NAME"));
 printf("<br>SERVER_PROTOCOL = %s\n", getenv("SERVER_PROTOCOL"));
 printf("<br>SERVER_PORT = %s\n", getenv("SERVER_PORT"));
printf("<br>CONTENT_TYPE = %s\n", getenv("CONTENT_TYPE"));
printf("<br>CONTENT_LENGTH = %s\n", getenv("CONTENT_LENGTH"));
printf("<br>REQUEST_METHOD = %s\n", getenv("REQUEST_METHOD"));
printf("<br>REMOTE_USER = %s\n", getenv("REMOTE_USER"));
printf("<br>REMOTE_IDENT = %s\n", getenv("REMOTE_IDENT"));
printf("<br><p>Variables:\n\n");

/* se imprimen los datos enviados al CGI a través del formulario */
for(i = 0; elements[i].name[0]; i++)
  printf("<p>%s=%s\n", elements[i].name, elements[i].val);

printf("</body></html>\n");

}   /**********termina main*************/
```

Sur 105 No. 343
Col. Héroes de Churubusco C.P. 09090, Tel. 55815126
http://www.samicom.com.mx/site/
e-mail: servicioclientes@samicom.com.mx

```
void splitword(char *out, char *in, char stop)
{
int i, j;

for(i = 0; in[i] && (in[i] != stop); i++)
  out[i] = in[i];

out[i] = '\0'; /* terminate it */
if(in[i])
  ++i;

for(j = 0; in[j]; )  /* shift the rest of the in */
  in[j++] = in[i++];
}

char x2c(char *x)
{
register char c;

/* nota: (x & 0xdf)pone en mayúscula a x  */
c  = (x[0] >= 'A' ? ((x[0] & 0xdf) - 'A') + 10 : (x[0] - '0'));
c *= 16;
c += (x[1] >= 'A' ? ((x[1] & 0xdf) - 'A') + 10 : (x[1] - '0'));
return(c);
}

/* Esta función repasa el URL carácter por carácter y convierte todas las
secuencias de escape (hexadecimales codificados) en caracteres.
También convierte en espacios los signos +
*/

void unescape_url(char *url)
{
register int i, j;

for(i = 0, j = 0; url[j]; ++i, ++j)
  {
  if((url[i] = url[j]) == '%')
    {
    url[i] = x2c(&url[j + 1]);
```

```
    j += 2;
    }
  else if (url[i] == '+')
    url[i] = ' ';
  }
url[i] = '\0';  /* terminar en la nueva longitud */
}
```

El método POST

El método GET pasa los datos al programa CGI usando la variable estándar QUERY_STRING, el método POST pasa los mismos datos, en el mismo formato, usando el flujo de archivo “de entrada estándar” (stdin).

Podemos modificar el formulario que tenemos de ejemplo con sólo cambiar la etiqueta <FORM> y en ella escribir METHOD=POST. Con esto en los datos recibidos la variable QUERY_STRING estará vacía, pero las variables CONTENT_TYPE y CONTENT_LENGTH estarán llenas con el tipo MIME de los datos del formulario que se pasan al CGI (application/x-www-form-urlencoded) y el número de bytes de ese flujo de datos, respectivamente.

¿Por qué utilizar el método POST?

Debido a que, por naturaleza, el método GET está limitado al tamaño máximo de la variable estándar, que no se garantiza que sea grande entre plataformas y que probablemente sea de entre unos 256 y unos 4096 bytes. Si se piensa que llegará a tener esa cantidad de datos en un formulario será necesario utilizar POST en vez de GET.

Código del CGI que decodifica los datos del CGI. POST

```
#include <stdio.h>
#include <stdlib.h>
/* Estructura para los datos del CGI */
#define MAXQELEMENTS (16)
struct {
    char name[128];
    char val[128];
} elements[MAXQELEMENTS];


void splitword(char *out, char *in, char stop);
char x2c(char *x);
void unescape_url(char *url);
```

```
main()
{
char * ct; /* para el content-type */
char * cl; /* para el  content-length */
int   icl; /* content-length */
char * qs; /* query string */
int rc;
int i;

/* header mime-type */
printf("Content-type: text/html\n\n");


/* graba el content-type y content-length
   y los checa para su validación */

ct = getenv("CONTENT_TYPE");
cl = getenv("CONTENT_LENGTH");
if(cl == NULL)
  {
  printf("content-length es indefinido!\n");
  exit(1);
  }
icl = atoi(cl);

/* ¿tenemos una consulta válida? */
if(strcmp(ct, "application/x-www-form-urlencoded"))
  {
  printf("No entiendo el content-type %s\n");
  exit(1);
  }
else if (icl == 0)
  {
  printf("content-length es cero, no hay datos a decodificar \n");
  exit(1);
  }

/* asignación de memoria para el flujo de entrada */
if((qs = malloc(icl + 1)) == NULL)
  {
  printf("No se puede reservar memoria, contacta al webmaster\n");
```

```
  exit(1);
  }

if((rc = fread(qs, icl, 1, stdin)) != 1)
  {
  printf("No se puede leer la cadena de entrada (%d)! Contacta al webmaster\n",
rc);
  exit(1);
  }
qs[icl] = '\0';


