UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
"ACATLÁN"

CREACIÓN DE COMMON GATEWAY INTERFACE (CGI'S) UTILIZANDO EL LENGUAJE PERL

TESINA

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

PRESENTA

ENGELBERT RAFAEL BENAVIDES SEGURA.

Asesor: Lic. Christian Carlos Delgado Elizondo

Octubre 2003

TESIS
CON
FALLA DE
ORIGEN

AGRADECIMIENTOS

Agradecimientos

Para mi asesor de tesina, el Lic. Christian Carlos Delgado Elizondo

"...to whom you may impart griefd, joys, fears, hopes, suspicions, counsels, and whatsoever lieth upon your heart to oppress it, in a king of civil shrift or confession".Deseo agradecer también los valiosos comentarios y sugerencias de mis sinodales:

Francisco Bacon

Y también deseo agradecer a mi Padre que siempre me apoya, a mis hermanos que me dieron animo y a Je t'aime que siempre me escucha y aconseja.

ÍNDICE

La Web no es sólo un medio para colocar información de variados tipos y formatos a disposición de los usuarios de Internet. Utilizando la Web es posible también interactuar con el usuario de forma que pueda, por ejemplo, encargar un producto, suscribirse a algún servicio, reservar un pasaje, hacer una consulta a una base de datos, etc.

Normalmente cuando un navegador de Web (por ejemplo el Netscape) llama a un URL (Localizador Universal de Recursos) en particular, sucede lo siguiente: Primero la computadora contacta al servidor HTTP (Protocolo de transferencia de hipertexto) con dicho URL (HTTP es el protocolo que se utiliza en las comunicaciones en la Web entre el server y el browser). El servidor HTTP revisa si el archivo requerido por nuestra computadora se encuentra en su sistema, en caso afirmativo envía una copia del archivo como respuesta. Nuestro navegador entonces, muestra el archivo en el formato apropiado.

Además de todo esto, los servidores del Web están configurados de tal manera que cada vez que se requiere un archivo de un directorio determinado (usualmente el "cgi-bin"), dicho archivo no es enviado, sino que es ejecutado como un programa y la salida de este programa es enviada a nuestro navegador para que éste lo muestre. Esta función es conocida como "Common Gateway Interface" por sus siglas en ingles ("CGI") o "Interfas de Puerta de Enlace" y los programas a los que nos referimos son llamados programas CGI.

CGI no es un lenguaje, es sólo un simple protocolo que puede utilizarse para comunicar formas del Web y programas. Un programa CGI puede ser escrito en cualquier lenguaje que pueda leer STDIN (siglas de Standar Input, o entrada de datos estándar), escribir a STDOUT (siglas de Standar Output, o salida de datos estándar) y leer variables del ambiente, por ejemplo; virtualmente cualquier lenguaje de programación, incluyendo C, Perl y hasta scripts de Unix.

El lenguaje Perl (Practical Extraction and Report Languaje) nació en entornos UNIX para la realización de tareas administrativas. Es muy potente manipulando archivos y cadenas de texto, ya que en este entorno la mayoría de archivos de configuración del sistema, de las aplicaciones, entre otros; son archivos de texto, y a menudo de un tamaño considerable. Es un lenguaje interpretado, con una sintaxis similar a C. En la actualidad, el intérprete de Perl es gratuito, y puede ser instalado en una multitud de plataformas.

Debido a estas características, se asocia, sobre todo, a la programación en entornos web usando la interfaz CGI (Common Gateway Interface), ya que en estos entornos se conectan plataformas heterogéneas, tanto software como hardware, y es más fácil codificar una aplicación en un lenguaje que pueda ser ejecutado en cualquier máquina sin tener que ser recompilado, por lo que Perl sale ganando al poder ejecutarse en distintas máquinas con la única condición de que tenga instalado su intérprete correspondiente.

A diferencia de la mayoría de los lenguajes de programación, Perl no limita el tamaño de los datos, y sólo vendrá impuesto por la memoria que pueda obtener, por ejemplo, podría tratar todo un archivo de texto como fuese una cadena de caracteres. Perl utiliza ténicas sofisticadas para encontrar patrones, por lo que le permite escanear grandes cantidades de datos rápidamente. Aunque se suele utilizar para trabajar con textos, para lo que está optimizado, también puede trabajar con datos binarios.

Convenciones utilizadas en este documento:

En este documento se emplearán varios conceptos y terminologías de computación, de las cuales encontrará cada uno de sus significados en la sección de Glosario de Términos.

Además se emplearán las siguientes convenciones tipográficas:

- En el documento introduciré números como este ***(07B)*** o ***(02R)***, que indican de que bibliografía me base o en donde pueden tomar más datos al respecto; al final del documento se encuentra la sección de bibliografía y de referencias en donde se citan las fuentes utilizadas.

- Todos los pseudocódigos que manejaré llevarán el tipo de letra *Times New Roman Cursivas*

- Todos los códigos fuente de Perl y algunos más, llevarán el tipo de letra **`Courier New en negritas`** para poder identificarlos, además estarán encerrados en un recuadro como éste:

PLANTEAMIENTO DEL PROBLEMA

EN LA ACTUALIDAD LA MANERA EN QUE LA INFORMACIÓN SE PRESENTA EN UN SITIO WEB O EN INTERNET HA VENIDO EVOLUCIONANDO CON EL TRANSCURRIR DEL TIEMPO Y EL AVANCE DE LA TECNOLOGÍA, A MEDIDA QUE ESTE CRECIMEINTEO SE HA VENIDO DANDO HE TENIDO LA OPORTUNIDAD DE OBSERVARLO Y DESCUBRIR QUE LOS DISEÑOS QUE ANTERIORMENTE SÓLO ERAN PARA PERESENTAR LA INFORMACIÓN DE MANERA SIMPLE, TAMBIÉN HAN SUFRIDO CAMBIOS Y EN LA ACTUALIDAD SE PRESENTAN CON UN CONTENIDO DINÁMICO E INTERACTIVO LOGRANDO QUE LA SIMPLEZA DE UN SITIO WEB NO EXISTA.

PERO ME HE TOPADO CON UN PROBLEMA A RESOLVER, DICHA DIFICULTAD ES LA FALTA DE INFORMACIÓN QUE EXISTE ACERCA DE LAS HERRAMIENTAS QUE HACEN POSIBLE QUE EL CONTENIDO DE UN SITIO WEB SE TRANSFORME EN ALGO DINÁMICO Y FUNCIONAL.

FORMULACIÓN DEL PROBLEMA

EL PRESENTE TRABAJO PRETENDE RECABAR LA SUFICIENTE INFORMACIÓN QUE PERMITA CREAR UNA REFERENCIA EN LA CUAL SE PUEDA ESTABLECER BASES SUSTENTABLES ACERCA DE LA CREACIÓN DE SITIOS WEB DINÁMICOS, YA QUE EN LA ACTUALIDAD LA INFORMACIÓN QUE NOS LLEGA A MÉXICO DEL EXTRANJERO CUENTA CON UN RETRAZO TECNOLÓGICO LO CUAL HACE QUE DICHA INFORMACIÓN NO SEA SUFICIENTE PARA CONOCER COMO TRABAJA ESTA TECNOLOGÍA.

OBJETIVO GENERAL

PROPORCIONAR UNA REFERENCIA DE CONSULTA QUE PERMITA CONVERTIR SUS SITIOS WEB EN ALGO MÁS QUE UNA SIMPLE VERSIÓN EN LÍNEA, CON LA FINALIDAD DE AÑADIR "CONTENIDO DINÁMICO".

PROPORCIONAR UNA GUIA O REFERENCIA DE CONSULTA QUE PERMITA TERNER LAS BASES PARA PODER CREAR SITIOS WEB DINÁMICOS Y NO UN SIMPLE DOCUMENTO DE TEXTO.

PARA PODERLO LLEVAR A CABO, RECABARE LA SUFICIENTE INFORMACIÓN DE DIVERSOS DOCUMENTOS, PERMITIENDO ASÍ QUE TODO AQUEL QUE CONSULTE ESTE DOCUMENTO, LE RESULTE FACIL ENTENDER CÓMO CREAR UN SITIO WEB DINÁMICO.

METODOLOGÍA DE LA INVESTIGACIÓN

LA AUSENCIA DE MODELOS EXPLICATIVOS ACERCA DE CÓMO CREAR SITIOS VERDADERAMENTE DINÁMICOS GENERA HOY EN DÍA UN CONFLICTO ENTRE LOS CREADORES DE SITIOS WEB DE NUESTRO PAÍS, Y ADEMÁS SI OBSERVAMOS QUE PARA PODER COMBATIR ESTE TIPO PROBLEMAS TENEMOS QUE ESTAR A EXPENSAS DE LA POCA INFORMACIÓN QUE NOS BRINDAN LOS QUE CREAN LAS HERRAMIENTAS PARA HACER UN SITIO DINÁMICO DE ESTA MANERA NOS DAMOS CUENTA QUE LA INFORMACIÓN ES INSUFICIENTE.

ES POR ELLO QUE ESTA INVESTIGACIÓN PRETENDE SER UN ESFUERZO POR LOGRAR UN ESTUDIO TEÓRICO –METODOLÓGICO PARA ESTABLECER LAS BASES NECESARIAS PARA CREAR UN SITIO DINÁMICO A TRAVÉS DE LAS HERRAMIENTAS QUE PROPONDRÉ, YA QUE EL TRATAR DE ABARCAR TODAS LAS HERRAMIENTAS DESVIARÍA TAL ESTUDIO Y SERIA INCONSISTENTE.

AHORA BIEN, PARA PODER REALIZAR DICHO ESTUDIO ME FUNDAMENTARE EN UNA BASE METODOLÓGICA DE TIPO EXPLORATORIA PARA PROPORCIONAR LA FUENTE NECESARIA QUE PUEDA LOGRAR DESCRIBIR CON CLARIDAD COMO SE PUEDEN CREAR DICHOS SITIOS.

ESTA FORMA DE ACOMETER EL PROBLEMA SERÁ LA MEJOR MANERA DE CREAR UNA REFERENCIA DE CONSULTA QUE HAGA POSIBLE ATACAR LA FALTA DE INFORMACIÓN QUE ES NECESARIA PARA LA CREACIÓN DE UN SITIO WEB TOTALMENTE DINÁMICO EN NUESTRO PAÍS.

INTRODUCCIÓN

1.1 Historia de Internet *(01B)*

1.1.1 Concepto de Internet *(91B)*

Aunque con demasiada frecuencia se confunde el término de Internet con uno de sus componentes, el WWW, los conceptos que subyacen en el sustrato de la llamada "red de redes" son muy amplios y abarcan casi cualquier forma de comunicación electrónica.

El camino hacia Internet

Internet es un hijo de la Guerra Fría. En los años 60 el Departamento de Defensa norteamericano temía que un ataque nuclear destruyera su red de comunicación centralizada. Para contrarrestar esta amenaza empezó a buscar un nuevo diseño más fiable y fuerte.

La idea para la nueva red parte de RAND, una famosa organización de investigación para la defensa: se eliminaría la autoridad centralizada de forma que no fuera posible destruir la red atacando al nodo principal; los mensajes se dividirían en paquetes y cada uno tendría un doble identificador con la dirección de destino y el número de paquete. Cada uno de estos componentes viajaría sin una ruta preestablecida: si el paso por un nodo fuera imposible, se desviaría a otro hasta que finalmente alcanzase su destino. En el nodo final se iría recomponiendo el mensaje a medida que se fueran recibiendo los paquetes que lo forman.

De esta forma se asegura que la destrucción o corte de una parte de la red (incluso una parte importante) no supone la interrupción de la transmisión. Hay otras importantes consecuencias: todos los nodos son iguales, no hay un centro principal y otros subordinados, todos están al mismo nivel; resulta muy sencillo añadir nuevos nodos, incluso nuevas redes a la red principal; y los mensajes no tienen un camino prefijado, de hecho pueden seguir la ruta más larga y caprichosa imaginable y pasa por nodos completamente desconocidos para el emisor y el receptor.

El nacimiento oficial de Internet se fija en 1969 con ARPANET, una red experimental patrocinada por la Agencia de Investigaciones de Proyectos Avanzados (ARPA) del Departamento de Defensa (DoD) norteamericano. El propósito de esta primera red era explorar tecnologías experimentales de conexión enlazando centros de investigación entre sí y acercando recursos remotos como bases de datos y grandes sistemas a investigadores de todo el mundo.

La red contaba originalmente con cuatro nodos, en 1971 tenía 15, y en 1972, 40. El crecimiento continuó durante toda la década, la arquitectura de la red facilitaba que nuevos nodos y redes se conectaran a ella. Además se añadieron otros medios de transmisión como el satélite, lo que hacía aún más heterogénea la red. Esto forzó a que se buscaran nuevos protocolos que garantizaran la conexión entre distintas plataformas.

El resultado de este proceso es TCP/IP que se compone de un sistema para fragmentar y recomponer mensajes (TCP) y otro para direccionar paquetes (IP). Un sistema unificado de direcciones IP empezó a ser aceptado universalmente y pronto (1984) se vió complementado por un esquema de dominios simbólicos que se simplificaba y racionalizaba la identificación de los recursos en la red (DNS).

A principios de los años 80 se desgajó de ARPANET la red exclusivamente militar Milnet, lo que dio un caracter más civil y menos oficial a la red original. Por esas fechas el conjunto de redes patrocinado por el DoD empezaba a ser conocido como DARPA Internet.

Al mismo tiempo que ocurría esto, otras redes operaban de forma independiente como UUCP o USENET conectando sistemas Unix, o CSNET y BITNET como soporte a comunicaciones en el mundo académico y de investigación. Todas estas redes han acabado conectadas a Internet (y no formando parte de ella como muchas veces se dice erróneamente).

En 1986 NFSNET toma el relevo de ARPANET como red para investigadores. Por estas fechas un conjunto de redes privadas se empieza a hacer cargo del grueso de tráfico en la red lo que precipita el final de la presencia gubernamental. Finalmente ARPANET desaparece en 1990 (NFSNET lo hace en 1991) dejando un sucesor en forma de conjunto de redes: Internet.

En estos momentos la presencia técnica y financiera del gobierno norteamericano es prácticamente inexistente. Después de haber puesto en marcha la red para sus propios fines, considera que ha alcanzado un tamaño que le permite mantenerse por sus propios medios.

La medida no ha estado exenta de polémica, pero la Internet comercial es una realidad: CIX (Comercial Internet Exchange) o MCI son los componentes clave de la nueva NII (Nacional Information Infraestructure) norteamericana, que puede ser en breve el máximo exponente de las anunciadas superautopistas de la información.

La evolución de Internet está muy relacionada con la aparición de nuevos servicios sobre la red. Ideada inicialmente para proporcionar conexión remota a otras máquinas en un momento en el que el tiempo de proceso era escaso y caro, pronto se añadió la posibilidad de intercambiar mensajes con otros usuarios: el correo electrónico.

El crecimiento del correo llevó a aparejada la difusión de información por medios masivos, primero a través de las listas de correo y luego con las *News* de USENET.

Otro servicio importante es el de transferencia de archivos, encarnado por *ftp*, que ha sido hasta hace muy poco el más importante de la red.

A medio camino entre la difusión de información y la transferencia de archivos están nuevos servicios como *gopher* o el más popular de todos, WWW. Otros sistemas se han encargado de la búsqueda de información (*WAIS*) o de la transferencia en formatos especiales (IRC o MBONE).

1.1.2 Las dimensiones de la red *(01B)*

En su desarrollo, Internet ha seguido tres fases diferenciadas que se muestran en la Figura 1.01 (Alvares García Alonso. "HTML, CGI, JAVA, servidores...tecnológia WWW", Anaya Multimedia, Madrid 1996. pág.6).

En un primer momento, que coincide con los años iniciales de ARPANET, se trataba de una pequeña red que conectaba a unas decenas de ordenadores con unos cientos de usuarios. Los cuales participaban activamente en el desarrollo de la red y sus herramientas además tenían un conocimiento elevado de los mecanismos que subyacían a ésta. Internet y su cultura son acreedoras del trabajo de estos pioneros. Se estima que al final de esta fase se alcanzó la cifra de 2,000 sistemas conectados.

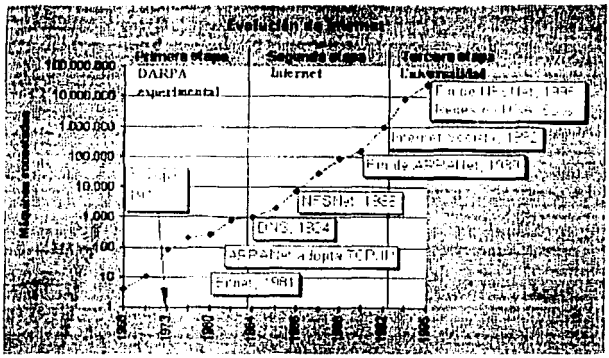


Figura 1.01 Evolución de Internet

La segunda fase, que empieza en los años 80 está marcada por la incorporación de nuevas redes al conjunto inicial. Los sistemas se estabilizan y universalizan. Los usuarios dependen cada vez más de herramientas generales que suplen su desconocimiento del funcionamiento interno de la red. Internet se expande a gran velocidad por el mundo académico y la comunidad científica. Aparecen las primeras compañías privadas (aún en el campo de los desarrollos técnicos). Esta fase termina con la conexión, en ocho años, de 1,000,000 de ordenadores.

La fase en que vivimos, la tercera, está marcada por la difusión generalizada de dos de las aplicaciones de Internet:

a) El WWW
b) El correo electrónico.

Ellas son responsables, en buena parte, de la universalización total de Internet. La red está entrando en las empresas y en muchos hogares y ha hecho variar, entre otras cosas, la forma de entender el negocio del software, haciendo ver que no tiene por qué ser una utopía el disponer de software de dominio público de calidad.
Muchos creen que estamos ante una revolución -más- en la informática, aunque son muchos que ven Internet con desconfianza, y otros simplemente como entretenimiento. Lo único cierto es que el usuario dispone ahora de herramientas comerciales y de un catálogo de servicios muy extenso que no para de crecer, y que las empresas han debido revisar planes y estrategias para adaptarse a algo muy nuevo aunque siempre ha estado ahí.

La cantidad de usuarios de Internet tiene un crecimiento del 160% anual y se extiende, al menos en 93 países (conexión IP, el e-mail llega a 154). El número de usuarios se cuantifica hacia mediados de 1996 en unos 60 millones, aunque el crecimiento es tan rápido que en estos momentos esa cifra puede haber sido rebasada.

Sin embargo, hay que puntualizar que las estadísticas no son tan fiables como sería de desear, distintos estudios presentan cifras muy dispares. Ni siquiera hay acuerdo sobre una posible desaceleración del crecimiento de la red. En la Tabla 1.01 se muestra el resultado actualizado de 1996 del IDS[*], un estudio que se actualiza periódicamente y que se considera en general fiable (Alvares García Alonso. "HTML, CGI, JAVA, servidores... tecnológia WWW", Anaya Multimedia, Madrid 1996, pág.7).

Tabla 1.01 Crecimiento de Internet

Clase de red[**]

Fecha	Hosts	Dominios	A	B	C
Junio 1996	12.881.000	488.000	95	5.892	128.378
Enero 1996	9.472.000	240.000	92	5.655	87.924
Junio 1995	6.642.000	120.000	91	5.390	56.057
Enero 1995	4.852.000	71.000	91	4.979	34.340
Junio 1994	3.212.000	46.000	89	4.493	20.628
Enero 1994	2.217.000	30.000	74	4.043	16.422
Junio 1993	1.776.000	26.000	67	3.728	9.972
Enero 1993	1.313.000	21.000	54	3.206	4.998

La Tabla 1.01 Nos muestra el crecimiento que hemos tenido respecto de la Internet, esto se puede verificar comparando el crecimiento de Host y de Dominios para las diferentes tipos de redes [***] durante los años de 1993 a 1996.

[*] Puede encontrar más información en la página http://www.isc.org/ds/

[**] La clase de red hace referencia en realidad a su amplitud y a su dirección IP. Las redes de tipo A son las más amplias y, por esa razón, las menos numerosas.

[***] Clase A- Para redes muy grandes (hasta 16 Millones de máquinas), Clase B- Redes Grandes (Hasta 65.000 máquinas), Clase C-Redes medianas y pequeñas(hasta 254 máquinas)

Buena parte del crecimiento de la red se atribuye al tráfico de documentación multimedia basada en WWW. Este aumento se manifiesta tanto en el volumen de información transferida como en el número de conexiones realizadas.

Además, rompiendo con la tendencia mantenida en el pasado, las peticiones y transferencias se producen tanto desde grandes sistemas en empresas como desde ordenadores personales en el domicilio de los usuarios finales. Se rompe así el "monopolio" de Internet por parte de centros académicos y de investigación y se entra de lleno en la difusión universal de información y servicios.

Tan importante es conocer el número de usuarios de la red como saber quiénes son estos usuarios. Había una ignorancia casi total en este punto y sólo se aventuraban conjeturas hasta la realización de la encuesta Nielsen,* en la que se hizo un muestreo exhaustivo de la población de Estados Unidos y Canadá. Las conclusiones son muy interesantes (aunque muy discutidas):

Los "internautas" son mayoritariamente hombres (77%), aunque está aumentando rápidamente el número de mujeres, con alto nivel cultural (64% han pasado por la universidad) y con un buen nivel adquisitivo (25% por encima de los 50,000 dórales anuales). Se calcula que el 17% de la población tiene acceso a Internet, aunque sólo un 11% lo emplea; los usuarios de WWW son el 8% de todo el país. El lugar desde dónde se accede se reparte entre el domicilio (64%), el trabajo (54%), y la universidad o centro de enseñanza (30%); no debe dejarse de notar que hay usuarios que se llevan el "trabajo a casa" y hacen doblete. (Alvares García Alonso. "HTML, CGI, JAVA, servidores... tecnológia WWW", Anaya Multimedia, Madrid 1996, pág.8).

En cuanto al uso que se hace de la red, de la encuesta se desprende que un 72% la usa para WWW, un 73% para el correo, que la tarea más habitual es "explorar Internet" con el 90%, que la búsqueda de información varía entre el 65 y 55% (dependiendo de su naturaleza), la obtención de software es utilizada por el 31% de los encuestados, el mismo porcentaje que aprovecha la red para usar otro ordenador, y que, por último, sólo el 14% la usa en estos momentos para telecomprar.

* ACNielsen, líder mundial en investigación de mercados, información y análisis, fue fundada en 1923 por Arthur Conell Nielsen. Desde entonces ha ido extendiendo los productos y servicios, encontrándose presente en más de 100 países.

1.1.3 Los dominios de la red *(01B)*

Los distintos elementos presentes en la red se identifican por una serie de direcciones numéricas que, en realidad, poca gente emplea. En su lugar se usan unas direcciones simbólicas basadas en el DNS (Domain Name System). DNS es un sistema de bases de datos de nombres y direcciones rápidas por todo el mundo. Estas bases de datos se encargan de traducir las direcciones simbólicas DNS en direcciones numéricas IP.

Las direcciones DNS se basan en el uso de una jerarquía de dominios. En el primer nivel se encuentran seis dominios especiales propios de Estados Unidos junto a dominios nacionales dónde cada país se representa con dos letras. (Alvares García Alonso. "HTML, CGI, JAVA, servidores... tecnológia WWW", Anaya Multimedia, Madrid 1996, pág.8).Los dominios especiales son:

a) com (empresas)
b) edu (centros de enseñanza)
c) gob (organismos públicos)
d) mil (defensa)
e) net (redes)
f) org (organizaciones).

Ejemplos de dominios nacionales son:

a) mx (México)
b) uk (Reino Unido)
c) de (Alemania)
d) ca (Canadá)
e) au (Australia)
f) it (Italia)
g) jp (Japón)
h) fr (Francia)
i) es (España)
j) us (Estados Unidos).

Una organización se puede registrar dentro de uno de los dominios anteriores según su localización geográfica* y actividad, y a partir de ese momento podemos crear nuestros propios subdominios. Por ejemplo, la Universidad Nacional Autónoma de México cuenta con su propio dominio que es: **unam.mx** (la dirección se construye disponiendo los dominios de izquierda a derecha a medida que se hacen más específicos), a partir de ese momento podría crear sus propios subdominios.

Por ejemplo; para representar la seccion de Cómputo y telecomunicaciones podría usar la dirección **unam.mx/servicios/computo**, y para la representación de Radio UNAM sería **unam.mx/radiounam/**

* Aunque no necesariamente. En México hay muchas empresas, especialmente multinacionales, que utilizan el dominio "com".

Tradicionalmente, el predominio ha correspondido al dominio .edu, a fin de cuentas la universidad (y en concreto la norteamericana) ha sido el principal impulsor de Internet. Sin embargo, y como prueba de los cambios en el uso de la red, el mayor número de máquinas corresponde en la actualidad al dominio .com. La Tabla 1.02 contiene datos de la ya mencionada IDS actualizados a Julio de 1996 con el número de ordenadores por dominio. (Alvares García Alonso. "HTML, CGI, JAVA, servidores... tecnológia WWW", Anaya Multimedia, Madrid 1996, pág.10).

Tabla 1.02 Principales dominios Internet

Dominio	*Significado*	*Número de Ordenadores*
com	Empresas USA	3,323,647
edu	Universidades USA	2,114,851
net	Redes	1,232,902
uk	Reino Unido	579,492
de	Alemania	579,492
jp	Japón	496,427
us	Estados Unidos	432,727
mil	Defensa USA	431,939
ca	Canadá	424,356
au	Australia	397,460
gob	Administración USA	361,065
org	Organizaciones	327,148
fi	Finlandia	277,207
nl	Holanda	214,704
fr	Francia	189,786
se	Suecia	186,312
no	Noruega	120,780
it	Italia	113,776
ch	Suiza	102,691
za	Sudáfrica	83,349
nz	Nueva Zelanda	77,886
dk	Dinamarca	76,955
at	Austria	71,090
es	España	62,447
...	...	...

Este sistema de dominios podría ser la fuente para tener una guía de Internet, pero no es así. Durante mucho tiempo se mantuvo una lista de ordenadores conectados a la red, pero el rápido crecimiento de ésta y su caracter descentralizado han hecho que hoy en día sólo se pueda contar con aproximaciones y que los estudios se parezcan más a encuestas que a verdaderos censos.

1.1.4 Software Público *(01B)*

Una de las mayores contribuciones de Internet al mundo de la informática está en su papel decisivo de el desarrollo del software.

En lo que respecta a sistemas operativos y estándares, a Internet se debe, bien por ser el medio para el que se construyó, o el que contribuyó a popularizarlo, la existencia y difusión del protocolo TCP/IP y todas sus extensiones; de Unix, empezando por BSD y acabando por Linux; del sistema de ventanas X Windows, padre e inspiración de otras interfases gráficas de usuario: de las aplicaciones para correo electrónico; del formato PostScript para documentación, de los lenguajes Tcl y Perl.

Recientemente la Internet se ha convertido en objeto de atención por sí misma y no sólo por sus productos. El protocolo HTTP, el lenguaje HTML y Java, y aplicaciones como Mosaic y Netscape no habrían tenido razón de ser sin este medio

Pero culturalmente, una de sus contribuciones más importante es la introducción del concepto de software de libre distribución: (programas de calidad desarrollados por personas e instituciones que sin ánimo de lucro o sólo pidiendo una modesta contribución o el reconocimiento de su labor dejan su trabajo a disposición de los restantes usuarios de la red). Este software, al igual que Internet debía ser capaz de operar en distintas plataformas, lo que convierte a la capacidad de adaptarse y funcionar en cada una de ellas en una necesidad.

Lo que era una necesidad para los primeros sistemas Unix conectados a la red ha pasado a ser un elemento habitual para cualquier usuario de ordenadores. El software de libre distribución y el *shareware* se encuentran por todas partes. Además se trata de aplicaciones desarrolladas con gran calidad y con funcionalidades similares y en muchos casos superiores a las de programas comerciales. A pesar de que sólo una pequeña parte de los usuarios de *shareware* acaba pagando derechos a los autores del software, el número de títulos no deja de aumentar y se multiplican las ediciones de CD-ROM con colecciones de software de libre distribución.

El campo del software público y el *shareware* han dado lugar a interesantes fenómenos. Es clásico el ejemplo de Doom y sus secuelas. Un juego que se ha distribuido masivamente y que ha creado toda una subcultura dentro del mundo de los juegos de ordenador. Además, la existencia de software al que podemos acceder libremente desde la red no impide que se puedan vender versiones comerciales y esto resulte un buen negocio, un gran contraste con la habitual venta de software caro y falto de funcionalidades.

Posiblemente el área más beneficiada por el uso de este tipo de aplicaciones sea el propio WWW, dónde desde servidores y *browsers* hasta herramientas de calidad o editores pueden ser aplicaciones no comerciales.

1.1.5 Servicios disponibles en Internet *(01B)*

Hay un gran número de servicios que siguen resultando muy útiles y que jugaron un papel decisivo en las etapas tempranas de desarrollo de la red. Además, hay que tener en cuenta que el propio Web hace uso de ellos pues los *browsers* unen a la propia navegación en el WWW la capacidad de recibir y enviar correo, leer las *News*, recibir archivos de FTP o recoger archivos *gopher*, otros de los servicios que se han utilizado son los siguientes:

- **E-mail.**
- **FTP**
- **Telnet**
- **News**
- **Gopher**
- **Archie**
- **WWW**

Estos servicios se verán más a detalle en el capítulo IV.

1.2 La World Wide Web *(02R)*

1.2.1 El proyecto World Wide Web *(02R)*

En 1989 la red mundial de datos, el memex global, ya existía en potencia. La Internet, que se originó en el ámbito militar durante la guerra fría (Hardy 1993), se había desarrollado más allá de los propósitos originales como resultado de su uso por parte de la comunidad científica internacional, que necesitaba nuevos sistemas de distribución de la información. Lo único que se requería, eran vías de acceso sencillas y homogéneas. Este era uno de los objetivos que Tim Berners-Lee se planteó en 1989 cuando presentó a sus superiores del CERN (Consejo Europeo para la Investigación Nuclear) la propuesta original para el proyecto World Wide Web. Otro era la posibilidad de gestionar conexiones no lineales.

"World Wide Web" (abreviado "Web"; escrito también "WWW" o incluso "W3") significa algo así como "red (o telaraña) global". La propaganda oficial del CERN lo define como un "sistema hipermedia distribuido" (Boutell 1994). En principio se pensó como un medio para la distribución de la información entre equipos de investigadores geográficamente dispersos; concretamente se dirigía a la comunidad de físicos de altas energías vinculados al CERN (Berners-Lee 1994).

En su primera propuesta, Berners-Lee exponía las desventajas del uso de sistemas incompatibles e inconexos: "En el CERN, una diversidad de datos está ya disponible: informes, datos experimentales, datos personales, listas de direcciones de correo electrónico, documentación informática, documentación experimental y muchos otros conjuntos de datos están girando continuamente en discos de ordenadores.

Es sin embargo, imposible "saltar" de un conjunto a otro de una manera automática: una vez encontrado el nombre de Joe Bloggs se lista en una descripción incompleta de algún software en línea, no se encuentra directamente su dirección actual de correo electrónico. Usualmente, tendrás que utilizar un método de consulta distinto en un ordenador distinto con un interface distinto. Una vez que has localizado la información, es difícil guardar sus conexiones o hacer una anotación privada que puedas después encontrar rápidamente."

La conclusión era que "hay un enorme beneficio potencial en la integración de una variedad de sistemas de un modo que permita a los usuarios seguir conexiones que apuntan de un elemento de información a otro".

Se pretendía pues que los recursos disponibles en formato electrónico, que residen en ordenadores distintos conectados a la red, fuesen accesibles para cada investigador desde su terminal, de un modo transparente y exento de dificultades, sin necesidad de aprender a utilizar varios programas distintos. Además, debería posibilitarse el salto entre elementos de información conexos. Los recursos existentes deberían integrarse en una red hipertextual, distribuida y gestionada por ordenadores.

Las primeras instalaciones del WWW para uso interno del CERN estuvieron listas en 1991. Ese mismo año el sistema se abrió ya a Internet. Desde entonces, para acceder al World Wide Web no se requiere más que una terminal conectada a Internet, pero la máxima facilidad de uso y el máximo rendimiento se alcanzan con una pantalla gráfica (un modelo Next o Macintosh, un X-Terminal o una PC con tarjeta gráfica).

Entonces el sistema nos ofrece hipertextos como el que muestra la Figura 1.02, nodos de la telaraña global. Las palabras subrayadas, y las imágenes recuadradas, son ligas que nos conducen a otros nodos.

Para viajar hasta ellos basta con situarse con el ratón sobre el link y pulsar el botón. El nodo de llegada puede ser otro hipertexto, o también un nodo no hipertextual integrado en la red: un servidor *gopher*, un grupo de netnews, una búsqueda en una base de datos *WAIS*, etc.

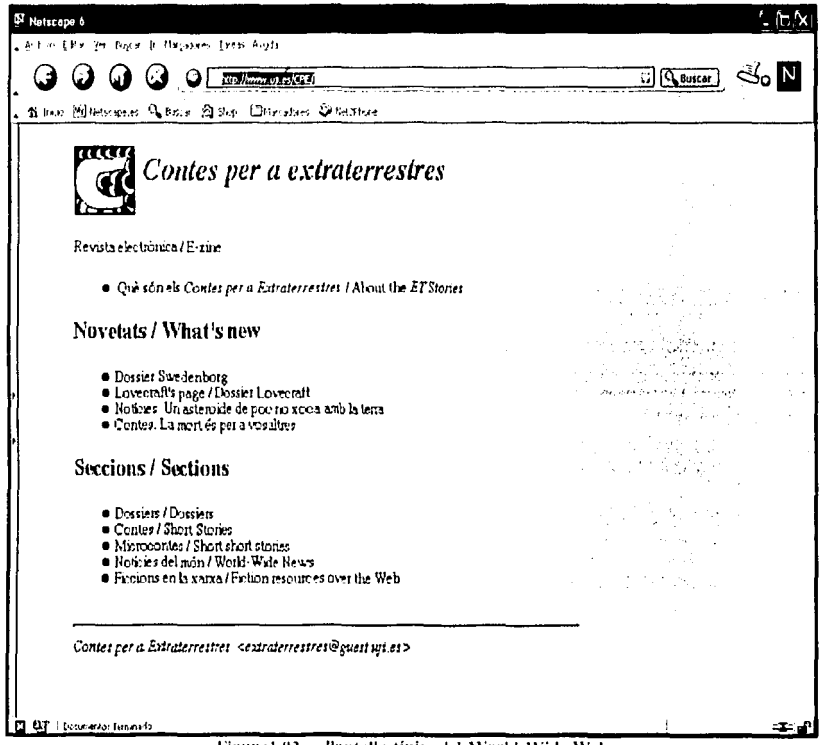


Figura1.02.-- Pantalla típica del World-Wide Web

El éxito del WWW, el crecimiento de la telaraña, ha sido espectacular. Durante 1993 se pasó de 50 a 500 nodos. En 1994 se contabilizan ya miles de servidores en el WWW que distribuyen todo tipo de información.

A finales de la década de los ochenta la interconexión de miles de redes de área local había convertido la Internet en el mayor almacén de datos que jamás hubiese existido, pero también en el más caótico. Las posibilidades eran enormes, pero las dificultades resultaban frustrantes:

a) Formatos incompatibles
b) Programas distintos
c) Protocolos heterogéneos.

Se imponía pues la necesidad de simplificar el acceso a este caudal de información, hacerlo más sencillo y homogéneo. *WAIS*, desarrollado a partir de 1989 por un grupo de empresas sólo fue una solución parcial, los datos debían indexarse con el nuevo software y distribuirse por medio de un nuevo protocolo, es decir, había que realizar un trabajo de adaptación de lo ya existente al nuevo sistema.

El *Gopher* de la Universidad de Minnesota, ampliamente difundido desde 1991, aportó algo más; por medio de un sistema simple de ventanas (o de menús) se accede a todo tipo de archivos de texto, imágenes, bases de datos, etc., sin tener que preocuparse por su localización física en la red, el formato o el protocolo de recuperación: *ftp* y *wais*, por ejemplo, son protocolos que el *gopher* maneja desde el principio, además del suyo propio.

1.2.2 Hipertexto e hipermedia *(02R)*

La experiencia de la proliferación del conocimiento y de la angustia derivada de no poder abarcarlo todo no es nueva, no ha surgido con los ordenadores y la conectividad. Ya en 1945 Vannevar Bush (Bush, 1945) se lamentaba: "La suma de la experiencia humana se está expandiendo a un ritmo prodigioso y los medios que utilizamos para seguir el hilo a través del consiguiente laberinto de ítems momentáneamente importantes son los mismos que usábamos en los días de los barcos de vela".

En su opinión el problema no era tanto una cantidad excesiva de publicaciones como el nulo avance de las tecnologías con que se gestionaba su manejo. Con los rudimentos tecnológicos de su época en mente, Bush fue capaz de idear un sistema llamado memex que permitiría archivar el conocimiento de un modo más eficaz: una especie de escritorio futurista en el que se guardarían, microfilmados, libros, actas, archivos, etc.

Cada elemento de información se visualizaría en pantalla tecleando su código memotécnico correspondiente y, esto es lo más importante, podríamos registrar las conexiones observadas entre elementos distintos. Un usuario del memex que contase con una buena base de datos podría anotar conexiones entre, un artículo de enciclopedia sobre el escritor angloamericano H. Ph. Lovecraft, una fotografía suya y alguno de sus cuentos. Al leer el artículo, la simple pulsación de un botón le permitiría hojear "El horror de Dunwich" o visualizar la fotografía. Más tarde podría conectar con este conjunto la biografía de Lovecraft escrita por Pierre Bourbonnais.

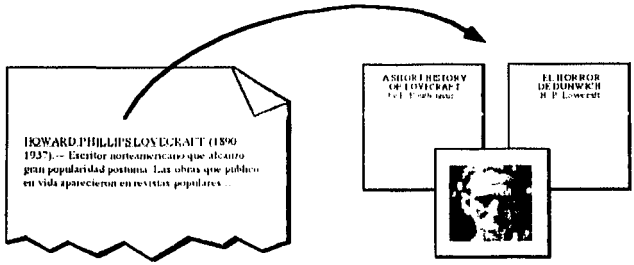


Figura 1.03. Una liga nos conduce a otros textos, imágenes, etc.

Bush remarcaba que este tipo de asociación no lineal de ideas era el modo de funcionamiento natural de la mente humana, y confiaba en que dispositivos semejantes al memex lo reproducirían en el futuro más adecuadamente. Es un hecho que los artículos de una enciclopedia, las notas al pie o las referencias bibliográficas contienen conexiones no lineales de aquel tipo, pero los medios tradicionales resultan inadecuados para gestionarlas.

Cuando nos encontramos con una referencia bibliográfica que nos interesa, todo lo que podemos hacer es acudir a una biblioteca o una librería. Con el memex, idealmente, pulsaríamos un botón para consultar en nuestra pantalla el libro en cuestión.
En el futuro, profetizaba Bush, las enciclopedias serían redes de conexiones que el usuario podría anotar y modificar a su antojo.

Bush era un visionario. En 1945 sus ideas no eran técnicamente realizables. Ni lo eran aún en 1965, cuando otro visionario, Ted Nelson, las ordenó conceptualmente. Fue Nelson quien acuñó el término "hipertexto" para referirse a "un cuerpo de material escrito o gráfico interconectado de un modo complejo que no se puede representar convenientemente sobre el papel; puede contener anotaciones, adiciones y notas de los estudiosos que lo examinan" (Nelson 1965).

La idea es que el lector examina los nodos de una red, y pasa de unos a otros siguiendo las conexiones (links, en inglés). El hecho de que los nodos pueden contener texto, pero también pueden integrar otros medios: imagen, sonido, etc. es lo que se quiere remarcar con otro término complementario: "hipermedia".

Durante las dos décadas siguientes se vivió el auge de los ordenadores, el almacenamiento digital y las redes. El propio Nelson cobró conciencia de lo apropiado de estas nuevas tecnologías para la realización del sueño de una red de elementos de información libremente accesible alrededor del mundo. Sin embargo, se diría que sus ideas sólo han llegado a concretarse recientemente con el World Wide Web.

1.2.3 La arquitectura de la World Wide Web *(02R)*

El diseño del World Wide Web sigue el modelo cliente-servidor: un paradigma de división del trabajo informático en el que las tareas se reparten entre un número de clientes que efectúan peticiones de servicios de acuerdo con un protocolo, y un número de servidores que las atienden (Malkin, 1993). En el Web, nuestras estaciones de trabajo son clientes que demandan hipertextos a los servidores. Para poner en marcha un sistema como éste ha sido necesario:

a) Diseñar e implementar un nuevo protocolo que permitiera realizar saltos hipertextuales, esto es, de un nodo de origen a uno de destino, que podría ser un texto o parte de un texto, una imagen, un sonido, una animación, fragmento de vídeo, etc. Es decir, cualquier tipo de información en formato electrónico. Este protocolo se denomina HTTP (HyperText Transfer Protocol) y es el "lenguaje" que "hablan" los servidores del WWW.

b) Inventar un lenguaje para representar hipertextos que incluyera información sobre la estructura y el formato de representación y, especialmente, indicar origen y destino de saltos hipertextuales. Este lenguaje es el HTML o (HyperText Markup Language).

c) Idear una forma de codificar las instrucciones para los saltos hipertextuales de un objeto a otro de la Internet. Dada la variedad de protocolos, y por tanto, formas de almacenamiento y recuperación de la información, en uso en la Internet, esta información es vital para que los clientes (ver el siguiente punto) puedan acceder a dicha información.

d) Desarrollar aplicaciones cliente para todo tipo de plataforma y resolver el problema de cómo acceder a información que está almacenada y es accesible a través de protocolos diversos (*FTP*, *NNTP*, *Gopher*, *HTTP*, *X.500*, *WAIS*, etc.) y representar información multiformato (texto, gráficos, sonidos, fragmentos de vídeo, etc.). A este fin se han desarrollado diversos clientes, entre los que destaca la familia Mosaic, del NCSA (National Center for Supercomputer Applications) de la Universidad de Chicago, y su sucesor Netscape Navigator, de Netscape Communications Corporation.

Pero, veamos con cierto detenimiento los rasgos más sobresalientes de estos elementos clave del sistema.