/* Separación de datos por nombre valor y sustitución de caracteres especiales*/
for(i = 0; *qs && i < MAXQELEMENTS; i++)
  {

  /* se dividen los valores por '&'*/
  splitword(elements[i].val, qs, '&');

   /* se convierten los caracteres especiales*/
  unescape_url(elements[i].val);
  /* se separan los datos por nombre valor */
  splitword(elements[i].name, elements[i].val, '=');
  }
  /*****************código HTML*****************/
printf("<html><head><title>variables CGI Estándar</title></head>\n");
printf("<body>\n");
printf("<p><b>Variables CGI Estándar</b>\n");
printf("<br><p><b>Acerca del Usuario</b>\n");
printf("<br>HTTP_USER_AGENT = %s\n", getenv("HTTP_USER_AGENT"));
printf("<br>HTTP_ACCEPT = %s\n", getenv("HTTP_ACCEPT"));
printf("<br>REMOTE_HOST = %s\n", getenv("REMOTE_HOST"));
printf("<br>REMOTE_ADDR = %s\n", getenv("REMOTE_ADDR"));
printf("<br><p><b>Acerca del Servidor</b>\n");
printf("<br>SERVER_NAME = %s\n", getenv("SERVER_NAME"));
printf("<br>SERVER_SOFTWARE = %s\n", getenv("SERVER_SOFTWARE"));
printf("<br>GATEWAY_INTERFACE = %s\n", getenv("GATEWAY_INTERFACE"));
printf("<br><p><b>Variables específicas de solicitud</b>\n");
printf("<br>QUERY_STRING = %s\n", getenv("QUERY_STRING"));
printf("<br>SCRIPT_NAME = %s\n", getenv("SCRIPT_NAME"));
printf("<br>SERVER_PROTOCOL = %s\n", getenv("SERVER_PROTOCOL"));
printf("<br>SERVER_PORT = %s\n", getenv("SERVER_PORT"));
```

```
printf("<br>CONTENT_TYPE = %s\n", getenv("CONTENT_TYPE"));
printf("<br>CONTENT_LENGTH = %s\n", getenv("CONTENT_LENGTH"));
printf("<br>REQUEST_METHOD = %s\n", getenv("REQUEST_METHOD"));
printf("<br>REMOTE_USER = %s\n", getenv("REMOTE_USER"));
printf("<br>REMOTE_IDENT = %s\n", getenv("REMOTE_IDENT"));
printf("<br><p><b>Variables:</b>\n\n");


/* impresión de datos */
for(i = 0; elements[i].name[0]; i++)
  printf("<p>%s=%s\n", elements[i].name, elements[i].val);
  printf("</body></html>\n");
}

/*Funciones especiales*/
void splitword(char *out, char *in, char stop)
{
int i, j;

for(i = 0; in[i] && (in[i] != stop); i++)
  out[i] = in[i];

out[i] = '\0'; /* terminate it */
if(in[i])
  ++i;

for(j = 0; in[j]; )  /* shift the rest of the in */
  in[j++] = in[i++];
}

char x2c(char *x)
{
register char c;

/* note: (x & 0xdf) makes x upper case */
c  = (x[0] >= 'A' ? ((x[0] & 0xdf) - 'A') + 10 : (x[0] - '0'));
c *= 16;
c += (x[1] >= 'A' ? ((x[1] & 0xdf) - 'A') + 10 : (x[1] - '0'));
return(c);
}
```

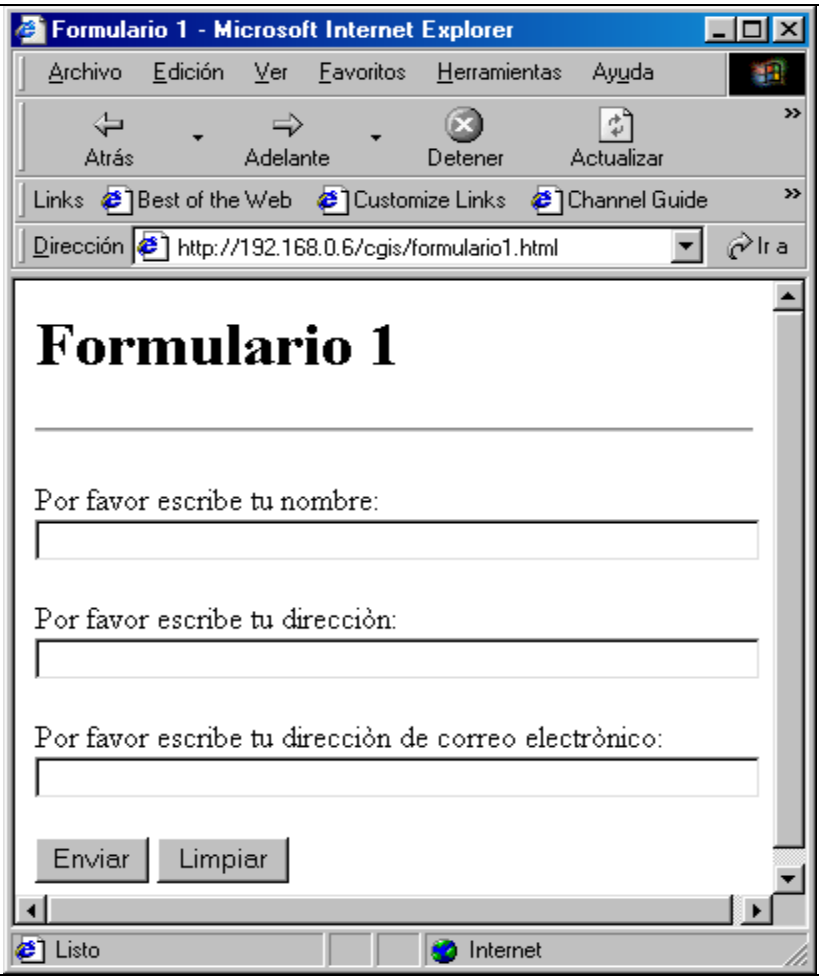
```
void unescape_url(char *url)
{
register int i, j;