1.2.4 HTTP: HyperText Transfer Protocol *(02R)*

El HTTP (HyperText Transfer Protocol) es el protocolo de alto nivel del World Wide Web que rige el intercambio de mensajes entre clientes y servidores del Web. Un protocolo es: "Una descripción formal de los formatos de los mensajes y las reglas que deben seguir dos ordenadores para intercambiar dichos mensajes. Los protocolos pueden describir detalles de bajo nivel de las interfaces de máquina a máquina (por ejemplo, el orden en el cual deben enviarse bits y bytes a través de un cable) o intercambios de alto nivel entre programas (por ejemplo, la forma en que dos programas transfieren un archivo a través de la Internet)." (Malkin y LaQuey Parker, 1993, pág. 39).

El HTTP es un protocolo genérico orientado a objetos que no mantiene la conexión entre transacciones (Berners-Lee, 1993). Ha sido especialmente diseñado para atender las exigencias de un sistema hipermedia distribuido como es el World Wide Web. Sus características principales son:

- **Ligereza:** reduce la comunicación entre clientes y servidores a intercambios discretos, de modo que no sobrecarga la red y permite saltos hipertextuales rápidos.

- **Generalidad:** puede utilizarse para transferir cualquier tipo de datos, según el estándar MIME. Esto incluye también los que desarrollen en el futuro, ya que el cliente y el servidor pueden negociar en cualquier momento el modo de representación de los datos, el cliente notifica al servidor una lista de formatos que entiende, y en adelante el servidor sólo remitirá al cliente datos que este sea capaz de manejar. El cliente debe aceptar al menos dos formatos:

  a) text/plain (texto normal)
  b) text/html (hipertexto codificado en HTML, el lenguaje en el que se escriben los hipertextos del Web).

- **Extensibilidad:** contempla distintos tipos de transacción entre clientes y servidores, y la futura implementación de otros nuevos. Esto abre posibilidades más allá de la simple recuperación de objetos de la red: búsquedas, anotaciones, etc.

El esquema básico de cualquier transacción HTTP entre un cliente y un servidor es el siguiente (Berners-Lee, 1993):

Conexión: El cliente establece una conexión con el servidor a través del puerto 80 (puerto estándar), u otro especificado.

Petición: El cliente envía una petición al servidor.

Respuesta: El servidor envía al cliente la respuesta (esto es, el objeto demandado o un código de error).

Cierre: Ambas partes cierran la conexión.

1.2.5 HTML: HyperText Markup Language *(02R)*

El HTML (HyperText Markup Language) es el lenguaje en el que se escriben los hipertextos del World Wide Web. Cumple la norma SGML, y permite añadir a un documento de texto:

1. La especificación de estructuras del texto. Por ejemplo, títulos, encabezamientos, límites de los párrafos, listas de elementos.
2. Estilos: texto enfatizado, citas, etc.
3. Objetos multimedia: imágenes o sonido.
4. Conexiones hipertextuales a otros objetos de la red: partes sensibles del documento desde dónde podríamos saltar a otras partes del Web.

Todo este "valor añadido" al texto se codifica como etiquetas ("tags") que se insertan en el propio texto. La Figura 1.04 muestra un ejemplo que nos permitirá hacernos una idea de todo ello.

```
<HTML>

<HEAD>
<TITLE>IT stories contents
</TITLE>
<BASE HREF="http://www.uji.es/CPE/">
</HEAD>

<BODY>

<H1><IMAGE SRC="et.gif" ALIGN=middle>
    (I>Contes per a estrateerrestres</I></H1>

<P> Revista electr&ograve;nica / E-zine

<MENU>
<LI><A HREF="signatures/estrateerrestres.html">
    Qu&egrave; s&oacute;n els (I>Contes per a Estrateerrestres</I> /
    About the (I>IT Stories</I></A>
</MENU>

<H2>Novetats / What's new</H2>

<MENU>
<LI><A HREF="dossiers/swedenborg/index.html">Dossier Swedenborg</A>
<LI><A HREF="dossiers/hpl/index.html">Lovecraft's page / Dossier Lovecraft</A>
<LI>Hot&iacute;cies
    <A HREF="news/asteroide.html">
      Un asteroide de poc no xoca amb la terra</A>
<LI>Contes
    <A HREF="contes/mort.html">
      La mort &eacute;s per a vosaltres</A>
</MENU>

<H2>Seccions / Sections</H2>

<MENU>
<LI><A HREF="dossiers/index.html">Dossiers / Dossiers</A>
<LI><A HREF="contes/index.html">Contes / Short Stories</A>
<LI><A HREF="microcontes/index.html">Microcontes / Short short stories</A>
<LI><A HREF="news/index.html">Hot&iacute;cies del n&oacute;n / World-Wide News</A>
<LI><A HREF="web/index.html">Ficcions en la xarxa / Fiction resources over the Web</A>
</MENU>

<HR>
<ADDRESS>
<A HREF="signatures/estrateerrestres.html">
 Contes per a Estrateerrestres</A>
<A HREF="mailto:estrateerestres@guest.uji.es">
&lt;estrateerrestres@guest.uji.es&gt;</A>
</ADDRESS>

</BODY>

</HTML>
```

Figura 1.04. Documento HTML

Las etiquetas del HTML se delimitan por medio de los signos "<" y ">". Por ejemplo, la etiqueta `<P>` marca el inicio de cada párrafo. Otras, la mayor parte, van por parejas: `<TITLE>` y `</TITLE>` abren y cierran, respectivamente, el título del documento.

Los ligas se abren y cierran con las etiquetas `<A>` y `</A>`. El objeto de la red a dónde nos lleva el link se codifica en la etiqueta de apertura por medio de una notación que se ha convertido de hecho en un estándar de Internet: los llamados URL.

1.2.5 URL: Uniform Resource Locator *(02R)*

Los URL (Uniform Resource Locator) son una notación estándar para la especificación de recursos presentes en Internet. Constituyen la piedra angular del Web, ya que hacen posible que un link de HTML se refiera a cualquier objeto de la red.

Un URL representa de un modo compacto la localización y el método de acceso de cualquier recurso de la red (Berners-Lee, Masinter y McCahill, 1994). No sólo hay más de dos millones de ordenadores conectados a los varios miles de redes que forman la Internet, sino que existen múltiples protocolos o formas diferentes de acceder a la información (*ftp*, *gopher*, *http*, etc.). Los URL aportan esos dos datos esenciales: dónde se encuentra un recurso y cómo se puede acceder a él.

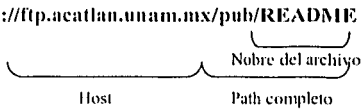
La sintaxis de los URL es la siguiente:

URL::<esquema>:<parte-específica-del-esquema>

El esquema es un término convenido que representa el método de acceso a un recurso. La parte específica del esquema informa sobre su localización en la red, de un modo que depende de cada método de acceso. Un ejemplo nos ayudará a entender esto.

Cuando utilizamos *ftp* anónimo para copiar un archivo de un ordenador remoto a nuestro ordenador necesitamos saber el: host o nombre del ordenador remoto dónde se encuentra el archivo y path o vía que conduce al archivo dentro de la estructura de archivos del ordenador remoto.

Supongamos que el archivo se llama README, y que está en el directorio pub del host ftp.acatlan.unam.mx; el URL de tal objeto sería éste:



Al recuperar un archivo mediante *ftp* anónimo usamos "anonymous" como nombre de usuario, y nuestra dirección de correo electrónico como password. En los URL esta información se omite dado que es conocida. Sin embargo, es posible incluirla si, por ejemplo, no se trata de *ftp* anónimo, sino que se necesita especificar un usuario real y su password.

La sintaxis genérica de los URL para objetos accesibles por *ftp* es la siguiente:

URL:ftp://[user[:password]@host[:port]/path[;type=<typdecode>]

El "port" puede omitirse si el servidor de *ftp* emplea el port estándar de *ftp* (que es el 21). Este principio de omitir lo ya conocido se sigue en todos los URL. Si los distintos servidores siguen las recomendaciones de la Internet no es necesario incluir información redundante.

El "path" es la lista ordenada de subdirectorios por los que hay que pasar para llegar al archivo, separados por "/", seguida del nombre del archivo.

El "type" es "d", "a", "i".

1) "d" indica que se requiere la transmisión de una lista de nombres de archivos (un directorio).
2) "a" solicita una transmisión de líneas de texto.
3) "i" solicita una transmisión binaria.

En la actualidad existen esquemas definidos para los siguientes servicios:

Esquema	Sintaxis
ftp (File Transfer Protocol)	ftp://user:password@host:port/path;type=<typecode>
http (HyperText Transfer Protocol)	http://<host>:<port>/<path>?<searchpart>
gopher (gopher)	gopher://<host>:<port>/<gopher-path>
mailto (correo electrónico)	mailto:<rfc822-addr-spec>
news (USENET news)	news:<newsgroup-name>
nntp (USENET news especificando un servidor nntp, NetNews Transfer Protocol)	nntp://<host>:<port>/<newsgroup-name>/<article-number>
wais (Wide Area Information Server)	wais://<host>:<port>/<database> o wais://<host>:<port>/<database>?<search> o wais://<host>:<port>/<database>/<wtype>/<wpath>

Ejemplos

- *ftp://ftp.uji.es/pub/archivo.doc*
- *http://www.uji.es*
  (URL de la página de entrada del servidor Web del Departamento de Educación de la Universitat Jaume I, en el host www.uji.es)
- *mailto:jordi@edu.uji.es*
  (Este URL posibilita el envío de un mensaje de correo electrónico a la dirección jordi@edu.uji.es)

- *news:comp.infosystems.gopher*
  (URL del grupo de news comp.infosystems.gopher)

- *nntp://news.uji.es/comp.infosystems.gopher*
  (Este URL especifica el grupo de *news* comp.infosystems.*gopher* almacenado en el servidor news.uji.es)

- *wais://wais.uji.es/tractatus?ethics*
  (Este URL especifica la búsqueda del término "ethics" en la base de datos "tractatus" del servidor *WAIS* wais.uji.es)

La utilidad, y la necesidad, de una notación que, como ésta, introduzca algo de orden en el caos de la red es obvia. Los URL se idearon para un proyecto concreto y limitado, el del WWW, pero ha cundido el ejemplo. Ahora mismo se está produciendo un amplio debate en el seno de Internet, concretado en un grupo de trabajo de la IETF (Internet Engineering Task Force) para el desarrollo de sistemas universales de designación y caracterización de objetos persistentes de la red, inspirados en los URL pero que irían más allá: debería ser posible, por ejemplo, asignar un URN (Uniform Resource Name) invariable para un objeto, aunque cambiara su path e incluso su método de acceso. Un sistema distribuido (similar al DNS o Domain Name System) resolvería un URN en uno o varios URL aplicando criterios de optimización de recursos (como proximidad al solicitante).

1.2.6 La interfase de usuario del Word Wide Web *(02R)*

Dado que los nodos que forman el Web atienden peticiones en protocolos distintos, los programas cliente del Web (también llamados "Web *browsers*") deben ser lo más parecido a un cliente universal capaz de presentar al usuario cualquier recurso de la red, dado su URL.

Actualmente existe un número de "Web *browsers*" para distintos sistemas y plataformas que satisfacen aquel requisito en mayor o menor medida.

El más popular ha sido quizá el NCSA Mosaic, del National Center for Supercomputing Applications de la Universidad de Illinois, con versiones para X Windows, Macintosh y MS Windows, sustituido recientemente por Netscape Navigator, de Netscape Communications Corporation. Aunque hoy en día existen bastantes Web *browsers* como los son: Internet Explorer, el Nestcape, el MSN Explorer, Opera, Neoplanet

1.2.7 La Internet como telaraña: El World Wide Web *(02R)*

Al principio se ha presentado el World Wide Web como un proyecto de integración de recursos de la red. Después se ha dicho que con el Web se cumplían los viejos anhelos de Vannevar Bush. Nos parecerá obvio que un sistema de acceso a la Internet debe ser hipermedia, porque la información no se ceñirá a un sólo medio y porque ha de ser posible seguir las conexiones entre los elementos. Pero, ¿Es el Web el sistema hipermedia perfecto?

En algunos aspectos fundamentales dista de serlo. Sobre todo, en la posibilidad de personalizar las conexiones entre los elementos de información. Realizar anotaciones para comentar las páginas no es suficiente, y una adecuada personalización de páginas exige que escribamos nuestro propio código HTML, lo que no está (¿aún?) al alcance de todos.

El HTML, por su parte, también tiene puntos flacos. El hecho de que las marcas se integren en el propio texto dificulta el mantenimiento de éste. La modificación del texto hace necesario volver a aplicar las marcas.

También es arduo mantener las ligas, pero esto no es tanto un problema del HTML como del sistema de URL. Ya se ha dicho que se intenta superar los URL mediante la especificación de URN: nombres permanentes de objetos, independientes de sus localizaciones y métodos de acceso transitorios, que unos servidores de nombres resolverían en los URL correspondientes.

Finalmente, parece que no basta con el acceso hipermedia a la red. La Internet continúa siendo un almacén caótico. Sólo se ha ordenado el interface de usuario, el acceso a los datos, pero estos continúan desordenados. Para solventar este desorden se requieren sistemas de indexación y catalogación que pueden estar basados en los actuales, como *WAIS*.

1.2.8 Sitios Web *(02R)*

El concepto de los sitios Web

Una Página de Internet o Página Web es un documento que contiene información específica de un tema en particular y que es almacenado en algún sistema de cómputo que se encuentre conectado a la red mundial de información denominada Internet, de tal forma que este documento pueda ser consultado por cualesquier persona que se conecte a esta red mundial de comunicaciones. Un Sitio Web es un conjunto de páginas Web relacionadas entre sí.

Los principales beneficios de tener un Sitio Web

Los beneficios de tener un Sitio Web son simplemente facilitar la comunicación entre gobiernos, instituciones educativas, empresas, asociaciones y personas físicas, con el propósito de establecer una relación aún más estrecha entre ellos. Particularmente en el aspecto comercial, fomentar una mayor comunicación entre clientes y empresas estableciendo un modelo de operación del negocio más orientado al cliente.

Algunos de los softwares que se utilizan para desarrollar sitios Web son:

- **Perl**

Perl que significa "Practical Extraction and Report Language" es un lenguaje de programacion medianamente nuevo, el cual surgio de otras herramientas de UNIX como son: sed, grep, awk, c-shell. Principalmente sirve para labores de procesamiento de texto, lo cual lo hace en una forma facil y clara, no como es en C o Pascal; tambien sirve para la programacion de software de sistemas; y ahora ultimo se ha consolidado como "el" lenguaje para programar aplicaciones para WWW, por ejemplo un programa que ponga en pantalla las notas de un alumno dada su matricula

- **PHP**

PHP, acrónimo de "PHP: Hypertext Preprocessor", es un lenguaje interpretado de alto nivel. La mayoría de su sintaxis es similar a C, Java y Perl, con solamente un par de caracteristicas PHP específicas. La meta de este lenguaje es permitir escribir a los creadores de páginas web, páginas dinámicas de una manera rápida y fácil.

- **Flash**

Flash es el software de creación más avanzado para crear animación interactiva escalable para el Web Tanto si crea logotipos animados, controles de navegación de sitios Web, animaciones de gran formato o sitios Web completos de Flash, descubrirá que la capacidad y flexibilidad de Flash es el medio ideal para desarrollar su propia creatividad

- **ASP**

ASP es una tecnología desarrollada por MS para crear páginas Web de contenido dinámico apoyándose en scripts ejecutados en el servidor. Básicamente una página ASP es una mezcla entre una página HTML y un programa que da como resultado una página HTML que es enviada al cliente (navegador).

- **JavaScript**

JavaSript es un lenguaje compacto, y basado en objetos, diseñado para el desarrollo de aplicaciones cliente-servidor a través de Internet. Netscape Navigator 2.0 es capaz de interpretar sentencias JavaScript sumergidos en programas CGI.

En una aplicación cliente para un browser, las sentencias JavaScript sumergidas en un documento HTML pueden reconocer y responder a eventos generados por el usuario, como clicks del mouse, información en formularios y navegación de documento a documento.

1.3 Historia de Perl *(04B)*

1.3.1 Orígenes *(04B)*

Perl comenzó como resultado de la frustración de un hombre y, según él mismo expresa, por su inmoderada pereza. Se trata de un lenguaje único en forma que no pueden explicarse simplemente describiendo los detalles técnicos del lenguaje. Perl, es un estado tanto mental como una gramática del lenguaje.

Una de las rarezas del lenguaje es que su nombre ha generado otras definiciones. Originalmente, Perl significó Practical Extraction Report Language (Lenguaje práctico de extracción y reporte). Sin embargo, los programadores también se refieren a él como Pathologically Eclectic Rubbish Lister (Listador de tonterías patológicamente ecléctica). O incluso como Pratically Everything Really Likable (Prácticamente todo lo realmente deseable).

Dediquemos algunos minutos a examinar las fuerzas externas que provocaron el surgimiento de Perl —esto nos dará una visión de la forma en que se *concibió* el uso de Perl—. En el año de 1986, Larry Wall se encontraba trabajando en una tarea que comprendía la generación de reportes a partir de muchos archivos de texto con referencias cruzadas. Como programador de UNIX y debido al problema que representaba la manipulación del contenido de archivos de texto, comenzó a utilizar awk para la tarea. Pronto se vio que awk no estaba hecho para ese trabajo y que no había ningún otro candidato obvio para el mismo, por lo que tuvo que escribir algo de código.

Ahora viene la parte interesante. Larry pudo haber escrito sólo una utilería para manejar el trabajo en particular que le ocupaba y continuar con su vida. No obstante, se dio cuenta de que no pasaría mucho tiempo antes de que tuviera que escribir otra utilería especial para manejar alguna otra cosa que las herramientas tradicionales no pudieran resolver. (Es posible que percibiera que la mayoría de los programadores *siempre* estaban escribiendo utilerías especiales para manejar cosas que las herramientas estándar no podían solventar.)

Así pues, en lugar de desperdiciar más su tiempo, inventó un nuevo lenguaje y escribió un intérprete para él. Si esto parece una paradoja, en realidad no lo es: siempre requiere un poco más de esfuerzo prepararse con las herramientas adecuadas, pero si se hace bien el esfuerzo es redituable.

El nuevo lenguaje enfatizó sobre la administración del sistema y el manejo de texto. Después de algunas revisiones, podía manejar expresiones regulares, señales y también sockets de red. Llego a conocerse como Perl y pronto se hizo popular entre los programadores frustrados y perezosos de UNIX.

1.4 La importancia de utilizar Perl *(03R)*

Perl surgió como una opción para una gran cantidad de herramientas de UNIX en las cuales basa su propia sintaxis, buscando el mínimo sacrificio de su desempeño por una máxima facilidad de programación e integración, sigue la filosofía de mantener un ambiente que sea capaz de detectar y corregir pequeñas omisiones del programador, y de proporcionarle una forma abreviada de realizar múltiples tareas. En una palabra, es una utilería que pretende facilitar el proceso de grandes volúmenes de información sin sacrificar el rendimiento.

Algunas de las ventajas del uso del lenguaje PERL son las siguientes:

- Construcción de pequeños programas que pueden ser usados como filtros para obtener información de archivos, realizar búsquedas.
- Se puede utilizar en varios entornos, como puede ser Windows 95, OS/2, ..., sin realizar cambios de código, siendo únicamente necesario la introducción del intérprete PERL correspondiente a cada sistema operativo.
- También es uno de los lenguajes más utilizados en la programación de CGI scripts, que son guiones o scripts que utilizan el interface CGI *(Common Gateway Interface)*, para intercambio de información entre aplicaciones externas y servicios de información. Como ejemplo de ello tenemos los programas de búsqueda usados por cualquier browser.
- El mantenimiento y depuración de un programa en PERL es mucho más sencillo que la de cualquier programa en C.

1.4.1 ¿Qué fuentes de información existen? *(03R)*

Los libros que son ya clásicos sobre Perl son los libros del camello y la llama de la editorial O'Reilly además, existen magníficas introducciones y manuales de referencia que se pueden obtener vía Internet. Aun cuando es imposible mencionar con precisión las fuentes de información de un medio tan dinámico con algo tan estático como este documento. Debe notarse, además que estas referencias están en inglés.

Para buscar información, Yahoo! por supuesto:
http://www.yahoo.com/Computers_and_Internet/Languajes/Perl/

Fuente de información general y referencia de documentos:
http://www.cpan.org
http://pubweb.nexor.co.uk/public/perl/perl.html
http://www.khoros.unm.edu:80/staff/neilb/perl/perl5.html

Documentación:
http://www-cgi.cs.cmu.edu/cgi-bin/perl-man
http://www.perl.com/perl/info/documentation.html
ftp://ftp.cdrom.com/pub/perl/CPAN/doc/manual/html/index.html

Se debe recalcar que por la misma naturaleza de Perl, los recursos disponibles y las herramientas que se pueden utilizar cambian muy a menudo, por lo que es indispensable dedicar algún esfuerzo a mantenerse al día para evitar un desperdicio mayor de esfuerzo por no utilizar los nuevos recursos disponibles.

1.4.2 Filosofía de Perl *(03R)*

"Hay más de una forma de hacerlo"
-Larry Wall, autor del lenguaje de programación Perl.

Perl no establece ninguna filosofía de programación (de hecho, no se puede decir que sea orientado a objetos, modular o estructurado aun cuando soporta directamente todos estos paradigmas), los objetivos que se tuvieron en cuenta al diseñar la sintaxis de Perl fueron la facilidad de aprendizaje, facilidad de uso y la claridad de código las cuales, se consideran que son necesarias (aunque pueden escribirse programas en Perl complejos e inteligibles si así se desea).

Por si fuese poco, Perl no es ni un compilador ni un intérprete, esta en un punto intermedio, cuando mandamos a ejecutar un programa en Perl, se compila el código fuente a un código intermedio en memoria, se le optimiza (como si fuésemos a elaborar un programa ejecutable) pero es ejecutado por un motor, como si se tratase de un intérprete. El resultado final, es que utilizamos algo que se comporta como un intérprete pero que tiene un rendimiento comparativo al de programas compilados. Sin embargo, ya existen compiladores de Perl con la versión 5.

En fin, Perl no nos forza a nada, pero como es lógico hay ciertas reglas que hay que seguir para facilitar nuestro trabajo:

- **Claridad.** En la mecánica de programación actual, los programas deben de ser entendibles por la persona que nos ayede en tareas de mantenimiento, de lo contrario perjudicamos tanto a nuestros compañeros de trabajo como a nuestra propia libertad para progresar y mantenernos libres de preocupaciones.

- **Indentación.** Una costumbre ya clásica de la programación, y a lo largo de los ejemplos de este documento, indentamos el código dos espacios hacia adelante al abrir cada bloque, y terminamos la indentación al terminar el bloque, de modo que las llaves de apertura y cierre quedan a la vista y en la misma columna, solas en sus renglones (esto incrementa algo el número de líneas, pero facilita de sobremanera la búsqueda y corrección de los diversos bloques de control).

- **Nombres de variables**. Por lo general, se bebe dar la máxima claridad a los nombres de las variables sin hacerlos demasiado grandes, el nombre de los contadores y variables que guardan valores concernientes a un pequeño segmento de código por lo regular son de un par de letras (e1, e2, ... ex para los contadores, s1, s2, para cadenas de entrada etc.) mientras que las variables que afectan a diversos segmentos (a modo de regla, que tienen su definición en una pantalla distinta a dónde se usan). Pero, para que se tenga una manera más adecuada de entender los programas más adelante en el capítulo III se notará que esta regla se rompe, aunque lo más adecuado es seguir esta regla lo más posible.

- **Comentarios**. Para facilitar la comprensión de un programa no hay como explicarlo, y los comentarios son el medio ideal para hacerlo, hay por lo menos tres comentarios que siempre deben incluirse en un programa: ¿Qué hace el programa?, ¿Quién lo escribió? y ¿Cuándo inició y terminó de escribirlo?, sobre todo en el contexto de una organización, estos tres simples comentarios pueden hacer la diferencia entre desechar un programa como indescifrable o dedicarle algún tiempo para revisarlo. Además, es prudente comentar dentro del código la forma en que el programa deberá ejecutarse, parámetros, y su sintaxis, así como comentar las estructuras de control como un modo de explicar la funcionalidad al detalle y recalcar con comentarios las funciones que cumplen las variables.

- **Sencillez**. Es cómodo en ocasiones el comprimir una serie de instrucciones en una sola línea, queda al criterio decidir cuando se gana en claridad con un código más o menos extenso, pero no debe titubearse en comentar el código que sea "comprimido".

CAPÍTULO II

COMPILAR E INSTALAR

OBJETIVO ESPECÍFICO:

DETERMINAR LAS CARACTERÍSTICAS QUE SE NECESITAN PARA INSTALAR PERL, ASÍ COMO ESTABLECER LAS BASES DE MANERA QUE PUEDA GENERARSE UN PRIMER PROGRAMA.

2.1 Obtención e instalación de Perl *(04B) (18R)*

2.1.1 Costo y licencia de uso *(04B) (18R)*

Perl es gratuito. Hay libertad de copiar todo el código fuente y la documentación, compilarlo, imprimirlo y pasarlo a otras manos. Cualquier programa que nosotros escribamos en Perl es nuestro para hacer con él lo que nos plazca; por lo que a Perl concierne, no hay que pagar derechos ni existen restricciones para su distribución.

Sin embargo, no es por completo "del dominio público", y por una buena razón. Si el código fuente fuera por completo del dominio público sería posible que cualquiera le hiciera alteraciones menores, lo compilara y lo vendiera, en otras palabras, que se lo robara a su creador. Por otra parte, sin distribuir el código fuente, resulta difícil asegurarse de que pueda usarlo todo aquel que lo desee.

La Licencia Pública General (GNU) es una forma de distribuir software gratuito sin el peligro que alguien se aproveche de nosotros. Bajo este tipo de licencia, podría distribuirse gratuitamente el código fuente y ser usado por todo el mundo, pero cualquier tipo de programas derivados del uso de dicho código deben ser liberados bajo el mismo tipo de licencia. En otras palabras, si nosotros derivamos el código fuente a partir de un código fuente con licencia GNU, tenemos que liberar el código a todo aquel que lo desee.

Por lo general esto es suficiente para proteger los intereses del autor, pero pude producir un torrente de versiones derivadas del paquete original. Esto podría privar al autor original de una mención en el desarrollo de su propia creación. También puede conducir a confusión por parte de los usuarios finales al hacerse más difícil establecer cuál es la versión definitiva del paquete y si un script en particular funcionará con una versión determinada, y así sucesivamente.

Es por lo anterior que Perl se libera bajo los términos de la licencia "Artística". Ésta es una variante de la (GNU), la cual dice que cualquiera que libere un paquete derivado de Perl debe dejar en claro que no se trata en realidad del paquete Perl. Todas las modificaciones deben indicarse con claridad, renombrar en su caso los ejecutables y distribuir los módulos originales junto con las versiones modificadas. El efecto consiste en que se reconoce claramente al autor original como el "propietario" del paquete. También se aplican los términos generales de la Licencia Pública General GNU.

2.1.2 ¿Tenemos nosotros Perl instalado? *(04B) (18R)*

Resulta imprescindible tener instalado Perl en nuestra computadora antes de continuar, porque al leer los ejemplos querremos probarlos. Si aún no tenemos instalado Perl perderemos tiempo e iniciativa.
Es muy sencillo verificar si nuestro sistema ya tiene instalado Perl. Simplemente debemos escribir en la línea de comandos: `Perl -v`

La Figura 2.01 nos muestra el resultado de poner la línea de comandos `Perl -v`

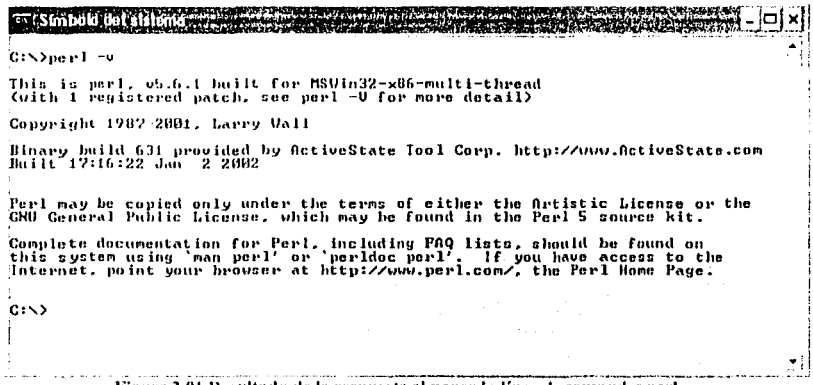


Figura 2.01 Resultado de la respuesta al poner la línea de comandos perl -v

Si obtenemos un mensaje de error o si tenemos una versión anterior como la 4 de Perl, debemos instalar Perl nosotros mismos. La siguiente sección describe la manera de obtener e instalar Perl en diferentes plataformas.

2.1.3 Obtención de Perl *(04B) (18R)(07R)*

Las nuevas versiones de Perl se liberan a través de Internet y se distribuyen a sitios Web y archivos *ftp* en todo el mundo. Por lo general, los binarios UNIX no se ponen a disposición en Internet, por lo que regularmente es mejor construir Perl en nuestro sistema, de modo que podamos estar seguros de que funcione. Al fin y al cabo, todos los sistemas UNIX tienen un compilador C y Perl.

Cada sistema operativo tiene su propia forma de obtener e instalar Perl.

Para UNIX y Linux: La Página Base de Perl (Perl Home Page) contiene un vínculo de software (http://www.perl.com/pub/a/language/info/software.html#unix) que nos permitirá bajar (esto es, tomar) una copia del más reciente código fuente de Perl, esta página explica también porque no están disponibles los binarios de Perl.

Para Windows 95/98/Windows NT: La Página base de hip communications, inc. (http://www.perl.com/pub/a/language/info/software.html#win32) contiene un vínculo para bajar una copia del i86 Release Binary. Este vínculo nos permite bajar una copia de un archivo zip que contiene los archivos de Perl en forma comprimida.

Con los archivos de distribución se incluyen las instrucciones para compilar o instalar Perl en cada sistema operativo.

Requerimientos para instalar Perl en Linux:

Requisitos:

- **Hardware**
  Recomendado 35MB espacio del disco duro para una instalación típica

- **Sistema operativo**
  Mínimo 2.0 Kernel

- **Ayuda en línea**
  Web Browser

Instalando ActivePerl

- **Paquete Debian**

  Este paquete instala perl en: /usr/local/ActivePerl-5.6

  Para instalar:

  ```
  #dpkg -i ActivePerl-5.6.1.630-i686-linux.deb
  ```

- **Paquete RPM**

  Este paquete instala perl en: /usr/local/ActivePerl-5.6

  Para instalar en una situación diferente, use la opción '--prefix' con el comando rpm.

  Para instalar:

  ```
  #rpm -i ActivePerl-5.6.1.630-i686-linux.rpm
  ```

- **Paquete ActiveState**

  Copie ActivePerl-5.6.1.630-i686-linux.tar.gz en un directorio temporal.

  Para instalar:

```
#tar -xzf ActivePerl-5.6.1.630-i686-linux.tar.gz
#cd ActivePerl-5.6.1.630
#sh install.sh
```

Requerimientos para instalar Perl en Solaris:

Requisitos:

- **Hardware**
  Recomendado 35MB espacio del disco duro para una instalación típica

- **Sistema operativo**
  Solaris 2.6

- **Ayuda Online**
  Web Browser

Instalando ActivePerl

- **Formato Pkgadd**

  Este paquete instala perl en: /usr/local/ActivePerl-5.6

  Copie ActivePerl-5.6.1.630-sun4-solaris.gz en un directorio temporal.

  Para instalar:

```
#gunzip ActivePerl-5.6.1.630-sun4-solaris.gzsun4- solaris.tar.gz
#pkgadd -d ActivePerl-5.6.1.630-sun4-solaris
```

- **Instalador Formato ActiveState**
  Copie ActivePerl-5.6.1.630-sun4-solaris.tar.gz en un directorio temporal.

  Para instalar:

```
#tar -xzf ActivePerl-5.6.1.630-sun4-solaris.tar.gz
#cd ActivePerl-5.6.1.630
#sh install.sh
```

Requerimientos para instalar Perl en Windows:

Requisitos

- **Hardware**
  Recomendado 35MB espacio del disco duro para una instalación típica

- **Sistema operativo**

  Windows XP
  - Ningún requisito extra

  Windows 2000
  - Ningún requisito extra

  Windows NT
  - Service pack 5+
  - Microsoft Windows Installer 1.1+ (disponible en: http://download.microsoft.com/download/platformsdk/wininst/1.1/NT4/EN-US/InstMsi.exe)
  - Internet Explorador 5+ (disponible en: http://windowsupdate.microsoft.com)

  Windows ME
  - Ningún requisito extra

  Windows 98
  - Microsoft Windows Installer 1.1+ (disponible en: http://download.microsoft.com/download/platformsdk/wininst/1.1/W9X/EN-US/InstMsi.exe)
  - Internet Explorador 5+ (disponible en: http://windowsupdate.microsoft.com)
  - DCOM para Windows 98 (disponible en: http://www.microsoft.com/com/resources/downloads.asp)

  Windows 95
  - Microsoft Windows Installer 1.1+ (disponible en: http://download.microsoft.com/download/platformsdk/wininst/1.1/W9X/EN-US/InstMsi.exe)
  - Internet Explorador 5 (disponible en: http://windowsupdate.microsoft.com)
  - DCOM para Windows 95 (disponible en: http://www.microsoft.com/com/resources/downloads.asp)
  - MSVCRT (disponible en: ftp://ftp.microsoft.com/softlib/mslfiles/msvcrt.exe

Ejecute el paquete de instalación ActivePerl para iniciar el wizard de la instalación, el cual lo guiará a través de la instalación y le dará opciones sobre qué y dónde instalar el material.

2.1.4 ¿Qué sucede durante la instalación? *(07R)*

Perl es instalado por default en `C:\Perl`. La unidad seleccionada por defecto "`c`", será el mismo sobre la cual el S.O. está instalado o la unidad que tenga el mayor espacio disponible. Esto puede ser modificado con sólo seleccionar el botón Browser y escogiendo la unidad apropiada y el directorio.

Si usted, ejecuta Perl en la línea de comandos la escritura será ejecutada por el primer Perl.exe encontrado en la lista de caminos en la variable de ambiente de la ruta. Para asegurar que el script que usted desea que se ejecute, puede especificar la ruta completa dónde se localiza el Perl.exe que quiere ejecutar (tecleando `perl -v` en el prompt de comandos, y le dirá qué versión de Perl está primero en la ruta).

Instalando ActivePerl cambiará su ambiente de ruta inconstante y pueden cambiar la configuración del registro, como asociaciones de archivo que pueden afectar su Web Server. Si desea usar una copia previamente instalada de Perl, necesitará modificar esta configuración

Ejecutando los scripts Perl en su Web Server

Perl, Perlscript y Perl para ISAPI son en suma todos útiles para su Web Server. Sin embargo ellos tienen requerimientos específicos (requerimientos de sistema), y puede requerir alguna configuración adicional para trabajar apropiadamente. Para más información de compatibilidad, instalación y configuración eche un vistazo en la configuración del Web Server en la sección de FAQ de ActivePerl

Instalando ActivePerl desde la línea de comandos

Usted puede instalar ActivePerl desde la línea de comandos usando el programa "msiexec", por ejemplo

```
c:\> msiexec  /i <msi_file>
```

Seleccionando las características de ActivePerl, Usted puede controlar cuales características son instaladas usando las propiedades de "ADDLOCAL" en la línea de comandos. En esta propiedad puede poner una coma, delimitando la lista de características a ser instaladas localmente, por ejemplo.

```
c:\> msiexec /i <msi_file>ADDLOCAL = "Perl,PerlIS"
```

ActivePerl tiene las siguientes características:

- **PERL_FEATURE** - The Perl core
- **PPM** - Programmer's Package Manager
- **PERLIS** - Perl for ISAPI interpreter
- **PERLSE** - Perl ActiveX Scripting Engine
- **EXAMPLES** - Some simple examples
- **DOCUMENTATION** - ActivePerl documentation

Por defecto se instalan todos los rasgos de ActivePerl en Windows XP

Configurando ActivePerl.

Pueden especificarse las propiedades en la línea de comandos para configurar las opciones de instalación de ActivePerl

Por ejemplo:

```
C:\> msiexec /i<msi_file>TARGETDIR = "c:\Perl" PERL_PATH = "YES"
```

EL instalador MSI de ActivePerl reconoce las siguientes propiedades en la línea de comandos:

- **TARGETDIR:** Usando esta propiedad usted puede especificar la ruta absoluta dónde ActivePerl se instalará. Esta propiedad es opcional. Si no lo especifica, ActivePerl será instalado por default en una localidad determinada por el servicio de instalación de Windows, usualmente la raíz de la unidad con el mayor espacio disponible. El valor de esta opción es requeridá y debe ser la ruta absoluta al directorio en la cual ActivePerl será instalado

- **P._IISMAP:** Poniendo en está propiedad "yes" hace que el instalador cree un script global de IIS que traza para ".pl" y Perl. Ésta es optativa. El valor por defecto es "no".

- **PLX_IISMAP:** Poniendo en esta propiedad "yes", hace que el instalador cree un script

- **Global de IIS:** que traza para .plx" y PerlIs. Esta es opcional. El valor por defecto es "no"

- **PERL_PATH:** Poniendo en esta propiedad "yes", encauza el directorio de Perl/bin a ser agregado a la ruta en un ambiente variable. Esta propiedad es optativa. El valor por defecto es no agregar el directorio Per/bin a la ruta.

Ejecutando el instalador de ActivePerl en modo silencioso:

Para ejecutar el instalador de ActivePerl en modo silencioso, usted necesita especificar la opción "`/q`" en la línea de comandos. Esto causa que el instalador corra sin las opciones de instalación. Por ejemplo:

```
C:\> msiexec /i <msi_file> /q
```

Autentificarse durante la instalación:

Para autentificarse durante la instalación de ActivePerl, usted necesita especificar en la línea de comandos la opción "`/l`". Esta opción requiere que provea el nombre del archivo clave como un argumento. Por ejemplo:

```
C:\> msiexec /i <msi_file> /l <log_file>
```

Para más información puede ver Readme de ActivePerl.

2.2 Bajo qué plataformas se utiliza *(07B) (08B) (18R)*

2.2.1 ¿Dondé Puede Usarse? *(07B) (08B) (18R)*

Las plataformas dónde Perl se ha desarrollado más, son los servidores UNIX, por sus necesidades de administración y lo robusto de su manejo de memoria y de procesos, además de la facilidad de Perl para realizar los así llamados CGI's, interfaces para comunicar recursos del servidor con un servicio de Internet particular (como podría ser WWW o *gopher*).

En otras plataformas, para computadoras personales en particular, se han desarrollado versiones que mantienen un razonable grado de funcionalidad, pero en realidad, el sistema DOS no tiene un manejo lo bastante bueno de los procesos o de la memoria para permitir a Perl dar un buen desempeño, además de que no es común ver en computadoras personales necesidades de administración de la magnitud de un servidor institucional. Sin embargo, puede practicarse la programación de Perl en cualquier computadora personal, o incluso elaborar programas de reporteo en él, sin embargo, es algo que no se ha popularizado hasta hoy.

2.2.2 Plataformas soportadas *(07B) (08B) (18R)*

Perl fue diseñado originalmente para el sistema operativo UNIX y está distribuido generalmente como código fuente en lugar de archivos ejecutables binarios.

Esto permite al usuario la posibilidad de adaptar la instalación a sus necesidades y requisitos y le ayuda también a mantener un cierto grado de portabilidad del lenguaje.

Sin embargo, hoy en día exiten distribuciones binarias de terceras partes. Las más comunes de ellas son Perl para Win32, para sistemas de Windows de 32 bits y MacPerl, para el sistema operativo Macintosh. Junto con estos archivos binarios, el código fuente real se encuentra disponible para puertos diversos, incluyendo OS/2, VMS, Machintosh, Windows de 16 bits y de 32 bits, Windows NT y otros.

En esos puertos no nativos puede encontrarse que algunas de las funciones y otros comandos funcionan de forma diferente o no funcionan en absoluto. Es necesario asegurarnos de comprobar la documentación de Perl para cualquier cuestión de despliegue en esos sistemas y estar atentos ante cualquier desarrollo nuevo que pueda ayudar en la búsqueda.

Para mantener una visión actualizada de los puertos más recientes, puede visitarse la página de inicio de Perl en http://www.perl.com/perl.

2.3 Su primer programa en Perl *(07B) (08B) (18R)*

Su primer programa Perl consiste en un archivo ordinario de texto que contiene una serie de enunciados Perl.

En esta sección veremos cuales son las herramientas necesarias para desarrollar y poner en funcionamiento un programa escrito en Perl, y configuraremos ya un entorno de trabajo en nuestra computadora que nos permitirá ejecutar los primeros programas.

Debemos tener en cuenta que aunque la idea final es escribir código scripts que correrán sobre servidores de páginas Web, también podremos ejecutar programas Perl en nuestra computadora.

2.3.1 Los útiles de trabajo *(07B) (08B) (18R)*

En la lista siguiente tendremos las herramientas o editores que necesitaremos para escribir y ejecutar programas Perl en nuestra computadora.

- Editor de textos capaz de guardar los archivos en "modo texto" (ASCII).
- Intérprete Perl para Windows 95/98/XP o NT (según el entorno empleado).

Para desarrollar programas que se ejecuten en servidores Web necesitarán además:

- Conexión con Internet. (Aunque no es tan necesario)
- Espacio en el disco servidor dónde se alojarán las páginas y programas que realicemos.
- Acceso al intérprete Perl del servidor.
- Aunque no es imprescindible, nos resultará muy útil un editor de páginas Web.

Los elementos que emplearemos y a los que corresponden las figuras y descripciones de los ejemplos son:

- Sistema Operativo: Windows XP
- Editor de MS-DOS: Versión 2.0.026 (incluido en Windows XP)
- Intérprete Perl: Versión v5.6.1

2.3.2 El primer programa Perl en sistemas Windows *(07B) (08B) (18R)*

Una vez instalado el intérprete estamos en condiciones de ejecutar el primer programa.

Para realizar los ejercicios de esta primera parte, nos resultará útil tener abiertos en Windows XP el editor de textos con el que vamos a trabajar, y al mismo tiempo, una ventana DOS desde la que podremos llamar al intérprete para ir probando el código que escribamos.

Al tener los dos programas en ejecución, podremos pasar de uno a otro de forma inmediata pulsando sobre sus iconos en la barra de herramientas de Windows XP.

Suponiendo que el tratador de textos empleado es Editor de MS-DOS, el proceso inicial será el descrito por las siguientes figuras:

Figura 2.02 Ejecución del Editor de MS-DOS

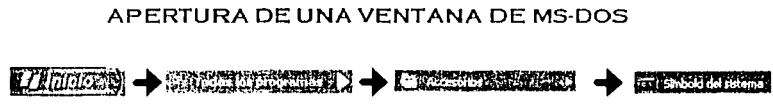


Figura 2.03 Apertura de una ventana de MS-DOS

Seleccionar la ventana del editor y teclear el listado siguiente:

```
#saludo                                    (Comentario)
system "cls";                              (Limpia la pantalla)
print "Mi primer programa";
sleep (8);                    (Detiene la pantalla durante 8seg.)
```

La Figura 2.04 nos muestra como luciría nuestro editor MS-DOS al escribir este listado

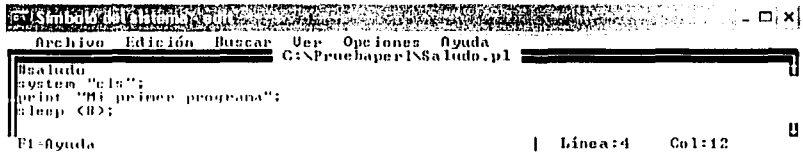


Figura 2.04 Imagen del editor MS-DOS con el listado

Hay que grabar el listado con el nombre `Saludo.pl` en el directorio `c:\Prueperl`.

Pasemos ahora a la ventana MS-DOS y entramos en el directorio `c:\Prueperl`

Tecleamos: `perl Saludo.pl`

Si seguimos los pasos correctamente, en la ventana aparecerá

Mi primer programa, como en la Figura 2.05

Si nos devuelve algún error, revisemos nuevamente todo el proceso.

Si la respuesta ha sido "Comando o nombre de archivo incorrecto" querrá decir que no hemos instalado correctamente el intérprete.

Otro error muy común es olvidar el punto y coma (;) al final de una línea.

La Figura 2.05 nos muestra la compilación correcta de mi primer programa

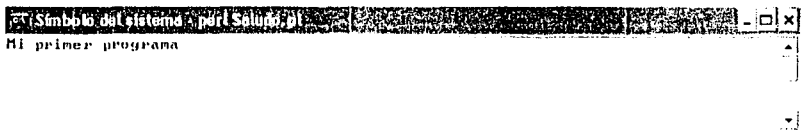


Figura 2.05 Imagen compilación correcta de mi primer programa

Si la respuesta ha sido la esperada... ¡enhorabuena!, ya estamos en disposición de ejecutar en nuestra computadora cualquier programa Perl, y ya veremos como es más fácil de lo que imaginamos.

2.3.3 Creación del programa *(07B) (08B) (18R)*

Un programa Perl consiste en un archivo ordinario de texto que contiene una serie de enunciados Perl. Los enunciados están escritos en lo que parece una amalgama de C, Shell script de UNIX y lengua inglesa. En rigor, eso es lo que es.

El código Perl puede tener un flujo bastante libre. Las amplias reglas sintácticas que determinan dónde comienza y termina un enunciado son:

- Se ignoran los espacios a la izquierda de la línea. Podemos comenzar un enunciado Perl en cualquier punto que deseemos: al inicio de la línea, con sangría para dar claridad (se recomienda), o incluso, si deseamos, alineado a la derecha (definitivamente no se recomienda, ya que el código sería difícil de entender).
- Los enunciados se terminan con un punto y coma.
- Los espacios, tabuladores y líneas en blanco son irrelevantes fuera de las cadenas: un espacio es tan bueno como cien. Esto significa que podemos, con fines de claridad, dividir los enunciados en varias líneas. Una cadena es básicamente una serie de caracteres encerrados entre comillas.
- Se ignora todo lo que aparece después de un signo de libra o gato (#), excepto en las cadenas. Use este signo para condimentar el código con útiles comentarios.

El siguiente es un enunciado de Perl

```
print ("Mi nombre es Engelbert\n");
```

No hay premio en adivinar lo que sucede cuando Perl ejecuta este código: imprime **Mi nombre es Engelbert**. Si la `\n` no le es familiar, no se preocupe: simplemente significa que Perl debe imprimir un caracter de línea nueva después del texto o, en otras palabras, ubicarse al principio de la siguiente línea.

La impresión de mayor cantidad de texto consiste ya sea en ligar enunciados como éste o en dar varios argumentos a la función `print()`.

```
print ("Mi nombre es Engelbert,\n");
print (" Yo vivo en la ciudad de México,\n", "Y trabajo en
Symantec.\n");
```

Así, pues, ¿cómo se ve un programa Perl completo? El siguiente es un pequeño ejemplo completo, con la línea de invocación en la parte superior y unos cuantos comentarios.

```
#!/usr/local/bin/perl -w
print ("Mi nombre es Engelbert,\n");
print (" Yo vivo en la ciudad de México,\n", "Y trabajo en
Symantec.\n");
```

Sin embargo, no se trata de un programa de Perl típico; es sólo una secuencia lineal de comandos sin ninguna complejidad.

Nosotros podemos crear cualquier programa Perl iniciando cualquier procesador de textos:

En UNIX podemos usar emacs o vi.

2.3.4 Invocación *(07B) (08B) (18R)*

Suponiendo que Perl esté instalado correctamente y funcionando en su sistema, la forma más simple de ejecutar un programa Perl consiste en escribir lo siguiente:

```
perl Nombreprograma.pl
```

Nombreprograma debe sustituirse con el nombre del programa que intentemos ejecutar. Si nosotros creamos el programa test.pl, podemos ejecutarlo como sigue:

```
perl test.pl
```

Este ejemplo asume que Perl se encuentra en la trayectoria de ejecución; de no ser así, tendremos que suministrar también la trayectoria completa a Perl. En UNIX, por ejemplo, el comando podría ser.

```
/usr/local/bin/perl test.pl
```

En tanto que en Windows XP, podríamos tener la necesidad de utilizar:

```
C:\perl\bin\perl test.pl
```

Los sistemas UNIX tienen otra forma de invocar un programa. Sin embargo, nosotros necesitaremos hacer dos cosas. La primera es colocar una línea como la siguiente

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

Al comienzo del archivo Perl. Esto indica a UNIX que el resto de este archivo de script se ejecuta mediante `/usr/local/bin/perl`. El segundo paso es hacer que el propio archivo del programa sea ejecutable cambiando el modo:

```
Chmod +x test.pl
```

Ahora podemos ejecutar el archivo del programa en forma directa y dejar que éste indique al sistema operativo qué intérprete al ejecutarlo. La nueva línea de comandos es simplemente:

```
test
```

2.3.5 Comentarios en su programa *(04B)*

Es de suma importancia poner comentarios en los programas Perl. Los comentarios nos permitirán describir la intención más allá de la mecánica de nuestro programa. Por ejemplo, es muy fácil entender que nuestro programa sumara 66 a otro valor. Pero dentro de dos años, nosotros podremos olvidar cómo fue, en primera instancia, que calculó el número 66. Los comentarios se colocan dentro de un archivo de programa utilizando el caracter #. Todo lo que aparece después del # se ignora. Por ejemplo:

```
#Esta línea entera se ignora
print ("Perl es fácil.\n");   #Esta es media línea de comentarios.
```

CAPÍTULO III

PERL BÁSICO

OBJETIVO ESPECÍFICO:

ESTABLECER LA SINTAXIS QUE POR LO GENERAL UTILIZA ESTE INTÉRPRETE, PARA PODER DESARROLLAR Y ENTENDER LOS PROGRAMAS NECESARIOS QUE HACEN POSIBLE EL ESTABLECIMIENTO DE UN CONTENIDO DINÁMICO EN UN SITIO WEB.

3.1 Constantes numéricas y de cadena *(04B)(08B)*

Antes de continuar, se debe aclarar que para entender los conceptos de programación aquí manejados, lo más conveniente es que se tenga un nivel de programación medio o avanzado en conocimiento del lenguaje C, C++ o cualquier otro lenguaje.

En esta sección veremos algunas de las formas en las que Perl maneja los datos. Todos los programas de cómputo usan datos de alguna manera. Algunos los usan para personalizar el programa. Por ejemplo, un programa de correo podría necesitar el recordar su nombre para saludarlo al iniciar. Otro programa —digamos uno que busque archivos en su disco duro — podría recordar los últimos parámetros de búsqueda en caso de que quisiéramos hacer la misma búsqueda otra vez.

Una *constante* es un valor que se representa *tal cual*, o que se indica textualmente en su código fuente. Cuando en los programas vemos los cuatro caracteres 45.5, en realidad se refieren a un valor de cuarenta y cinco y medio. Perl utiliza cuatro tipos de constantes.
La siguiente es una rápida mirada de ellos:

- **Números:** Éste es el tipo de datos más básico.
- **Cadenas:** Una cadena es una serie de caracteres que se manejan como una unidad.
- **Arreglos:** Un arreglo es una serie de números y cadenas que se manejan como una unidad. Se puede pensar que un arreglo es como una lista.
- **Arreglos asociativos:** Éste es el tipo de datos más complicado. Se puede pensar en él como una lista en la que cada valor tiene un elemento de búsqueda asociado.

3.1.1 Constantes numéricas *(04B) (08B)*

Las constantes numéricas se usan con frecuencia. Representan un número con el que su programa necesitará trabajar. La mayor parte del tiempo usará números en base diez[▲], la base que todo el mundo usa. Sin embargo, Perl le permitirá también utilizar números en base 8 (octales)[▲▲], o en base 16 (hexadecimales).[▲▲▲]

[▲] En la notación decimal —o en base diez—cuando se ve un valor de 15 significa (1* 10) +5 ó 15_{10}. El subíndice indica qué base se está empleando

[▲▲] En la notación octal —o en base ocho—cuando se ve el valor 15, significa (1*8)+5 ó 13_{10}.

[▲▲▲] En la notación hexadecimal —o en base 16— cuando se ve el valor 15, significa (1*16)+5 ó 21_{10}. La base 16 requiere de 6 caracteres adicionales además del 0 al 9, de modo que cada posición puede tener un total de 16 valores. Las letras de la A a la F se usan para representar los valores del 11 al 16.
Así que el valor BD_{16} es igual a $(B_{16} * 16) * D_{16} \cup (11_{10} * 16) + 13_{10}$, que es igual a 189_{10}.

Si nosotros utilizáramos números grandes o muy pequeños, encontraremos también de utilidad la notación científica.[*]

Números

Veamos algunos tipos diferentes de números que se pueden utilizar en el código de un programa.

Primero, he aquí algunos enteros.

Un entero. Los enteros son números sin componentes decimales.

*Un entero en formato octal. Este número es 35, u (8*4)+3, en base 10.*

*Un entero en formato hexadecimal. Este número también es 35, o (16*2)+3, en base 10.*

```
123
040
0x23
```

Ahora veamos algunos números y fracciones, también conocidos como *valores de punto flotante*. Con frecuencia veremos que se hace referencia a estos valores como *valor flotante* con fines de sencillez.

Un flotante con un valor en el lugar de los décimos. Puede también expresarse como 100 y $^{5}/_{10}$.

Un flotante con un valor de fraccionario hasta la posición de milésimos. Puede expresarse también como 54 y $^{534}/_{1000}$.

```
100.5
54.534
```

El siguiente es un número muy pequeño

- Un valor flotante muy pequeño. Podemos representar este valor en la notación científica como 3.4E-5.

```
.000034
```

[*] Si olvidamos todas las matemáticas que aprendimos en secundaria. La notación científica se ve como 10.23E+4, lo que equivale a 1,023,000. Si se usara un número negativo, puede también representar números pequeños. Por ejemplo, 10.23E-4 es .001023. Simplemente hay que desplazar el punto decimal a la derecha si el exponente es positivo, y a la izquierda si es negativo.

3.1.2 Constantes de cadena *(04B)(08B)*

Las *constantes de cadena* son grupos de caracteres encerrados entre comillas a fin de que se puedan usar como un sólo dato. Se usan con frecuencia en los programas para identificar nombres de archivo, mostrar mensajes y solicitar entradas. En Perl puede usar las comillas sencillas (**'**), las dobles (**"**) o las invertidas (**`**).

Cadenas de comillas sencillas

Los siguientes ejemplos enseñaran cómo usar constantes de cadena. Las constantes de cadena se usan ampliamente para identificar nombres de archivos o cuando se exhiben mensajes a los usuarios. Veremos primero las cadenas de comillas sencillas y luego las de comillas dobles.
Una cadena de comillas sencillas es muy simple. Sólo necesitamos encerrar el texto que deseemos emplear entre comillas sencillas

Una constante que describe esto es como la siguiente

```
'Perl es un lenguaje muy sencillo'
```

Las cadenas son muy sencillas, ¿no? Pero ¿qué pasaría si quisiéramos utilizar una comilla sencilla dentro de una constante? Si lo hiciéramos, Perl pensaría que quiere terminar antes la cadena y daría por resultado un error de compilación. Perl usa el caracter de diagonal invertida (\)* para indicar que la función normal de la comilla sencilla —termina una cadena— debe ignorarse por un momento.

Una constante que comenta sobre la capacidad de la diagonal invertida sería como sigue, además hay que notar cómo se usan las comillas sencillas.

```
'La palabra Je t\'aime traducida quiere decir TE AMO'
```

Las comillas sencillas se usan aquí de manera específica a fin de que las comillas dobles pueden emplearse para rodear las palabras habladas. Sólo hay una cosa más que debemos saber al respecto sobre las comillas sencillas.
Podemos agregar un corte de línea a una cadena de comillas sencillas simplemente agregando el corte de línea al código fuente, como se muestra en el Listado 3.01.

Listado 3.01. Uso de cortes de línea insertados para saltar a una nueva línea.

```
print'Precio de los productos
Pan:        $34.45
Fruta:      $45.00
            =====
            $79.45';
```

* El caracter de diagonal invertida se conoce también como *un caracter de escape*, tal vez debido a que permite que el siguiente caracter escape de su interpretación normal.

El programa produce la siguiente salida:

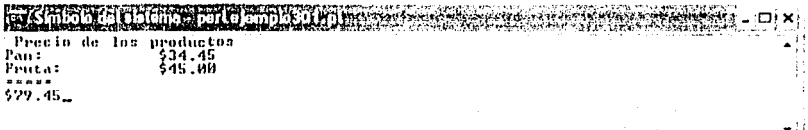


Figura 3.01 Una nota de artículos exhibida en una sola constante larga entre comillas sencillas

Podemos ver que con las constantes de comillas sencillas, incluso los cortes de línea en el código fuente forman parte de la cadena.

Cadenas de comillas dobles

Las cadenas de comillas dobles comenzaron siendo simples y luego se hicieron un poco más complejas que las cadenas de comillas sencillas. Con las cadenas de comillas dobles, podemos usar la diagonal invertida para incorporar a la cadena algunos caracteres especiales.

La cadena básica de comillas dobles es una serie de caracteres comprendidos entre comillas dobles. Si necesitáramos usar las comillas dobles dentro de la cadena, podemos utilizar el caracter de diagonal invertida.

> *Esta constante es similar a las otras en las que utilizábamos el caracter de escape.*
> *Sólo que las comillas son diferentes.*
> *Otra constante que usa comillas dobles dentro de una cadena de comillas dobles.*

```
"El que ama de verdad no es el que enciende el fuego"
"Si no el que lo \"Conserva\""
```

Hay que notar como se usa la diagonal invertida en la segunda línea para dar escape a los caracteres de comillas dobles. Y cómo puede emplearse la comilla sencilla sin una diagonal invertida.

Una diferencia principal entre las cadenas de comillas dobles y sencillas consiste en que las cadenas de comillas dobles tienen algunas *secuencias de escape* especiales que se pueden usar. Las secuencias de escape representan caracteres que no se registran con facilidad mediante el teclado, o que son difíciles de ver dentro de una ventana de editor.

La Tabla 3.01 muestra todas las secuencias de escape que Perl comprende. Los ejemplos a continuación de la Tabla ilustrarán algunos casos.

Tabla 3.01 Secuencias de escape

Secuencia de escape	Descripción o caracter
`\a`	Alarma \ campana
`\b`	Retroceso (Backspace)
`\e`	Escape
`\f`	Salto de hoja (Form Feed)
`\n`	Línea nueva (NewLine)
`\r`	Retorno de carro (Carriage Return)
`\t`	Tabulador (Tab)
`\v`	Tabulador vertical
`\$`	Signo de pesos
`\@`	Ampersand
`\%`	Signo de porcentaje
`\0nnn`	Cualquier byte octal
`\xnn`	Cualquier byte hexadecimal
`\cn`	Cualquier caracter de control
`\l`	Cambia el caracter siguiente a minúsculas
`\u`	Cambia el caracter siguiente a mayúsculas
`\L`	Cambia los siguientes caracteres a minúsculas hasta encontrar una secuencia `\E`. Nótese que se requiere usar aquí una `E` mayúscula, la minúscula no funcionará.
`\Q`	Cita metacaracteres como constantes.
`\U`	Cambia los siguientes caracteres a mayúsculas hasta encontrar una secuencia `\E`. Nótese que se requiere usar aquí una `E` mayúscula, la minúscula no funcionará.
`\E`	Termina la secuencia `\L`, `\Q` o `\U`. Observe que necesitará usar `E` mayúscula aquí; la minúscula no funcionaría.
`\\`	Diagonal invertida

Esta constante representa lo siguiente: José Jacobo tiene 34 años. La \u se usa dos veces en la primera palabra para poner en mayúsculas los caracteres j. Y la notación hexadecimal se emplea para representar la edad mediante los códigos ASCII para el 3 y el 4.

Esta constante representa lo siguiente: La olla estaba CALIENTE! La \U pone en mayúsculas todos los caracteres hasta que se ve una secuencia. \E.

```
"\ujose\ujacobo tiene \x33\x34 años."

"La olla estaba \Ucaliente\E!"
```

Para más información sobre los códigos ASCII, ver la sección de Glosario de Términos, "Tabla de caracteres ASCII".

Veamos ahora las secuencias de escape `\t` y `\n`. El Listado 3.02 —un programa que exhibe una notación con varios artículos— producirá la salida que se presenta en la Figura 3.02.

Exhibe una constante como primera, segunda y tercera líneas de la salida.
Exhibe una línea de separación.
Exhibe una línea de separación.
Exhibe el total.

Listado 3.02 Uso de los caracteres de tabulador y de línea nueva para imprimir

```
print "Precio de los productos

Pan:\t\$34.45\n";
print  "Fruta:\t";
print "\$45.00\n";
print "\t====\n";
print "\t479.45\n";
```

El programa produce la siguiente salida:

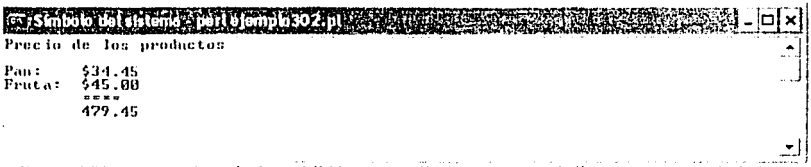


Figura 3.02 Una nota de artículos exhibida por medio de caracteres de línea nueva y de tabulador.

Este programa usa dos métodos para provocar un corte de línea.

- El primero consiste simplemente en incluir el corte de línea en el código fuente.
- El segundo es utilizar `\n` o caracter de línea nueva.

- Lo más recomendable es usar el caracter `\n`, de modo que al observar el código en el futuro, podamos asegurar de que la intención fue provocar un corte de línea y no simplemente se oprimió la tecla Enter por error.

Cadenas de comillas invertidas

Se podría argumentar que las cadenas de comillas invertidas no son en realidad un tipo de datos. Ello se debe a que Perl usa las cadenas de comillas invertidas para ejecutar comandos del sistema. Cuando Perl ve una cadena de comillas invertidas, transfiere su contenido a Windows, UNIX o a cualquier otro sistema operativo que se este utilizando.

Veamos cómo usar la cadena de comillas invertidas para exhibir un listado del directorio de todos los archivos de texto en el directorio Perl.

La Figura 3.03i muestra cómo se vería la salida de dicho programa.

Imprime el listado del directorio.

```
print `dir *.txt`;
```

```
El volumen de la unidad C es HACKER
El número de serie del volumen es: 2430-18DF

Directorio de C:\Perl\bin\Ejemplos

06/06/2002  10:20 p.m.                 0 otsE.txt
06/06/2002  10:21 p.m.                 0 sE.txt
06/06/2002  10:21 p.m.                 0 roP.txt
06/06/2002  10:22 p.m.                 0 IT.txt
06/06/2002  10:22 p.m.                 0 emlateJ.txt
               5 archivos              0 bytes
               0 dirs  16,032,563,200 bytes libres
```

Figura 3.03i Uso de una cadena de comillas invertidas para exhibir un directorio.

Todas las secuencias de escape utilizadas con cadenas de comillas dobles se pueden usar con cadenas de comillas invertidas.

3.1.3 Constantes de arreglo *(4) (8)*

Perl usa los *arreglos* —o listas— para almacenar una serie de elementos. Nosotros podríamos usar un arreglo para almacenar todas las líneas de un archivo, para ayudar a clasificar una lista de direcciones, o para almacenar una diversidad de elementos.

Impresión de un arreglo

En esta ocasión, veremos la impresión de un arreglo y cómo se representan los arreglos en el código fuente de Perl.

Este ejemplo muestra un arreglo vacío, un arreglo de números y un arreglo de cadenas.

Imprime el contenido de un arreglo vacío,
Imprime el contenido de un arreglo de números.
Imprime el contenido de un arreglo de cadenas.
Imprimir el contenido de un arreglo con diferentes tipos de datos.

Listado 3.03 Impresión de algunas constantes de arreglo

```
print "Aquí hay un arreglo vacío:" . () . "< -- Nada por allí!\n";
print (12, 014, 0x0c, 34.34, 23.e-3);
print "\n";
print ("Este ", "es", 'un', "arreglo", 'con', "cadenas de comillas
sencillas y dobles");
print "\n";
print ("El", 30, "es", 'un', "numero mezclado con cadenas", 0x08,
"intercambiables");
```

La Figura 3.03 presenta la salida del Listado 3.03

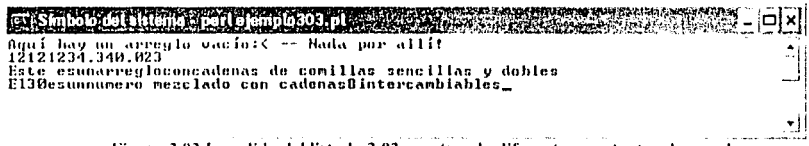


Figura 3.03 La salida del listado 3.03, mostrando diferentes constantes de arreglo.

La cuarta línea del listado muestra que podemos mezclar, en un mismo arreglo, cadenas de comillas sencillas y dobles. Puede también mezclarse números y cadenas en forma intercambiable como se muestra en la última línea.

3.2 Variables *(04B) (08B)*

En la sección anterior, aprendimos sobre las *constantes*, valores que no cambian durante la ejecución de un programa debido a que las respetamos en el código fuente *exactamente* como deberían usarse. Sin embargo, la mayoría de las veces, necesitaremos modificar los valores que use un programa. Para ello, requerimos reservar partes de la memoria de la computadora para contener los valores cambiantes. Y, necesitaremos llevar un registro de en dónde se encuentran esas pequeñas áreas de memoria, a fin de que podamos referirnos a ellas mientras se ejecuta un programa.

Perl, al igual que otros lenguajes de cómputo, utiliza las variables para llevar un registro del uso de la memoria de la computadora. Cada vez que necesitemos almacenar una nueva pieza de información, debemos asignar una variable.

Ya hemos visto cómo Perl utiliza los números, cadenas y arreglos. Ahora, veremos cómo usará variables para contener esta información. Perl tiene tres tipos de variables:

Tipo de variable	Descripción
Escalares	Contienen el valor de un número o de una cadena a la vez. Los nombres de variables escalares comienzan siempre con `$`
Arreglos	Contienen una lista de valores. Los valores pueden ser números, letra, o incluso otro arreglo. Los nombres de las variables de arreglo comienzan siempre con `@`
Arreglos asociativos	Usan cualquier valor como un índice dentro de un arreglo. Los nombres de las variables de arreglo asociativos comienzan siempre con `%`

Los diferentes caracteres de inicio nos ayudan a comprender cómo se está utilizando una variable al observar el código Perl de alguien más. Si vemos una variable llamada `@Valor`, automáticamente sabemos que se trata de una variable de arreglo.

También proporcionan un *espacio de nombres* diferentes para cada tipo de variables. Los espacios de nombres separan un conjunto de nombres de otro. De ahí que Perl pueda llevar un registro de variables escalares en una tabla de nombres (o espacio de nombres) y las variables de arreglo en otras. Esto nos permitirá utilizar `$nombre`, `@nombre` y `%nombre` para referirnos a valores distintos.*

* Los nombres de variables en Perl son sensibles al uso de mayúsculas y minúsculas. Esto significa que `$VarNAme`, `$varName` y `$VARNAME` se refieren a variables diferentes.

3.2.1 Variables escalares *(04H) (08B)*

Las variables escalares se usan para llevar un registro de piezas de información individuales. Las podemos utilizar para almacenar el título de un libro o el número de habitaciones de una casa. Para una variable escalar podemos emplear casi cualquier nombre, siempre que comience con $.

Veamos algunos ejemplos de variables.

Esta variable escalar contendrá el número de habitaciones.
Esta variable escalar contendrá el título de un libro.

```
$numeroDeHabitaciones
$tituloLibro
```

La mayoría de los programadores tratan de usar nombres descriptivos para sus variables. Prácticamente no hay límite para la longitud de un nombre de variables en Perl, pero de preferencia debemos de mantenerlos en menos de 15 caracteres.

Asignación de valores a las variables escalares

Ahora que ya sabemos cómo lucen los nombres de las variables escalares, veremos cómo podemos asignarles un valor. La asignación de valores a una variable se hace mediante el signo de igual (=).

Asignar un valor de 23 a la variable de nombre $numeroDeHabitaciones.
Asignar el valor El paciente Ingles a la variable llamada $tituloLibro.

```
$numeroDeHabitaciones = 23;
$tituloLibro = "El paciente Ingles";
```

Nótese que estamos asignando valores constantes a las variables. Después de asignar los valores, podemos modificarlos.

Cambio de valores en variables escalares

El siguiente ejemplo hará una asignación de variables y luego modificará el valor de la misma utilizando una segunda asignación. La segunda asignación incrementará el valor en cinco.*

Asignar un valor de 23 a la variable de nombre $numeroDeHabitaciones.
Suma 5 a la variable $numeroDeHabitaciones.

```
$numeroDeHabitaciones = 23;
$numeroDeHabitaciones = $numeroDeHabitaciones +5;
```

* En Perl, no tenemos nunca que declarar, definir o asignar tipos de datos sencillos (por ejemplo, escalares, arreglos o arreglos asociativos). Al usar la variable por vez primera, Perl le asigna un valor ya sea de cero si necesitáramos un número, o una lista vacía si requiere de un arreglo. El uso de un nombre de variable equivale a definirlo.

3.2.2 Variables de Arreglo *(04B) (08B)*

Los nombres de variables de arreglo siempre comienzan con un caracter @.

Las reglas para nombrar variables de arreglo son las mismas que para las variables escalares. No hay reglas. Bueno, ninguna de la que tengamos que preocuparnos.

Asignación de valores a variables de arreglo

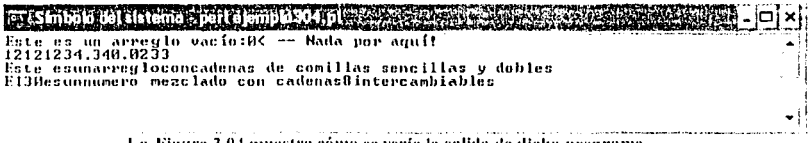
Al igual que con las variables escalares, se emplea el signo igual (=) para asignar valores a las variables de arreglo.
Usaremos uno de los ejemplos de la sección anterior, "Constantes numéricas y de cadena" —ligeramente modificado— de modo que estemos familiarizados con parte del ejemplo.

Asignar valores a variables de arreglo.
Imprime las variables de arreglo.

Listado 3.04 Asignación de valores a variables de arreglo

```
@arregloVacio = ( );
@numeroArreglo =(12, 014, 0x0c, 34.34, 23.3E-3);
@arregloSencilloDoble= ("Este ", "es", 'un', "arreglo", 'con', "cadenas
de comillas sencillas y dobles");
@arregloIntercambiable=  ("El", 30, "es", 'un', "numero mezclado con
cadenas", 0x08, "intercambiables");
print "Este es un arreglo vacío:" . @arregloVacio . "< -- Nada por
aquí!\n";
print  @numeroArreglo; print "\n";
print @arregloSencilloDoble; print "\n";
print @arregloIntercambiable; print "\n";
```

La Figura 3.04 presenta la salida del Listado 3.04



La Figura 3.04 muestra cómo se vería la salida de dicho programa.

En este ejemplo, asignamos valores constantes a variables de arreglo y luego los exhibimos mediante el comando `print`.

Uso de los elementos del arreglo

Los elementos individuales de un arreglo se acceden poniendo un signo $ como prefijo al nombre del arreglo y utilizando un índice que indica a Perl qué elemento deseamos usar.

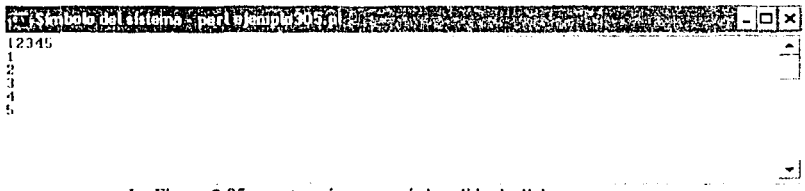
El Listado 3.05 crea un arreglo de cinco elementos e imprime después de cada elemento individual.

Crea un arreglo de cinco elementos.
Imprime el arreglo.
Imprime cada elemento del arreglo.

Listado 3.05 Acceso de los elementos de un arreglo

```
@arreglo = (1..5);
print @arreglo;   print "\n";
print $arreglo [0];     print "\n";
print $arreglo [1];     print "\n";
print $arreglo [2];     print "\n";
print $arreglo [3];     print "\n";
print $arreglo [4];     print "\n";
```

La Figura 3.05 presenta la salida del Listado 3.05



La Figura 3.05 muestra cómo se vería la salida de dicho programa.

3.2.3 Variables de Arreglo Asociativo *(04B) (08B)*

Ahora es momento de ver los arreglos asociativos. Éstos son definitivamente los más complicados de los tres tipos de datos. Y, sin embargo, sólo son otro tipo de arreglos.
Nosotros ya vimos que los elementos de los arreglos se pueden acceder con índices tanto positivos como negativos. Pues bien, con los arreglos asociativos podemos emplear *cualquier* tipo de dato escalar como índice. Los nombres de los arreglos asociativos comienzan con el caracter %.

Nosotros veremos que a los arreglos asociativos se les llama algunas veces *hashes*. El término "Hash" se refiere a cómo se almacenan en la memoria los elementos de un arreglo asociativo. "Hash" es un término más breve que "arreglos asociativos" y, por lo tanto, mucho más fácil de escribir y de referirse a él.

Asignación de valores a variables de arreglo asociativo

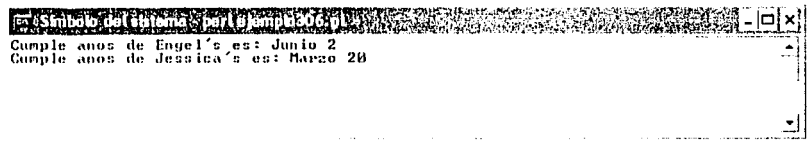
Antes de abordar más a detalle los arreglos asociativos, veremos cómo asignarle valores. Al definir todo un arreglo, podemos usar la misma representación empleada para los arreglos, sólo hay que recordar que se necesitan dos partes para cada elemento del arreglo asociativo. También podemos asignar valores a elementos individuales de un arreglo asociativo mediante el uso de llaves ({ }) alrededor de la llave de índice.

Crear un arreglo asociativo con tres elementos. Cada elemento consta de dos valores: la llave de búsqueda y su valor asociado.
Agregar un elemento al arreglo asociativo.

Listado 3.06 Asignación de valores a variables de arreglo asociativo

```
%arregloAsociativo = ("Jacobo A. ","Dic 2", "Engel B.","Junio 2", "Rubi
C.", "Feb 14");
$arregloAsociativo {"Jessica S."}= "Marzo 20";
print "Cumple anos de Engel´s es: " . $arregloAsociativo {"Engel B."} .
"\n";
print "Cumple anos de Jessica´s es: ".$arregloAsociativo {"Jessica S."}
. "\n";
```

La Figura 3.06 presenta la salida del Listado 3.06



La Figura 3.06 muestra cómo se vería la salida de dicho programa.

Al asignar nuevos valores a las llaves, Perl ampliará el arreglo asociativo según sea necesario. Se usara una tabla interna para llevar el registro de qué llaves están definidas.
Si intentáramos accesar una llave indefinida, Perl retornará un valor nulo o cadena en blanco.

Uso de la variable especial $"

Perl tiene diversas variables especiales. Cada una de estas variables tiene un significado predefinido, nos dedicaremos a hablar de la variable especial $"[▲]

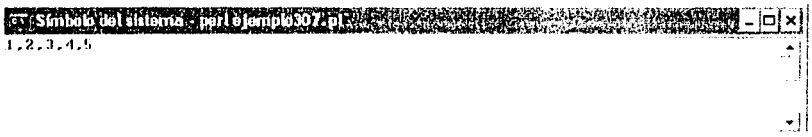
Asignar el caracter coma a la variable especial $".
Crear un arreglo de cinco elementos.
Imprimir el arreglo con comas entre los elementos.

[▲] En ocasiones, pudiéramos necesitar imprimir los elementos de un arreglo separados por comas o por otro caracter. La variable $" controla cuál es el separador que emplea Perl al imprimir el arreglo. Por lo regular la variable tiene asignado el caracter de espacio. Sin embargo, podemos asignarle cualquier caracter que se desee

Listado 3.07 Uso de variables especiales

```
$" = ",";
@arreglo = (1..5);
print "@arreglo\n";
```

Este programa imprimirá:



La Figura 3.07 muestra cómo se vería la salida de dicho programa.

Por supuesto, debido a que `$"` es una variable escalar, podremos también haberle asignado una cadena más larga. Por ejemplo, podremos usar `$" = ",  "` para agregar una coma y un espacio entre los elementos del arreglo.

3.3 Operadores *(04B) (08B)*

En un lenguaje de cómputo, los *operadores* indican a la computadora qué acciones realizar. Perl tiene más operadores que la mayoría de los lenguajes.

Los operadores son instrucciones que le damos a una computadora de modo que pueda efectuar alguna tarea u operación. Todos los operadores hacen que se realicen acciones sobre los *operandos*. Un operando puede ser cualquier cosa sobre la que podamos realizar una operación. En términos prácticos, cualquier operando en particular será una constante, una variable o una expresión.

Los operandos son también de naturaleza *recurrente*. En Perl, la expresión `3+5` -dos operandos y un operador más- se puede considerar como un operando con un valor de 8. Por ejemplo, `(3+5) - 12` es una expresión que consta de dos operandos, el segundo de los cuales se le resta al primero. El primer operando es `(3+5)` y el segundo es `12`.

En esta sección se expondrá la mayoría de los operadores disponibles para nosotros en Perl. Nosotros encontraremos muchos tipos de operadores y cómo determinar su orden de precedencia.

La precedencia es muy importante en cualquier lenguaje de cómputo y Perl no es la excepción. El *orden de precedencia* indica cuál de los operadores debe evaluarse primero.

Debemos pensar en los operadores del mismo modo en que daríamos instrucciones al conductor de un automóvil. Podríamos decir "dé vuelta a la izquierda" o "dé vuelta a la derecha". Estas órdenes se pueden considerar como operadores direccionales. De la misma forma en que `+` y `-` significan "suma esto" o "resta esto". Por otra parte, si nosotros decimos "alto" mientras el automóvil está en movimiento, esta orden deberá imponerse al resto.
Esto significa que "alto" tiene precedencia sobre "dé vuelta a la izquierda" y "dé vuelta a la derecha".

3.3.1 Tipos de operadores *(04B) (08B)*

Perl maneja muchos tipos de operadores. La Tabla 3.02 muestra los tipos de operadores en el lenguaje Perl. En esta sección se expone a detalle los tipos de uso más comunes.

Tabla 3.02 Los tipos de operadores en Perl

Tipos de operadores	*Descripción*
Aritméticos	Estos operadores son iguales a los que aprendimos en la escuela primaria. `+`, `-`, `*`. Suma, resta y multiplicación son el pan y mantequilla de la mayoría de los enunciados matemáticos.
De asignación	Estos operadores se usan para asignar un valor a una variable. El álgebra usa operadores de asignación. Por ejemplo en el enunciado `X=6`, el signo de igual es el operador de asignación.
De vinculación	Estos operadores se usan durante las comparaciones de cadenas.
A nivel de bit	Estos operadores afectan los bits individuales que conforman un valor. Por ejemplo el valor 3 es también 11 en la notación binaria o `((1x2)+1)`. Cada carácter en la notación binaria representa a un *bit*, que es la parte más pequeña que podemos modificar de la memoria de la computadora.
Coma	El operador de coma tiene dos funciones. Sirve para separar arreglos o elementos de una lista.
Prueba de archivos	Estos operadores se usan para probar diversas condiciones asociadas con los archivos.
De lista	Los operadores de lista son algo curioso en Perl. Se asemejan a llamadas de funciones en otros lenguajes.
Lógicos	Estos operadores instrumentan la lógica booleana o de verdadero / falso. En el enunciado "Si Juan tiene fiebre **AND** Juan tiene congestión nasal **OR** dolor de oído **AND** Juan no e mayor de 60 años, entonces Juan tiene un resfriado", **AND**, **OR** y **NOT** están actuando como operadores lógicos.
Relacionales numéricos	Estos operadores permiten verificar la relación de una variable numérica con otra. Por ejemplo, (¿es 5 mayor que 12?).
Postfijo	Un miembro de este grupo de operadores -`( )`, `[ ]`, `{ }`- aparecen al final de los objetos afectados.
De rango	Los operadores de rango se usan para crear un rango de elementos en los arreglos. También se pueden utilizar en un contexto escalar.
De referencia	Los operadores de referencia se usan para manipular variables.
De cadena	El operador de concatenación de cadenas se usa para unir dos cadenas. El operador de repetición de cadenas se usa para repetir una cadena.
Relacionales de cadena	Estos operadores permiten verificar la relación de una variable de cadena con otra. Por ejemplo, (es `"abc"` mayor que `"ABC"`?)
Ternarios	El operador ternario se usa para elegir entre dos opciones con base en una condición dada. Por ejemplo, si el parque está a menos de una milla, Juan puede caminar; en caso contrario, debe conducir.

A continuación hablaremos de los operadores más importantes.

3.3.2 Los operadores aritméticos binarios *(04B) (08B)*

Existen seis *operadores aritméticos binarios* suma, resta multiplicación, exponenciación, división y módulo. Aunque pudiéramos no estar familiarizados con el operador de módulo, el resto actúa exactamente como nosotros esperamos que lo hagan. La Tabla 3.03 lista los operadores aritméticos que pueden actuar sobre dos operandos: los operadores aritméticos binarios. En otras palabras, el operador de suma (`+`) se puede usar para sumar dos números como los siguientes: `4+5`. Los demás operadores binarios actúan de manera similar.

Tabla 3.03 Los operadores aritméticos binarios

Operadores	*Descripción*
`op1 + op2`	Suma
`op1 - op2`	Resta
`op1 * op2`	Multiplicación
`op1 ** op2`	Exponenciación
`op1 / op2`	División
`op1 % op2`	Módulo

3.3.3 Los operadores aritméticos unitarios *(04B) (08B)*

Los operadores aritméticos unitarios actúan sobre un sólo operando. Se usan para cambiar el signo a un valor, para incrementarlo o disminuirlo. *Incrementar* un valor significa sumar uno a dicho valor. *Disminuir* un valor significa restar uno a ese valor. La Tabla 3.04 lista los operadores unitarios de Perl.

Tabla 3.04 Los operadores aritméticos unitarios

Operador	*Descripción*
Modificación del signo de op1	
`+op1`	Operando positivo
`-op1`	Operando negativo
Modificación del valor de op1 antes de utilizarlo	
`++op1`	Pre-incremento del operando en una unidad
`--op1`	Pre-disminución del operando en una unidad
Modificación del valor de op1 después de utilizarlo	
`op1++`	Post-incremento del operando en una unidad
`op1--`	Post-disminución del operando en una unidad

Los operadores aritméticos comienzan a complicarse al introducir los operadores unitarios. Usar el operador unitario de más (`+`) no hace nada, y Perl lo ignora. No obstante, el operador unitario negativo, cambia el significado de un valor de positivo a negativo o viceversa. Por ejemplo, si tuviéramos una variable llamada `$primeraVar` igual a `34` y luego imprimiéramos `-$primeraVar`, exhibiría `-34`.

Los operadores + + y - - son ejemplos de la notación abreviada de Perl. Si estos operadores aparecen frente al operando, éste se incrementa o disminuye antes de utilizar su valor. Si aparecen después del operando, entonces el valor del mismo se utiliza y después se incrementa o disminuye, según corresponda.

El operador de pre-incremento

Este ejemplo muestra cómo usar el operador de pre-incremento (++).

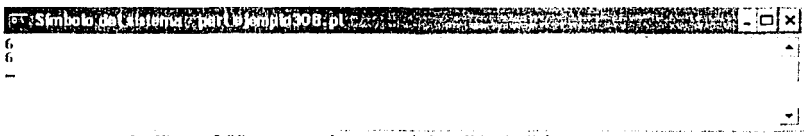
Se asigna un valor de 5 a la variable $ numPaginas.
Se incrementa en 1 la variable $numPaginas.
Se imprime la varible $numPaginas.

Se asigna un valor de 5 a la variable $numPaginas.
Se incrementa la variable $numPaginas utilizando el operador de incremento previo y luego se imprime.

Listado 3.08 Uso del operador de pre-incremento

```
#Manera original
$numPaginas =5;
$numPaginas = $numPaginas +1;
print ($numPaginas, "\n");


#Nueva manera
$numPaginas = 5;
print (++$numPaginas, "\n");
```

Al ejecutarse este programa, la salida lucirá como la siguiente:



La Figura 3.08 muestra cómo se vería la salida de dicho programa.

Podemos ver que la nueva forma de codificación es más breve que la forma original. El enunciado `print (++$numPaginas, "\n");` primero incrementa la variable `++$numPaginas` y luego permitirá que el comando `print` la utilice.

El operador de pre-decremento

El ejemplo muestra cómo utilizar el operador de pre-decremento (-).