for(i = 0, j = 0; url[j]; ++i, ++j)
  {
  if((url[i] = url[j]) == '%')
    {
    url[i] = x2c(&url[j + 1]);
    j += 2;
    }
  else if (url[i] == '+')
    url[i] = ' ';
  }
url[i] = '\0';  /* terminate it at the new length */
}
```

8. Presentación de datos recibidos.

Una vez que hemos recibido, formateado y almacenado los datos del en el CGI podemos hacer uso de ellos como mejor nos plazca. Para poder utilizar un dato dentro del programa, por ejemplo si queremos imprimirlo, sólo basta con teclear la siguiente sentencia:

printf("%s\n" , elements[n].val);
donde *%s* corresponde al tipo de variable que vamos a imprimir y en este caso es una cadena y que corresponde al elemento *n* de la estructura de valores, pero ¿cómo saber a qué número de elemento corresponde cada campo del formulario?, la respuesta es muy sencilla, cada elemento corresponde al número de dato con que es recibido, por ejemplo:



En el formulario de la figura anterior, el campo del *nombre* corresponde al primer dato del formulario, la *dirección* al segundo, el *mail* al tercero y así sucesivamente hasta llegar al último campo, pero en la estructura de datos el último dato corresponde al elemento n-1 del número total de datos del formulario, teniendo así que:

Dato del formulario	Elemento de la estructura de valores
Nombre	Elements[0].val
Dirección	Elements[1].val
Mail	Elements[2].val
Sexo	Elements[3].val
Novelas	Elements[4].val
Cuentos	Elements[5].val
Biografías	Elements[6].val
Comics	Elements[7].val

De esta manera podemos identificar qué dato llamar para mostrarlo en la respuesta que envíe el CGI. Los datos siempre son almacenados en forma de cadenas, en caso de se necesite uno de esos datos como entero, será necesario convertirlo con la función *atoi*.

9. Base de datos POSTGRES

Pocas herramientas de software son tan necesarias como las bases de datos. No hay empresa o institución que pueda prescindir de ellas; son el alma de los programas contables, de la nómina y de los inventarios en las empresas tradicionales, pero de igual manera alojan la información que consumimos en todo el mundo gracias a la Internet.

Esa importancia de las bases de datos ha generado una millonaria y gigantesca industria mundial, representada por empresas de software como Oracle, Informix, Sybase y otras, las cuales facturan anualmente varios millones de dólares por el licenciamiento de sus manejadores de bases de datos relacionales.

Afortunadamente, la actual revolución que ha significado el software de código abierto (open source), nos ha permitido contar con PostgreSQL, el servidor de bases de datos más avanzado en esta categoría.

Un poco de historia.

El más antiguo antecesor de PostgreSQL es Ingres, desarrollado en la Universidad de California en Berkeley de 1977 a 1985. Con el fin de mejorar Ingres, Michael Stonebraker generó en 1986 un nuevo servidor de bases de datos y lo llamó Postgres, es decir posterior a Ingres. Ya para 1994, Jolly Chen y Andrew Yu le agregaron la funcionalidad del lenguaje de consulta estructurado (SQL, por sus siglas en inglés), una norma mundial establecida varios lustros antes y lo llamaron Postgres95 (1994-1995).

Durante 1996 se dieron dos cambios importantes: se cambió el nombre a PostgreSQL y se formó un grupo especial de desarrollo, comandado por Marc. G. Fournier en Toronto, Canadá.

Desarrollo de PostgreSQL

Al igual que muchos proyectos de código abierto, PostgreSQL es desarrollado por un gran número de programadores que utilizan la Internet como el medio para discutir, acordar y enviar las mejoras que lo han convertido en el líder en su área.

A diferencia del software comercial, cada 3 ó 5 meses de libera una nueva versión, con nuevas características, con menos errores (bugs) o más acorde a las normas de SQL92. El código fuente, fundamentalmente en lenguaje C, cuenta con mas de 250,000 líneas; y no cuesta un solo centavo.

PostgreSQL se encuentra disponible para los sistemas operativos tipo Unix y para Windows. Se puede obtener en formato de archivo comprimido (*.tar.gz) en http://www.postgresql.org o como un archivo RPM (RedHat Package Manager) que fácilmente se puede instalar y configurar, en http://www.redhat.com.

Una distribución clásica de PostgreSQL ofrece mucho más que el motor o servidor de la base de datos; siempre lo acompañan algunas herramientas que facilitan su configuración y administración: un cliente interactivo en modo de texto, otro en modo gráfico, utilerías para extraer la base de datos hacia un archivo de comandos SQL, una interfase para programadores (API) para hacer aplicaciones en lenguaje C, documentación para usuarios, programadores, administradores, etc.

El servidor de la base de datos se puede accesar por medio de varios lenguajes: C, Perl, Visual Basic, Delphi, Phyton, etc.; lo cual le da una gran versatilidad para usarse como motor de base de datos de prácticamente cualquier aplicación que requiera un manejador robusto, eficiente y que cumpla con las normas internacionales.

Bruce Momjiam, miembro del grupo de desarrollo de PostgreSQL, está escribiendo un libro que publicará Addison-Wesley (PostgreSQL: Introduction and Concepts), el cual se puede obtener de manera gratuita en formato PDF, en http://www.postgresql.org/docs/awbook.html. A fines de marzo de 2000, Bruce hizo públicos los primeros diez capítulos del libro, que constituyen un buen material para adentrarse es esta base de datos.

Para mayor información puede visitarse el site del proyecto de castellanización de la documentación en: http://lucas.hispalinux.es y consultar los manuales y tutoriales de Postgres.

Iniciando Postgres

Una vez que ya sabemos algo sobre Postgres vamos a ver cómo se maneja, primero que nada no hay que preocuparnos por su instalación, pues al instalar la versión de Linux Red Hat 7.1 Postgres se instala automáticamente y ya sólo hay que darla de alta con el comando **setup** y a través del servicio "**System Services**" y habilitando **[*]postgresql**.

Para iniciar el servicio de Postgres se debe de ejecutar el siguiente comando:

[root@server /root]#/etc/rc.d/init.d/postgresql start

Una vez iniciado el servicio de postgres, es necesario entrar al sistema con el usuario propietario de Postgres que tiene el mismo nombre de la base, para entrar con el usuario de la base al sistema, se hace de la siguiente manera:
[root@server /root]#su –l postgres
y nos aparece el siguiente prompt:
bash-2.04$

Para crear una base de datos:

Ahora vamos a crear una base de datos de prueba a través del siguiente comando:
bash-2.04$createdb prueba
una vez creada la base de datos, la información generada se almacena en el siguiente directorio:
/var/lib/pgsql
que es el directorio home del usuario *postgres* y la base y todos los archivos realcionados con ella están en el directorio:
/var/lib/pgsql/base/prueba

Para trabjar con la base de datos:

Crear tablas y ejecutar queries es necesario abrir una sesión SQL , para ello es necesario correr el siguiente comando:
bash-2.04$psql prueba y obetenemos el siguiente mensaje y prompt:

```
Welcome to psql, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

prueba=#
```

A partir del prompt **prueba#** se pueden ejecutar las sentencias necesarias para generar tablas, hacer inserciones, consultas, actualizaciones, etc a través del lenguaje SQL.

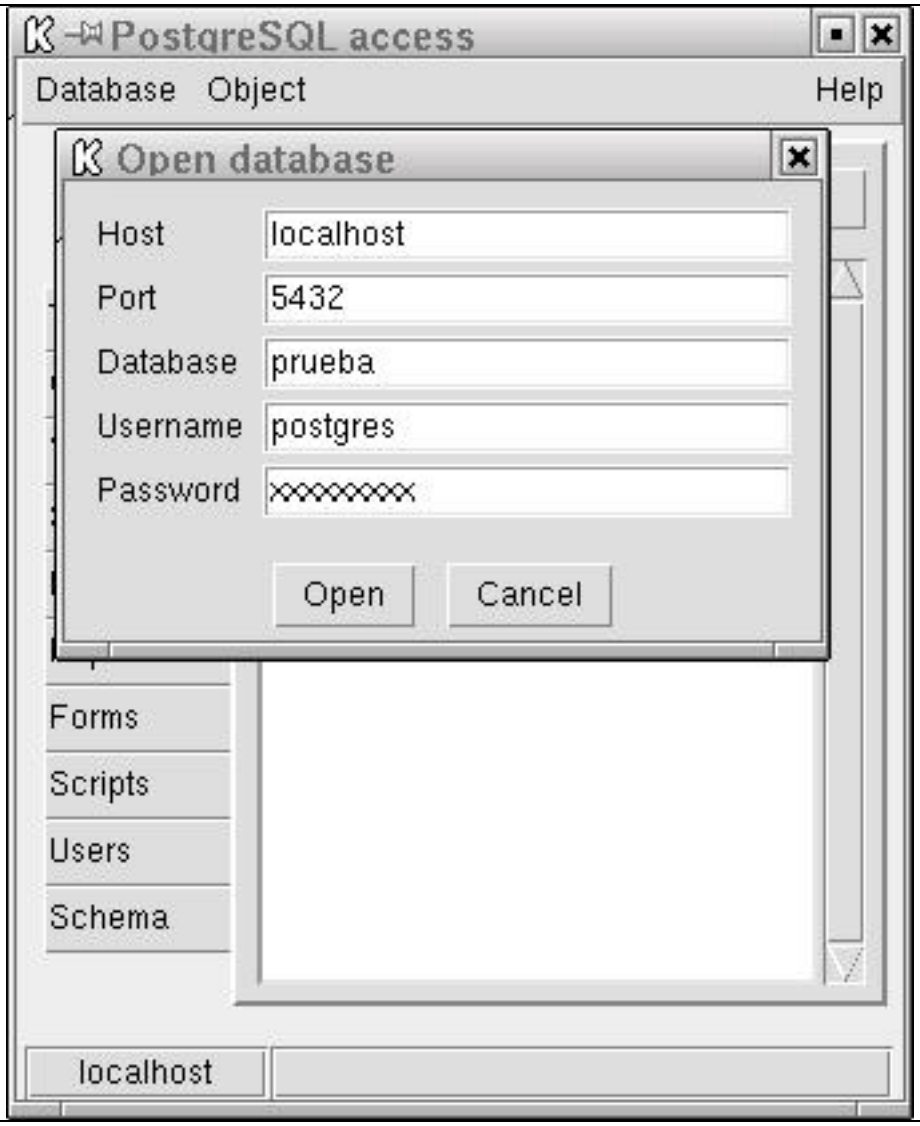
Herramienta gáfica para utilizar postgres:

Existen varias herramientas gráficas para trabajar con Postgres y una de ellas es: **pgaccess**, que nos permite crear nuevas bases, tablas, queries, formas,

crear usuarios, asignar perimsos, entre otros aspectos tanto de operación como de administración y para empezar a trabajar con él hay que hacer unos ajustes previos, como asignarle un password al usuario *postgres* y esto se hace de la siguiente manera:
[root@server /root]#passwd postgres y obtenemos los siguiente:

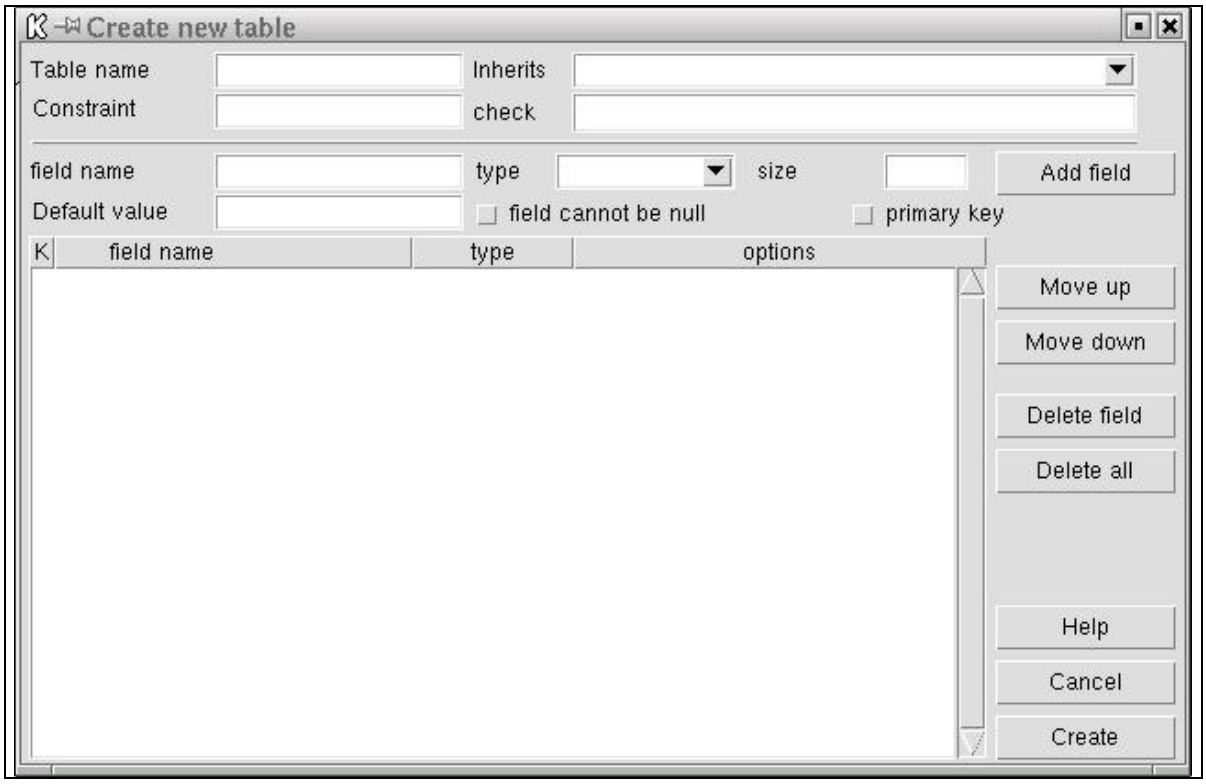
```
Changing password for user postgres
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
[root@server /root]#
```

Una vez que se la ha asignado un password a postgres podemos invocar a la herramienta *pgaccess* de la siguiente manera:
[root@server /root]# pgaccess & y nos aperecerá una ventana donde hay que indicar que base deseamos abrir, con que usuario y el password del mismo, veamos esta imagen que nos hace referencia a esto:



Una vez que le damos los datos correctos nos aparecerá una ventana como la siguiente:

Donde podemos empezar a trabajar con la base de datos prueba pudiendo crear tablas, queries, vistas, secuencias, funciones, reportes, formas, scripts, administrar usuarios y esquemas. Para ilustrar lo sencillo que es trabajar con esta herramienta, la siguiente figura ilustra la interfaz que nos permite generar tablas de una manera muy sencilla e intuitiva.



Como se puede ver en la figura, para crear una tabla sólo es necesario darle un nombre e ir creando los campos de acuerdo a nuestras necesidades y listo se tiene una tablas en unos cuantos momentos y de esta manera se trabaja para crear formas, vistas, queries, etc.

Hasta este momento hemos visto la facilidad con que puede trabajarse con Postgres y ahora veremos la manera de interactuar con ella a través de programas CGI.

10. Conexión a la base de datos a través de un CGI

Para conectarnos a la base de datos a traves de un CGI, es necesario crear un programa que contenga la funciones necesarias que puedan inetractuar con Postgres y para facilitarnos la vida, vamos a utilizar el precompilador llamado **ecpg** y que en realidad trabaja con los programas escritos en lenguaje C, pero con algunas varientes que permiten incluir sentencias SQL en el código sin necesidad de escribir grandes cantidades decódigo para llevarlas a cabo y quien se encarga de escribir todas esas líneas de código para ejecutar las sentencias SQL es el precompilador **ecpg**. Así que ahora nuestros programas ya no tendrán la extensión ***.c**, si no ***.pgc** y el precompilador **ecpg** nos entregará un programa con extensión ***.c** el que finalmete vamos a compilar para crear nuestro programa CGI.

Veamos el código de un programa que se encargue de conectarse a la base de datos prueba:

```
exec sql include header_test;
exec sql include sqlca;
exec sql type c is char reference;
typedef char* c;

#include<stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define FALSE 0
#define TRUE 1
#define    UNAME_LEN     20
#define    PWD_LEN       40

/* Esta es la estructura para las variables enviadas al CGI  */
struct {
  char name[128];
  char val[128];
} elements[16];

/* forward declarations */
void splitword(char *out, char *in, char stop);
char x2c(char *x);
void unescape_url(char *url);

char msg[128];
```

```
/**********declaración de variables **********/
exec sql begin declare section;

char nombre[50];
char apellido_materno[50];
char apellido_paterno[50];
char usuario[15];
char password[15];
char fecha[17];
char query[250];
exec sql end declare section;


main()
{
 char * qs; /* qs es para almacenar la variable QUERY_STRING */
 int i;
/************DEBUG de POstgres********/

FILE *dbgs;

if ((dbgs = fopen("log", "w")) != NULL)
 ECPGdebug(1, dbgs);


 printf("Content-type: text/html\n\n");

/* se asigna la variable QS a qs; se aborta el programa si está vacía/
if((qs = getenv("QUERY_STRING")) == NULL)
  {
  printf("No hay información a decodificar.\n");
  exit(1);
  }
  printf("%s\n",qs);
/* Separación de datos por nombre valor y sustitución de caracteres especiales */

for(i = 0; qs[0] != '\0'; i++)
  {
  /* se dividen los valores por '&'*/
  splitword(elements[i].val, qs, '&');
  /* se convierten los caracteres especiales*/
```

Sur 105 No. 343
Col. Héroes de Churubusco C.P. 09090, Tel. 55815126
http://www.samicom.com.mx/site/
e-mail: servicioclientes@samicom.com.mx

```
  unescape_url(elements[i].val);
  /* se separan los datos por nombre valor */
  splitword(elements[i].name, elements[i].val, '=');
  }
/*******código HTML******/
 printf("<html><head><title>variables CGI Estándar</title></head>\n");
  printf("<body>\n");
  printf("<p><b>Variables CGI Estándar</b>\n");
  printf("<br><p><b>Acerca del Usuario</b>\n");
  printf("<br>HTTP_USER_AGENT = %s\n", getenv("HTTP_USER_AGENT"));
  printf("<br>HTTP_ACCEPT = %s\n", getenv("HTTP_ACCEPT"));
  printf("<br>REMOTE_HOST = %s\n", getenv("REMOTE_HOST"));
printf("<br>REMOTE_ADDR = %s\n", getenv("REMOTE_ADDR"));
  printf("<br><p><b>Acerca del Servidor</b>\n");
  printf("<br>SERVER_NAME = %s\n", getenv("SERVER_NAME"));
 printf("<br>SERVER_SOFTWARE = %s\n", getenv("SERVER_SOFTWARE"));
 printf("<br>GATEWAY_INTERFACE = %s\n",
getenv("GATEWAY_INTERFACE"));
 printf("<br><p><b>Variables específicas de solicitud</b>\n");
 printf("<br>QUERY_STRING = %s\n", getenv("QUERY_STRING"));
 printf("<br>SCRIPT_NAME = %s\n", getenv("SCRIPT_NAME"));
 printf("<br>SERVER_PROTOCOL = %s\n", getenv("SERVER_PROTOCOL"));
 printf("<br>SERVER_PORT = %s\n", getenv("SERVER_PORT"));
printf("<br>CONTENT_TYPE = %s\n", getenv("CONTENT_TYPE"));
printf("<br>CONTENT_LENGTH = %s\n", getenv("CONTENT_LENGTH"));
printf("<br>REQUEST_METHOD = %s\n", getenv("REQUEST_METHOD"));
printf("<br>REMOTE_USER = %s\n", getenv("REMOTE_USER"));
printf("<br>REMOTE_IDENT = %s\n", getenv("REMOTE_IDENT"));
printf("<br><p>Variables:\n\n");
printf("%s\n",elements[0].val);
/* print 'em all out */
for(i = 0; elements[i].name[0]; i++)
  printf("<p>%s=%s\n", elements[i].name, elements[i].val);
/************función de conexión a postgres ************/
login();