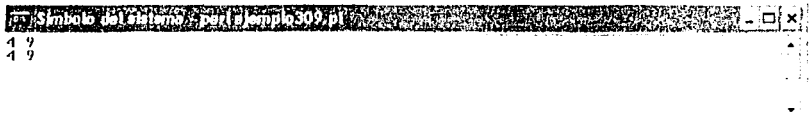
Se asigna un valor de 5 a la variable $numPaginas.
Se decrementa en 1 la variable $numPaginas.
Se asigna la variable $totalPaginas el valor de $numPaginas +5.
Se imprimen las varibles $totalPaginas y $numPaginas.

Se asigna un valor de 5 a la variable $numPaginas.
Se decrementa la variable $numPaginas y luego se asignan $numPaginas +5 a $total Paginas.
Se imprimen las variables $numPaginas y $totalPaginas

Listado 3.09 Uso del operador de pre-decremento

```
#Manera original
$numPaginas =5;
$numPaginas = $numPaginas -1;
$totalPaginas = $numPaginas + 5;
print ("$numPaginas $totalPaginas \n");

#Nueva manera
$numPaginas = 5;
$totalPaginas = --$numPaginas + 5;
print ("$numPaginas $totalPaginas \n");
```

Al ejecutarse este programa, la salida lucirá como la siguiente:



La Figura 3.09 muestra cómo se vería la salida de dicho programa.

El enunciado `$totalPaginas = --$numPaginas +5;` decrementará primero la variable `$numPaginas` y luego permitirá que el operador de suma la utilice.

3.3.4 Los operadores lógicos *(04B) (08B)*

Los *operadores lógicos* se utilizan principalmente para controlar el flujo del programa. Por lo regular, los encontraremos como parte de un `if`, un `while` u otro enunciado de control.

Tabla 3.05 Los operadores lógicos

Operador	*Descripción*
`op1 && op2`	Realiza un `AND` lógico de los dos operandos.
`op1 || op2`	Realiza un `OR` lógico de los dos operandos
`!op1`	Realiza un `NOT` lógico del operando.

El concepto de operador lógico es simple. Permite que un programa tome una decisión con base en varias condiciones. Cada operando se considera como una condición que puede evaluarse como verdadera o falsa. Entonces, el valor de la condición se utiliza para determinar el valor general del `op1` operador `op2`, o el argumento `!opt`. Los siguientes ejemplos demuestran distintas formas del uso de las condiciones lógicas.

El operador "`AND`" (`&&`)

El operador `&&` se utiliza para determinar si ambos operandos o condiciones son verdaderos. La Tabla 3.06 muestra los resultados del uso del operador `&&` sobre los cuatro conjuntos de valores verdadero /falso.

Tabla 3.06 Los operadores lógicos

`op1`	`op2`	`op1&&op2`
0	0	0
1	0	0
0	1	0
1	1	1

Si el valor de $primeraVar es 10 AND el valor de $segundaVar es 9, entonces imprime el mensaje de error.

```
if ($primeraVar == 10 && $segundaVar ==9){
print ("Error!");
};
```

Si alguna de las dos condiciones es falsa o incorrecta, entonces se ignora el comando print.

El operador "`OR`" (`||`)

El operador `||` se usa para determinar si alguna de las condiciones es verdadera. La Tabla 3.07 muestra los resultados del uso de `||` sobre los cuatro conjuntos de valores verdadero/falso.

Tabla 3.07 Tabla de resultados del operador ||

op1	op2	op1 || op2
0	0	0
1	0	1
0	1	1
1	1	1

Si el valor de $primeraVar es 9 OR el valor de $primeraVar es 10, entonces se imprime el mensaje de error.

```
if ($primeraVar == 9 || $pimeraVar ==10){
print ("Error!");
};
```

Si alguna de las condiciones es verdadera, entonces se ejecuta el comando print.*

El operador "NOT" (!)

El operador ! se usa para convertir valores de verdadero a falso y viceversa. En otras palabras, se niega un valor. Perl considera que cualquier valor diferente de cero es verdadero, incluso los valores de cadena. La Tabla 3.08 muestra el resultado del uso del operador ! sobre los valores de verdadero y falso.

Tabla 3.08 Tabla de resultados del operador !

op1	!op1
0	1
1	0

Asignar el valor de 10 a $primeraVar.
Negar $primeraVar -!10 es igual a 0- y asignar el nuevo valor a $segundaVar.
Si la variable $segundaVar es igual a cero, entonces imprime la cadena "cero"

Listado 3.10 Uso del operador "NOT" (!)

```
$primeraVar = 10;
$segundaVar= !$primeraVar;
if ($segundaVar ==0){
print("cero\n");
};
```

* Si el primer operando del operador || se evalúa como verdadero, no se evaluará el segundo operando. Esto puede ser una fuente de errores si no somos cuidadosos. Por ejemplo, en el siguiente fragmento de código:
`if($primeraVar++ || $segundaVar++) {print("\n");}`
la variable $segundaVar no se incrementará si `$primeraVar++` se evalúa como verdadera.

Al ejecutarse este programa, la salida lucirá como la siguiente:

La Figura 3.10 muestra cómo se vería la salida de dicho programa.

Nosotros podemos remplazar el 10 de la primera línea con "diez", 'diez', o cualquier valor diferente de cero, no nulo.

3.3.5 Los operadores relacionales numéricos *(04B) (08B)*

Los operadores *relacionales numéricos*, que se listan en la Tabla 3.09, se usan para verificar la relación entre dos operandos. Nosotros podemos ver si un operando es igual a otro, si es mayor que otro, o si es menor.[▲]

Tabla 3.09 Los operadores relacionales numéricos

Operador	*Descripción*
Los operadores de igualdad	
`op1 = = op2`	Este operador retorna verdadero si `op1` es igual a `op2`. Por ejemplo, `6 = = 6` es verdadero.
`op1 ! = op2`	Este operador retorna verdadero si `op1` no es igual a `op2`. Por ejemplo, `6 != 7` es verdadero.
Los operadores de comparación	
`op1 < op2`	Este operador retorna verdadero si `op1` es menor que `op2`. Por ejemplo, `6<7` es verdadero.
`op1 < = op2`	Este operador retorna verdadero si `op1` es menor o igual a `op2`. Por ejemplo, `7<=7` es verdadero.
`op1 > op2`	Este operador retorna verdadero si `op1` es mayor que `op2`. Por ejemplo, `6>5` es verdadero.
`op1 > = op2`	Este operador retorna verdadero si `op1` es mayor o igual que `op2`. Por ejemplo `7>=7` es verdadero.
`op1 < = = > op2`	Este operador retorna 1 si `op1` es mayor que `op2`, 0 si `op1` es igual a `op2`, y -1 si `op1` es menor que `op2`.

[▲] Es importante tener presente que el operador de igualdad es un par de signos de igual y no sólo uno. Se cometen muchos errores en los programas debido a que la gente olvida esta regla y usa un sólo signo de igual al verificar condiciones.

Ejemplo: uso de operador < = >

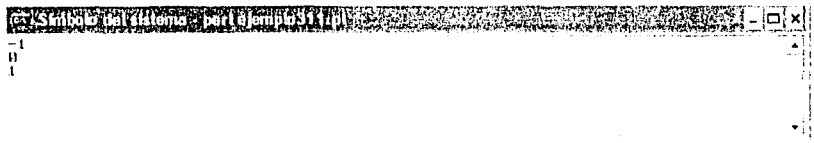
El operador de *comparación numérica* se usa para determinar con rapidez la relación ente un operador y otro. Se usa con frecuencia en actividades de clasificación.*

Establecer tres variables
Imprimir la relación entre cada variable con la variable $mediaVar.

Listado 3.11 uso de operador < = >

```
$bajoVar = 8;
$mediaVar = 10;
$lagaVar = 12;
print($bajoVar  <=> $mediaVar,"\n");
print($mediaVar  <=> $mediaVar,"\n");
print($lagaVar  <=> $mediaVar,"\n");
```

El programa produce la siguiente salida:



La Figura 3.11 muestra cómo se vería la salida de dicho programa.

El `-1` indica que `$bajoVar (8)` es menor que `$mediaVar (10)`. El `0` indica que `$mediaVar` es igual que a sí misma. Y, el `1` indica que `$larga (12)` es mayor que `$mediaVar (10)`.

3.3.6 Los operadores de asignación *(04B) (08B)*

El último tipo de operadores son los operadores de *asignación*. Nosotros ya utilizamos el operador básico de asignación (=) para valorar variables en algunos de los ejemplos.
Además. Perl tiene operadores de asignación abreviados que combinan el operador de asignación básico con otro operador. Por ejemplo, en vez de decir `$primeraVar = $primeraVar + $segundaVar`, nosotros podríamos decir `$primeraVar += $segundaVar`.

La ventaja de usar operadores abreviados –además de teclear menos- consiste en que se ponen en claro las intenciones con respecto a la asignación.

La Tabla 3.10 lista una parte de los operadores de asignación de Perl.

* Es posible que en ocasiones veamos que al operador < = > se le denomine, por su aspecto, operador nave espacial.

Tabla 3.10 Los operadores de asignación.

Operador	*Descripción*
`var = op1;`	Este operador asigna el valor de `op1` a `var`.
`var += op1;`	Este operador asigna el valor de `var + op1` a `var`.
`var -= op1;`	Este operador asigna el valor de `var - op1` a `var`.
`var *= op1;`	Este operador asigna el valor de `var * op1` a `var`.
`var /= op1;`	Este operador asigna el valor de `var / op1` a `var`.
`var %= op1;`	Este operador asigna el valor de `var % op1` a `var`.
`var .= op1;`	Este operador asigna el valor de `var . op1` a `var`.
`var **= op1;`	Este operador asigna el valor de `var ** op1` a `var`.
`var x= op1;`	Este operador asigna el valor de `var x op1` a `var`.
`var <<= op1;`	Este operador asigna el valor de `var << op1` a `var`.
`var >>= op1;`	Este operador asigna el valor de `var >> op1` a `var`.
`var &= op1;`	Este operador asigna el valor de `var & op1` a `var`.
`var |= op1;`	Este operador asigna el valor de `var | op1` a `var`.
`var ^= op1;`	Este operador asigna el valor de `var ^ op1` a `var`.

Los ejemplos en esta sección no describirán los diferentes operadores de asignación. Su uso es directo. Sin embargo, al asignar valores a arreglos, existen algunas situaciones especiales. La primera consiste en asignar valores a rebanadas o partes de arreglos, y la segunda es la asignación de elementos de arreglos a escalares. Comenzaremos con las rebanadas de arreglos.

Ejemplos: Asignación utilizando rebanadas de arreglos.

Si recordamos la sección de, "Variables", las rebanadas de arreglos nos permiten acceder en forma directa a varios elementos de un arreglo, utilizando los operadores de coma, o bien de rango. Por ejemplo la variable `@arreglo (10, 12)` se refiere al décimo y duodécimo elemento del arreglo `@arreglo`.

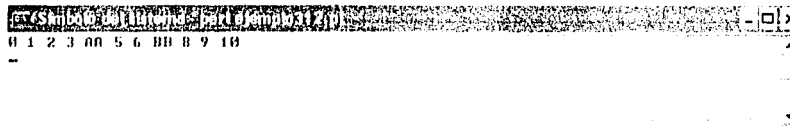
Podemos usar el operador de asignación conjuntamente con rebanadas de arreglos para asignar valores a varios elementos de un arreglo en un sólo enunciado. Si tenemos un arreglo de 10 elementos y deseamos modificar los elementos 4 y 7, podemos hacer algo como lo siguiente:

Crear un arreglo con 10 elementos.
Asignar valores a los elementos 4 y 7.
Imprimir el arreglo

Listado 3.12 Asignación utilizando rebanadas de arreglos

```
@arreglo = (0..10);
@arreglo[4,7] = ("AA","BB");
print ("@arreglo\n");
```

El programa produce la siguiente salida:



La Figura 3.12 muestra cómo se vería la salida de dicho programa.

Podemos considerar la asignación de la rebanada de arreglo de la siguiente manera. El arreglo de la izquierda es el destino y el de la derecha es el origen. De modo que, el arreglo de destino obtiene la asignación de los valores en el arreglo de origen.

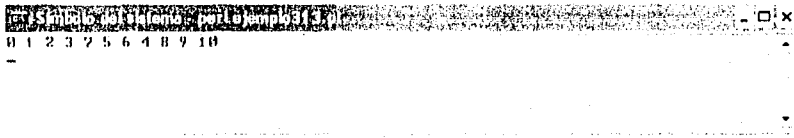
Una rebanada de arreglo es una forma rápida y conveniente de intercambiar dos elementos del mismo arreglo.

Crear un arreglo con 10 elementos.
Intercambiar los valores de los elementos 4 y 7.
Imprimir el arreglo.

Listado 3.13 Intercambio de los elementos de un arreglo

```
@arreglo = (0..10);
@arreglo[4,7] = @arreglo[7,4];
print "@arreglo\n";
```

El programa produce la siguiente salida:



La Figura 3.13 muestra cómo se vería la salida de dicho programa.

Hay que notar que el 4° y 7° elemento han intercambiado posiciones. También podemos usar el operador de rango al utilizar la asignación de rebanadas de arreglo.

Ejemplo: Asignación de un arreglo a variables escalares

En ocasiones, es posible que necesitemos tomar elementos de un arreglo y asignarlos a variables escalares. Esta capacidad es en especial útil dentro de las funciones.

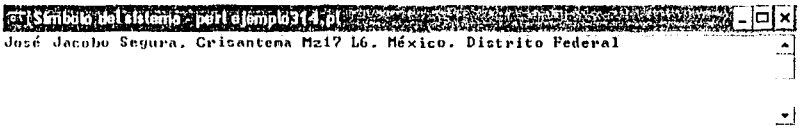
En el siguiente ejemplo, tomaremos un arreglo que contenga una dirección y separamos los elementos en cuatro variables escalares.

Crear un arreglo con la dirección de José Jacobo Segura
Asignar cada elemento del arreglo a una variable escalar por separado.
Imprimir las variables escalares.

Listado 3.14 Asignación de un arreglo a variables escalares

```
@arreglo = ("José Jacobo Segura", "Crisantema Mz17 L6", "México",
"Distrito Federal");
($nombre, $calle, $ciudad, $estado) = @arreglo;
print ("$nombre, $calle, $ciudad, $estado\n");
```

El programa produce la siguiente salida:



La Figura 3.14 muestra cómo se vería la salida de dicho programa.

El primer elemento de `@arreglo` se asigna a la primera escalar a la izquierda del operador de asignación. Debido a que las escalares están encerradas entre paréntesis, Perl las ve como otra lista.

Si no pudiéramos hacer este tipo de asignación de varios elementos de arreglos a varias escalares, podríamos hacer lo siguiente:

```
@arreglo = ("José Jacobo Segura", "Crisantema Mz17 L6", "México",
"Distrito Federal");
$nombre = @arreglo[0];
$calle = @arreglo[1];
$ciudad = @arreglo[2];
$estado = @arreglo[3];
print ("$nombre, $calle, $ciudad, $estado\n");
```

Si el arreglo tiene más elementos que escalares, se ignoraran los elementos adicionales. En forma correspondiente, si no hay suficientes elementos, algunas de las variables escalares tendrán un valor indefinido.

3.3.7 Orden de precedencia *(04B) (08B)*

La precedencia es muy importante en cualquier lenguaje de cómputo y Perl no es la excepción. El *orden de precedencia* indica cuál de los operadores debe evaluarse primero.

Ahora que estamos más familiarizados con la mayoría de los operadores de Perl, podemos explorar el tema con más detalle. La Tabla 3.11 es una lista completa de operadores y cómo se clasifican en términos de precedencia, entre más alto sea el nivel, mayor es su precedencia.

Los operadores al mismo nivel tienen igual precedencia y se evalúan de izquierda a derecha. En caso contrario, primero se evalúan los niveles con precedencia más alta.

Perl utiliza la *asociatividad* para decir qué operadores corresponden entre sí. Por ejemplo, al operador unitario menos, tiene una asociatividad de derecha a izquierda debido a que afecta al operando inmediatamente a su derecha.

Tabla 3.11 Orden de precedencia y asociatividad de los operadores de Perl

Nivel	*Operador*	*Descripción*	*Asociatividad*
22	(), [], {}	Llamadas a funciones, paréntesis, índices de arreglos	De izquierda a derecha
21	->	Operador infinito de desreferenciación	De izquierda a derecha
20	++, --	Incremento automático, decremento automático	Ninguna
19	**	Exponenciación	De derecha a izquierda
18	!, ~, +, -, \	Negación lógica, negación a nivel de bit, más unitario, menos unitario, referencia	De derecha a izquierda
17	=~, !~	Correspondencia, no correspondencia	De izquierda a derecha
16	*, /, %, x	Multiplicación, división, Módulo, repetición	De izquierda a derecha
15	+, -, .	Suma, resta, concatenación de cadenas	De izquierda a derecha
14	<<, >>	Desplazamiento a la izquierda a nivel de bit, desplazamiento a la derecha a nivel de bit	De izquierda a derecha

13		Operador de prueba de archivos	Ninguna
12		Operadores relacionales	Ninguna
11		Operadores de igualdad	Ninguna
10	`&`	`And` a nivel de bit	De izquierda a derecha
9	`|, ^`	`Or` a nivel de bit, `Xor` a nivel de bit	De izquierda a derecha
8	`&&`	`And` lógico	De izquierda a derecha
7	`||`	`Or` lógico	De izquierda a derecha
6	`..`	Operador de rango	Ninguna
5	`?:`	Operador ternario o condicional	De derecha a izquierda
4		Operadores de asignación	De derecha a izquierda
3	`,`	Operador coma	De izquierda a derecha
2	`not`	Operadores lógicos de baja precedencia	De izquierda a derecha
1	`and`	Operadores lógicos de baja precedencia	De izquierda a derecha
0	`or, xor`	Operadores lógicos de baja precedencia	De izquierda a derecha

Ejemplo: Orden de precedencia

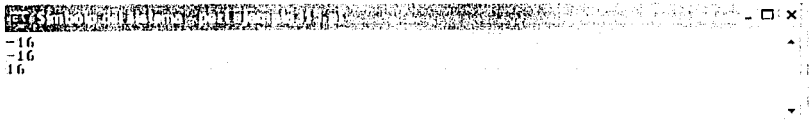
Aunque no es posible mostrar ejemplos de todas las ramificaciones de la precedencia de operadores, sí podemos observar uno a uno o dos a fin de dar una idea del concepto. Primero, un ejemplo utilizando un operador ternario y varios operadores aritméticos:

Asignar una expresión a $primeraVar
Asignar una expresión a $segundaVar utilizando paréntesis para indicar un orden de precedencia preferido.
Asignar una expresión a $terceraVar utilizando paréntesis de una manera distinta para indicar un orden de precedencia preferido.
Imprimir $primeraVar, $segundaVar, $terceraVar

Listado 3.15 Orden de precedencia

```
$primeraVar = -2 **4;
$segundaVar = -(2**4);
$terceraVar = (-2)**4;

print "$primeraVar\n";
print "$segundaVar\n";
print "$terceraVar\n";
```

El programa produce la siguiente salida:



La Figura 3.15 muestra cómo se vería la salida de dicho programa.

En este ejemplo, podemos observar el nivel de precedencia de la exponenciación es más alto que el nivel del menos unitario ya que la primera y segunda variables son iguales.*

* Si nosotros utilizamos siempre paréntesis para indicar cómo deseamos que se evalúen los operadores, nunca tendremos que preocuparnos por la precedencia de operadores del código.

3.4 Funciones *(04B) (08B)*

Las funciones son bloques de código a los que se asignan nombres de modo que podamos usarlas según lo requiéramos. Las funciones nos ayudaran a organizar el código en partes que sean fáciles de comprender y de trabajar con ellas. Nos permiten construir programas paso a paso, probando el código sobre la marcha.

Después de concebir la idea para un programa, necesitamos desarrollar un esquema del mismo, ya sea en nuestra mente o sobre papel. Cada paso en el esquema podría ser una función en el programa. Esto se denomina *programación modular*. La programación modular es muy buena pues nos permite ocultar los detalles de modo que los lectores del código fuente puedan entender la intención global del programa.

Por ejemplo, si nuestro programa tiene una función que calcula el área de un círculo, se podría usar la siguiente línea de código para invocarla:*

```
$areaDelPrimerCirculo = areaDelCirculo ($primerRadio);
```

Al observar la llamada de la función, el lector sabe lo que el programa esta haciendo. No es necesaria una comprensión detallada de la función en sí.**

Veamos un poco más de cerca la llamada a la función. Lo primero que hay en la línea es una variable escalar y un operador de asignación. Lo primero que vemos es el nombre de la función `areaDelCirculo( )`. Los paréntesis directamente a la derecha y ningún signo de `$`,`@` o `%` al inicio del nombre indican que se trata de una llamada a una función.

Dentro de los paréntesis hay una lista de parámetros o valores que se transfieren a la función. Podemos pensar en un parámetro tal como en un balón de fútbol americano. Cuando se pasa, al receptor (por ejemplo, la función), tiene varias opciones: correr (modificarlo en cierto modo), pasar (llamar a otras subrutinas), o perder del balón (llamar al manejador de errores).***

---

* Llamar a una función significa que Perl detiene la ejecución de la serie actual de líneas del programa. El flujo del programa salta al código de programa que esta dentro de la función Cuando ésta termina, Perl salta de regreso al punto en el que se invocó a la función. La ejecución del programa continúa hacia adelante a partir de ese punto.

** Para hacer más fácil la comprensión del lector sobre las funciones que realice un programa es más sencillo que imaginemos nombres de funciones y variables que ayuden a la gente a entender el programa. Si la línea de código

fuera `$areaFC = areaCirc ($fRad);`

su significado no hubiera sido tan claro.

*** Perl nos permite usar el caracter `&` para iniciar nombres de funciones.

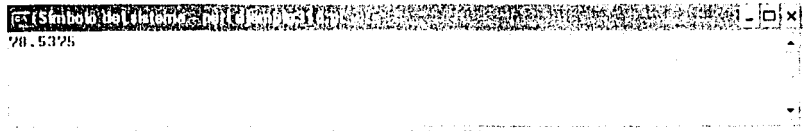
El Listado 3.16 muestra un breve programa que llama y define la función `areaDelCirculo()`.

Asignar a SareaDelPrimerCirculo el valor que retorna la función areaDelCirculo
Imprimir SareaDelPrimerCirculo.
Definir la función areaDelCirculo.
Obtener el primer parámetro del arreglo de parámetros @_.
Calcular el área y retornar el nuevo valor.

Listado 3.16 Cálculo del área de un círculo

```
$areaDelPrimerCirculo = areaDelCirculo(5);
print ("$areaDelPrimerCirculo\n");

sub areaDelCirculo {
      $radio = $_[0];
      return (3.1415 * ($radio ** 2));
}
```

El programa produce la siguiente salida:



La Figura 3.16 muestra cómo se vería la salida de dicho programa.

El hecho de que se imprima algo indica que el flujo del programa regresó a la línea de impresión después de llamar a la función `areaDelCirculo()`.
Una definición de función es muy simple. Consta de:

```
sub nombredelafuncion{
}
```

Eso es todo. Las definiciones de función en Perl nunca son más complejas, como lo pudimos atestiguar en lo escrito anteriormente.

La parte complicada es cuando intervienen los *parámetros*. Los parámetros son valores que se pasan a la función. Los parámetros se especifican dentro de los paréntesis que siguen inmediatamente al nombre de la función.

En el Listado 3.16, la llamada a la función fue `areaDelCirculo(5)`. Sólo hubo un parámetro, el número 5. Aunque sólo hay un parámetro, Perl crea (internamente) un arreglo de parámetros para que lo use la función.[▲]

[▲] Debido a que los parámetros siempre se pasan como listas, también se conoce a las funciones de Perl como operadores de lista. Y, si sólo se usa un parámetro, en ocasiones se les hace referencia como operadores unitarios.

Dentro de la función `areaDelCirculo( )`; el arreglo de parámetros se denomina `@_`. Todos los parámetros especificados durante la llamada a la función se le almacenan en el arreglo `@_`, a fin de que la función pueda recuperarlos. Nuestra pequeña función hizo esto con la línea:

```
$radio = $_[0];
```

Esta línea de código asigna el primer elemento del arreglo a `@_` a la variable escalar `$radio`.

La siguiente línea de la función:

```
return (3.1415 * ($radio ** 2));
```

Calcular el área del círculo y retorna el valor recién calculado. En este caso, el valor de retorno se asigna a la variable escalar `$areaDelPrimerCirculo`.[▲▲]

Es probable que hayamos utilizado lenguajes de cómputo que hacen una distinción entre función y subrutina, siendo la diferencia que una función retorna un valor y una subrutina no. Perl no hace tal distinción. Todas son funciones –retornen o no un valor.

Ejemplo: uso del arreglo de parámetros (@_)

Todos los parámetros para una función se almacenan en un arreglo denominado `@_`. Un efecto colateral de ello es que nosotros podemos saber cuántos parámetros se transfirieron evaluando `@_` en un contexto escalar.

Llamar a la función primerSub( ) con diversos parámetros.
Definir la función primerSub().
Asignar a $numParametros el número de elementos en el arreglo @_
Imprimir cuántos parámetros se pasaron.

Listado 3.17 uso del arreglo de parámetros (@_)

```
primerSub(1, 2, 3, 4, 5, 6);
primerSub(1..3);
primerSub("A".."Z");

sub primerSub {
      $numParametros = @_;
      print ("El numero de parámetros es: $numParametros\n");
}
```

[▲▲] Si preferimos, no es necesario usar la función `return( )` para retornar un valor, debido a que Perl retorna de manera automática el valor de la última expresión evaluada.

Este programa imprime:

La Figura 3.17 muestra cómo se vería la salida de dicho programa.

Perl permite pasar cualquier número de parámetros a una función. La función decide cuáles parámetros utilizar y en qué orden. El arreglo `@_` se usa como cualquier otro arreglo.

3.4.1 Funciones de cadena *(04B) (08B)*

El primer conjunto de funciones que veremos son las que se relacionan con las cadenas. Estas funciones nos permiten, entre otras cosas, determinar la longitud de una cadena, buscar una sub-cadena, y cambiar los caracteres de la cadena de mayúsculas a minúsculas. La Tabla 3.12 muestra las funciones de cadena de Perl.

Tabla 3.12 Funciones de cadena

Función	*Descripción*
`chomp (CADENA) o chomp (ARREGLO)`	Usa el valor de la variable `$/` para eliminar terminaciones de la CADENA o de cada elemento de un ARREGLO. La terminación de la línea sólo se elimina si corresponde con el valor actual de `$/`.
`chop (CADENA) o chop (ARREGLO)`	Elimina el último caracter de una CADENA o de cada elemento de un ARREGLO. Retorna el último caracter eliminado.
`chr (NÚMERO)`	Retorna el caracter representado por NÚMERO en la tabla de caracteres ASCII. Por ejemplo `chr (65)` retorna la letra A.
`crypt (CADENA1, CADENA2)`	Encripta la CADENA1. Por desgracia Perl no proporciona una función para desencriptar. Esta función no está disponible bajo los sistemas operativos de Windows.
`index (CADENA, SUBCADENA, POSICIÓN)`	Retorna la posición de la primera ocurrencia de la SUBCADENA dentro de la CADENA, a partir de la POSICIÓN o después de ella. Si no se especifica POSICIÓN, la búsqueda comienza desde el principio de la CADENA.
`join (CADENA, ARREGLO)`	Retorna una cadena que consta de todos los elementos del ARREGLO unidos por la CADENA. Por ejemplo `join(">>", ("AA", "BB", "CC"))` retorna `"AA>>BB>>CC"`.
`lc (STRING)`	Retorna una cadena con todas las letras de CADENA en minúsculas. Por ejemplo `lc ("ABCD")` retorna `"abcd"`.

`lcfirst (CADENA)`	Retorna una cadena con la primera letra de CADENA en minúscula. Por ejemplo `lcfirst ("ABCD")` retorna `"aBCD"`.
`length (CADENA)`	Retorna la longitud de CADENA.
`rindex (CADENA, SUBCADENA, POSICIÓN)`	Retorna la posición de la última ocurrencia de la SUBCADENA dentro de la CADENA, a partir de la POSICIÓN o después de ella. Si no se especifica la POSICIÓN, la búsqueda comienza al final de la CADENA.
`split (PATRÓN, CADENA, LÍMITE)`	Divide una cadena con base en cierto delimitador. En un contexto de arreglo, retorna una lista de las cosas que encontró. En un contexto escalar, retorna el número de cosas encontradas.
`substr (CADENA, DESPLAZAMIENTO, LONGITUD)`	Retorna una porción de la CADENA de acuerdo a lo determinado por el DESPLAZAMIENTO y la LONGITUD. Si no se especifica la LONGITUD, retorna todo a partir del DESPLAZAMIENTO hasta el final de la CADENA. Se puede emplear un DESPLAZAMIENTO negativo para comenzar desde la derecha de la CADENA.
`uc (CADENA)`	Retorna una cadena con todas las letras de CADENA en mayúsculas. Por ejemplo, `uc ("abcd")` retorna `"ABCD"`.
`ucfirst (CADENA)`	Retorna Una cadena con la primera letra de CADENA en mayúscula. Por ejemplo, `ucfirst ("abdc")` retorna `"Abcd"`.

▲

Ejemplo: modificación del valor de una cadena.

Con frecuencia nos encontramos en la necesidad de cambiar parte del valor de una cadena, por lo regular esto siempre sucede en medio de la cadena. Cuando surge esta necesidad, acudimos a la función `substr()`. Normalmente, la función `substr()` retorna una subcadena con base en tres parámetros: la cadena a utilizar, la posición dónde iniciar y la longitud de la cadena a retornar.

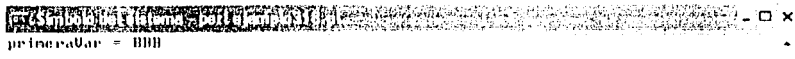
Asignar a $primeraVar de retorno de substr().
Imprimir $primeraVar.

Listado 3.18 modificación del valor de una cadena

```
$primeraVar = substr("0123BBB789", 4, 3);
print ("primeraVar = $primeraVar\n");
```

▲ Como regla general, si Perl encuentra un número en dónde espera una cadena, convierte silenciosamente el número a una cadena sin que se requiera que nosotros hagamos nada.

Este programa imprime:



La Figura 3.18 muestra cómo se vería la salida de dicho programa.

La función `substr()` comienza en la quinta posición y retorna los tres siguientes caracteres. La cadena retornada puede emplearse para una asignación como en el ejemplo anterior, como un elemento de un arreglo para concatenación de cadenas, o para otro centenar de opciones.

3.4.2 Funciones de arreglos *(04B) (08B)*

Los arreglos constituyen una gran parte del lenguaje Perl y este tiene muchas funciones para ayudarnos a trabajar con ellos. Algunas de las acciones que realizan los arreglos comprenden la eliminación de elementos, la verificación de la existencia de un elemento, la inversión de todos los elementos en un arreglo y la clasificación de los elementos. La Tabla 3.13 lista todas las funciones que podemos usar con los arreglos.

Tabla 3.13 Funciones de arreglos

Función	*Descripción*
`defined (VARIABLE)`	Retorna verdadero si la VARIABLE tiene un valor real y nulo si aún no se ha asignado un valor a la variable. Esto no se limita a los arreglos; se puede verificar cualquier tipo de dato. Vea también la función **`exists`** para información sobre las llaves de arreglos asociativos.
`delete (LLAVE)`	Elimina la pareja llave valor del arreglo asociativo dado. Si elimina un valor del arreglo **`%ENV`**, se modifica el ambiente del proceso actual, no el del proceso padre.
`each (ARREGLO_ASOC)`	Retorna una lista de dos elementos que contiene una pareja de llave y valor del arreglo asociativo dado. La función se usa principalmente a fin de iterar sobre los elementos de un arreglo asociativo. Retorna una lista nula después de leído el último elemento.
`exists (LLAVE)`	Retorna verdadero si la llave es parte del arreglo asociativo especificado. Por ejemplo **`exists ($arreglo{"naranja"})`** retorna verdadero si el arreglo asociativo **`%arreglo`** tiene una llave con el valor de "Naranja".
`join (CADENA, ARREGLO)`	Retorna una cadena que consiste en todos los elementos del ARREGLO unidos por la CADENA. Por ejemplo, `join (">>", ("AA", "BB", "CC"))` retorna `"AA>>BB>>CC"`.

`keys (ARREGLO_ASOC)`	Retorna una lista que contiene todas las llaves en un arreglo asociativo dado. La lista no tiene un orden en particular.
`map (EXPRESIÓN, ARREGLO)`	Evalúa la EXPRESIÓN para cada elemento del ARREGLO. Se asigna la variable especial `$` a cada elemento del ARREGLO inmediatamente antes de evaluar la EXPRESIÓN.
`pack (CADENA, ARREGLO)`	Crea una estructura binaria de los elementos del ARREGLO, usando la CADENA como guía.
`pop (ARREGLO)`	Retorna el último valor de un ARREGLO. También reduce en uno el tamaño del ARREGLO.
`push (ARREGLO, ARREGLO2)`	Agrega el contenido de ARREGLO2 a ARREGLO1. Esto aumenta el tamaño de ARREGLO1 según sea necesario.
`reverse (ARREGLO)`	Cuando se usa en un contexto de arreglo, invierte los elementos de un ARREGLO dado. Cuando se usa en un contexto escalar, el arreglo es convertido a una cadena, y ésta se invierte.
`scalar (ARREGLO)`	Evalúa el ARREGLO en un contexto escalar y retorna el número de elementos en el ARREGLO.
`shift (ARREGLO)`	Retorna el primer valor de un ARREGLO. Además reduce en uno el tamaño del ARREGLO.
`sort (ARREGLO)`	Retorna una lista que contiene los elementos del ARREGLO clasificados.
`splice (ARREGLO1, DESPLAZAMIENTO, LONGITUD, ARREGLO2)`	Reemplaza Elementos del ARREGLO1 con elementos del ARREGLO2. Retorna una lista que contiene todos los elementos eliminados. Al determinar el valor del DESPLAZAMIENTO, recordemos que la variable `$ [` puede cambiar la base de los índices del arreglo.
`split (PATRÓN, CADENA, LÍMITE)`	Divide una cadena con base en cierto delimitador. En un contexto de arreglo, retorna una lista de las cosas que encontró. En un contexto escalar, retorna el número de cosas encontradas.
`undef (VARIABLE)`	Siempre retorna el valor indefinido. Además, hace indefinida la VARIABLE, la cual debe ser una escalar, un arreglo completo, o un nombre de subrutina.
`unpack (CADENA, ARREGLO)`	Realiza la acción opuesta `pack()`.
`unshift (ARREGLO1, ARREGLO2)`	Arregla los elementos del ARREGLO2 al principio del ARREGLO1. Nótese que los elementos agregados conservan su orden original. Retorna el tamaño del nuevo ARREGLO1.
`values (ARREGLO_ASOC)`	Retorna una lista que contiene todos los valores en un arreglo asociativo dado. La lista no tiene un orden en particular.

Al igual que con las funciones de cadena sólo se explorarán unas cuantas de estas funciones. Una vez que hayamos visto los ejemplos, podremos manejar las demás sin ningún problema.

Ejemplo: impresión de un arreglo asociativo.

La función `each()` retorna parejas de llave y valor de un arreglo asociativo, uno por uno en una lista. Esto se denomina *iterar* sobre los elementos de un arreglo. Iteración es sinónimo de hacer un ciclo. Así, que también podría decirse que la función `each()` comienza al inicio de un arreglo y realiza un ciclo a través de cada elemento hasta llegar al final del arreglo. Esta capacidad nos permite trabajar con parejas llave-valor de manera fácil y rápida.

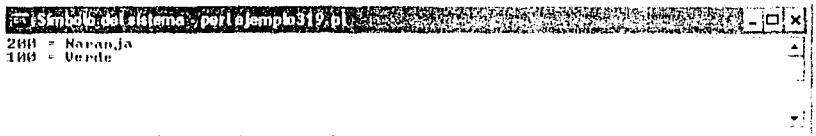
La función `each()` no realiza por sí misma el ciclo. Requiere de una pequeña ayuda de algunos enunciados de control de Perl. Para este ejemplo, usaremos el ciclo `while` para imprimir un arreglo asociativo. El enunciado de control `while (CONDICIÓN) { }` ejecuta de manera continua todo el código de programa comprendido entre los corchetes hasta que la CONDICIÓN sea falsa.

Crear un arreglo asociativo con parejas de número-color.
Usar el ciclo while para iterar sobre los elementos del arreglo.
Imprimir la pareja llave-valor.

Listado 3.19 impresión de un arreglo asociativo

```
%arreglo = ("100", "Verde", "200", "Naranja");

while (($llave, $valor) = each (%arreglo)){
print("$llave = $valor\n");
}
```

Este programa imprime:



La Figura 3.19 muestra cómo se vería la salida de dicho programa.

La función `each()` retorna falso cuando se llega al final del arreglo. Por lo tanto, puede usarla como base de la condición del ciclo `while`. Al llegar al fin del arreglo, el programa continúa la ejecución después del corchete de cierre. En este caso, el programa simplemente termina.

3.5 Enunciados *(04B) (08B)*

Si nosotros observamos un programa Perl desde un muy alto nivel, está hecho de enunciados. Los *enunciados* son una unidad completa de instrucción para que procese la computadora. La computadora ejecuta –en secuencia– cada enunciado que ve hasta que se procesa un salto o bifurcación.

Los enunciados pueden ser muy simples o muy complejos. El enunciado más sencillo es como éste:

```
123;
```

el cual es una constante numérica seguida de un punto y coma. El punto y coma es muy importante, indica a Perl que está completo el enunciado. Un enunciado más complicado podría ser:

```
$tamanoLibro = ($numeroDePaginas >= 1200 ? "Largo" : "Normal");
```

el cual dice que si el número de páginas es 1,200 o mayor, entonces asigna "Largo" a `$tamanoLibro`; en caso contrario, asigna "Normal" `$tamanoLibro`.

En Perl, todo enunciado tiene un valor. En el primer ejemplo, el valor del enunciado es 123. En el segundo ejemplo, el valor del enunciado podría ser "Largo", o bien, "Normal", dependiendo del valor de `$numeroDePagina`. El último valor que se evalúa se convierte en el valor para el enunciado.

Como en el lenguaje humano, en el que nosotros conformamos enunciados a partir de partes del habla –pronombres, verbos y modificadores- podemos también separar en partes los enunciados de Perl. Las partes son las constantes, variables y funciones que ya hemos visto.

Las frases del lenguaje humano –como "pasear al perro"– tienen también su contraparte en los lenguajes de cómputo. El equivalente en cómputo es una expresión. Las *expresiones* son una secuencia de constantes, variables y funciones conectadas por uno o más operadores que se evalúan en un sólo valor, escalar o de arreglo. Se puede promover una expresión a enunciado arreglando un punto y coma. Esto se hizo en el primer ejemplo previo. La simple adición de un punto y coma a la constante la convirtió en un enunciado ejecutable por Perl.

También las expresiones pueden tener efectos colaterales. Las funciones que se llaman hacen cosas que no son obvias de inmediato (como la asignación de variables globales), o se pueden utilizar los operadores de incremento previo y posterior para modificar el valor de una variable.

3.5.1 Bloques de enunciados *(04B) (08B)*

Un *bloque de enunciados* es un grupo de enunciados encerrados entre llaves. Perl visualiza a un bloque de enunciados como un enunciado. El último enunciado ejecutado se convierte en el valor del bloque de enunciados. Esto significa que en cualquier parte que podemos usar un sólo enunciado —como el enunciado *map*— podemos utilizar un bloque de enunciados. También podemos crear variables que sean locales para el bloque de enunciados. Así que, sin meternos en el problema de crear una función, podemos aun aislar una fracción de código de otra.

La siguiente es la forma en que se usa con frecuencia un bloque de enunciados:

```
$primeraVar = 10;
{
      $segundaVar >>=2;
      $segundaVar ++;
}
$terceraVar =20;
```

El bloque de enunciados sirve para enfatizar que el código interior se separa del resto del programa. En este caso, la inicialización de `$segundaVar` es un poco más compleja que las otras variables. Utilizar un bloque de enunciados no modifica en forma alguna la ejecución del programa, simplemente es una ayuda visual para señalar secciones de código y una forma de crear variables locales.

3.5.2 Bloques de enunciados y variables locales *(04B) (08B)*

Normalmente, es buena idea colocar toda la inicialización de variables en la parte superior del programa o función. Sin embargo, si estamos dando mantenimiento a un código ya existente, es posible que necesitemos usar un bloque de enunciados y variables locales para minimizar el impacto de los cambios en el resto del código, en especial si se nos acaba de asignar la responsabilidad de un programa que alguien más escribió.

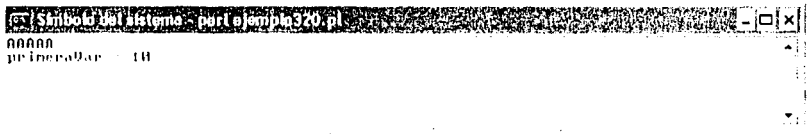
Podemos usar la función `my()` para crear variables cuyo alcance esté limitado al bloque de enunciados. Esta técnica es muy útil para variables temporales que no se necesitarán en ninguna otra parte del programa. Por ejemplo, podríamos tener un enunciado complejo que quisiéramos dividir en otros más pequeños para hacerlo más comprensible. O es posible que deseemos insertar algún enunciado `print` para ayudar a depurar una parte del código y requiramos de algunas variables temporales para adecuar el enunciado `print`.

Asignar diez a $primeraVar.
Iniciar el bloque de enunciados.
Crear una versión local de $primeraVar con un valor de A.
Imprimir $primeraVar cinco veces.
Terminar el bloque de enunciados.
Imprimir la variable global $primeraVar.

Listado 3.20 Bloques de enunciados y variables locales

```
$primeraVar = 10;
{
      my ($primeraVar) = "A";
      print $primeraVar x 5 . "\n";
}
print ("primeraVar = $primeraVar\n");
```

Este programa imprime:



La Figura 3.20 muestra cómo se vería la salida de dicho programa.

Podemos ver que el valor de `$primeraVar` no ha sido modificado por el bloque de enunciados aunque dentro de él se usa una variable `$primeraVar`. Esto demuestra que la variable utilizada dentro del bloque de enunciados tiene de hecho un alcance local.*

3.5.3 Tipos de enunciados *(04B) (08B)*

Así como hay diversos tipos de expresiones, existen también varios tipos de enunciados. La Tabla 3.14 lista siete diferentes tipos de enunciados.

Tabla 3.14 Tipos de enunciados en Perl

Tipo de enunciado	*Descripción*
Enunciados sin acción	Estos enunciados evalúan un valor pero no realizan acciones.
Enunciados de acción	Estos enunciados realizan alguna acción
Enunciados de asignación	Estos enunciados asignan un valor a una o más variables. Se exponen, junto con el operador de asignación.
Enunciados de decisión	Estos enunciados le permiten verificar una condición y elegir entre una o más acciones.
Enunciados de salto	Estos enunciados permiten modificar de manera incondicional el flujo del programa hacia otro punto de su código.

* Siempre es bueno usar bloques de enunciados cuando requerimos de manera temporal enviar la salida de una depuración a un archivo. Después, una vez localizados todos los errores y que la necesidad de depurar esté concluida, podemos eliminar fácil y rápidamente el bloque de enunciados ya que todo el código está en un solo punto.

Enunciados de ciclo	Estos enunciados le permiten efectuar una serie de enunciados en forma repetida en tanto cierta condición sea verdadera o hasta que cierta condición sea verdadera.
Enunciados modificados	Estos enunciados le permiten usar las palabras claves `if`, `unless`, `until` y `while` para cambiar el comportamiento de un enunciado.*

Los *enunciados sin acción* son evaluados por Perl y tienen un valor pero no realizan ninguna acción. Por ejemplo, el enunciado de Perl 10 + 20; tiene un valor de 30, pero debido a que no se modificaron variables, no se hizo ningún trabajo. El valor de 20 no está almacenado en ningún lado, y se olvida rápidamente después del enunciado.

¿De qué sirve un *enunciado sin acción* si no se realiza ningún trabajo? Muchos programadores de Perl usan estos sencillos enunciados como variables de retorno en funciones. Por ejemplo:

```
sub primeraVar {
      doAlgo();
      condition == verdadero ? "Exito" : "Fracaso");
}
```

Debido a que al salir de una función Perl retorna el valor del último enunciado evaluado, podemos emplear enunciados sin acción para indicar a Perl qué valor debe retornar al programa principal Notaremos que incluso aunque se usó el operador ternario, no se hizo ningún trabajo ya que no hay llamadas a funciones u operadores unitarios.

Los *enunciados de acción* expresiones para realizar alguna tarea. Pueden incrementar o disminuir una variable y llamar a una función.

Los *enunciados modificados* usan expresiones conjuntamente con una palabra clave modificadora para efectuar cierta acción. Existen cuatro palabras clave modificadoras: `if`, `unless`, `until` y `while`. La sintaxis básica de un enunciado modificado es

EXPRESION modificador (CONDICION);

Veamos algunos ejemplos de enunciados modificados.

* Una palabra clave es una palabra que está reservada para el uso de Perl. Estas palabras (`if`, `elseif`, `while`, `unless`, `until`, `for`, `last`, `next`, `redo` y `continue`) son parte integral del lenguaje y le proporcionan la capacidad de controlar el flujo del programa.

Ejemplo: uso del modificador `if`

El modificar `if` indica a Perl que la expresión debe evaluarse sólo si se cumple una determinada condición. La sintaxis básica de un enunciado modificado con un modificador `if` es

EXPRESION `if` (CONDICION);
Ésta es una forma compacta de decir

```
if (CONDICION) {
      EXPRESION;
}
```

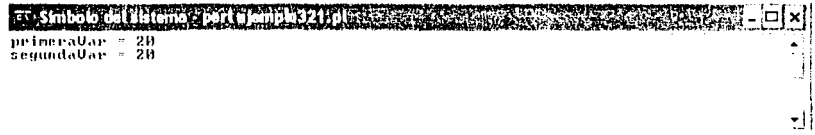
El siguiente es un ejemplo que muestra que el modificador `if` puede evitar la evaluación de una expresión.

Inicializar a 20 las variables $primeraVar y $segundaVar.
Incrementar $primeraVar siempre y cuando $segundaVar sea igual a 10.
Imprimir los valores de $primeraVar y $segundaVar.

Listado 3.21 Uso del modificador `if`

```
$primeraVar = 20;
$segundaVar = 20;

$primeraVar++ if ($segundaVar == 10);

print("primeraVar = $primeraVar\n");
print("segundaVar = $segundaVar\n");
```

Este programa imprime:



La Figura 3.21 muestra cómo se vería la salida de dicho programa.

Este programa no incrementa `$primeraVar` debido a que el valor de `$segundaVar` es 20 al momento de evaluar la condición. Si modificara el 10 por 20 en la condición, Perl incrementaría `$primeraVar`.

Ejemplo: uso del modificador `unless`

El modificador `unless`, es el opuesto al modificador `if`. Este modificador evalúa una expresión a menos que una condición sea verdadera. La sintaxis básica de un enunciado modificado con un modificador `unless` es:

EXPRESION `unless` (CONDICION);

Ésta es una forma compacta de decir

```
if (! CONDICION) {
EXPRESION;
}
```

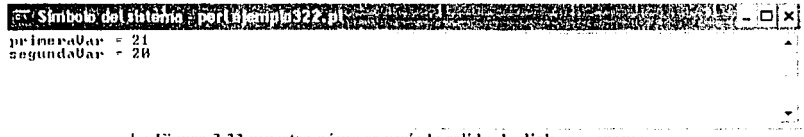
Este modificador ayuda a mantener comprensible con claridad el código del programa, debido a que no tenemos el operador lógico `not` para cambiar el valor de una condición para poder evaluar una expresión. Recordemos el ejemplo anterior.

Iniciar a 20 las variables $primeraVar y $segundaVar.
Incrementar $primeraVar a menos que $segundaVar sea igual a 10.
Imprime los valores de $primeraVar y $segundaVar.

Listado 3.22 Uso del modificador `unless`

```
$primeraVar = 20;
$segundaVar = 20;

$primeraVar++ unless ($segundaVar == 10);

print("primeraVar = $primeraVar\n");
print("segundaVar = $segundaVar\n");
```

Este programa imprime:



La Figura 3.22 muestra cómo se vería la salida de dicho programa.

Si estuviéramos utilizando todo el modificador `if`, el enunciado modificado se leería

```
$primeraVar++ if ($segundaVar != 10);
```

El modificador `unless` es más directo. Siendo todo igual, el concepto de que `$segundaVar` sea igual a 10 es más fácil de captar que el concepto de que `$segundaVar` no sea igual a 10.

Ejemplo: uso del modificador `until`

El modificador `until` es un poco más complejo que los modificadores `if` o `unless`. `until` evalúa la expresión en forma repetida hasta que la condición se hace verdadera. La sintaxis básica de un enunciado modificado con el modificador `until` es

EXPRESION `until` (CONDICION);

Ésta es una manera compacta de decir

```
until (CONDICION) {
EXPRESION;
}
```

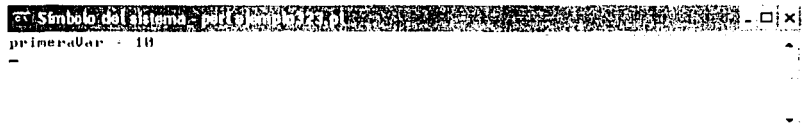
La expresión sólo se evalúa cuando la condición es falsa. Si la condición es verdadera al encontrar el enunciado, la expresión no se evaluará nunca. El siguiente ejemplo lo demuestra:

Inicializar $primeraVar a 10.
Evaluar en forma repetida $primeraVar++ hasta que sea verdadera la condición $primeraVar > 2.
Imprimir el valor de $primeraVar.

Listado 3.23 Uso del modificador `until`

```
$primeraVar = 10;
$primeraVar++ until ($primeraVar > 2);

print ("primeraVar = $primeraVar\n");
```

Este programa imprime:



La Figura 3.23 muestra cómo se vería la salida de dicho programa.

Este ejemplo muestra que la expresión `$primeraVar++` no se ejecutó nunca debido a que la condición era verdadera la primera vez que se evaluó. Si se hubiera ejecutado el valor de `$primeraVar++` hubiese sido 11 al imprimir. En este caso, el modificador `until` trabajó exactamente como el modificador `unless`.

Ejemplo: uso del modificador `while`

El modificador `while` es el opuesto al modificador `until`. Evalúa repetidamente la expresión mientras que la condición sea verdadera. Cuando la condición se hace falsa, el enunciado termina. La sintaxis básica de un enunciado modificado con un modificador `while` es

EXPRESION `while` (CONDICION);

Ésta es una forma compacta de decir:

```
while (CONDICION) {
EXPRESION;
}
```

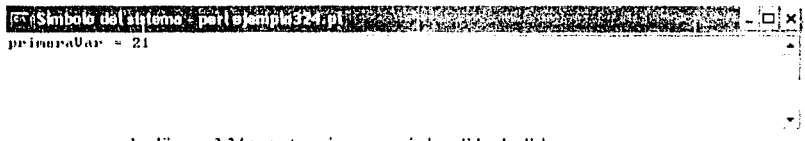
La expresión se evalúa sólo cuando la condición es verdadera. Si la condición es falsa al momento de encontrar el enunciado, la expresión no se evaluará nunca. El siguiente es un ejemplo del modificador `while`.

Inicializar $primeraVar a 10.
Evaluar en forma repetida $primeraVar++ mientras que sea verdadera la condición $primeraVar < 20.
Imprimir el valor de $primeraVar.

Listado 3.24 Uso del modificador `while`

```
$primeraVar = 10;
$primeraVar while ($primeraVar++ < 20);

print ("primeraVar = $primeraVar\n");
```

Este programa imprime:



La Figura 3.24 muestra cómo se vería la salida de dicho programa.

Podemos comparar este ejemplo directamente con el ejemplo anterior dado para el modificador `until`. Los operadores en condiciones también son de naturaleza opuesta.

3.6 Enunciados de control *(04B) (08B)*

En la sección anterior, "*Enunciados*", se expusieron los enunciados sin acción, con acción y modificados. Esta sección expone otros tres tipos de enunciados: de decisión, de ciclo y de salto o bifurcación.

Veremos como utilizar el enunciado `if` para decidir sobre uno o más cursos de acción. Los enunciados de ciclo se usan para repetir una serie de enunciados hasta que una condición dada sea verdadera o falsa. Y, por último, redondearemos esta sección observando los enunciados de salto o bifurcación, los cuales nos permiten controlar el flujo del programa desplazándose en forma directa al inicio o al final de un bloque de enunciados.

3.6.1 Enunciados de decisión *(04B) (08B)*

Los *enunciados de decisión* usan la palabra clave `if` para ejecutar un bloque de enunciados con base en la evaluación de una expresión o para elegir entre ejecutar uno de dos bloques de enunciados con base en dicha evaluación. Se emplean con mucha frecuencia. Por ejemplo, es posible que un programa requiera ejecutar una sección de código si un cliente es del sexo femenino y otra sección de código, si el cliente es de sexo masculino. A continuación se muestra la ejemplificación de la sintaxis del enunciado `if`

Ejemplo: el enunciado `if`

La sintaxis del enunciado `if` es la siguiente:

```
if (CONDICION) {
      # Bloque de código ejecutado
      # si la condición es del sexo femenino.
}else {
      # Bloque de código ejecutado
      # si la condición es del sexo masculino.
}
```

En ocasiones, necesitaremos elegir entre varios bloques de enunciados, como cuando requerimos ejecutar un bloque distinto de código para cada mes del año. Para este tipo de decisiones, nosotros emplearemos el enunciado `if...elsif`, el cual tiene la siguiente sintaxis:

```
if (CONDICION_UNO) {
      # Bloque de código ejecutado
      # si la condición uno es cierta.
}elseif (CONDICION_DOS){
      # Bloque de código ejecutado
      # si la condición uno es cierta.
}else {
      # Bloque de código ejecutado
      # si todas las demás condiciones son falsas.
}
```

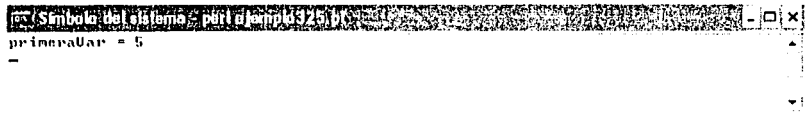
Las expresiones condicionales pueden utilizar cualquier de los operadores expuestos en la sección de, "*Operadores*". Se puede incluso usar los operadores de asignación ya que el valor de un expresión de asignación es el valor que se está asignando. Esta última oración pudiera parecer algo confusa, así que veamos un ejemplo.

Asignar un valor de 10 a $primeraVar.
Restar cinco a $primeraVar y si el valor resultante es verdadero (por ejemplo, diferente de cero), entonces ejecuta el siguiente bloque de enunciados.

Listado 3.25 Uso del enunciado `if` *

```
$primeraVar = 10;
if ($primeraVar -= 5){
      print ("primeraVar = $primeraVar\n");
}
```

Este programa imprime:



La Figura 3.25 muestra cómo se vería la salida de dicho programa.

Este ejemplo, además de demostrar el uso de los operadores de asignación dentro de expresiones condicionales, muestra también que la parte `else` del enunciado `if` es opcional. Si se hubiera incluido en el código la parte de `else`, sólo se hubiera ejecutando cuando `$primeraVar` iniciara con un valor de 5.

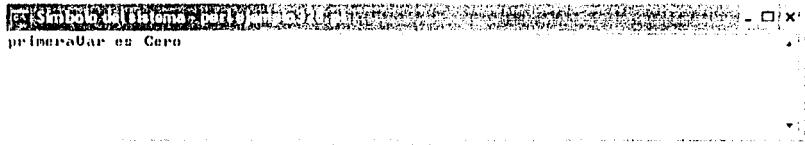
Asignar un valor de 10 a $primeraVar
Restar cinco a $primeraVar y si el valor resultante es verdadero (en otras palabras, diferente de cero), entonces imprime $primeraVar. Si no, imprime "$primeraVar es cero".

Listado 3.26 Uso del enunciado `if...else`

```
$primeraVar = 5;
if ($primeraVar -=5) {
      print("primeraVar = $primeraVar\n");
} else {
      print("primeraVar es Cero\n");
}
```

* En Perl, las llaves que rodean el bloque de enunciados no son opcionales. Incluso los bloques de enunciados de una sola línea deberán encerrarse entre llaves.

Este programa imprime:



La Figura 3.26 muestra cómo se vería la salida de dicho programa.

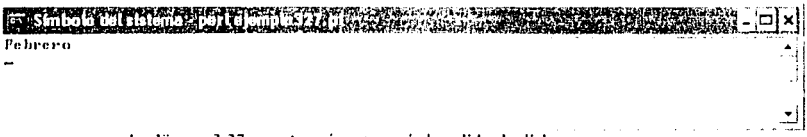
El Listado 3.61 muestra el uso de la cláusula `else` del enunciado `if`. Los enunciados en la cláusula `else` se ejecutaron debido a que el valor de `$primeraVar` menos cinco fue cero. También podemos usar el enunciado `if` para seleccionar entre varios bloques de enunciados. Con este fin se usa la norma `if ….elsif` del enunciado.

Inicializar $mes con un valor de 2.
Si el valor de $mes es 1, entonces imprime Enero.
Si el valor de $mes es 2, entonces imprime Febrero.
Si el valor de $mes es 3, entonces imprime Marzo.
Imprime cualquier mensaje para cualquier otro valor de $mes.

Listado 3.27 Uso del enunciado `if…elsif`

```
$mes = 2;

if ($mes ==1){
      print ("Enero\n");
}
elsif ($mes ==2){
      print ("Febrero\n");
}
elsif ($mes ==3){
      print ("Marzo\n");
}
else {
      print ("Ninguno de los primeros tres meses\n");
}
```

Este programa imprime:



La Figura 3.27 muestra cómo se vería la salida de dicho programa.

La cláusula `else` al final de cada `elsif` sirve para captar cualquier valor desconocido o no previsto y es un buen lugar para colocar mensajes de error. Es frecuente que esos mensajes de error incluyan la escritura del valor erróneo en un archivo de bitácora con el fin de evaluar los errores. Después de la evaluación, podemos decidir si se requiere modificar el programa para manejar el valor no previsto mediante otra cláusula `elsif`.

3.6.2 Enunciados de ciclo *(04B) (08B)*

Un ciclo se usa para repetir la ejecución de un bloque de enunciados hasta que se cumple una determinad condición. Se puede usar un ciclo para hacer una iteración a través de un arreglo en busca de un valor. También se pueden usar los ciclos para controlar cantidades.
En realidad, el número de usos distintos de los ciclos es limitado. Existen tres tipos de ciclos: `while`, `until` y `for`.

Ejemplo: ciclos `while`

Los ciclos `while` se usan para repetir un bloque de enunciado en tanto una condición sea cierta. Hay dos formas de ciclo: una, en la que se verifica la condición antes de la ejecución de los enunciados (el ciclo `do ....while`), y en otra dónde la condición se verifica después de ejecutar los enunciados (el ciclo `while`).

El ciclo `do....while` tiene la sintaxis:

```
do {
      DECLARACIONES
} while (CONDICION);
```

El ciclo `while` tiene la sintaxis:

```
while (CONDICION) {
      DECALARACIONES
}
continue {
      DECALARACIONES
}
```

Los enunciados en el bloque `continue` del ciclo `while` se ejecutan justo antes de que el ciclo inicie la siguiente iteración.

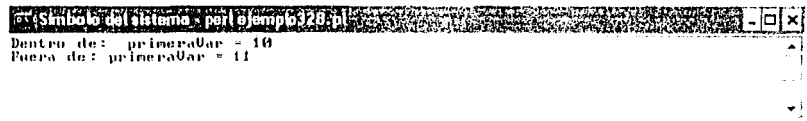
El bloque de enunciados de un ciclo `do...while` siempre se ejecutará por lo menos una vez. Esto se debe a que la condición se verifica después de ejecutar el bloque de enunciados nunca antes. El siguiente es un ejemplo de un ciclo `do...while`.

Inicializar $primeraVar con el valor de 10.
Iniciar el ciclo do...while
Imprimir $primeraVar.
Verificar la condición while; si es verdadera, regresa al inicio del bloque de enunciados.
Imprimir el valor de $primeraVar.

Listado 3.28 ciclos `while`

```
$primeraVar = 10;
do{
      print("Dentro de:  primeraVar = $primeraVar\n");
      $primeraVar++;
} while ($primeraVar < 2);

print ("Fuera de: primeraVar = $primeraVar\n");
```

Este programa imprime:



La Figura 3.28 muestra cómo se vería la salida de dicho programa.

Este ejemplo muestra que el bloque de enunciados se ejecuta incluso aunque la condición `$primeraVar < 2` sea falsa al iniciar el ciclo. Esta capacidad es útil en ocasiones al contar en forma descendente, como al imprimir las páginas de un reporte.

Ejemplo: ciclos `until`

Los ciclos `until` se usan para repetir un bloque de enunciados mientras una condición sea falsa. Como en el ciclo `while` anterior, también hay dos formas de ciclo `until`: una, en dónde la condición se verifica antes de ejecutar el bloque de enunciados (el ciclo `do...until`), y otra, en la que se verifica la condición después de ejecutar los enunciados (el ciclo `until`).

El ciclo `do...until` tiene la siguiente sintaxis:

```
do {
      DECALARACIONES
} until (CONDICION);

until (CONDICION) {
      DECALARACIONES
}
```

Una vez más, el tipo de ciclo que utilicemos dependerá de las necesidades que tengamos en el momento.

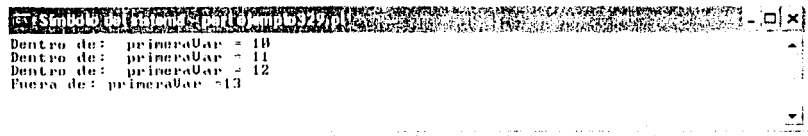
Este ejemplo nos muestra que el bloque de enunciados no se evalúa nunca si la condición es verdadera al inicio del ciclo `until`. El siguiente es otro ejemplo de un ciclo `until` que nos muestra la ejecución de un bloque de enunciados:

Inicializar $primeraVar con el valor de 10.
Iniciar el ciclo until y verificar la condición
Imprimir el valor de $primeraVar
Incrementar el valor de $primeraVar.
Regresar al inicio del ciclo del bloque de enunciados y verificar de nuevo la condición.
Imprimir el valor de $primeraVar.

Listado 3.29 ciclos `until`

```
$primeraVar =10;
until ($primeraVar > 12) {
      print("Dentro de:  primeraVar = $primeraVar\n");
      $primeraVar++;
};
print ("Fuera de: primeraVar =$primeraVar\n");
```

Este programa imprime:



La Figura 3.29 muestra cómo se vería la salida de dicho programa.

Ejemplo: ciclos `for`

Una de las tareas más importantes en programación consiste en realizar un ciclo un número determinado de veces. Ya sea que necesitemos ejecutar una cierta función para cada cliente en su base de datos o imprimir una página en un reporte, puede utilizar el ciclo `for`. Su sintaxis es:

```
for (INICIALIZACIÓN, CONDICIÓN; INCREMENTO/DECREMENTO) {
      DECLARACIONES
}
```

Primero se ejecuta la expresión *inicialización*, antes de comenzar el ciclo. Se puede usar para inicializar las variables que se utilizan dentro del ciclo. Por supuesto esto puede hacerse en una línea antes del ciclo `for`. Sin embargo, incluir la inicialización dentro del ciclo ayuda a identificar las variables del mismo.

Al inicializar variables, debemos asegurarnos de no confundir el operador de igualdad (==) con el operador de asignación (=). El siguiente es un ejemplo de cómo se vería un error de este tipo:

```
for ($inicio == 0; $inicio < 0; $inicio++)
```

Se debe quitar uno de los signos de igual. Si piensa que tiene un problema en la programación del ciclo `for`, asegurémonos de revisar los operadores.

La expresión *condición* se usa para determinar si debe continuar o concluir el ciclo. Cuando la expresión condicional se evalúa como falsa, el ciclo terminará.

La expresión *incremento/decremento* se emplea para modificar las variables del ciclo en cierta forma, cada vez que ha sido ejecutado el bloque de código. El siguiente es un ejemplo de un ciclo `for` básico:

> *Comienza el ciclo inicializando a cero la variable $primeraVar. Esta variable se incrementará cada vez que se ejecute el bloque de enunciados. Dicho bloque se ejecutará mientras que $primeraVar sea menor que 100*
> *Imprimir el valor de $primeraVar cada vez que recorre el ciclo.*

Listado 3.30 ciclos `for`

```
for ($primeraVar =0; $primeraVar < 100; $primeraVar++){
      print("Dentro de:  primeraVar = $primeraVar\n");
};
```

Este programa imprime:

```
Dentro de:   primeraVar = 0
Dentro de:   primeraVar = 1
Dentro de:   primeraVar = 2
Dentro de:   primeraVar = 3
Dentro de:   primeraVar = 4
Dentro de:   primeraVar = 5
Dentro de:   primeraVar = 6
Dentro de:   primeraVar = 7
- - - - - - - - - - - - -
Dentro de:   primeraVar = 90
Dentro de:   primeraVar = 91
Dentro de:   primeraVar = 92
Dentro de:   primeraVar = 93
Dentro de:   primeraVar = 94
Dentro de:   primeraVar = 95
Dentro de:   primeraVar = 96
Dentro de:   primeraVar = 97
Dentro de:   primeraVar = 98
Dentro de:   primeraVar = 99
```

La Figura 3.30 muestra cómo se vería la salida de dicho programa.

Este programa exhibirá los números del 0 al 99. Al terminar el ciclo, `$primeraVar` será igual a 100.

En las expresiones de inicialización y de *incremento/decremento*, podemos usar el operador de coma para evaluar dos expresiones a la vez.*

* El operador de coma nos permite usar dos expresiones en dónde normalmente Perl sólo nos permite tener una. El valor del enunciado se convierte en el valor del último enunciado evaluado.

Comienza el ciclo inicializando SprimeraVar en 100 y SsegundaVar en 0. Cada vez que se ejecute el bloque, la variable SprimeraVar disminuirá y SsegundaVar se incrementará.

El bloque de enunciados se ejecutará mientras que SprimeraVar sea mayor que 0. Imprimir el valor de SprimeraVar y de SsegundaVar cada vez que recorre el ciclo.

Listado 3.31 ciclos `for` usando el operador de (,)

```
for ($primeraVar = 100, $segundaVar = 0;
      $primeraVar > 0;
      $primeraVar--, $segundaVar++){

      print("Dentro de:   primeraVar = $primeraVar   segundaVar =
      $segundaVar\n");
}
```

Este programa imprime:

```
Dentro de:  primeraVar = 100  segundaVar = 0
Dentro de:  primeraVar = 99   segundaVar = 1
Dentro de:  primeraVar = 98   segundaVar = 2
Dentro de:  primeraVar = 97   segundaVar = 3
Dentro de:  primeraVar = 96   segundaVar = 4
Dentro de:  primeraVar = 95   segundaVar = 5
Dentro de:  primeraVar = 94   segundaVar = 6
Dentro de:  primeraVar = 93   segundaVar = 7
Dentro de:  primeraVar = 92   segundaVar = 8
Dentro de:  primeraVar = 91   segundaVar = 9
. . . . . .
Dentro de:  primeraVar = 12   segundaVar = 88
Dentro de:  primeraVar = 11   segundaVar = 89
Dentro de:  primeraVar = 10   segundaVar = 90
Dentro de:  primeraVar = 9    segundaVar = 91
Dentro de:  primeraVar = 8    segundaVar = 92
Dentro de:  primeraVar = 7    segundaVar = 93
Dentro de:  primeraVar = 6    segundaVar = 94
Dentro de:  primeraVar = 5    segundaVar = 95
Dentro de:  primeraVar = 4    segundaVar = 96
Dentro de:  primeraVar = 3    segundaVar = 97
Dentro de:  primeraVar = 2    segundaVar = 98
Dentro de:  primeraVar = 1    segundaVar = 99
```

La Figura 3.31 muestra cómo se vería la salida de dicho programa.

Un uso más común del operador de coma pudiera ser inicializar algunas variables de bandera que deseemos que el ciclo modifique.

Ejemplo: ciclos `foreach`

Los arreglos son tan útiles que Perl proporciona una forma especial del enunciado `for` sólo para ellos. El enunciado `foreach` se usa solamente para iterar sobre los elementos de un arreglo. Resulta muy útil para encontrar el elemento más grande, imprimir los elementos, o simplemente ver si un determinado elemento es miembro de un arreglo.

Su sintaxis es:

```
foreach CICLO_VAR (ARREGLO) {
      DECLARACIONES
}
```

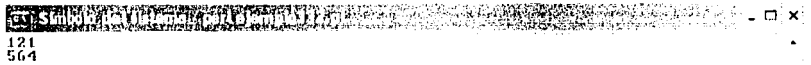
La variable `CICLO_VAR` se le asigna al valor de cada elemento del arreglo, hasta que éste termina. Veamos cómo utilizar `foreach` para localizar el elemento más grande del arreglo.

Llamar dos veces a la función max() con parámetros diferentes cada vez.
Define la función max().
Crear una variable local $max, y obtener luego el primer elemento del arreglo de parámetros.
Si el elemento actual es mayor que $max, realizar un ciclo a través del arreglo de parámetros comparando cada elemento con $max.
Retornar el valor de $max.

Listado 3.32 ciclos `foreach`

```
print max(45..121, 12..23) . "\n";
print max(23..34, 356..564) . "\n";

sub max {
      my ($max) = shift(@_);

      foreach $temp (@_) {
            $max = $temp if $temp > $max;
      }
      return ($max);
}
```

Este programa imprime:



La Figura 3.32 muestra cómo se vería la salida de dicho programa.

En este ejemplo podemos encontrar dos cosas importantes. Una es el uso de la función `shift()` para valuar una variable local y eliminar al mismo tiempo del arreglo, su primer elemento. Si usamos sólo `shift()`, se pierde el valor del primer elemento.

El otro punto importante es el uso de `$temp` dentro del ciclo `foreach`. Algunos programadores de Perl desaprueban el uso de variables temporales en esa forma. Perl tiene una variable interna `$_`, que puede utilizarse en su lugar. Si no se especifica la variable de signo, se asignará a `$_` el valor de cada elemento del arreglo al iterar al ciclo.

3.6.3 Palabras clave de salto *(04B) (08B)*

Perl tiene cuatro palabras clave para modificar el flujo de sus programas. La Tabla 3.15 lista las palabras clave junto con una breve descripción.

Tabla 3.15 Palabras clave de salto de Perl

Palabra clave	*Descripción*
`last`	Salta fuera del bloque actual de enunciados.
`next`	Omite el resto del bloque de enunciados y continúa con la siguiente iteración del ciclo.
`redo`	Reiniciar el bloque de enunciados.
`goto`	Salta a la etiqueta especificada

Cada una de estas palabras clave se describe con más detalle en su propia sección, las cuales se presentan a continuación.

Ejemplo: la palabra clave `last`

La palabra clave `last` se usa para salir de un bloque de enunciados. Esta capacidad es útil si estamos buscando un valor en un arreglo. Al localizar el valor, puede concluir antes el ciclo.

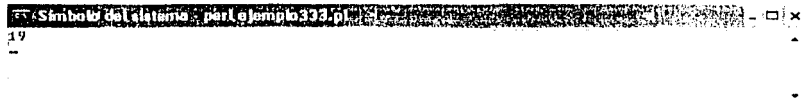
Crear un arreglo conteniendo las 26 letras del alfabeto.
Usa un ciclo for para iterar sobre el arreglo. La variable índice comenzará en cero y se incrementará en tanto sea menor que el número de elementos en el arreglo.

Prueba el elemento del arreglo para ver si es igual a "T". Nótese que se usa el operador de igualdad de cadena. Si el elemento del arreglo es "T", entonces sale del ciclo.

Listado 3.33 la palabra clave `last`

```
@arreglo = ("A".."Z");
for ($inicio = 0; $inicio < @arreglo; $inicio++){
      if ($arreglo[$inicio] eq "T") {
            last;
      }
}
print ("$inicio\n");
```

Este programa imprime:



La Figura 3.33 muestra cómo se vería la salida de dicho programa.

Este es un ciclo directo, excepto por la forma en que calcula el número de elementos en el arreglo. Dentro de la expresión condicional, la variable `@arreglo` se evalúa en un contexto escalar. El resultado es el número de elementos en el arreglo.

Ejemplo: la palabra clave `next`

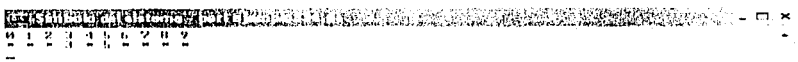
La palabra clave `next` nos permite saltar el resto del bloque de enunciados y comenzar en la siguiente iteración. Un uso de este comportamiento pudiera ser para seleccionar elementos específicos de los arreglos para procesarlos e ignorar el resto. Por ejemplo:

Crear un arreglo de 10 elementos.
Imprimir el arreglo.
Efectuar una interacción sobre el arreglo.
Ignorar los elementos tercero y quinto.
Cambiar el elemento actual por un asterisco.
Imprimir el arreglo para verificar que haya sido modificado.

Listado 3.34 la palabra clave `next`

```
@arreglo = (0..9);
print("@arreglo\n");
for ($inicio = 0; $inicio < @arreglo; $inicio++) {
      if ($inicio == 3 || $inicio ==5) {
            next;
      }
      $arreglo[$inicio] = "*";
}
print("@arreglo\n");
```

Este programa imprime:



La Figura 3.34 muestra cómo se vería la salida de dicho programa.

Este ejemplo cambia por asteriscos todos los elementos del arreglo, con excepción del tercero y el quinto, sin importar sus valores anteriores. La siguiente palabra clave fuerza a Perl a saltarse el enunciado de asignación y a ir directamente a la expresión de incremento/decremento.

Ejemplo. La palabra clave `redo`

La palabra clave `redo` hace que Perl reinicie el bloque actual de enunciados. Antes de reiniciar el bloque no se evalúan ni la expresión de *incremento/decremento* ni la expresión condicional. Por lo regular, esta palabra clave se usa al obtener entradas desde fuera del programa, ya sea del teclado o de un archivo. Resulta esencial que las condiciones que provocaron la ejecución del enunciado `redo` se modifiquen a fin de evitar un ciclo infinito.[▲] Este ejemplo demostrará la palabra clave `redo` con alguna entrada desde el teclado:

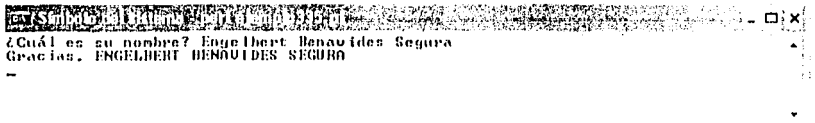
Iniciar un bloque de enunciados.
Imprimir un mensaje solicitando un nombre.
Leer una cadena del teclado. El control regresa al programa cuando el usuario oprime la tecla Enter.
Eliminar el caracter de línea nueva al final de la cadena.
Si la cadena es la longitud cero, significa que el usuario simplemente presionó la tecla Enter sin registrar un nombre, así que exhibe un mensaje de error y repite con redo el bloque de enunciados.
Imprimir un mensaje de agradecimiento con el nombre en mayúsculas.

Listado 3.35 la palabra clave `redo`

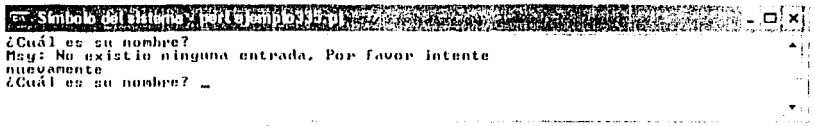
```
{
      print ("¿Cuál es su nombre? ");
      $nombre = <STDIN>;
      chop($nombre);

if (! length($nombre)) {
      print("Msg: No existio ninguna entrada, Por favor intente
            nuevamente\n");
      redo;
      }
      print("Gracias, " . uc($nombre) . "\n");
}
```

[▲] Vale la pena destacar que el bloque de enunciados en este ejemplo actúa como una construcción de ciclo de un sólo tiempo. Podemos usar dentro del bloque de enunciados cualquiera de las palabras clave de salto.

Este programa imprime:



La Figura 3.35 muestra cómo se vería la salida de dicho programa cuando pone su nombre.



La Figura 3.35 muestra cómo se vería la salida de dicho si no teclea ningún caracter.

El enunciado `redo` nos ayuda a tener un flujo de programa más directo. Sin él, necesitaríamos usar un ciclo `do...until`.

Ejemplo: la palabra clave `goto`

La palabra clave `goto` permite que su programa salte directamente a cualquier etiqueta. Sin embargo, debido a que Perl también proporciona los enunciados de ciclo y otras palabras clave de salto, su uso se ha visto menospreciado por la mayoría de los programadores. El uso de `goto` en los programas hace con frecuencia que la lógica del programa sea intrincada. Si escribimos un programa que cree que necesite de un `goto` a fin de ejecutase, entonces usémoslo, pero antes, intentemos reestructurar el programa para evitarlo.

3.7 Referencias *(04B) (08B)*

Una *referencia* es una variable escalar que apunta a una ubicación de memoria que contiene algún tipo de datos. Todo en el programa Perl se almacena dentro de la memoria de la computadora. Por lo tanto, todas las variables y funciones se encuentran en alguna ubicación de memoria. Las referencias se usan para contener las direcciones de memoria. Cuando una referencia es *desreferenciada*, recuperamos la información a que se refiere la referencia.

3.7.1 Tipos de referencias *(04B) (08B)*

Existen cinco tipos de referencias. Una referencia puede apuntar a un escalar, un arreglo, un hash, una función u otra referencia. La Tabla 3.16 muestra cómo se valúan con el operador de asignación los diferentes tipos y cómo desreferenciarlos utilizando llaves.

Tabla 3.16 Los cinco tipos de referencias

Asignación de la referencia	*Como desreferenciar*
`$refScalar =\$scalar;`	`${$refScalar}` es un valor escalar
`$refArreglo = \@arreglo;`	`@{$refArreglo}` es un valor de arreglo.
`$refHash = \%hash;`	`%{$refHash}` es un valor hash.
`$refFuncion = \&funcion`	`&{$refFuncion}` es la ubicación de una función.
`$refRef = \$refScalar;`	`${${$refScalar}}`es un valor escalar.

En esencia, todo lo que necesitamos para crear una referencia es agregar una diagonal invertida al frente de un valor o variable.

Ejemplo: transferencia de parámetros a funciones

En la sección de "Funciones", hablamos de la transferencia de parámetros a funciones. En ese momento, no podíamos pasar más de un arreglo a una función. Esto, debido a que al buscar parámetros, las funciones sólo ven un arreglo (el `arreglo@`). Para superar esta limitante, podemos emplear las referencias.

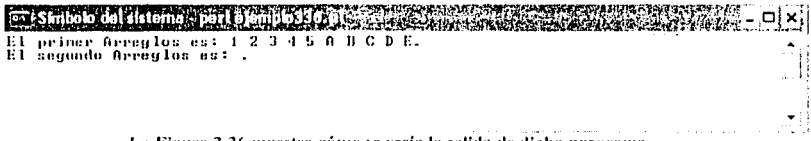
Comencemos por pasar dos arreglos a una función para mostrar que la función sólo ve un arreglo.

Llamar a primeraSub() con dos arreglos como parámetros.
Definir la función primeraSub().
Crear variables locales y asignar elementos del arreglo de parámetros.
Imprimir los arreglos locales.

Listado 3.36 transferencia de parámetros a funciones

```
primeraSub( (1..5 ), ("A"..."E"));

sub primeraSub {
my(@primerArreglo, @segundoArreglo) = @_;

print ("El primer Arreglos es: @primerArreglo.\n");
print ("El segundo Arreglos es: @segundoArreglo.\n");
}
```

Este programa imprime:



La Figura 3.36 muestra cómo se vería la salida de dicho programa.

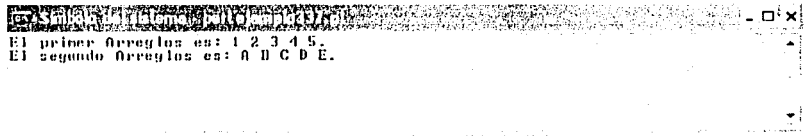
Dentro de la función `primeraSub()`, se asignó a la variable `@primerArreglo` todo el arreglo de parámetros, sin dejar nada a la variable `@segundoArreglo`. Por medio de la transferencia de referencias a `@aregloUno` y `@arregloDos`, podemos conservar los arreglos para su uso dentro de la función. Se requieren muy pocos cambios para permitir el uso de referencias en el ejemplo anterior. Observemos.

Llamar a primeraSub() utilizando el operador de diagonal invertida para pasar una referencia a cada arreglo.
Definir la función primeraSub().
Crear dos variables escalares locales y asignar elementos del arreglo de parámetros.
Imprimir las variables locales, desreferenciándolas para que luzcan como arreglos. Esto se hace con la notación @{}.

Listado 3.37 transferencia de parámetros a funciones

```
primeraSub( \(1..5 ), \("A"..."E"));        #Uno

sub primeraSub {
my($ref_primerArreglo, $ref_segunArreglo) = @_;        #Dos

print ("El primer Arreglos es: @{$ref_primerArreglo}.\n"); #Tres
print ("El segundo Arreglos es: @{$ref_segunArreglo}.\n"); #Tres
}
```

Este programa imprime:



La Figura 3.37 muestra cómo se vería la salida de dicho programa.

Para que este ejemplo usara referencias se hicieron tres cosas:

1. En la línea marcada como "Uno", se agregaron diagonales invertidas para indicar que debían transferirse una referencia al arreglo.
2. En la línea marcada como "Dos", se tomaron las referencias del arreglo de parámetros y se asignaron a variables escalares.
3. en las líneas marcadas como "Tres", se desreferenciaron los valore escalares. Desreferenciar significa que Perl usará la referencia como si fuera un tipo de datos normal, en este caso, una variable de arreglo.

Ejemplo: creación de un registro de datos

Los arreglos asociativos (*hashes*) de Perl son muy útiles cuando se trata de almacenar información de modo que se facilite su recuperación. Por ejemplo, podría almacenar información de clientes como la siguiente:

```
%registro =   ( "Nombre"        =>    "Jacobo Segura",
                "Dirección"     =>    "Crisantema M-17 L-6",
                "Ciudad"        =>    "México",
                "Estado"        =>    "D.F.",
                "C.P."          =>    "01840"
);
```

El arreglo asociativo `%registro` puede considerarse como un *registro de datos* con cinco *miembros*. Cada miembro es una partida individual de información. El *registro de datos* es un grupo de miembros relacionados con un mismo tema.

Por ejemplo, podemos accesar el miembro correspondiente al estado diciendo `$registro{"Estado"}`. Todos los miembros pueden accesar en forma similar.

Por supuesto, una base de datos con un sólo registro no es muy útil. Por medio de referencia, podemos construir un arreglo de registros múltiples. El Listado 3.85 muestra dos registros y cómo inicializar un arreglo de base de datos.

Declarar un registro de datos de nombre %registroUno como un arreglo asociativo.
Declarar un registro de datos de nombre %registroDos como un arreglo asociativo.

Declarar un arreglo denominado @basededatos con referencias a los arreglos asociativos como elementos.

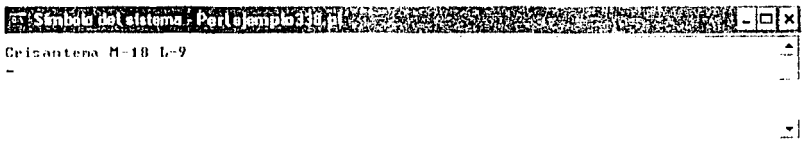
Listado 3.38 Una base de datos con dos registros

```
%registroUno =    ( "Nombre"          =>    "Jacobo Segura",
                    "Direccion"       =>    "Crisantema M-18 L-9",
                    "Ciudad"          =>    "México",
                    "Estado"          =>    "D.F.",
                    "C.P."                  =>    "01840"
);

%registroDos =    ( "Nombre"          =>    "Rubi Segura",
                    "Dirección"       =>    "Crisantema M-17 L-6",
                    "Ciudad"          =>    "México",
                    "Estado"          =>    "D.F.",
                    "C.P."                  =>    "01840"
);

@basededatos = ( \%registroUno, \%registroDos );
```

Podemos imprimir el miembro de la dirección (`Direccion`) del primer registro como sigue:

```
print (%{$basededatos[0]}-> {"Direccion"} . "\n");
```

Este programa imprime:



La Figura 3.38 muestra cómo se vería la salida de dicho programa.