printf("</body></html>\n");

strcpy(msg, "commit");  /**********commit final**************/
        exec sql commit;

strcpy(msg, "disconnect"); /*********se desconecta d ela base**********/
```

Sur 105 No. 343
Col. Héroes de Churubusco C.P. 09090, Tel. 55815126
http://www.samicom.com.mx/site/
e-mail: servicioclientes@samicom.com.mx

```
        exec sql disconnect;
      if (dbgs != NULL)
      fclose(dbgs);

exit(0);

}   /***********termina main*************/


 /*********función login*********/

  login()
{
strcpy(msg,"connect");
exec sql connect to unix:postgresql://localhost:5432/prueba;
 printf("<br><p><b>Conexión a Postgres realizada</b>\n\n");
}

/***************funciones de separado*************/
void splitword(char *out, char *in, char stop)
{
int i, j;

for(i = 0; in[i] && (in[i] != stop); i++)
  out[i] = in[i];

out[i] = '\0'; /* terminate it */
if(in[i])
  ++i;

for(j = 0; in[j]; )  /* shift the rest of the in */
  in[j++] = in[i++];
}

char x2c(char *x)
{
register char c;

/* note: (x & 0xdf) makes x upper case */
c  = (x[0] >= 'A' ? ((x[0] & 0xdf) - 'A') + 10 : (x[0] - '0'));
c *= 16;
```

```
c += (x[1] >= 'A' ? ((x[1] & 0xdf) - 'A') + 10 : (x[1] - '0'));
return(c);
}

/* this function goes through the URL char-by-char
   and converts all the "escaped" (hex-encoded)
   sequences to characters

   this version also converts pluses to spaces. I've
   seen this done in a separate step, but it seems
   to me more efficient to do it this way. --wew
*/

void unescape_url(char *url)
{
register int i, j;

for(i = 0, j = 0; url[j]; ++i, ++j)
  {
  if((url[i] = url[j]) == '%')
    {
    url[i] = x2c(&url[j + 1]);
    j += 2;
    }
  else if (url[i] == '+')
    url[i] = ' ';
  }
url[i] = '\0';  /* terminate it at the new length */
}
```

Observando el código anterior, las partes de código realtadas en negritas son las que se le han agregado al código en lenguaje C para la decodificación de datos enviados a través del método GET. Para poder compilar este programa hay que hacer lo siguiente:

a) Contar con la librería: header_test.h en el directorio donde se están compilando los CGI
b) Salvar el código con extensión *.pgc

c) Llamar al precompilador de la siguiente manera: **ecpg programa.pgc –I/usr/include/pgsql**

d) Si todo es correcto en el código *.pgc el precompilador generará un programa con el mismo nombre que el extensión *.pgc, pero con extensión *.c, el cual se compila de la siguiente manera: **cc programa.c –I/usr/include/pgsql –lecpg –lpq –lcrypt**

e) Una vez compilado el programa *.c, se generará un ejecutable llamado a.out, el cual constituye el programa CGI y será copiado al directorio cgi-bin del servidor de la siguiente manera cp a.out /var/www/cgi-bin/programa.cgi

f) Finalmente para que se realice la conexión exitosa a la base de datos, s necesario crear con pgaccess un usuario llamado apache, al cual se le darán todos los privilegios ya que apache es el usuario que se encarga de administrar y ejecutar todas las peticiones hechas al servidor Web y por ende él es quien va a realizar la conexión a la base de aquí que exista la necesidad de que el usuario exista en la base de datos.

Código de la librería header_test.h

```
exec sql include sqlca;

#define ERROR -1
#define TRUE 1
#define FALSE 0


void
Finish(msg)
{
sqlprint();
/****finish transaction****/
exec sql        rollback;
/******** and remove test table *******/
exec sql drop table meskes;
exec sql        commit;
exec sql        disconnect;