Analicemos la expresión de desreferenciación en este enunciado `print`. Recordemos que debemos trabajar de izquierda a derecha y evaluar siempre primero los corchetes y los paréntesis. Ignorando la función `print()` y el caracter de línea nueva, puede evaluar esta línea de código de la siguiente manera:

- El corchete más interno es `[0]`, el cual significa que estaremos viendo el primer elemento de un arreglo.
- Los operadores de corchetes cuadrados tienen una asociatividad de izquierda a derecha, así que buscaremos el nombre del arreglo asociativo a la izquierda. El nombre del arreglo es basededatos.
- En seguida vienen las llaves, que indican a Perl la desreferenciación. Las llaves tienen también una asociatividad de izquierda a derecha, de modo que buscaremos a la izquierda para ver el tipo de referencia. En este caso vemos un `%`, que significa un arreglo asociativo.

- El -> es el operador infijo de desreferenciación. Indica a Perl que lo que se desferencia a la izquierda (en este caso la referencia a basededatos) se conecta con algo a la derecha.
- Ese "algo" a la derecha es el valor llave o "`Direccion`". Hay que notar que se encuentra entre llaves, tal como si estuviéramos usando una llave de arreglo asociativo normal.

La declaración de variables en el ejemplo anterior emplea tres variables para definir la estructura de datos. Podemos condensar la declaración en una variable como observamos en el Listado 3.39.

Declarar un arreglo de nombre @basededatos con dos arreglos asociativos como elementos. Debido a que los arreglos asociativos no se están asignando directamente a una variable se les considera anónimos.
Imprimir el valor asociado con la llave "Nombre" del primer elemento del arreglo @basededatos.
Imprimir el valor asociado con la llave "Nombre" del segundo elemento del arreglo @basededatos.

Listado 3.39 Declaración de la estructura de la base de datos en una sola instrucción

```
@basededatos = (
```

```
@basededatos  = (
      { "Nombre"          =>    "Jacobo Segura",
      "Direccion"         =>    "Crisantema M-18 L-9",
      "Ciudad"            =>    "México",
      "Estado"            =>    "D.F.",
      "C.P."                    =>    "01840"
},
      { "Nombre"          =>    "Rubi Segura",
      "Dirección"         =>    "Crisantema M-17 L-6",
      "Ciudad"            =>    "México",
      "Estado"            =>    "D.F.",
      "C.P."                    =>    "01840"
      }
);

print (%{$basededatos [0]} -> {"Nombre"} . "\n");
print (%{$basededatos [1]} -> {"Nombre"} . "\n");
```

Este programa imprime:

```
Símbolo del sistema - Perl ejemplo339.pl
Jacobo Segura
Rubi Segura
_
```

La Figura 3.39 muestra cómo se vería la salida de dicho programa.

Analicemos el código de desreferenciación de la primera línea.

- El corchete más interno es `[0]`, el cual significa que estaremos viendo el primer elemento de un arreglo.
- Los operadores de paréntesis cuadrados tienen una asociatividad de izquierda a derecha, así que buscaremos el nombre del arreglo asociativo a la izquierda. El nombre del arreglo es basededatos.
- En seguida vienen las llaves, que indican a Perl la desreferenciación. Las llaves tienen también una asociatividad de izquierda a derecha, de modo que buscaremos a la izquierda para ver el tipo de referencia. En este caso vemos un `%`, que significa un arreglo asociativo.
- El `->` es el operador infijo de desreferenciación. Indica a Perl que lo que se desferencia a la izquierda (en este caso la referencia a basededatos) se conecta con algo a la derecha.
- Ese "algo" a la derecha es el valor llave o "`Nombre`". Hay que notar que se encuentra entre llaves, tal como si estuviéramos usando una llave de arreglo asociativo normal.

Aunque las declaraciones de estructura en los dos últimos ejemplos se ven diferentes, son equivalentes. Podamos confirmarlo ya que las estructuras se desreferencian del mismo modo. ¿Qué está pasando aquí? Perl está creando referencias a arreglos asociativos *anónimos* que se convierten en elementos del arreglo `@basededatos`.

En el ejemplo anterior, cada arreglo asociativo tiene un nombre: `%registroUno` y `%registroDos`. En el ejemplo actual, no hay un nombre de variable asociado en forma directa a los arreglos asociativos. Si usamos una variable anónima en nuestros programas, Perl proporcionará una referencia a ella de manera automática.

Podemos explorar un poco más el concepto de los registros de datos utilizando este ejemplo básico. Hasta ahora, hemos usado referencias de arreglos asociativos como elementos de un arreglo. Cuando se almacena un tipo de dato dentro de otro tipo de dato, esta acción se denomina *anidar* tipos de datos. Podemos anidar tipos de datos con la frecuencia y profundidad que deseemos.

En este punto del ejemplo, se usó `(%{$basededatos [0]} -> {"Nombre"}` para desreferenciar el miembro "Nombre" del primer registro. Este tipo de desreferenciación usa un suscrito de arreglo para indicar a Perl cuál es el registro que buscamos.

CAPÍTULO IV

PERL E INTERNET

OBJETIVO ESPECIFICO:

GENERAR UNA ADECUADA DOCUMENTACIÓN PARA PODER ESTABLECER LAS BASES NECESARIAS Y ENTENDER QUE ES UN CGI ADEMÁS DE LOS REQUERIMIENTOS NECESARIOS PARA CREARLOS Y PONERLOS EN MARCHA.

4.1 Uso de los protocolos de Internet *(04H) (08H)*

Una de las razones por las que Internet ha florecido tan rápido es porque todos pueden entender los *protocolos* que se hablan en la red. Un protocolo es un conjunto de comandos y respuestas. Existen dos estratos de protocolos. El estrato de bajo nivel se denomina TCP/IP y aunque es crucial para Internet, podemos efectivamente ignorarlo por ahora. Los protocolos de alto nivel como ***ftp, smtp, pop, http*** y ***telnet.*** Ellos utilizan TCP/IP como un medio para comunicarse entre computadoras. Todos los protocolos tienen el mismo patrón básico:

- **Inicio de una conversación-** Su computadora (el cliente) inicia una conversación con otra computadora (el servidor).
- **Sostenimiento de una conversación-** Durante la conversación, se envían comandos y se acusan de recibidos.
- **Fin de una conversación-** La conversación termina.

La Figura 4.01 muestra el protocolo para el envió de correspondencia. El usuario final crea un mensaje de correo y luego el sistema emisor utiliza el protocolo de correspondencia para sostener una conversación.

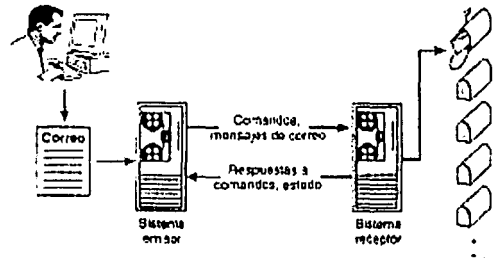


Figura 4.01 Todos los protocolos siguen este modelo de comunicación.

La conversación en Internet se hace con sockets, de manera parecida a utilizar un teléfono o gritar por una ventana.

4.1.1 Sockets *(04B) (08B)*

Los *sockets* son los vínculos de bajo nivel que permiten las conversaciones en Internet.

La Tabla 4.01 lista todas las funciones de Perl que se relacionan con los sockets, de modo que tengamos una útil referencia.

Tabla 4.01 Funciones de sockets de Perl

Función	*Descripción*
`accept (NEWSOCKETS, SOCKET)`	Acepta una conexión de socket de los clientes en espera de una conexión. El SOCKET original se deja de lado y se crea un nuevo socket para que el proceso remoto converse con él. SOCKET debe haber sido abierto antes con la función `socket()`. Retorna verdadero si tiene éxito y falso en el caso contrario.
`bind( SOCKET, DIRECCION_EMPACADA)`	Vincula una dirección de red al identificador de socket. Retorna verdadero si tiene éxito y falso en caso contrario.
`conecct(SOCKET, DIRECCION_EMPACADA)`	Intenta una conexión a un socket. Retorna verdadero si tiene éxito y falso en caso contrario.
`getpeername(SOCKET)`	Retorna la dirección empacada del lado remoto de la conexión. En caso necesario se puede utilizar esta función para rechazar conexiones por cuestiones de seguridad.
`getsockname (SOCKET)`	Retorna la dirección empacada del lado local de la conexión.
`getsockopt(SOCKET, NIVEL, NOMOPCIÓN)`	Retorna la opción de socket solicitada, o indefinido en caso de error.
`listen(SOCKET, TAMAÑOCOLA)`	Crea una cola para SOCKET con tantas ranuras como indica TAMAÑOCOLA. Retorna verdadero si tiene éxito y falso en el caso contrario.
`recv(SOCKET, BUFER, LONG, BANDERAS)`	Intenta recibir del SOCKET, en un BÚFER, el número de bytes especificado por LONG. Retorna la dirección del emisor, o el valor indefinido si hay un error. El BÚFER crecerá o se reducirá a la longitud realmente leída. Sin embargo, debe usted inicializar el BÚFER antes de usarlo. Por ejemplo `my ($buffer) = '';`.
`select(RBITS, WBITS, EBITS, TIEMPOLÍMITE)`	Examina los descriptores de archivo para ver si están listos o tienen pendientes condiciones de excepción.
`send(SOCKET, BUFER, BANDERAS [DESTINO])`	Envía un mensaje a un socket. En sockets no conectados, debemos especificar un DESTINO. Retorna el número de caracteres enviados, o el valor indefinido si hay un error.
`setsockpot(SOCKET, NIVEL, NOMOPCIÒN, VALOPCIÒN)`	Asigna la opción de socket solicitada. Retorna indefinido si hay un error. Si no deseamos pasar un argumento, VALOPCION pudiera especificarse como indefinida.
`shutdown(SOCKET, METODO)`	Apaga una conexión de socket en la manera indicada por METODO. Si `METODO = 0`, se ignorará toda la información entrante. Si `METODO = 1`, se detendrá toda la información saliente. Si `METODO = 2`, entonces se desactivará tanto el envío como la recepción.
`socket(SOCKET, DOMINO, TIPO, PROTOCOLO)`	Abre un TIPO específico de socket y lo adjunta al nombre SOCKET. Para más detalles, consulte "El lado del servidor de una conversación". Si tiene éxito retorna verdadero, y falso si no lo tiene.
`socketpair(SOCK1, SOCK2, DOMAIN, TIPO, PROTOCOLO`	Crea un par de sockets sin nombre del tipo especificado, en el dominio especificado. Retorna verdadero si tiene éxito y falso en caso contrario.

La siguiente sección estará dedicada a ver ejemplos de protocolos específicos.

La Tabla. 4.01 proporciona una lista de los protocolos de alto nivel que podemos utilizar. Para investigar más acerca de los protocolos, lo podemos encontrar en detalle en los documentos que están en el sitio Web *http://ds.internic.net/ds/dspg0intdoc.html*.

Tabla 4.01 Una pequeña muestra de protocolos

Protocolos	*Números*	*Descripción*
auth	113	Autentificación
echo	7	Revisa al servidor para ver si está ejecutando
finger	79	Le permite recuperar información sobre un usuario
ftp	21	File Transfer Protocol (Protocolo para transferencia de archivos)
nntp	119	Network News Transfer Protocol – Usenet News Groups (Protocolo para transferencia de noticias en red – Grupo de noticias Usenet)
pop	109	Post Office Protocol (Protocolo de oficina postal) correo entrante
smtp	25	Simple Mail Transfer Protocol (Protocolo simple para transferencia de correo) correo entrante
time	37	Time Server (Servidor de hora)
telnet	23	Nos permite conectarnos a una computadora anfitriona (host) y usarla como si fuera una terminal conectada directamente

También a los protocolos se les llama servicios. De ahí el término, servidor de correo o servidor *ftp*. En forma subyacente a todos los protocolos de alto nivel se encuentran el muy popular protocolo de control de transferencia / Protocolo Internet o TCP/IP. Para utilizar los protocolos de alto nivel, no es necesario que sepamos sobre TCP/IP. Todo lo que necesitamos saber es que TCP/IP habilita a un servidor para que *escuche* y responda a conversaciones entrantes. Dichas conversaciones llegan a algo denominado puerto. Un *puerto* es un lugar imaginario en dónde los paquetes entrantes de información pueden arribar (tal como un barco arriba a un puerto marítimo). Cada tipo de servicio (por ejemplo, correo o transferencia de archivos) tiene su propio número de puerto.

4.1.2 Tipo de servicios *(04H) (08H)*

e-mail.

El primero y más universalizado es el correo electrónico que permite intercambiar mensajes entre usuarios. El primitivo sistema se ha ido complicando hasta contar con interfases gráficas de usuario permitiendo el intercambio de información en distintos formatos como imágenes, programas o archivos de datos propios de determinadas aplicaciones. Netscape ha sido pionero en la integración del correo en el Web. Desde la versión 2.0 de su Navigator, dispone de un sistema de gestión de correo electrónico bastante digno que incluye un libro de direcciones con alias y la organización de los mensajes en carpetas.

FTP

La transferencia de archivos basados en el file transfer protocol, es una de las aplicaciones más populares de la red. Tradicionalmente era el servicio que ocupaba el grueso del tráfico en la red, aunque recientemente ha sido superada por WWW.

Su principal cometido es el intercambio de archivos en cualquier formato y se organizan en directorios que pueden estar acompañados de información suplementaria.

Telnet

En acceso a login remoto es la tercera de las funcionalidades básicas (y clásicas) de Internet y uno de sus primeros atractivos. Permite que un usuario entre a cuentas abiertas en máquinas remotas. Normalmente se complementa con la transferencia de archivos.

Existen varios protocolos para realizar esta operación pero el usado universalmente es *telnet*, que suele ser el mecanismo elegido en WWW para abrir sesiones en otras máquinas.

News

No se trata estrictamente de un servicio de Internet. En realidad residen exclusivamente en la red USENET y hay usuarios que sólo pueden acceder a este servicio y no al resto de Internet, pero parte de su tráfico circula por Internet y se ha convertido en uno de sus servicios más populares. Se trata de un sistema a medio camino entre el correo y la transferencia de archivos: la información esta organizada en grupos temáticos, originalmente técnicos pero que en la actualidad cubren cualquier área posible de interés. Los usuarios envían mensajes que permanecen a la vista de todos los que se conecten al mismo grupo y que pueden responder directamente a las cuestiones o comentarios planteados a través del grupo.

Ha sido usado profusamente como un servicio técnico paralelo en el que una duda de un usuario podría ser leída y resuelta por conocedores del tema procedentes de todo el mundo.

Gopher

Se trata de un sistema de menús jerárquico a medio camino entre la rusticidad de *ftp* y el multimedia de WWW. Cada entrada puede apuntar por igual a un archivo, a un nuevo menú (directorio) en la misma máquina o a otro menú en otra máquina. Las entradas se pueden seleccionar a través de un número, con cursores o por medio de iconos, dependiendo del software cliente (visualizador). Junto a las entradas que apuntan a otros elementos puede haber texto que informe o guíe en la navegación. A su vez, seleccionar un archivo texto hace que éste se visualice en la propia aplicación.

Archie

Es una utilidad que facilita la localización de información a través de la red. A medida que el número de servidores ha ido creciendo se ha convertido en un verdadero problema localiza una referencia de información determinada. Los servidores Archie permiten realizar búsquedas de información a partir de direcciones o textos, devolviendo una lista con las referencias y sus localizaciones.

WWW

Desarrollado por el CERN, es un sistema multimedia apoyado en Internet y el responsable en gran medida de la reciente e inusitada popularidad de la red. WWW exporta documentos basados en texto pero que incorporan imágenes y enlaces. Estos enlaces pueden llevar a nuevos documentos, como referencias en algún otro sistema (*NEWS, telnet, ftp, gopher*), o archivos en cualquier formato soportado por el cliente: imágenes, filmaciones digitalizadas, sonido o software.

4.2 El concepto de CGI *(27R)*

4.2.1 ¿Qué es el CGI? *(27R)*

Cuando el World Wide Web inicio su funcionamiento como lo conocemos, empezando a tomar popularidad aproximadamente en 1993, sólo se podía apreciar texto, imágenes y enlaces. La introducción de Plugins en los navegadores permitió mayor interactividad entre el usuario y el cliente, aunque estaba limitado por la velocidad y la necesidad de tener que bajar e instalar cada plugin que se necesitará, por lo que estos se desarrollaron mayormente en áreas de video, audio y realidad virtual.

El **CGI** (Por sus siglas en inglés **"Common Gateway Interface"**) es la interfaz estándar de programación entre servidores Web y programas externos. Ésta es una de las áreas más emocionantes y divertidas de la programación actual.

En sí, es un método para la transmisión de información hacia un compilador instalado en el servidor. Su función principal es la de añadir una mayor interacción a los documentos Web que por medio del HTML se presentan de forma estática.

El CGI es utilizado comúnmente para contadores, bases de datos, motores de búsqueda, formularios, generadores de email automático, foros de discusión, chats, comercio electrónico, juegos en línea y otros.

Esta tecnología tiene la ventaja de correr en el servidor cuando el usuario lo solicita por lo que es dependiente del servidor y no de la computadora del usuario.

De acuerdo a la traducción de la NCSA: "Un documento HTML es estático", lo que significa que existe en un estado constante; es un archivo de texto que no cambia. Un script CGI por otro lado, es ejecutado en tiempo real, lo que permite que regrese información dinámica. Por ejemplo, digamos que queremos conectar nuestras bases de datos de Unix al World Wide Web para permitir que las personas de todo el mundo la manipulen. Básicamente, lo que debemos hacer es crear un script CGI que será ejecutado por el servidor para transmitir información al motor de la base de datos, recibir los resultados y mostrárselos al cliente.

Los programas que maneja el CGI pueden estar compilados en diferentes lenguajes de programación. El más popular para el desarrollo de contenidos Web es el lenguaje Perl de distribución gratuita, aunque también podemos mencionar: C, C++ y Java.

El funcionamiento de esta tecnología es muy sencillo. Los scripts residen en el servidor, dónde son llamados, ejecutados y regresa información dé vuelta al usuario.

La Figura 4.02 Nos muestra una mejor aclaración de la forma cómo funciona este proceso

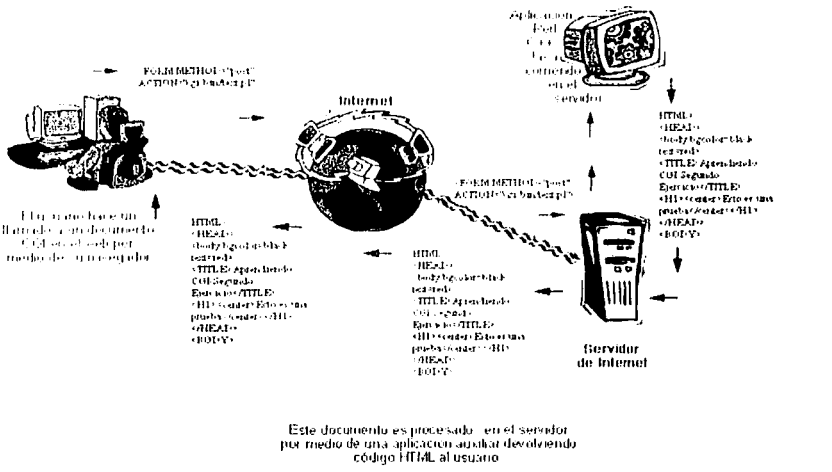


Figura 4.02 Función del Proceso al llamar un CGI

La CGI estándar no existe en forma aislada; depende de los estándares de HTML y HTTP.

HTML es el estándar que permite a los navegadores Web comprender el contenido de documentos. HTTP (Protocolo de transferencia de hipertexto) es el método utilizado para transmitir las páginas World Wide Web a través de la red.

Casi cualquiera puede armar algo de HTML y colgar una "página base" en la Web. Aunque la mayoría de los sitios ahí son, aburridos. ¿Por qué? El hecho es que la mayoría de los sitios se construyen como una simple serie de documentos HTML que nunca cambian. El sitio es completamente estático. No es probable que alguien visite una página estática más de una o dos veces. Pensemos en los sitios que visitamos con más frecuencia. Es probable que tengan algún contenido interesante, es cierto, pero más importante es que tengan un contenido dinámico.

Así que, ¿qué debe hacer un Webmaster (amo del Web)? Ningún Webmaster dispone del tiempo para actualizar sus sitios Web diariamente en forma manual. Por fortuna, la gente que desarrolló la Web pensó en este problema y nos dio la CGI. Esta interfaz le ofrece una forma de hacer sitios Web dinámicos e interactivos.

Cada una de las palabras del acrónimo de **Common Gateway Interface** nos ayudara a comprender la interfaz.

- **Interface** –utiliza un método bien definido para interactuar con un servidor. Web.
- **Gateway** –proporciona a los usuarios una forma de tener acceso a diferentes programas como generadores de base de datos o de imágenes.
- **Common** – interactúa con diferentes sistemas operativos.

Las aplicaciones CGI pueden realizar casi cualquier tarea que podamos imaginar. Por ejemplo, podemos crear páginas Web sobre la marcha, accesar bases de datos, mantener sesiones de *Telnet*, generar gráficas y reunir estadísticas.
El concepto básico tras la CGI es muy sencillo; sin embargo, la creación real de aplicaciones CGI no lo es. Esto requiere de verdaderas habilidades de programación. Necesitamos ser capaces de depurar programas y efectuar conexiones lógicas entre una idea y otra. También requiere de la capacidad de visualizar la aplicación que desearíamos crear.

4.2.2 ¿Por qué utilizar Perl para CGI? *(04B) (05R) (12B)*

Por diversas razones, Perl es el estándar *de facto* para la programación CGI, aunque quizás las más importantes sean:

- **Manejo de sockets** –crea programas que establecen una interfaz fluida con los protocolos de Internet. Nuestro programa CGI puede enviar una página Web en respuesta a una transacción y enviar una serie de mensajes de e-mail para informar a los interesados que la transacción se llevó a cabo.
- **Cotejo de patrones** –ideal para el manejo de datos y textos de búsqueda.
- **Manejo flexible de textos** –no hay que preocuparse por los detalles. La forma en que Perl maneja las cadenas, en términos de asignación y liberación de memoria, se desvanece en el fondo mientras nosotros programamos. Podemos ignorar los detalles de cómo concatenar, copiar y crear nuevas cadenas.

La ventaja de un lenguaje interpretado en aplicaciones CGI es la simplicidad en su desarrollo, depuración y revisión. Al eliminar el paso de compilación, podemos movernos con mayor rapidez de una tarea a otra, sin la frustración que puede a veces surgir de depurar programas compilados. Por supuesto, ningún lenguaje interpretado lo hará. Perl tiene la ventaja distintiva de tener una funcionalidad rica y capaz.

Hay ocasiones en las que una aplicación CGI madura, debe transportarse al lenguaje C u otro lenguaje compilado. Éstas son aplicaciones Web en las que la velocidad es importante. Si esperamos tener un sitio muy activo, probablemente queramos cambiar a un lenguaje compilado debido a que se ejecutan más rápido.

4.2.3 Las aplicaciones CGI *versus* los applets de Java *(04B) (05R) (12B)*

CGI y Java son dos cosas totalmente diferentes. CGI es una especificación que puede ser utilizada por cualquier lenguaje de programación. Las aplicaciones CGI se ejecutan en un servidor Web. Java es un lenguaje de programación que se ejecuta del lado del cliente.
Las aplicaciones CGI deben ser diseñadas para aprovechar la naturaleza centralizada de un servidor Web. Son excelentes para examinar bases de datos, procesar HTML a partir de datos y otras aplicaciones que requieren una limitada interacción con el usuario.
Las aplicaciones de Java son buenas cuando requerimos de un alto grado de interacción con el usuario: por ejemplo, en juegos o animación.

Los programas en Java deben mantenerse relativamente chicos en cuestion de líneas de programa debido a que se transmiten al cliente a través de Internet. Por otra parte, las aplicaciones CGI pueden ser tan grandes como necesitemos, ya que residen y se ejecutan en el servidor Web.

Podemos diseñar un sitio Web para usar aplicaciones tanto CGI como Java. Por ejemplo, es probable que quisieramos usar Java del lado del cliente para realizar la validación de campos al reunir información sobre un formulario. Luego, una vez validada la entrada, la aplicación Java puede enviar información a una aplicación CGI en el servidor Web, en dónde reside la base de datos.

4.2.4 ¿Deberíamos usar módulos CGI? *(04B) (05R) (12B)*

Podemos usar los módulos CGI que están disponibles en Internet. El módulo más actualizado se llama cgi.pm, aunque para poder utilizarlo debemos emplear Perl v5.002 o incluso una versión más reciente. El cgi.pm es muy comprensible y cubre muchos protocolos distintos además del CGI básico estándar.

Podemos encontrarnos con que cgi.pm es excesivo para aplicaciones CGI simples. De ser así podemos consultar el cgi.lite.pl. Esta biblioteca no hace tanto como el cgi.pm pero probablemente nos encontremos con que es más fácil de usar.

Podemos encontrar ambos scrpts en uno de los sitios WEB de CPAN.

4.3 Creación de CGI'S *(04B) (05R) (12B)*

Introducción

El Web no es sólo un medio para colocar información de variados tipos y formatos a disposición de los usuarios de Internet. Utilizando el Web es posible también interactuar con el usuario de forma que el pueda, por ejemplo, encargar un producto, suscribirse a algún servicio, reservar un pasaje o hacer una consulta a una base de datos. La intención de este texto es acercar a todos los que se han iniciado en HTML

4.3.1 ¿Cómo funciona esto? *(04B) (05R) (12B)*

En el momento en que accedemos a una página del tipo que mencionamos, podemos apreciar distintos elementos que permitirán que nuestros deseos/necesidades puedan ser transmitidos a las personas que manejan esa página. La mecánica es simple, el explorador de un cliente Web puede iniciar una aplicación CGI completando un formulario HTML o haciendo clic en un vínculo de una página HTML del servidor Web.

La aplicación CGI puede aceptar información escrita por el usuario y tratarla de cualquier modo que se pueda programar, y después devolver los resultados en una página HTML o enviar la información a una base de datos. Lo que se resume es, que la información es enviada al programa, el script CGI es procesado y en consecuencia genera una salida. En la Figura 4.03 se muestra el esquema de esta explicación

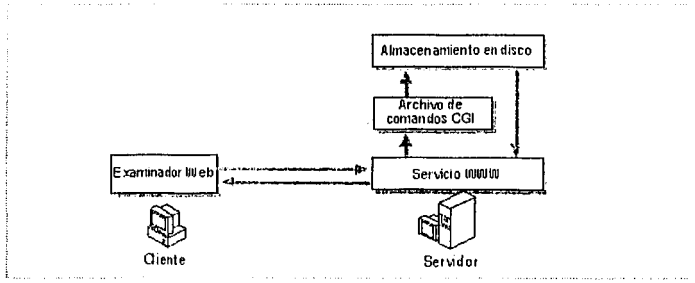


Figura 4.03 Esquema del Proceso de la llamada a un CGI

Los datos que ingresemos serán encapsulados en una variable de entorno denominada `QUERY_STRING`, y en el momento de apretar el botón de "submit o Firmar" será leída e interpretada por nuestro script. Dentro de ella irán todos los valores correspondientes a los datos que ingresemos, de manera que el script tendrá que separarlos uno por uno para su proceso. Esto se logra fácilmente mediante el uso de funciones que vienen incluidas en librerías especializadas para cada lenguaje en particular.

Existen otras variables de entorno que pueden ser accedidas por un script CGI cuando éste se ejecuta tales como:

- `QUERY_STRING` Data enviada al script CGI por el usuario. Esta puede ser la salida de un FORM, u otra información generada dinámicamente o estáticamente.
- `REMOTE_ADDR` La dirección de Internet de la máquina que hizo el request.
- `CGI_VERSION` Número de revisión de la especificación CGI.
- `CGI_REFERER` El URL del documento que contiene la llamada al script

El script eventualmente retorna una página HTML o una imagen que es mostrada como resultado de la ejecución

Para diferenciar estos dos tipos de salida se envía un "header" al comienzo de la transmisión de la respuesta que las identifica según la siguiente especificación (extensiones MIME).
Tipo de Información retornada Texto:

- Una página HTML / `Content-type: text/html`
- Una imagen .GIF / `Content-type: image/gif`

Los scripts normalmente están ubicados en el directorio **cgi-bin** del Web-Server o bien en alguno similar (CGI-WIN, CGI-DOS). La ubicación del directorio cgi-bin es determinada por el administrador del Web, o "webmaster" como se le llama comúnmente, y esta puede ser una ubicación física real o bien una lógica.

Si estuviéramos ejecutando el servidor Website bajo Windows XP o Win9x, la trayectoria traducida pudiera ser `c:/Apache/cgi-bin/test.pl`.

Es importante mencionarlo ya que la mayoría de los Web Servers sólo permiten que se ejecuten programas desde estos directorios en particular

¿Por dónde empezar?

Para comenzar analizaremos la creación de "forms o formularios", nuestro "front-end o interfaz gráfica" que permitirá a los usuarios el ingreso de la información deseada. Estos "forms" hacen uso de elementos comunes para aquellos familiarizados con las GUI modernas, Windows o X-Windows, como son los buttons, check boxes, radio buttons, pulldown menus, text boxes, etc...

Una vez que hayamos dominado bien este tema podremos pasar a la diagramación o programación de los scripts CGI, es decir los programas que procesarán los datos ingresados por los usuarios. La mejor manera de aprender este tema, es ir leyendo el documento y practicando en el browser (bastará con usar Netscape y el Edit de DOS) de todos modos se puede elegir el sistema que resulte más cómodo.

Esta sección ofrece un repaso en extremo breve de HTML y los formularios. A continuación veremos cómo se envía la información de formularios a los programas CGI, y con ello se explica la creación de ellos.

Una vez presentada la introducción al procesamiento de formularios, se desarrollara una aplicación de un Libro de Visitas.

4.3.2 Un breve repaso de HTML *(04B) (05R) (12B)*

El *Lenguaje de Marcación de Hipertexto* o HTML (por sus siglas de *Hypertext Markup Language)*, lo utilizan los programadores Web para describir el contenido de una página Web. No es un lenguaje de programación. Nosotros simplemente utilizamos HTML para indicar qué es una determinada porción de texto, como un párrafo, un encabezado o texto con formato especial. Todas las directivas de HTML se especifican mediante conjuntos correspondientes de paréntesis angulares y se denominan por lo regular *etiquetas*. Por ejemplo `<B>` significa que el texto que sigue deberá exhibirse en **negritas**. Para detener el texto en negritas, se usa la directiva `</B>`. La mayoría de las eriquetas de HTML se presenta en pares y rodean al texto afectado.

Los documentos HTML necesitan tener ciertas etiquetas a fin de que se les considere "correctos". El conjunto de etiqueta `<HEAD>..</HEAD>` rodean la información de encabezado de cada documento. Dentro del encabezado, nosotros podemos especificar un titulo de documentos con las etiquetas `<TITLE>..</TITLE>`[▲]

Después del encabezado del documento, necesitaremos tener un conjunto de etiquetas `<BODY>..</BODY>`. Dentro del cuerpo del documento, nosotros debemos especificar los encabezados de texto mediante un conjunto de etiquetas `<H1>..</H1>`. Cambiar el número después de la `H` modifica el nivel de encabezado. Por ejemplo, `<H1>` es el primer nivel, `<H2>` es el segundo nivel y así consecutivamente.
Podemos emplear la etiqueta `<P>` para indicar un salto de línea. Las etiquetas `<B>..</B>` e `<I>..</I>` se usan para indicar texto en negritas y en cursivas respectivamente.

El texto y las etiquetas de todo el documento HTML beben estar rodeados por un conjunto de etiquetas `<HTML>..</HTML>`. Por ejemplo:

[▲] Las etiquetas de HTML no son sensibles al uso de mayúsculas o minúsculas. Por ejemplo, `<TITLE>` es lo mismo que `<title>`. Sin embargo, el usar sólo mayúsculas en las etiquetas HTML hace que los documentos HTML sean más fáciles de comprender ya que se pueden identificar más fácilmente las etiquetas.

Listado 4.01 El formulario HTML utilizando etiquetas

```
<HTML>
<HEAD><TITLE>Este es el título</TITLE></HEAD>
<BODY>
<H1>Este es el encabezado de nivel uno </H1>
Este es el primer párrafo.
<P> Este es el segundo párrafo y tiene <I> cursiva</I>.
<H2>Este es el encabezado de nivel dos </H2>
Este es el tercer párrafo y tiene <B>negritas</B>.
</BODY>
</HTML>
```

La mayoría de las veces, debemos insertar o modificar texto dentro de las etiquetas del cuerpo del documento `<BODY>. .</BODY>`.

Esto es suficiente sobre las generalidades de HTML. La siguiente sección expone los Server Side Includes. Hoy en día, los Server Side Includes están reemplazando algunos programas CGI básicos, así que es importante conocerlos.

4.3.3 Server Side Incluyes *(04B) (05R) (12B)*

Una de las características más recientes incorporadas a los servidores Web es la de los Server Side Includes o SSI. Un SSI es un conjunto de funciones integradas dentro de los servidores Web que dan a los desarrolladores de HTML la capacidad de insertar datos en documentos HTML por medio de directivas especiales. Esto significa que nosotros podemos tener documentos dinámicos sin necesidad de crear programas CGI completos.

La información insertada puede tomar la forma de un archivo local o un archivo referenciado por un URL. También se puede incluir información de un conjunto limitado de variables, similar a las variables de ambiente. Por último, podemos ejecutar programas que puedan insertar texto dentro del documento.*

La mayor parte de los servidores Web requieren que se cambie la extensión del archivo de html a shtml a fin de que el servidor sepa que debe buscar directivas SSI. La extensión del archivo depende de la configuración del servidor, aunque shtml es una opción común.

Todas las directivas SSI se ven como comentarios HTML dentro de un documento.
En esta forma, las directivas SSI simplemente serán ignoradas en los servidores Web que no las manejen.

La Tabla 4.07 muestra una lista parcial de directivas SSI que maneja el servidor WebSite de O'Reilly. No todos los servidores Web manejarán todas las directivas de la tabla. Debemos revisar la documentación del servidor Web para determinar qué directivas maneja.

* Nota: La única diferencia real entre los programas CGI y los programas SSI es que los programas CGI deben enviar a la salida un encabezado HTTP como su primera línea de salida.

Tabla 4.07 Una lista parcial de directivas SSI

Directiva	*Descripción*
`<!--#config timefmt="'%c"—>`	Cambia el formato empleado para exhibir fechas.
`<!—#config sizefmt='%d bytes"—>`	Cambia el formato empleado para exhibir tamaños de archivo. También podría usted especificar bytes (para exhibir tamaños de archivos con comas o abbrev (para exhibir los tamaños de archivo en kilobytes o megabytes).
`<!—#config errmsg="##ERROR!##"—>`	Cambia el formato utilizado para exhibir mensajes de error provocados por directivas SSI impredecibles. Los mensajes de error se envían también a la bitácora de errores del servidor.
`<!—#echo var=?—>`	Exhibe el valor de la variable especificada por ?. Muchas de las variables posibles se muestran en esta tabla.
`<!—#echo var ="DOCUMENT_NAME"—>`	Exhibe la trayectoria completa y el nombre de archivo del documento actual.
`<!—#echo var='DOCUMENT_URI"—>`	Exhibe la trayectoria virtual y el nombre de archivo del documento actual
`<!—#echo var="LAST_MODIFIED"—>`	Exhibe la última vez que fue modificado el archivo, en el formato dia/mes/año y 16:45:40.
`<!—#echo var="DATE_LOCAL" —>`	Exhibe la fecha y hora utilizando la zona de horario local.
`<!—#echo var="DATE_GMT"—>`	Exhibe la fecha y hora utilizando el horario GMT.
`<!—#exec cgi=" /cgi-bin/ssi.exe" —>`	Ejecuta un programa CGI especificado. Debe estar activado para utilizarse. También puede usar una opción `cmd=` para ejecutar comandos de shell.
`<!—#flastmod virtual= "/docs/demo/ssi.txt—>`	Exhibe la fecha de última modificación del archivo especificado dada una trayectoria virtual.
`<!—#flstmod file="ssi.txt"—>`	Exhibe la fecha de última modificación del archivo especificado dada una trayectoria relativa.
`<!—#fsize virtual="/docs/demo/ssi.txt"—>`	Exhibe el tamaño del archivo especificado dada una trayectoria virtual.
`<!—#fsize file"ssi.txt"—>`	Exhibe el tamaño del archivo especificado dada una trayectoria relativa
`<!—#include virtual="/docs/demo/ssi.txt"—>`	Exhibe un archivo dada una trayectoria virtual.
`<!—#include file="ssi.txt" —>`	Exhibe un archivo dada una trayectoria relativa. La trayectoria relativa no puede comenzar con la secuencia de caracteres ../ o con el caracter / para evitar riesgos de seguridad.

SSI proporciona un conjunto bastante rico de características para el programador.

Nosotros podremos usar SSI si tuvieramos un conjunto de documentos existentes a los que quisieramos agregar fechas de modificación. También podríamos tener un archivo que quisieramos incluir en varias de nuestras páginas, tal vez para actuar como encabezado o pie de página. Podríamos utilizar sólo el comando de inclusión SSI sobre cada una de esas páginas en vez de copiar en forma manual el documento a cada página.

Cuando están disponibles, los SSI proporcionan una buena forma de hacer más interesantes a las páginas simples.

Antes de que estuvieran disponibles los SSI, se requería de un programa CGI a fin de generar automáticamente el texto de la fecha de la última modificación, o para agregar pies de página genéricos a todas las páginas.

4.3.4 Formularios HTML *(04B) (05R) (12B)*

Los formularios HTML se diseñan para permitir que el diseñador de página Web interactúe con los usuarios permitiéndoles llenar un formulario. El formulario puede estar compuesto de elementos como cuadros de entrada, botones, casillas de verificación, botones de opción y listas de selección. Todos los elementos del formulario se especifican mediante etiquetas HTML rodeadas por las etiquetas `<FORM>..` `</ FORM>`. Es posible tener más de un formulario por documento HTML.

Existen varios codificadores u opciones que se usan con la etiqueta `<FORM>`. Los más importantes son `METHOD` y `ACTION`:

- `METHOD`—Especifica la manera en la que se transfiere la información a los scripts CGI. Los valores normales son `GET` y `POST`.
- `ACTION`—Especifica el URL del script CGI que se invocará cuando se haga clic sobre el botón de remisión. También puede especificar una dirección de e-mail mediante la notación `mailto`: Por ejemplo, el envío de correo se llevaría a cabo mediante `ACTION=" mailto:medined@planet.net"` y la invocación a un script CGI se haría por medio de `ACTION="/cgi-bin/generar.pl"`,

La mayoría de los elementos de campos se definen utilizando la etiqueta **<INPUT>**. Al igual que la etiqueta **<FORM>**, **<INPUT>** tiene diversos modificadores. Los más importantes son:

- **CHECKED**—Especifica que la casilla de verificación o botón de opción que se está definiendo está seleccionada. Este modificador sólo debe emplearse cuando el tipo de elemento sea checkbox o radio.
- **NAME**—Especifica el nombre de un elemento de formulario. La mayoría de estos elementos deben tener nombres únicos.
- **MAXLENGHT**—Especifica el número máximo de caracteres que puede ingresar el usuario en un elemento de formulario. Si **MAXLENGHT** es mayor que **SIZE**, el usuario podrá desplazar el texto para accesar aquél que no es visible.
- **TYPE**—Especifica el tipo de campo de entrada. Los tipos de campo más importantes son: **checkbox**, **hidden**, **password**, **radio**, **reset**, **submit** y **text**.
- **SIZE**—Especifica el tamaño de un campo de entrada.
- **VALUE**—Especifica el valor por omisión para un campo. El modificador **VALUE** es obligatorio en los botones de opción (radio buttons).

4.3.5 Forms simples *(04B) (05B) (12B)*

Comenzaremos con una página que tiene incluido el tag correspondiente al inicio de un form;

Listado 4.02 Formulario HTML utilizando un Form simple

```
<HTML>
<HEAD>
<TITLE> Página de Prueba </TITLE>
</HEAD>
<BODY>
<H1>Titulo Prueba de Forms </H1>
<FORM METHOD="post" ACTION="http://alguna.direccion/cgi-
bin/prog.pl"><-- aquí comenzaría el código para los elementos del
form --!>
. . . . . .
. . . . . .
</FORM> <-- y aquí terminaría --!>
</BODY>
</HTML>
```

Evidentemente "`prog.pl`" es el nombre del ejecutable (script CGI) que correrá en el Web Server una vez que el usuario envíe la información requerida, y `POST` es el método que se utilizará para realizar dicha acción. Existe otro método llamado `GET` que hace exactamente lo mismo, la diferencia radica en que este último envía la data dentro de la variable de entorno que habíamos mencionado (`QUERY_STRING`) "pegada" al URL del script y `POST` lo hace como un "stream" continuo de caracteres. Es preferible utilizar `POST` ya que el mismo no tiene limitaciones de tamaño y en cambio `GET` sí puede tenerlas en el Server.

Text Line

El elemento más simple que podemos utilizar en un FORM es el "Text line" que le permite al usuario ingresar texto en una línea. La sintaxis utilizada para este tag es la siguiente;

Listado 4.03 Formulario HTML utilizando un Text Line

```
<FORM METHOD="post" ACTION="http://alguna.direccion/cgi-
bin/text.pl">
Escriba su nombre<BR>
<INPUT TYPE="text" NAME="nom_var" SIZE=20 >
</FORM>
```

En el browser se verá:

Escriba su nombre

Cuando el usuario oprima la tecla "Enter" lo que haya escrito será enviado al Web Server. Veamos los parámetros de este tag. En `NAME` va el nombre de la variable que almacenará el texto que ingrese el usuario y en `SIZE` tenemos 20 que es la longitud en caracteres que va a tener la caja. Por medio de un script-CGI "capturaremos" la variable "`nom_var`" y leeremos su contenido, el cual podremos procesar posteriormente.

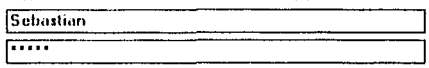
Password Text

En adición a Text Line existe el tipo "**password**" que sirve para ingresar claves o textos secretos, cuya sintaxis es muy parecida. No se trata para nada de un medio seguro para enviar información a través de la Web, pero, puede servirnos para este fin didáctico.

Listado 4.04 Formulario HTML utilizando un Password Text

```
<FORM METHOD="post" ACTION="http://alguna.direccion/cgi-
bin/pass.pl">
Ingrese su nombre y password.<BR>
<INPUT TYPE="text" NAME="nom_var" SIZE=20 >
<BR>
<INPUT TYPE="password" NAME="pass_var" SIZE=20 > <BR>
</FORM>
```

Obtendremos la siguiente salida:



Obviamente lo que escribamos en la caja de password no aparecerá como texto normal sino con una "*" representando cada caracter.

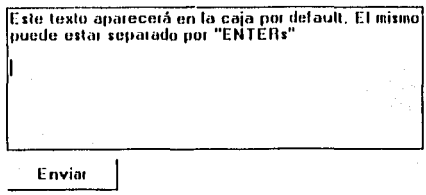
Text Boxes

Si necesitamos recibir más de una línea podemos utilizar los tags **<TEXTAREA> </TEXTAREA>** de la siguiente manera;

Listado 4.05 Formulario HTML utilizando un Text Boxes

```
<FORM METHOD="post" ACTION="http://alguna.direccion/cgi-
bin/text.pl">
Ingrese su comentario<BR>
<TEXTAREA NAME="caja" ROWS=5 COLS=40> Este texto aparecerá en la
caja por default. El mismo puede estar separado por
"ENTERs".</TEXTAREA> <BR>
<INPUT TYPE="submit" NAME="boton_env" VALUE="Enviar">
</FORM>
```

Obtendremos la siguiente salida:



El significado de cada parámetro es el siguiente:

- `NAME` Nombre de la variable que almacena el texto de la caja.
- `ROWS` Cantidad de filas de la caja de texto.
- `COLS` Cantidad de columnas de la columna.

Submit Button

El caso que acabamos de ver, el "Enter" sirve para pasar a la línea siguiente mientras se está escribiendo, de manera que implementamos otro método para enviar la información, que es el "Submit Button", cuando el mismo sea oprimido el texto que hayamos cargado será enviado.
Estos son los parámetros del "Submit Button":

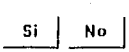
- `NAME` nombre del botón (para la referencia en el script)
- `VALUE` texto que aparece en el botón (por default es "`Submit`")

El parámetro "`NAME`" no es necesario en el caso anterior, pero es interesante citarlo ya que podemos llegar a utilizarlo en la siguiente situación;

Listado 4.06 Formulario HTML utilizando un Submit Button

```
<FORM METHOD="post" ACTION="http://alguna.direccion/cgi-
bin/text.pl">
Le gustó la página?<BR>
<INPUT TYPE="submit" NAME="boton_env" VALUE="Si">
<INPUT TYPE="submit" NAME="boton_env" VALUE="No">
```

Se obtendría:



Aquí la variable `boton_env` contendrá los valores "`si`" o "`no`" según cual de los dos botones oprima el usuario.

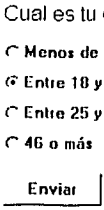
Radio Buttons

Los Radio Buttons permiten que el usuario elija una única opción entre varias. Son esos pequeños círculos que aparecen cuando tenemos que indicar el sexo, o bien un rango de edades o una opinión (muy bueno, bueno, malo). Veamos un ejemplo del código.

Listado 4.07 Formulario HTML utilizando un Radio Buttons

```
<FORM METHOD="post" ACTION="http://alguna.direccion/cgi-
bin/radio.pl">
Cual es tu edad?<BR>
<INPUT TYPE="radio" NAME="edad" VALUE="a"> Menos de 18<BR>
<INPUT TYPE="radio" NAME="edad" VALUE="b" CHECKED> Entre 18 y
24<BR>
<INPUT TYPE="radio" NAME="edad" VALUE="c"> Entre 25 y 45<BR>
<INPUT TYPE="radio" NAME="edad" VALUE="d"> 46 o más<BR>
<INPUT TYPE="submit" NAME="boton_env" VALUE="Enviar">
</FORM>
```

La pantalla mostraría:



El form retornar dentro de la variable "`edad`" el valor "`a`","`b`","`c`" o "`d`" según corresponda la opción que haya sido elegida por el usuario. La cláusula `CHECKED` permite tener un ítem seleccionado por default.

Check Boxes

Los Check Boxes sirven cuando necesitamos recibir un `input` con más de una opción seleccionada, se utilizan por ejemplo para marcar las características que más nos agradan de un determinado producto. La sintaxis es bastante parecida al anterior.

Listado 4.08 Formulario HTML utilizando un Check Boxes

```
<FORM METHOD="post" ACTION="http://alguna.direccion/cgi-
bin/check.pl">
Marque aquellos temas que sean de su interes:<BR>
<INPUT TYPE="checkbox" NAME="temas" VALUE="ntecs"> Nuevas
Tecnologías<BR>
<INPUT TYPE="checkbox" NAME="temas" VALUE="inves" CHECKED>
Investigación<BR>
<INPUT TYPE="checkbox" NAME="temas" VALUE="grafs" CHECKED> Grá
ficos / CAD<BR>
<INPUT TYPE="checkbox" NAME="temas" VALUE="redes"> Redes /
Comunicaciones<BR>
<INPUT TYPE="submit" NAME="boton_env" VALUE="Enviar">
</FORM>
```

En el browser tendríamos:

Marque aquellos temas que sean de su interes

☐ Nuevas Tecnologías

☑ Investigación

☑ Gráficos / CAD

☐ Redes / Comunicaciones

Enviar

Igual que con los radio buttons, la cláusula `CHECKED` permite tener marcados algunos cuadraditos por default. En la variable "`temas`" van a ir a parar aquellas opciones que sean marcadas por el usuario.

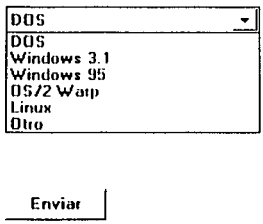
POP UP list

Podemos también utilizar menús descolgantes como manera de elegir una opción entre varias mediante el tag `<SELECT>` como podemos ver más abajo.

Listado 4.09 Formulario HTML utilizando un POP UP list

```
<FORM METHOD="post" ACTION="http://alguna.direccion/cgi-
bin/popup.pl">
Que sistema operativo usas?<BR>
<SELECT NAME="sistema">
<OPTION SELECTED> DOS
<OPTION> Windows 3.1
<OPTION> Windows 95
<OPTION> OS/2 Warp
<OPTION> Linux
<OPTION> Otro
</SELECT> <BR>
<INPUT TYPE="submit" NAME="boton_env" VALUE="Enviar">
</FORM>
```

Si presionamos con el mouse sobre la flechita la lista desplegable se verá así:



Al igual que en los otros ejemplos, la variable "`sistema`" almacenará el ítem elegido.

Forms Múltiples y el Reset Button

Todos los elementos que hemos mencionado pueden ser utilizados individualmente con un submit button (o sea con un form para cada uno) o bien pueden ser empleados varios de ellos en una misma página.▲

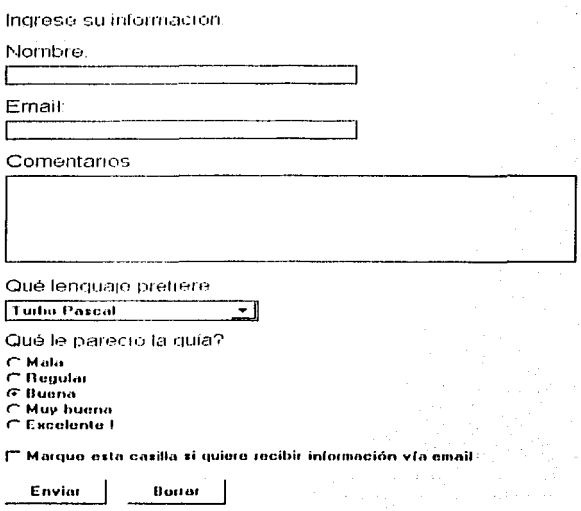
Para finalizar construiremos un ejemplo que contenga todos los tags vistos hasta ahora. Este form múltiple introduce el uso de un nuevo tipo de botón, el Reset Button, que en resumidas cuentas borra los datos que hayamos ingresado y deja los elementos en su opción de default.

Listado 4.10 Formulario HTML con Forms Múltiples

```
<FORM METHOD="post" ACTION="http://alguna.direccion/cgi-
bin/popup.pl">
Ingrese su información:<BR>
Nombre: <INPUT TYPE="text" NAME="nombre" SIZE=30 > <BR>
Email: <INPUT TYPE="text" NAME="email " SIZE=30 > <BR>
Comentarios.<BR>
<TEXTAREA NAME="caja" rows=5 cols=40></TEXTAREA> <BR>
<BR>Que lenguaje prefiere?<BR>
<SELECT NAME="lenguaje">
<OPTION SELECTED> Turbo Pascal
<OPTION> Turbo Pascal
<OPTION> Delphi
<OPTION> Visual Basic
<OPTION> Smalltalk/V
<OPTION> Cobol
</SELECT> <BR>
<BR>Que le pareció la guía? <BR>
<INPUT TYPE="radio" NAME="opin" VALUE="a">Mala <BR>
<INPUT TYPE="radio" NAME="opin" VALUE="b">Regular<BR>
<INPUT TYPE="radio" NAME="opin" VALUE="c" CHECKED>Buena<BR>
<INPUT TYPE="radio" NAME="opin" VALUE="d">Muy Buena<BR>
<INPUT TYPE="radio" NAME="opin" VALUE="d">Excelente!<BR>
<BR>
<INPUT TYPE="checkbox" NAME="email" VALUE="si">
Marque esta casilla si quiere recibir información vía email. <BR>
<BR>
<INPUT TYPE="submit" NAME="boton_env" VALUE="Enviar">
<INPUT TYPE="reset" NAME="boton_res" VALUE="Borrar">
</FORM>
```

▲ Se estila en la creación de un buen form, el poner líneas horizontales arriba y abajo del form (<HR> o alguna línea gráfica) y proveer instrucciones de como se deben llenar los blancos.

Salida en Pantalla.



Todo lo visto hasta ahora es posible practicarlo sin necesidad de un Server

Que necesitamos para empezar.

Para empezar necesitamos tener un poco de conocimiento de algún lenguaje de programación (el que sea). La idea de esta guía es utilizar como base Perl. Con eso bastará para dar una idea general, (que por otra parte es lo que se pretende alcanzar).

4.3.6 Nuestro primer programa CGI *(04B) (05R) (12B)*

Podemos emplear cualquier editor de texto o procesador de palabras existente para crear nuestros programas CGI, ya que son simples programas Perl que se invocan mediante un URL en vez de a través de la línea de comandos. El Listado 4.11 contiene un programa CGI muy fácil de entender, el cual no necesita que les sean suministrados datos de ningún tipo para funcionar.

Activar la opción de mensajes de advertencia.
Enviar el encabezado HTTP al navegador Web.
Enviar una línea de texto al navegador Web.

Listado 4.11. Un programa CGI muy pequeño

```
#!/usr/bin/perl
print "content-type: text/html\n\n";
print "<HTML>\n";
print "<HEAD>\n";
print "<TITLE>Aprendiendo CGI</TITLE>\n";
print "</HEAD>\n";
print "<BODY>\n";
print "Hola Intertips!\n";
print "</BODY></HTML>\n";
```

En el Listado anterior la primera línea se encarga de llamar al intérprete de Perl en una maquina que funciona con sistemas UNIX.

Nos detendremos un momento para ver ya cuáles son los puntos que deberemos tener en cuenta al crear programas.

La primera línea del programa debe ser siempre una directiva que indique al servidor a quién debe pasarle nuestro programa: al intérprete de Perl. En ella debemos incluir la trayectoria dónde está instalado, junto con su nombre. La línea debe empezar con los signos `#!`

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

Para llamar de igual forma al intérprete de Perl en una PC con Windows XP o Win9x bastara con cambiar el `#!/usr/bin/perl` por `#!c:\Perl\bin\Perl.exe`

Ahora el programa no se está ejecutando en nuestra computadora y por lo tanto la salida estándar no es la pantalla, si no un canal que envía los datos hacia el servidor, quien a su vez los transmitirá por la red hacia el navegador que los ha solicitado. Las instrucciones `print` que realicemos enviarán el contenido hasta el servidor WWW, éste lo examinará para ver si hay alguna directiva dirigida a él, el resto lo lanzará como respuesta al cliente que llamó al script o programa.

Para comprender cómo debe enviar la respuesta un programa debemos tener en cuanta los siguientes puntos:

- Los programas navegadores pueden recibir información en varios formatos: texto, HTML, imágenes GIF o JPG, sonido MIDI, etc.

- Cuando un servidor envía un archivo, sabe por la extensión del mismo el tipo de datos que contiene.

En la Tabla 4.08 nos muestra algunos ejemplos de extensiones MIME

Tabla 4.08 Ejemplos de extensiones MIME

Extensión	*Tipo de datos*
`.TXT`	Texto sin formato
`.HTM, .HTML o SHTML`	Documento en formato HTML.
`.JPG`	Imagen
`.GIF`	Imagen
`.MPG`	Video
`.MID`	Sonido
`.WAV`	Sonido

- Los scripts que envían datos de respuesta hacia el navegador antes de empezar a generar la respuesta deben mandar primero la línea con la cláusula **`content-type`** informando al servidor cuál van a ser los tipos de datos que deberán remitir al cliente.

En nuestro primer ejemplo mostrado en el Listado 4.11 la primera línea **`print`** envía esta cláusula advirtiendo que la respuesta se va a generar en formato HTML

```
print "content-type: text/html\n\n";
```

Cuando el servidor recibe una línea que comienza con la cláusula **`content-type`** sabe que se trata de una instrucción para él, la cual no debe enviar al cliente.

Para indicarle que ya hemos terminado de definir el tipo de datos y que lo que venga a continuación no forma parte de la directiva **`content-type`**, debemos enviar dos saltos de línea seguidos. Si nos olvidamos de ponerlos, el servidor tratará la información que reciba a continuación como parte de la descripción **`content-type`** y esta información por consiguiente no la enviaría hacia el cliente. Hay que recordarlo bien, por que éste es uno de los errores más comunes que pueden llevarnos a quebrarnos la cabeza, al ver que el script no genera ninguna respuesta.

4.3.7 CGI y las variables de ambiente *(04B) (05R) (12B)*

Al iniciar el programa CGI, el servidor Web crea e inicializa diversas variables de ambiente que nuestro programa puede accesar utilizando el arreglo asociativo `%ENV`.

La Tabla 4.09 contiene una breve descripción de cada variable de ambiente.

Tabla 4.09 Variables de ambiente CGI

Nombre de variable	*Descripción*
`AUTH_TYPE`	Proporciona, en forma opcional, el protocolo de autentificación utilizando para accesar su script. Si el servidor Web local maneja la autentificación y si ésta se usó para accesar su script.
`CONTENT_LENGTH`	Proporciona, en forma opcional, la longitud en bytes, del contenido proporcionado por el script a través del identificador de archivo STDIN. Se usa en particular en el método `POST` o en el procesamiento de formularios.
`CONTENT_TYPE`	Proporciona en forma opcional el tipo de contenido disponible del identificador de archivo STDIN. Se utiliza para el método `POST` o en el procesamiento de formularios. La mayor parte del tiempo, la variable estará en blanco y podemos asumir un valor de application / octet-stream.
`GATEWAY_INTERFACE`	Proporciona la versión de CGI manejada por el servidor Web local. La mayoría de las veces será igual a CGI/ 1. 1.
`HTTP_ACCEPT`	Proporciona una lista separada por comas de los tiempos MIME que aceptará el software del navegador. Podemos revisar esta variable de ambiente para ver si el cliente aceptará cierto tipo de archivo gráfico.
`HTTP_FORM`	Proporciona la dirección e-mail del usuario. No todos los servidores Web darán esta información. Por lo tanto, usaremos este campo para proporcionar un valor por omisión para un formulario HTML.
`HTTP_USER_AGENT`	Proporciona el tipo y versión del navegador Web del usuario. Por ejemplo, el navegador Web Netscape se denomina Mozilla.
`PATH_INFO`	Contiene de manera opcional cualquier información adicional de trayectoria de la solicitud HTTP que invocó al script.
`PATH_TRANSLATED`	Mapea la trayectoria virtual del script a la trayectoria física empleada para invocar al

	script.
QUERY_STRING	Contiene de manera opcional información del formulario cuando se emplea el método GET del procesamiento de formularios. También se usa para pasar información como la búsqueda de palabras clave para sripts CGI.
REMOTE_ADDR	Contiene la dirección decimal de punto del usuario.
REMOTE_HOST	Proporciona, en forma opcional el nombre de dominio del sitio desde el que se ha conectado el usuario.
REMOTE_IDENT	Proporciona, en forma opcional, una identificación del cliente cuando su servidor local ha contactado un servidor IDENTD en una maquina cliente. Será muy raro que veamos esto, ya que la consulta IDENTD es lenta.
REMOTE_USER	Proporciona, en forma opcional, el nombre utilizado por el usuario para accesar su script protegido.
REQUEST_METHOD	Por lo regular ya sea "GET" o "POST", el método mediante el cual se pondrá a disponibilidad del script información y formularios.
SCRIPT_NAME	Contiene la trayectoria virtual a script.
SERVER_NAME	Contiene el nombre de host configurado para el servidor.
SERVER_PORT	Contiene el número de Puerto sobre el que el servidor Web local está escuchando. El número estándar de puerto es 80.
SERVER_PROTOCOL	Contiene la versión de protocolo Web que usa este servidor. Por ejemplo HTTP/1.0.
SERVER_SOFTWARE	Contiene el nombre y versión del software de servidor Web. Por ejemplo Website/1.1e.

Después de haber dejado en claro porque se utilizan tanto en la primera línea el `#!` y la directiva `content-type`, así, como las variables de ambiente

Ahora vamos a hacer algo un poco más interesante, vamos a solicitar algunos datos con un FORM apropiado, y vamos a retornar una página que contenga los nombres de las variables utilizadas junto con el contenido de cada una ingresado por el usuario.

4.3.8 Un programa CGI que hace un Test de Física *(04B) (05R) (12B)*

Crear test a partir de un programa CGI es muy simple, sólo tenemos que crear una página HTML. Debemos asegurarnos de no poner en los nombres de los campos del test números. Cada entrada tiene que tener como valor un número que corresponderá a la puntuación de esa respuesta.

Debemos crear el archivo que contiene el resultado del test por ejemplo:

En un archivo que se llame quiz.dat.
En ese archivo tenemos algo como esto:

0:2: Tu resultado ha sido muy bajo. Repasa algún libro y vuelve en Septiembre.
3:4: No está; mal. Superas el nivel medio.
4:5: Sobresaliente. Pocos pueden decir que saben tanto de física como tú.
6:6: Eres muy bueno, acertaste hasta en la pregunta del CD-ROM!!!

Veamos el form que vamos a utilizar:

Listado 4.12 Formulario HTML para realizar el Test de Física

```
<HTML>
<HEAD><TITLE>Ejemplo de Test</TITLE></HEAD>
<BODY bgcolor="#FFFFFF">
<CENTER><H1>Test sobre tus conocimientos de Física </CENTER></H1>
<BR>
<FORM METHOD="POST" ACTION='/cgi-bin/quiz.pl'>

<br>La ecuación de Bernoulli:
<br><INPUT TYPE="radio" name="one" value="1"> Describe la
conservación de la energía para un fluido ideal
<br><INPUT TYPE="radio" name="uno" value="0"> Describe la
conservación de la energía para un fluido viscoso
<br><INPUT TYPE="radio" name="uno" value="0"> Describe la
conservación de la masa para un fluido
<br><INPUT TYPE="radio" name="uno" value="0"> Describe la
conservación de la energía para un fluido no pascaliano.
<br><INPUT TYPE="radio" name="uno" value="0"> Conocida ecuación
mediante la cual se establece el precio de los coches de lujo de
origen italiano
<br>
<br> Estamos en la cocina de nuestra casa a las 2 de la madrugada,
comiendo algo mientras se carga por FTP un programa de 1.5 Mb. De
pronto se produce un error crítico en el sistema (usamos Windows
XP) y suena un pitido de 400 Hz por los altavoces. Si nos acercamos
hacia el ordenador a una velocidad próxima a 200 metros por
segundo, ¿ que oiremos ?
<br><INPUT TYPE="radio" name="dos" value="1"> Un pitido más agudo
<br><INPUT TYPE="radio" name="dos" value="0"> Un pitido más grave
<br><INPUT TYPE="radio" name="dos" value="0"> Los gritos de los
vecinos
```

```
<br><INPUT TYPE="radio" name="dos" value="0"> Nuestra propia voz
maldiciendo el día en que decidimos instalar ese infecto sistema
operativo
<br>
<br>La segunda ley de Newton establece:
<br><INPUT TYPE="radio" name="tres" value="0"> La relación entre
la potencia suministrada a un objeto y la velocidad que éste
adquiere
<br><INPUT TYPE="radio" name="tres" value="1"> La relación entre la
variación de momento lineal y la aceleración de un objeto.
<br><INPUT TYPE="radio" name="tres" value="0"> Que las manzanas
caen de los árboles
<br><INPUT TYPE="radio" name="tres" value="0"> La fuerza con la que
se atraen dos masas por efecto de la gravedad
<br>
<br> Un CD-ROM está; girando con una velocidad angular constante.
Los bits de información, ¿estén acelerados?
<br><INPUT TYPE="radio" name="cuatro" value="0"> No, ya que el CD-
ROM gira a una velocidad angular constante.
<br><INPUT TYPE="radio" name="cuatro" value="2"> No, ya que los
bits de información no poseen masa.
<br><INPUT TYPE="radio" name="cuatro" value="0"> Si, ya que todo
movimento circular conlleva una aceleración perpendicular a la
trayectoria
<br><INPUT TYPE="radio" name="cuatro" value="1"> Esta no es  tan
fácil como las otras.
<br>
<br> Mi ilusión será pasar un rato con :
<br><SELECT NAME="cinco">
<OPTION VALUE="0">Mi ordenador</OPTION>
<OPTION VALUE="0">Mi novia/novio o mi esposa/marido</OPTION>
<OPTION VALUE="0">El galan Tom Cruse</OPTION>
<OPTION VALUE="0">La belleza Cameron Diaz</OPTION>
<OPTION VALUE="1">Galileo Galilei</OPTION>
<OPTION VALUE="1">Marie Curie</OPTION>
</SELECT>
<br>
<br><input type="submit" value="Muestrame mi puntuación"><input
type="reset" value="borrar">
</BODY>
</HTML>
```

La Figura 4.04 muestra cómo lucirá terminado el Formulario del Test de Física.

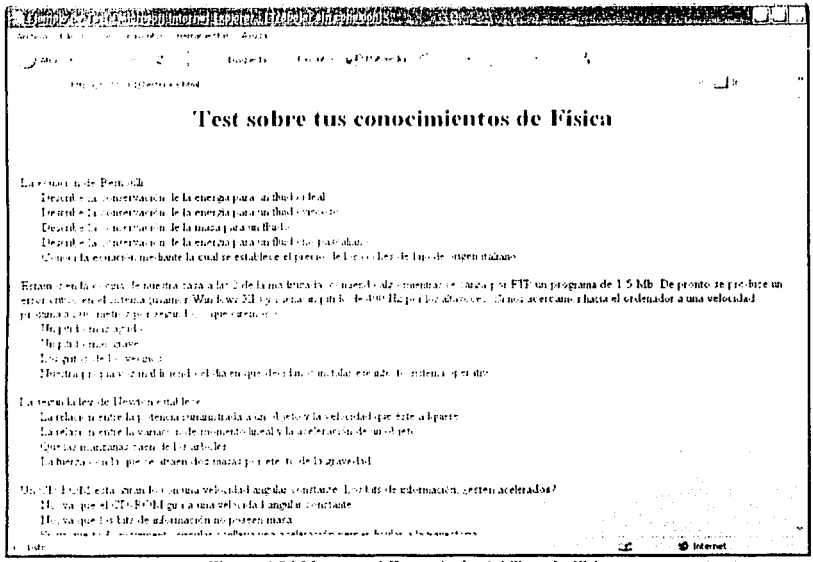


Figura 4.04 Muestra el Formulario del Test de Física

El Listado 4.13 contiene un programa CGI muy fácil de entender, cabe señalar que ahora la primera línea es: `#!c:\Perl\bin\Perl.exe` para poder ahora ver un ejemplo utilizando un sistema Windows.

Listado 4.13. Programa CGI que hace posible desarrollar un Test de Física

```
#!c:\Perl\bin\Perl.exe

# El archivo que contiene los resultados
$archivo_result = "quiz.dat";

# Empezamos la pagina con el body.
$body = "<BODY BGCOLOR=FFFFFF TEXT=000000>";

# El titulo
$titulo = "Test de Fisica";

# La minima puntuacion que se puede obtener.
$min = "0";
&form;

&cuenta_quiz;
```

```
sub imprimir_result
{
print "Content-type: text/html\n\n";
print "<HTML><HEAD><TITLE>$titulo</TITLE></HEAD>\n";
print "$body<CENTER><H1>$titulo</CENTER></H1><BR>\n";
print "Estos son sus resultados:<br>\n";
print "Su puntuacion: $cuenta\n";
print "<br>Significado: $signif <br><br>";
print "<br><hr><br></font></body></html>\n";
exit;
}

sub cuenta_quiz
{
open(CUENTA,"$archivo_result") || &error;
@lineas = <CUENTA>;
close(CUENTA);
$cuenta = 0;
foreach $respues(respuest, %entrada)
{
$cuenta = $respues + $cuenta;
}
if($cuenta < $min) { &error; }
foreach $linea(@lineas)
{
($bajo,$alto,$signif) = split(/\:/,$linea);
      if($cuenta == $bajo || $cuenta > $bajo && $cuenta ==
$alto || $cuenta < $alto)
      {
      &imprimir_result;
      }
}
&error;
}



sub error
{
print "Content-type: text/html\n\n";
print "<HTML><HEAD><TITLE>Error</TITLE></HEAD>\n";
print "<BODY><CENTER><H1>Error</CENTER></H1><BR>\n";
print "Ha ocurrido un error al intentar procesar los datos.
Asegurese ";
print "de haber contestado todas las preguntas.\n";
print "<br><hr><br></font></body></html>\n";
exit;

}

sub form
{
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
   if (length($buffer) < 5) {
         $buffer = $ENV{QUERY_STRING};
    }
   @pairs = split(/&/, $buffer);
```

```
    foreach $pair (@pairs) {
        ($nomb, $valor) = split(/=/, $pair);
          $valor =~ tr/+/ /;
          $valor     =~     s/%([a-fA-F0-9][a-fA-F0-9])/pack("C",
hex($1))/eg;
          $entrada{$nomb} = $valor;
    }

}
```

La Figura 4.05 muestra cómo lucirá terminado el Test de Física.

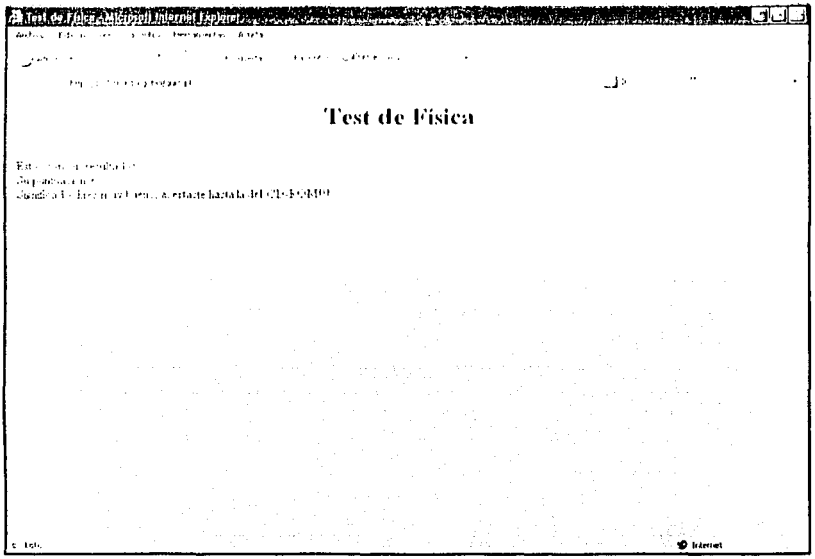


Figura 4.05 El Test de Física

Es evidente que a partir de este punto podríamos abordar cosas más complicadas si quisiéramos, ya que es posible utilizar todos los recursos de Perl para efectuar las acciones y procesos que queramos. Como sugerencia, la estructura que se ve aquí para un script más o menos decente es la siguiente (de todas maneras igual que en matemáticas de primer grado, "el orden de los factores no altera el producto", aunque sí lo hace más difícil de mantener:)

- Asignación de variables.
  Proceso principal del script que generar la salida que deseamos a partir de los datos de entrada.

- Envío de la página con los resultados del proceso

Lo mejor es comenzar a practicar con las estructuras más comunes, decisión (`IF`) e iteración (`FOR`), haciendo pequeños programas y probando las salidas obtenidas. Por ejemplo, un script que dependiendo del contenido de la variable "`nombre`" retorne una página con distintos mensajes según sea quien ha completado el form.

Los scripts pueden hacer cosas más complicadas, como verificar que los campos sean llenados correctamente, modificar archivos y hasta retornar una página que no haya sido elaborada por ellos. En el siguiente y último punto veremos como implementar estos métodos en la forma de un 'Libro de visitas'.

4.3.9 Creación de un Libro de Visitas para nuestro sitio *(04B) (05R) (12B)*

En esta sección, crearemos un Libro de Visitas para nuestro sitio Web. Un Libro de Visitas ofrece a los visitantes un lugar para agregar comentarios y ver los comentarios formulados por otros visitantes. De esta manera se incorpora un sentido comunitario al sitio Web que tenemos.

La aplicación de muestra del Libro de Visitas se presentará en dos etapas. Primero, se usa un formulario HTML para solicitar información, luego esta información se guarda y se exhiben los registros del Libro de Visitas mediante un programa CGI.

La Figura 4.06 muestra cómo lucirá terminado el Libro de Visitas.

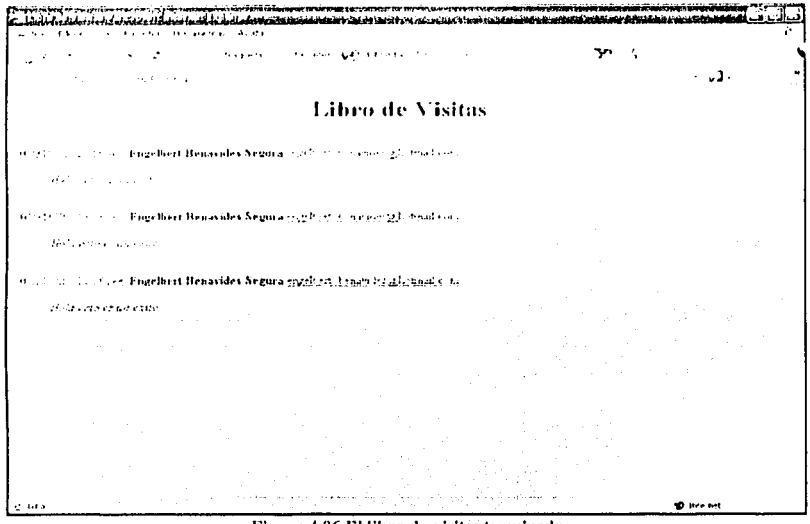


Figura 4.06 El libro de visitas terminado

El Libro de Visitas básico

Por lo regular, se llega a una aplicación de Libro de Visitas desde la página base de un sitio Web. Es posible que nosotros queramos agregar a nuestra página base un vínculo como el siguiente:

```
<A HREF="addlibro.htm">[Libro de Visitas]</A>
```

Después, debemos de colocar la página Web del Listado 4.14 en el directorio raíz virtual de nuestro servidor Web. Al hacer clic sobre el vínculo de hipertexto, llevaremos a los visitantes al formulario *Agregar Registro*.

Iniciar la página Web HTML.
Definir el encabezado de la página Web, el cual contiene el título.
Iniciar el cuerpo de la página.
Exhibir un encabezado.
Exhibir algunas instrucciones.
Iniciar un formulario HTML.
Iniciar una tabla HTML.
Cada fila de la tabla es otro campo de entrada,
Definir el botón de remisión.
Terminar la tabla.
Terminar el formulario.
Terminar el cuerpo de la página.
Terminar la página.

Listado 4.14 El formulario HTML de adición de registros al Libro de Visitas

```
<HTML>
<HEAD><TITLE>Libro de visitas</TITLE></HEAD>
<BODY>
<CENTER><H1> Libro de visitas </H1></CENTER>
Rellena los espacios en blanco del formulario. Para poder añadir
tus comentarios debes rellenar, al menos, los campos
correspondientes a tu nombre y los comentarios. Gracias.
<HR>
<FORM METHOD=POST ACTION="/cgi-bin/libro.pl">
<TABLE BORDER=0 CELLPADDING=10>
<TR>
<TD>Tu Nombre:</TD>
<TD><INPUT TYPE=text NAME=name SIZE=30></TD>
</TR>
<TR>
<TD>E-mail:</TD>
<TD><INPUT TYPE=text NAME=email SIZE=40></TD>
</TR>
<TR>
<TD VALIGN=top> Comentarios:</TD>
<TD><TEXTAREA NAME=comentarios COLS=60 ROWS=4></TEXTAREA></TD>
</TR>
</TABLE>
```

```
<INPUT TYPE=submit VALUE="Firmar"><INPUT TYPE=reset>
</FORM>
</BODY>
</HTML>
```

Lo único que pudiera ser necesario modificar a fin de que este formulario funcione, es el modificador **ACTION** de la etiqueta **<FORM>**. El directorio en dónde colocaremos el programa CGI pudiera no ser /cgi-bin. El archivo addlibro.html generará una página Web como de la Figura 4.07 siguiente:

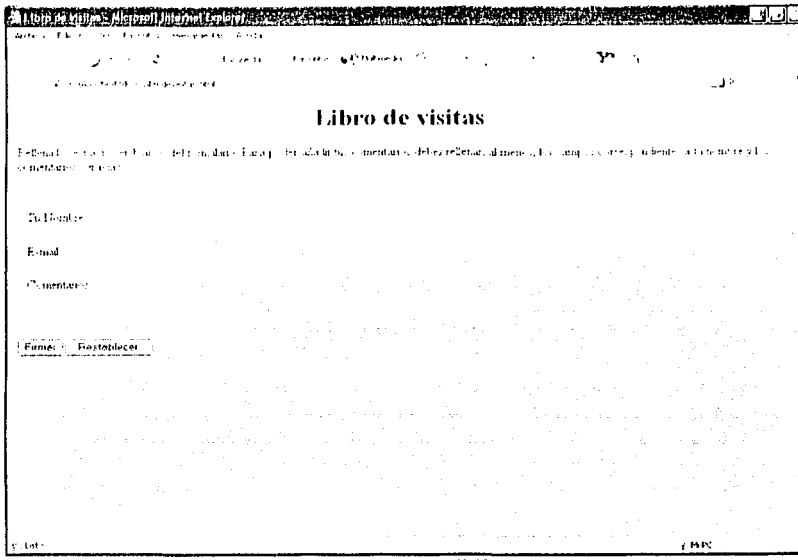


Figura 4.07 El formulario de adición de registros.

El programa CGI del Listado 4.15 se invoca cuando un visitante hace clic sobre el botón de remisión del formulario HTML de adición de registro. Este programa procesará la información del formulario. La guardará en un archivo de datos y luego creará una página Web para exhibir todos los registros en el archivo de datos.

Activar la opción de mensajes de advertencia.
Activar el programa strict.
Declarar una variable de arreglo asociativo para contener los datos de los campos del formulario HTML.
Obtener la hora local y simula que es uno de los campos del formulario.
Obtener los datos del formulario.
Guardar los datos en un archivo.

Enviar el encabezado HTTP al navegador Web.
Enviar el inicio de página e información de encabezado.
Enviar el encabezado y solicitar una línea horizontal.
Llamar la función readFormData para exhibir los regis tros del Libro de Visitantes.
Terminar la página Web.
Definir la función getFormData( ).
Declarar una variable local para contener la referencia al arreglo asociativo de campos de entrada.
Inicializar un búfer.
Si se usa el método GET, copia la información dentro del búfer.
Si se usa el método POST, lee la información dentro del búfer.
Iterar sobre el arreglo retomado por la función split().
Decodificar tanto el nombre del campo de entrada como su valor.
Comprimir en unas solas varias etiquetas <P>.
Convertir < en &it y > en &gt evitando su interpretación como etiquetas HTML..
Activar de nuevo las etiquetas HTML de negritas y cursivas.
Eliminar los caracteres de retorno de carro innecesarios.
Convertir dos líneas nuevas en una etiqueta de párrafo HTML.
Convertir las líneas nuevas individuales en espacios.
Crear un elemento en la variable de arreglo asociativo de campos de entrada.
Definirla función decodeURL ().
Obtener la cadena codificada del arreglo de parámetros.
Traducir todos lo signos de más en espacios.
Convertir los caracteres codificados como dígitos hexadecimales en caracteres normales.
Retornar la cadena decodificada.
Definir la función zeroFill, cambia "1" por "01".
Declarar una variable local para contener el número a rellenar.
Declarar una variable local para contener la longitud de cadena necesaria.
Determinar la diferencia entre la longitud actual de la cadena y la longitud requerida.
Si la cadena es lo suficientemente grande (como "12"), entonces la retorna.
Si la cadena no es suficientemente grande, le pone algunos ceros como prefijo.
Definir la función FormData()
Declara dos variables locales para contener el arreglo asociativo y el nombre del archivo
Abrir el archivo para adición.
Almacenar el contenido del arreglo asociativo en el archivo de datos.
Cerrar el archivo.
Definir la función readFormData()
Declarar una variable local para contener el nombre del archivo.
Abrir el archivo para lectura.
Imprimir sobre las líneas del archivo.
Realizar un split dentro en cuatro variables usando - como delimitador.
Imprimir el registro del Libro de Visitas utilizando un mínimo de etiquetas HTML.
Usar la regla horizontal para separar los registros.
Cerrar el archivo.

Listado 4.15 Un programa CGI para agregar un registro en el Libro de Visitas y exhibir una página HTML con dicho libro

```
#!c:\Perl\bin\Perl.exe

use strict;
    my(%campos);
    my($seg,    $min,    $hora,    $mdia,    $mes,    $ano)    =
(localtime(time))[0..5];
    my($archDatos) = "c:\\perl\\datos\\coment.dat";
    $mes  = zeroFill($mes, 2);
    $hora = zeroFill($hora, 2);
    $min  = zeroFill($min, 2);
    $seg  = zeroFill($seg, 2);
    $campos{'crontiempo'} = "$mes/$mdia/$ano, $hora:$min:seg";
    OptDatosForm(\%campos);
    SalvarDatosForm(\%campos, $archDatos);
    print("Content-type: text/html\n\n");
    print("<HTML>\n");
    print("<HEAD><TITLE>Libro de Visitas</TITLE></HEAD>\n");
    print("<H1><CENTER>Libro de Visitas</CENTER></H1>\n");
    print("<HR>\n");
    LeerDatosForm($archDatos);
    print("</BODY>\n");
    print("</HTML>\n");

sub OptDatosForm {
    my($hashRef) = shift;
    my($buffer) = "";
    if ($ENV{'REQUEST_METHOD'} eq "GET") {
        $buffer = $ENV{'QUERY_STRING'};
    }
    else {
        read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
    }
    foreach (split(/&/, $buffer)) {
        my($llave, $valor) = split(/=/, $_);
        $llave   = decifrarURL($llave);
        $valor = decifrarURL($valor);
        $valor =~ s/(<P>\s*)+/<P>/g;
        $valor =~ s/</&lt;/g;
        $valor =~ s/>/&gt;/g;
        $valor =~ s/&lt;b&gt;/<b>/ig;
        $valor =~ s!&lt;/b&gt;!</b>!ig;
        $valor =~ s/&lt;i&gt;/<b>/ig;
        $valor =~ s!&lt;/i&gt;!</b>!ig;
        $valor =~ s!\cM!!g;
        $valor =~ s!\n\n!<P>!g;
        $valor =~ s!\n! !g;
        %{$hashRef}->{$llave} = $valor;
    }
    $campos{'comentarios'} =~ s!\cM!!g;
    $campos{'comentarios'} =~ s!\n\n!<P>!g;
    $campos{'comentarios'} =~ s!\n!<BR>!g;
}
sub decifrarURL {
```

```
    $_ = shift;
    tr/+/ /;
    s/%(..)/pack('c', hex($1))/eg;
    return($_);
}
sub zeroFill {
    my($temp) = shift;
    my($len)  = shift;
    my($diff) = $len - length($temp);
    return($temp) if $diff <= 0;
    return(('0' x $diff) . $temp);
}
sub SalvarDatosForm {
    my($hashRef) = shift;
    my($archivo)    = shift;
    open(ARCH, ">>$archivo") or die("No se puede abrir el archivo
del Libro de Visitas.");
    print ARCH ("$hashRef->{'crontiempo'}~");
    print ARCH ("$hashRef->{'name'}~");
    print ARCH ("$hashRef->{'email'}~");
    print ARCH ("$hashRef->{'comentarios'}");
    print ARCH ("\n");
    close(ARCH);
}
sub LeerDatosForm {
    my($archivo)    = shift;
    open(ARCH, "<$archivo") or die("No se puede abrir el archivo
del Libro de Visitas.");
    while (<ARCH>) {
        my($crontiempo, $nombre, $email, $comentarios) = split(/~/,
$_);

        print("$crontiempo:            <B>$nombre</B>            <A
HREF=mailto:$email>$email</A>\n");
        print("<OL><I>$comentarios</I></OL>\n");
        print("<HR>\n");
    }
    close(ARCH);
}
```

Este programa no presenta ningún truco nuevo de Perl, así que se debe de entender con facilidad. Al invocar al programa, éste leerá la información del formulario y luego la guardará al final de un archivo de datos. Después de esto, el programa generará una página Web para exhibir todos los registros del archivo de datos.

4.4 Uso de Perl con servidores Web *(09B)*

4.4.1 Perl en un servidor WWW *(09B)*

Aunque un servidor WWW puede estar instalado en un red LAN e incluso en una PC aislado, aquí nos refereriremos a un equipo integrado en Internet dedicado a recibir y dar curso a solicitudes de las páginas que tiene almacenadas.

Lo primero que nos hará falta para ejecutar nuestros propios CGI'S en Internet es contar con acceso a un servidor WWW o instalar uno en el que podamos grabar nuestras propias páginas y programas.

Una de las formas más usuales para conseguirlo es a través de las páginas que conceden las empresas proveedoras de acceso a sus clientes o a través de una empresa especializada en el hospedaje de Webs.

Y otra de las formas es instalando nuestro propio servidor, y para ello en esta sección hablaremos de el servidor Apache que nos permite alojar nuestras propias páginas y además porque es un software de distribución libre.