/**********************debería de desmontar el disco y parar*****************/

exit(-1);
}
void
warn(void)
{
```

```
}
exec sql whenever sqlerror
do
Finish(msg);
exec sql whenever sqlwarning
do
warn();
```

11. Rutinas de sentencias SELECT, INSERT, UPDATE y DELETE

Para realizar este tipo de sentencias en necesario incluir en el programa una serie de variables que deben estar relacionadas con los campos de la o las tablas que se vayan a afectar y deben ser incluidas como sigue:

Suponiendo que se tiene una tabla que contenga los campos: id, nombre, apellido_paterno, apellido_materno, usuario, password y fecha, se deben de incluir en el programa una declaración de variables como esta:

```
exec sql begin declare section;
int id;
char nombre[50];
char apellido_materno[50];
char apellido_paterno[50];
char usuario[15];
char password[15];
char fecha[17];
char query[250];
exec sql end declare section;
```

La variable *query*, es necesaria para almacenar en ella el query que realice la sentencia deseada.

Rutina de selección (SELECT):

Función **selecciona()**

```
selecciona()

{
sprintf(query,"select
nombre,apellido_paterno,apellido_materno,login,password,fecha from usuarios
order by nombre");
exec sql prepare SM from :query;

        exec sql declare prep cursor for SM;
```

```
            exec sql open prep;

        exec sql whenever not found do break;

        while (1) {
                strcpy(msg, "fetch");
                exec sql fetch in prep into  :nombre, :apellido_paterno,
:apellido_materno, :usuario, :password, :fecha;
                /***************se imprimen variables*************/
              printf("<tr align=center> \n");
printf("%s \n",nombre);
printf("%s \n", apellido_paterno);
printf("%s \n",apellido_materno);
printf("%s \n",usuario);
printf("%s\n",password);
printf("%s</td>\n",fecha);



 /*****************se limpian variables para el siguiente ciclo********************/
              strcpy(nombre,"");
              strcpy(apellido_paterno,"");
              strcpy(apellido_materno,"");
              strcpy(usuario,"");
              strcpy(password,"");
              strcpy(fecha,"");
        }

        strcpy(msg, "close");
        exec sql close prep;
}/*************fin de funcion selecciona()**********************/
```

Rutina de inserción (INSERT)

Función **inserta()**

```
inserta()

{
strcpy(msg, "execute insert");
sprintf(command, "insert into usuarios values('%s','%s','%s','%s','%s','%s')",
element[0].val,element[1].val,element[2].val,element[3].val,element[4].val,element[5].val)
```

```
;
exec sql execute immediate :command;

  /***************se imprimen variables insertadas *************/
printf("<p>variables insertadas \n");
printf("<p>%s \n",element[0].val);
printf("<p>%s\n", element[1].val);
printf("<p>%s\n",element[2].val);
printf("<p>%s \n",element[3].val);
printf("<p>%s \n",element[4].val);
printf("<p>%s\n",element[5].val);
}
/*************fin de función inserta()**********************/
```

Rutina de actualización (UPDATE)

Función **actualiza()**

```
actualiza()

{
strcpy(msg, "execute update");
sprintf(command, "update  usuarios set nombre='%s' ,apellido_paterno='%s',
apellido_materno='%s',login='%s',password='%s',fecha='%s'  where id=%d
",element[6].val,element[7].val,element[8].val,element[9].val,element[10].val,
element[11].val,element[0].val);

exec sql execute immediate :command;

  /***************se imprimen variables actualizadas *************/
```

```
printf("<p>variables insertadas \n");
printf("<p>%s \n",element[6].val);
printf("<p>%s\n", element[7].val);
printf("<p>%s\n",element[8].val);
printf("<p>%s \n",element[9].val);
printf("<p>%s \n",element[10].val);
printf("<p>%s\n",element[11].val);


}/*************fin de función actualiza()**********************/
```

Rutina de borrado (DELETE)

Función **borra()**

```
borra()

{
strcpy(msg, "execute delete");
sprintf(command, "delete from  usuarios where id=%d'",element[0].val);
 exec sql execute immediate :command;
}
```

12. Integración de una aplicación práctica.

Hasta este punto hemos visto desde cómo poner en servicio un servidor Web, hasta la conexión con una base de datos a través deun programa CGI, pasando por el diseño de formularios, decodificación de datos, validadción de formularios y rutinas SELECT, INSERT, UPDATE y DELETE. Con estas herramientas estámos capacitados para crear un sistema ABC (Altas, Bajas y Cambios) de información en ambiente Web, el cual puede aplicarse prácticamente a cualquier tarea administrativa, por lo que la parte final de este curso es crear una aplicación dónde se ponga en uso todo lo aprendido en este curso y es aquí donde la imaginación y necesidades de los participantes se pondrá a prueba, así como su capacidad para trabajar en equipo.

13. Bibliografía.

- William E. Weinman. **"El Libro de CGI"** Ed. Prentice Hall

- Brian W. Kernighan & Dennis M. Ritchie. **"El Lenguaje de programación C"** 2ª edición. Ed. Prentice Hall.
- Jason J. Manger. **"Fundamentos de JavaScript"** Ed. Mc Graw Hill
- Ramón Soria. **"HTML Diseño y creación de páginas Web"** Ed. Alfaomega

Sitios Web:

- http://www.redhat.com
- http://www.postgres.org.mx
- http://lucas.hispalinux.es
- http://www.apache.org
- http://www.linux.org
- http://www.fsf.org/home.es.html