4.4.2 ¿Qué es lo bueno de Apache? *(09B)*

Apache estuvo en el lugar adecuado en el momento adecuado. Los creadores de sitios Web necesitan ciertas opciones y reparar los fallos, por lo que nació Apache (software de los usuarios para los usuarios). El modelo Open Source era ideal para este proyecto, ya que, especialmente en los primeros días del Web. Todo iba muy deprisa, y las empresas no se podían permitir el lujo de esperar a que un director de ingeniería decidiera si se podia vender un producto o servicio por la red. Necesitaban tener una posibilidad inmediatamente.

Actualmente Netcraft[▲] (http://www.netcraft.com/survey/) informa que 4.078.326 sitios Web están ejecutando Apache. Esto significa el 55,33% de todos los sitios Web examinados. El siguiente en la lista es Microsoft IIS, con un 22,0%.

Apache se ejecuta en más sitios Web que si sumamos el resto de servidores, ya que es un software mejor desarrollado. Algunas personas prefieren Apache a otros servidores porque es gratuito. Pero, incluso en organizaciones donde el precio es secundario, como IBM y la Familia Real Británica (http://www.royal.gov.uk/), Apache es el servidor elegido.

[▲] Netcraft ha hecho encuestas en la Web desde julio de 1995, momento en que registró 18.957 sitios en la Web. La empresa actualiza sus encuestas con caracter mensual, mostrando el crecimiento o declive de cada uno de los protagonistas, y ofrece comentarios sobre estas tendencias. Puede ver la encuesta en http://www.netcraft.net survey. Netcraft es una empresa de investigación en Internet, que ofrece encuestas como ésta, así como consultoría de seguridad y varios servicios Web e Internet.

Hay un viejo dicho en la industria del software: "Bueno, rápido, barato: elija cualquiera de los dos". El Proyecto Apache, en cierto modo, ha aunado las tres características.

4.4.3 Antecedentes de Apache *(09B)*

De acuerdo con las estadísticas de Netcraft (http://www.netcraft.com/), el servidor Web Apache se emplea más que el resto del conjunto de servidores Web. De los cerca de 7 millones de sitios Web que tiene la World Wide Web, cerca de 4 millones (el 55% ejecutan Apache. Si también se cuenta el software para servidor basado en código Apache, ésta cifra se acerca al 60%. Veremos cómo surgió Apache y por qué se ha popularizado tanto.

La Figura 4.08 muestra un gráfico de los servidores Web más populares y del número de sitios Web que utilizan esos servidores.

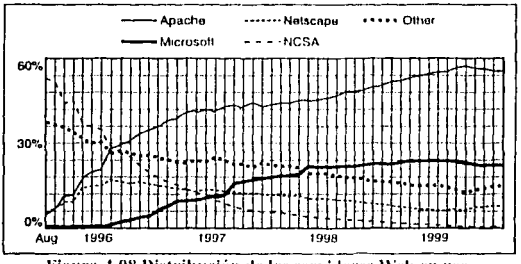


Figura 4.08 Distribución de los servidores Web en uso.

Al comienzo

La Web sigue siendo un fenómeno muy reciente. Tim Bemers -Fue uno más de los creadores de la Web a finales de los noventa cuando trabajaba en el CERN, el Laboratorio Europeo de Física de Partículas. Sus primeros desarrollos fueron para que los físicos de varias universidades de todo el mundo tuvieran un acceso instantáneo a la información, para permitirles colaborar en una serie de proyectos.

Tim definió los URL, HTTP y HTML y, con Robert Cailliau, escribió su primer servidor Web y el primer software para clientes Web, que posteriormente se bautizó con el nombre de navegador.

Hace sólo apenas unos años, tendríamos que haber explicado el significado de estos conceptos a casi todos. Ahora, casi no hay gente (por lo menos en las naciones desarrolladas) que no sepa lo que es la WWW.

Poco después del trabajo inicial de Tim, un grupo de Centro Nacional de Actividades de Supercomputación, *National Center for Supercomputing Activities* (NCSA) de la Universidad de Illinois, en Urbana Champaign (UIUC) desarrolló el servidor Web HTTPd NCSA y el navegador Web gráfico NCSA Mosaic. Mosaic no fue el primer navegador Web gráfico, aunque la gente lo cree. Ese honor le corresponde a Viola, escrito por Pei Wei y disponible con anterioridad a Mosaic. Pero Mosaic se apropió rápidamente de esta condición, y se convirtió en el navegador Web más usado en 1992.

HTTPd NCSA fue el servidor más utilizado en la Web durante los primeros años de su existencia. Sin embargo, en 1994, Rob McCool, que es el creador de HTTPd NCSA, dejó el NCSA, y el proyecto se terminó. Ya no había organización central que desarrollara nuevas características y que distribuyera un producto funcional.

Antes de que el código fuente del servidor se pusiera a disposición de todo el mundo, muchos de sus usuarios habituales habían desarrollado sus propias soluciones a los errores y sus propias características. Estas soluciones se compartían fortuitamente a través de Usenet, pero no había un mecanismo central que recuperara y distribuyera dichas soluciones.

Por consiguiente, Apache (al igual que la World Wide Web) fue ensamblado por voluntarios. Aunque la terminación del proyecto HTTPd NCSA dejara a los desarrolladores con un producto que no funcionaba bien en la época y con nadie a quien reclamar, al final se consiguió un producto muy superior.

¿Quién es el responsable?

En febrero de 1995, Brian Behlendorf y Cliff Skolnick ensamblaron una lista de envío, prepararon una computadora y consiguieron ancho de banda, donado por HotWired. Brian construyó un árbol CVS (Sistema de Versiones Simultáneas), en virtud del cual todo el que quisiera podía contribuir a crear nuevas características y a reparar errores. De esta forma, un grupo de desarrolladores podían recoger las modificaciones a sus códigos y crear un producto combinado. Comenzando con HTTPd 1.3 NCSA, empezaron a aplicar estas soluciones. La primera versión de este producto, llamado Apache, fue la versión 0.6.2, lanzada en abril de 1995.

Los ocho socios fundadores del Grupo Apache eran Behlendorf, Skolnick, Roy T. Fielding, Rob Hartill, David Robinson, Randy Terbush, Robert S. Thau y Andrew Wilson.

Poco después del primer lanzamiento, Thau diseñó una arquitectura completamente nueva. Comenzando con la versión 0.8.8 en agosto de 1995, Apache se incorporó a esta nueva base de código.

Netcraft muestra que Apache sobrepasa a NCSA como primer servidor HTTP a principios de 1996.

4.4.4 ¿Por qué Apache funciona tan bien? *(09H)*

Apache es un producto fantástico. Hace todo lo que se quiere que haga, y nada de lo que no se quiere. Es rápido, fiable y barato. ¿Qué más se podría pedir de una unidad de software?

Apache puede ser todo esto porque es *open source*. Esto significa que todo el que utilice el producto tiene acceso al código fuente. Si tiene una idea de algo que podría ser útil, puede escribir y entregar el código al Grupo Apache para su posible inclusión en el producto. Esto significa que las prestaciones de Apache son las prestaciones que la misma gente tiene al estar utilizando este software en sus sitios Web, y no las prestaciones que alguien ha sugerido en una reunión de marketing.

Además, cuando se encuentran fallos, las numerosas personas que tienen acceso al código podrán sugerir soluciones al problema (o, por citar a Eric Raymond, "con tanto ojos, cualquier fallo es superficial"). De ahí que la reparación de fallos siga rápidamente a su aparición. Esto contrasta con los productos de software cerrado, dónde, si se informa de un fallo, se está a merced del responsable del producto para esa reparación de fallos (en el caso de que atendieran nuestra observación).

Compilar e instalar Apache

Apache está disponible en forma binaria en varias plataformas, pero normalmente está disponible como código fuente. Esto significa que tendrá que tener un compilador C y compilar e instalar Apache. Los usuarios de Windows, que son los que suelen estar más acostumbrados a los programas de instalación gráficos e intuitivos, se congratularán de saber que este programa existe en Windows.

Requerimientos del sistema

Los requerimientos del sistema para ejecutar Apache son mínimos. Necesitaremos un mínimo de 12MB de espacio temporal en la unidad de disco duro para el proceso de instalación. Tras la instalación. Apache ocupa cerca de 3MB, además del espacio que se utilice para colocar el contenido Web.

También necesitará un compilador ANSI-C. El compilador GNU C, que se conoce como GCC, es el compilador recomendado, pero otros compiladores también trabajan bien si son compatibles con ANSI-C.

4.4.5 Conseguir Apache *(00h)*

El software de Servidor Apache está disponible en el sitio Web del Grupo Apache y en decenas de sitios *mirror* de todo el mundo. Lo mejor es tratar de encontrar un sitio *mirror* que nos sea próximo geográficamente. Evidentemente, la proximidad geográfica no significa necesariamente que un sitio esté próximo en términos de conectividad de red, pero supone un buen punto de partida.

Los URL importantes son:

- La Fundación Apache Software, que está en http://www.apache.org/, tiene más de un proyecto bajo la protección de la ASF, aunque Servidor Apache es el más conocido.

- El Proyecto Apache HTTP Server, en http://httpd.apache.org/, este sitio es la fuente de información más exacta y actualizada que existe sobre Servidor Apache. Toda la documentación del servidor está disponible en línea, así como la base de datos de fallos, los archivos de noticias, la información histórica y otros tipos de recursos relacionados con Apache. La Figura 4.09 muestra el sitio Web Servidor Apache.

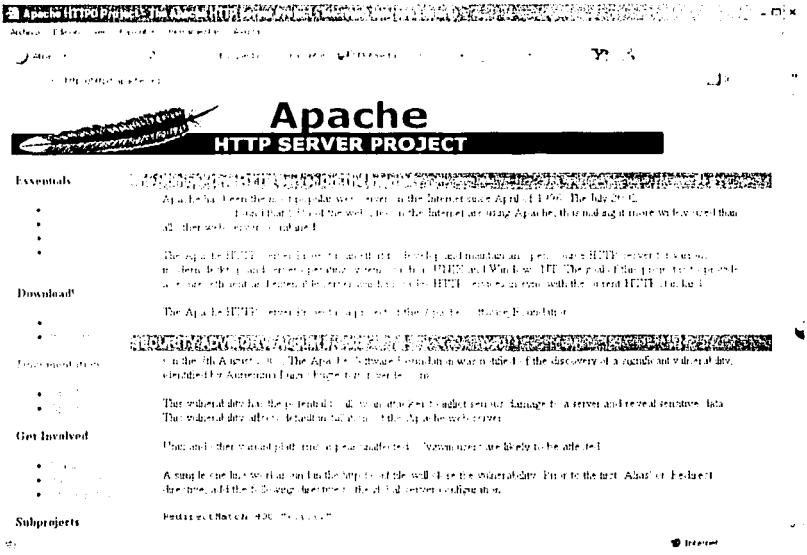


Figura 4.09 El sitio Web de Servidor Apache.

- Descargue Apache en http://www.apache.org/dist/, ésta es la ubicación principal para obtener el código fuente de Apache.

- Los *mirrors* del proyecto Apache en http://www.apache.org/mirrors/, este sitio enumera, por código de país, los sitios *mirror* oficiales de Apache.

Apache está disponible para su descarga en varias versiones, en código binario y en código fuente, en la página de descarga de Apache y en los distintos *mirrors*. Ciertas personas prefieren utilizar una versión más antigua del producto, porque con ello saben que el software que están usando está probado.

Hasta la fecha, la última versión de Apache para plataformas Unix es: la 1.3.9

Descargar la última versión siempre es lo más seguro, ya que el Grupo Apache Prueba el software antes de que se pueda descargar. Sin embargo, deberá leer la lista de fallos conocidos, para así poder estar al tanto de los temas problemáticos con el software y para evitar una versión que pueda tener un problema que le afecte directamente. Para ver los temas abiertos sobre una determinada versión, véase la página de información sobre fallos en http://www.apache.org/bug_report.html.

4.4.6 Instalación para usuarios impacientes del sistema Unix *(09/B)*

Si está impaciente y desea instalar Apache rápidamente, esto es lo que tiene que hacer: tendrá que estar conectado como *root* para ejecutar estos comandos satisfactoriamente:

```
cd apache_1.3.9
./configure --prefix=/usr/local/apache
make
make install
/usr/local/apache/bin/apachecti start
```

Puede cambiar el prefijo a otra cosa si desea instalarlo en otro lugar que no sea `/usr/local/apache`. Ésta es la ubicación predeterminada para instalar Apache.

Configurar Apache

Hay dos formas de configurar la construcción Apache. La forma más reciente, da APACI, le permite especificar opciones de línea de comandos. La forma antigua implica editar un archivo de configuración y seleccionar las opciones deseadas.

Compilar

Compilar es la parte más sencilla de todo el proceso. Cuando haya terminado con la fase de configuración, habiendo elegido el método deseado, escriba `make` para iniciar el proceso de integración. Esto puede durar varios minutos.*

Instalar

Instalar es casi tan fácil como compilar. Hay que escribir dos palabras: `make install`. Tendrá que estar conectado como *root* para ejecutar este comando, ya que la instalación está colocando archivos en directorios dónde la mayoría de los usuarios no tienen acceso de escritura.

4.4.7 Instalar y compilar Apache en sistemas Windows *(09H)*

La mayoría de usuarios de Microsoft Windows querrán instalar desde los binarios. Los binarios Windows están construidos de tal forma que todos los módulos disponibles se encuentran compilados y pueden activarse por medio de la directiva `Add-Module`.

Instalar Apache en Windows

Descargue el archivo de instalación en el sitio de descarga de Apache en http://www.apache.org/dist/. El nombre de archivo es ApacheW95ws2setup.exe, dónde la versión corresponde a la última que existente hoy en día que es la versión de Apache 2.0.39

La instalación es el proceso de instalación que se espera en Windows. Hay que hacer clic en Siguiente unas cuantas veces, y estará instalado.

Sugerencia

La ubicación predeterminada para instalar Apache en Windows es `C:\Archivos de programa\Apache`. Esto funciona, pero los espacios de las rutas de archivo pueden llevar a algunos problemas con la configuración. Por ejemplo, las rutas de archivos deberán aparecer entre comillas en el archivo de configuración. Olvidar estas comillas es una falta muy común. Puede evitar esto cambiando la instalación a `C:\httpd`, `C:\Apache2` o a alguna otra ruta que tenga sentido.

* En el improbable caso de que algo falle durante esta fase, verá varios mensajes de error que deberán apuntar al origen del problema. Si no sabe lo que significan estos mensajes de error, la mejor ayuda probablemente sea el grupo de noticias comp.infosystem.servers.unix o la base de datos de fallos del sitio Web de Apache.

Instalar como servicio de Windows XP

Si desea que Apache se ejecute todo el tiempo en su máquina Windows XP (por ejemplo, un servidor de producción), tendrá que instalar Apache como servicio de Windows XP. Estos servicios son, como su propio nombre lo indica, una característica de Windows XP, y no están disponibles en Windows 9x ya que no utilizan la tecnología NT.

Si no conoce bien los servicios de Windows XP, he aquí un breve resumen. Los servicios de Windows XP aseguran que la aplicación se inicia cuando el sistema se reinicia. Puede ser que las aplicaciones que no se instalen como servicios, se principien al iniciar sesión o de forma manual. Pero, si su servidor se reinicia por algún motivo, estas aplicaciones se reiniciarán. Cuando se instala Apache sobre Windows XP, Por defecto estará instalado como servicio. En dado caso de que no fuera así, es muy fácil convertirlo en un servicio. En el menú Inicio, Panel de Control y dentro de este mismo debe existir un icono llamado Rendimiento y mantenimiento, aquí debe existir otro icono que se llama Herramientas administrativas le damos doble clic y nos abrirá otra ventana en dónde seleccionaremos precisamente el icono que se llama servicios. La Figura 4.10 muestra como abrir el Panel de control en Windows XP.

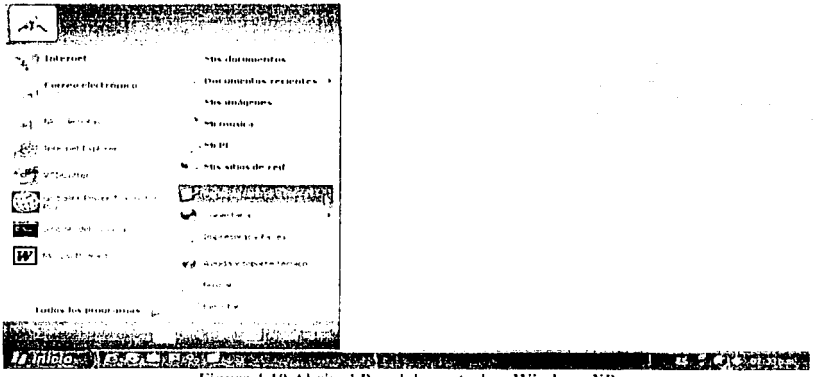


Figura 4.10 Abrir el Panel de control en Windows XP

La Figura 4.11 muestra la ventana que aparece al darle clic en el menú Panel de control en Windows XP.

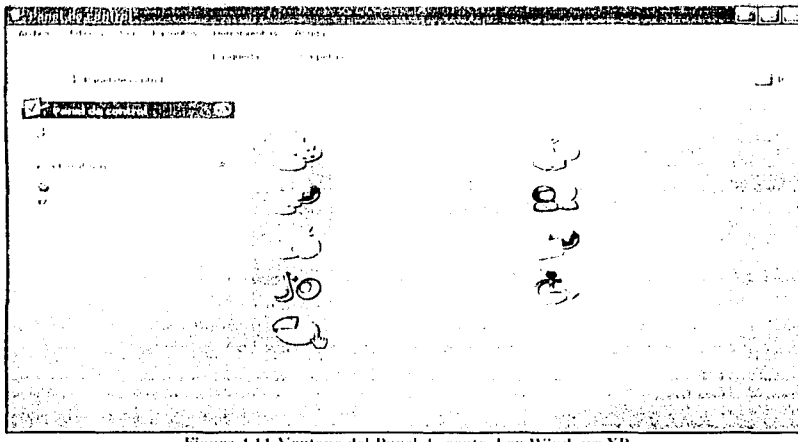


Figura 4.11 Ventana del Panel de control en Windows XP

La Figura 4.12 muestra la ventana que aparece al darle clic en el menú Rendimiento y mantenimiento en Windows XP.

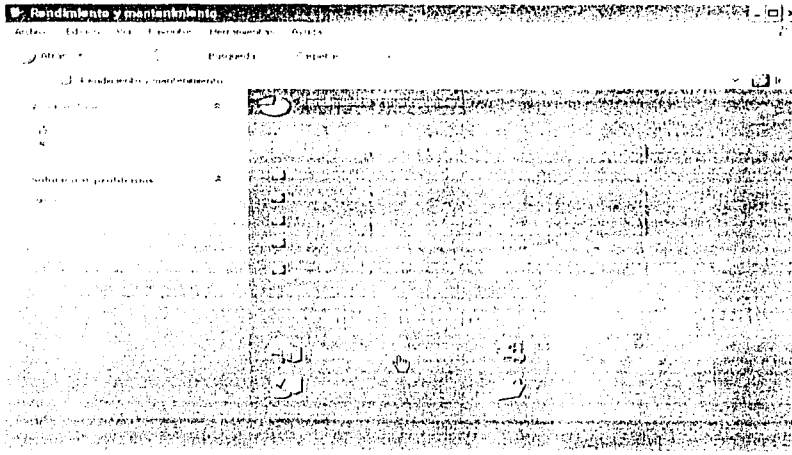


Figura 4.12 Opciones de Rendimiento (Herramientas Administrativas)

La figura 4.13 muestra la ventana en dónde elegiremos los servicios de Windows XP.

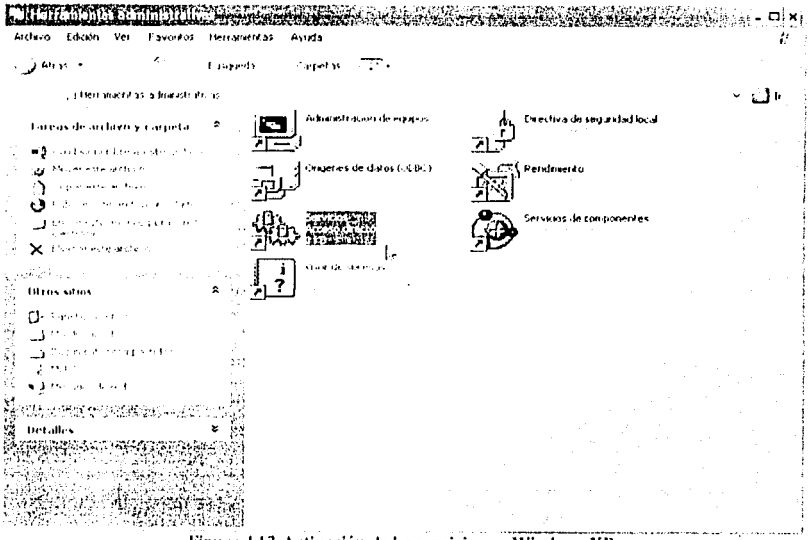


Figura 4.13 Activación de los servicios en Windows XP

4.4.8 Iniciar, detener y reiniciar el servidor *(09/11)*

Dependiendo de si está ejecutando Apache sobre Unix o Windows, las maneras de iniciar, detener y reiniciar el servidor son diferentes. Apache puede iniciarse manualmente en la línea de comandos o como parte del proceso de inicio del servidor. En Unix, el *script* del *shell* `apachectl` le da la posibilidad de iniciar, detener y reiniciar Apache en la línea de comandos.

Iniciar el servidor

Una vez que se instala y configura el servidor, ya se puede arrancar. Casi siempre se arranca corriendo el ejecutable en la línea de comandos.

En los sistemas operativos Unix y similares, Apache se suele abrir cuando se inicia la máquina y luego sigue ejecutándose mientras la máquina esté activada. El inicio se puede hacer manualmente en la línea de comandos o en un *script* de inicio.

En Windows XP, Apache se suele ejecutar como servicio de Windows XP; en otras versiones de Windows, se ejecuta como aplicación de consola.

Iniciar Apache en Unix

En Unix, puede iniciar Apache en la línea de comandos. También puede dejar que Apache se abra automáticamente al iniciar el sistema.

Iniciar en la línea de comandos en Unix

Puede iniciar Apache en la línea de comandos en una máquina Unix escribiendo el nombre del ejecutable `httpd`, con las opciones de línea de comandos que desee. El proceso del servidor Apache se inicia, cambia para ejecutarse como el usuario especificado en la directiva **User**.

Iniciar automáticamente en el tiempo de inicio

Todos los tipos de Unix proporcionan algún mecanismo para iniciar procesos de forma automática cuando arranca el sistema. Esto varía en los distintos tipos de Unix, y es necesario consultar la documentación para recibir instrucciones sobres cómo llevar a cabo esto en el sistema propio. El *script* `apachectl` puede resultar muy útil a la hora de proporcionar esta funcionalidad, ya que acepta `start` y `stop` como argumentos, lo que se espera en los *scripts* `/etc/rc.d` de los tipos de Unix que soporten ese mecanismo.

Detener o reiniciar en Unix

En sistemas operativos Unix y similares (como Linux), generalmente se detiene el servidor con el comando `kill`. Este comando es la forma que tiene Unix de enviar señales de terminación a un proceso y se puede enviar de varias formas.

Sin embargo, antes de enviar las señales de terminación, hay que saber a quién enviarlas. Si se comprueba la lista de procesos de la máquina Unix, se ve que hay más de un proceso `httpd` en ejecución. En una máquina Linux al dar el comando `ps ax`, listara los procesos y tiene este aspecto:

```
%ps ax
PID TTY STAT TIME COMMAND
.....
  1599 ?        S      0:00 /usr/sbin/dhcpd
  1740 ?        S      0:04 /usr/sbin/named
  4278 ?        S      0:00 smdb -D
  4287 ?        S      0:14 nmdb -D
13634 ?        S      0:00 /usr/local/apache/bin/httpd
16614 ?        S      0:00/usr/local/apache/bin/httpd
16615 ?        S      0:00/usr/local/apache/bin/httpd
16616 ?        S      0:00/usr/local/apache/bin/httpd
16617 ?        S      0:00/usr/local/apache/bin/httpd
16618 ?        S      0:00/usr/local/apache/bin/httpd
16620 ?        S      0:00/usr/local/apache/bin/httpd
16621 ?        S      0:00/usr/local/apache/bin/httpd
16629 ?        S      0:00/usr/local/apache/bin/httpd
16630 ?        S      0:00/usr/local/apache/bin/httpd
16631 ?        S      0:00/usr/local/apache/bin/httpd
11630 ?        S      0:00. /msql2d
13866 1        S      0:00 sh /usr/X11R6/bin/startx
26529 p0       S      0:00 -tcsh
26541 p0       R      0:00ps ax
```

En esta maquina muestra que hay 11 procesos `httpd` en ejecución. Uno es el proceso primario y el resto son los secundarios. Si terminamos cualquiera de los procesos secundarios, el proceso primario regenerará al secundario.

Detener Apache

Para detener Apache inmediatamente, emita el comando `kill -TERM` en el ID de proceso que se enumera en el archivo `httpd.pid`. Por ejemplo, se escribiría el siguiente comando en la línea de comandos:

```
kill -TERM 'cat /usr/local/apache/logs/httpd.pid'
```

Esto podría tardar unos cuantos segundos, ya que el proceso primario trata de matar a cada uno de sus secundarios, para luego matarse a sí mismo.

Reiniciar Apache

Existen dos formas de reiniciar el servidor Apache, dependiendo de lo rápido que se quiera ejecutar el reinicio:

- Para reiniciar inmediatamente, utilice una señal `HUP`. Por ejemplo, escribiría el siguiente comando:

```
kill -HUP 'cat /usr/local/apache/logs/httpd.pid'
```

- La señal `HUP` hace que el primario mate inmediatamente a todos los secundarios. Toda solicitud que atiendan estos secundarios terminarán. Luego vuelve a leer los archivos de configuración y vuelve abrir los archivos de registro. Luego regenera un nuevo conjunto de secundarios, que empiezan a atender solicitudes inmediatamente.

- Para reiniciar suavemente, utilice la señal `USR1`. Por ejemplo, escribiría el siguiente comando:

```
kill -USR1 'cat /usr/local/apache/logs/httpd.pid'
```

- La señal `USR1` le indica al primario que envíe una solicitud de terminación a cada secundario. Cada uno de los secundarios termina de atender la solicitud y luego se cierra. Si está inactivo, se cerrará inmediatamente. El primario vuelve a cargar los archivos de configuración y vuelve a abrir los archivos de registro. Cuando el secundario se cierra, se sustituye por una nueva generación de secundarios con la nueva configuración.

Probablemente sea mejor utilizar este último método en un servidor de producción, ya que no hará que las conexiones existentes se corten sin más; al contrario, terminará las transacciones que estén activadas en el momento del reinicio.

Iniciar Apache en Windows

En Windows, existen dos formas principales de iniciar el servidor Apache: como servicio Windows XP o con la nueva aplicación integrada que tiene Apache llamada Monitor Apache Servers.

Ejecutar Apache como servicio de Windows XP

Si está utilizando la versión Windows XP de Apache, probablemente le suene el concepto de los servicios Windows XP. En esencia, constituyen una forma de ejecutar programas en segundo plano en el instante en que se inicia Windows XP y de mantenerlos en ejecución mientras Windows XP esté activo. En su concepto, se parecen mucho a los demonios de Unix.

La ventaja principal de ejecutar Apache como servicio de Windows XP es que no hay que iniciar sesión en la máquina para comenzar el servicio al reiniciar la máquina, ya que ésta se inicia de forma automática. De este modo, cuando haya que reiniciar Windows XP, Apache se reiniciará cuando la máquina se active nuevamente.

Cuando se ejecuta como servicio, Apache se ejecuta en segundo plano, es decir, no hay icono de ventana o de barra de tareas, pero Apache aparece en la lista de procesos del Administrador de tareas.

Iniciar automáticamente al reinicio

El método más recomendado es que se instale Apache como servicio para iniciarse automáticamente al reinicio. Es así como se configura el servicio si se ha instalado desde el menú Inicio.
Para asegurarse de que el servicio está configurado para iniciarse automáticamente, abra el cuadro de diálogo Servicios del Panel de control de Windows. La Figura 4.14 muestra el cuadro de diálogo **Servicios**, con Apache configurado para que se inicie automáticamente.

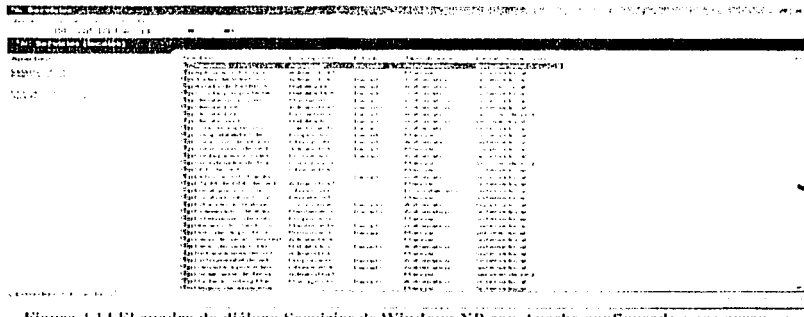


Figura 4.14 El cuadro de diálogo Servicios de Windows XP con Apache configurado para arrancar automáticamente.

Si no se ve la palabra automático en la columna tipo de inicio, que esta junto Apache en el cuadro de diálogo **Servicios**, seleccione Apache y haga doble clic en el mismo para ver las opciones adicionales del servicio véase la Figura 4.15. Seleccione en Tipo de inicio: Automático para que Apache se inicie automáticamente cuando se reinicia el sistema

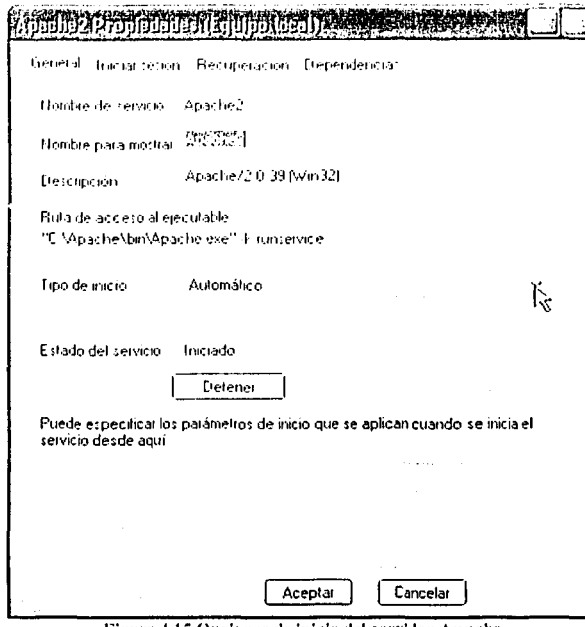


Figura 4.15 Opciones de inicio del servidor Apache

Iniciar el servidor con Monitor Apache Servers

Para iniciar de una manera sencilla, se utilizar la herramienta que viene con la versión 2.0.39 de Apache para Win32, que es el Monitor Apache Servers. Cuando se instala Apache sobre Windows XP, por defecto podremos encontrar el Monitor Apache Servers al darle en el menú de **Inicio**, elija Todos los programas, Apache HTTP Server 2.0.39, Control Apache Server y encontrara el Monitor Apache Servers. La Figura 4.16 muestra esto en el menú **Inicio**.

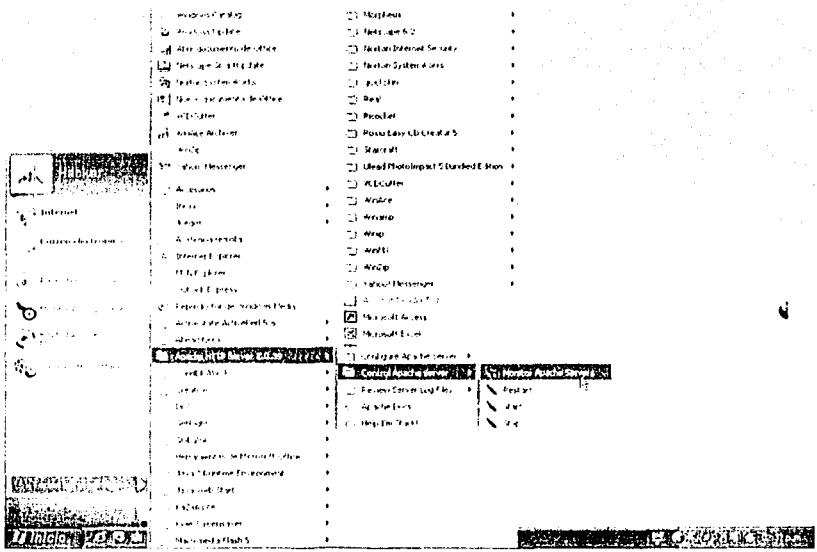


Figura 4.16 Abrir el Monitor Apache Servers

Cuando se selecciona éste elemento en el menú **Inicio**, se abrirá la ventana Apache Service Monitor en dónde indica el status de servicio de Apache que tiene en ese momento. Los status de servicio son dos: **iniciado o detenido.**

En la ventana Apache Service Monitor existen 8 botones o elementos que hacen diferentes funciones que son:

- **Ok:** Permite cerrar la ventana activa si es que el Servidor Apache se encuentra iniciado
- **Star**: Permite iniciar automáticamente al servidor Apache.
- **Stop:** Permite detener automáticamente al servidor Apache.
- **Restart:** Permite Reiniciar o restablecer automáticamente al servidor Apache.
- **Services:** Permite abrir automáticamente los servicios de Windows XP.
- **Connect:** Permite conectarse algún equipo remoto.
- **Disconnect:** Permite desconectarse del equipo remoto.
- **Exit:** Permite cerrar la ventana del Apache Service Monitor

La Figura 4.17 muestra la ventana de Apache Service Monitor.

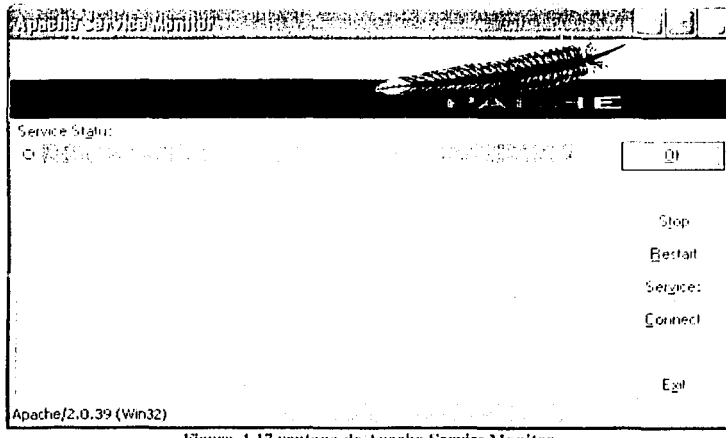


Figura 4.17 ventana de Apache Service Monitor

4.5 Puesta en marcha de un CGI *(04B) (08B)*

4.5.1 Pasos a seguir para poner en marcha un CGI *(04B) (08B)*

Aquí se explicarán los pasos a seguir para instalar un **CGI creado en Perl** (los Listados de la sección 4.3 nos servirán mucho porque son los que utilizaremos para poner en marcha los CGI).

Puntos que deben tomarse para poner en marcha un CGI en un servidor Web:

1. Debemos verificar que el servidor que utilizaremos tiene los **permisos para instalar CGI's** y, si es así, checar **en qué subdirectorio** debemos instalarlos.

2. Debemos ver **el path del intérprete Perl**. Es decir, en qué lugar del disco se encuentra el intérprete del lenguaje Perl. Ya que Perl no es un lenguaje compilado y no necesitamos un compilador, necesitamos lo que se conoce con el nombre de *intérprete*.

3. Debemos verificar **cuál es el path real de nuestro espacio Web** en el servidor. Usualmente suele ser algo como `/home/usuario` o `/usr/people/usuario` `/public_html`. Por ejemplo si tubieramos que nuestro path es `/home/usuarios_web/pepe`. Esto significa que nos corresponde en el disco un subdirectorio llamado `pepe` que cuelga de `usuarios_web`, que a su vez cuelga del subdirectorio `home`. Todos los path que empiezan por `/` son paths *absolutos*. En otras palabras, son paths descritos a partir de la raíz del disco (`/`). En UNIX/LINUX la raíz del disco es `/` (es el equivalente a `C:\ del DOS/WINDOWS`). El paths relativo es el path de un subdirectorio referido a otro, por ejemplo el path relativo del subdirectorio `/etc/X11` referido a `/usr/local/bin` sería `../../../etc/X11`.

4. Aunque no necesitemos más datos para instalar un CGI debemos buscar más cosas como: a) En dónde se encuentra el path del `sendmail`
   b) Y en dónde el del `date`.
   Posiblemente los necesitaremos si decidimos instalar más CGI's. Necesitaremos `sendmail` en aquellos CGI's que envíen correos. El programa `date` da la fecha y hora del sistema.

4.5.2 ¿Qué son los permisos? *(04B) (08B)*

Los controles de permisos de archivo pueden accesar archivos en los sistemas UNIX. Con mucha frecuencia, se escucha de programadores principiantes de CGI que intentan escribir archivos en un directorio en el que no tienen permiso de escritura. Los permisos en UNIX también se denominan *derechos*. UNIX puede controlar en diversas formas el acceso a archivos. Existen tres niveles de permisos para tres clases de usuarios. Para ver los permisos de un archivoen UNIX usemos el comando `ls` con la opción `-l` de la línea de comandos. Por ejemplo:

```
C:indyunix:~/public_html/pfind>ls -l

total 40
-rw-r--r--   1 dbewley  staff         139 Jun 18 14:14 home.html
-rwxr-xr-x   1 dbewley  staff        9145 Aug 14 07:06 pfind
drwxr-xr--   2 dbewley  staff         512 Aug 15 07:11 tmp
```

Cada línea de este listado indica una partida de directorio por separado. El primer caracter en la primera columna es por lo regular un guión o la letra d. Si una partida de directorio tiene una d, significa que se trata de un subdirectorio del directorio actual.
Los otros nueve caracteres son los permisos de archivo. Los permisos deben considerarse en grupos de tres, para las tres clases de usuarios. Dichas clases son:

- **Usuario** –el propietario del archivo o directorio. El nombre del propietario se exhibe en la tercera columna de la salida del comando ls.
- **Grupo** –el grupo que es propietario del archivo. Los archivos pueden tener propietarios tanto individuales como de grupo. Varios usuarios pueden pertenecer a un mismo grupo.
- **Otros** –cualquier usuario que no sea el propietario o pertenezca al grupo que es propietario del archivo.

Cada clase puede tener uno o más de los tres siguientes niveles de permisos:

- `r` –la clase puede leer el archivo o directorio.
- `w` –la clase puede escribir en el archivo o directorio.
- `x` –la clase puede ejecutar el archivo o listar el directorio.

	Propietario	Grupo	Todos
leer	400	40	4
escribir	200	20	2
ejecutar	100	10	1

Así los CGI deben tener prioridad 755 (400+200+100+40+10+4+1) y un archivo de estadísticas (que pueda ser consultado desde la página Web) debe tener prioridad 644 (400+200+40+4).

El permiso 755 (400+200+100+40+10+4+1) tiene una razón de ser. Por ejemplo, nosotros utilizaremos un CGI llamado Test.pl, que necesita asignarse el permiso 755. Esto se debe a que el propietario del CGI tiene todos los permisos estos son: el de leer=400, el de escribir=200 y el de ejecutar=100. En cambio el Grupo de usuarios sólo se le da el permiso de leer=40 y el de ejecutar=10: porque, si le damos el permiso de escritura, cualquier usuario del grupo puede modificar el CGI y con ello alterar su funcionamiento. De igual forma, a todos los demás usuarios se les asigna el permiso de leer=4 y ejecutar=1 por la misma razón que ya se explicó, que es la de proteger el CGI de ser modificado.

Ahora bién, a los subdirectorios y CGI's se les dan los permisos 705 y no 755, a los subdirectorios escribibles 707 y no 777 a los archivos HTML 604 y no 644, entre otros.

Estos permisos se sacan de la siguiente forma, por ejemplo:

- Para sacar el permiso 705 (400+200+100+4+1), en las propiedades del archivo se vería así:
  ```
  -rwx---r-x   1 alberto  staff       11816 May  9 09:19 test.pl
  ```
- Para el permiso 707 (400+200+100+4+2+1) en las propiedades del subdirectorio se vería así:
  ```
  -rwx---rwx   1 alberto  staff       11816 May  9 09:19 alberto
  ```
- Y para sacar el permiso 604 (400+200+4) en las propiedades del archivo HTML se vería así:
  ```
  -rw----r--   1 alberto  staff       11816 May  9 09:19 test.html
  ```

Si no está autorizado un permiso para el usuario que ejecutó el comando **ls**, su posición se cubre con un guión. Por ejemplo:

```
ls -l hform.html

-rwx------   1 alberto  staff       11816 May  9 09:19 slideshow.pl
```

Para este archivo, el propietario, **alberto**, tiene todos los derechos: lectura, escritura y ejecución. El grupo **staff** y cualquier otra persona no tienen ningún derecho.*

He aquí otro ejemplo:

```
ls -l pfind.pl

-rwxr-x--   1 alberto  staff       2863 Oct 10 1995  pfind.pl
```

* Los scripts de Perl no están compilados deben ser leídos por el intérprete de Perl cada vez que se ejecutan. Por lo tanto, los scripts de Perl, a diferencia de los programas compilados, deben tener permisos de ejecución y lectura.

Esta vez, tenemos acceso total mientras que el grupo staff puede leer y ejecutar el archivo. Los demás no tienen derechos en este archivo.
La mayoría de los archivos HTML tienen permisos como los siguientes:

```
ls -l schedule.html

-rw-r--r--   1 alberto  staff        2439 Feb  8 1996  schedule.html
```

Todos podemos leerlo, pero sólo el usuario puede modificarlo o eliminarlo. No hay necesidad de tener el permiso de ejecución ya que HTML no es un lenguaje ejecutable.

Podemos modificar los permisos en un archivo mediante el comando **chmod**. Este comando reconoce las tres clases de usuario como **u**, **g** y **o**, y los tres niveles de permisos como **r**,**w** y **x**. Concede y revoca permisos con un "**+**" o un "**-**" en conjunción con cada permiso que deseemos modificar. También aceptará una **a** para las tres clases de usuario a un tiempo.

La sintaxis del comando **chmod** es:

```
chmod <options> <file>
```

Los siguientes son algunos ejemplos del comando chmod en acción.

```
ls -l pfind.pl
-rw------   1 alberto  staff        2863 Oct 10 1995  pfind.pl

chmod u+x pfind.pl

ls -l pfind.pl
-rwx------   1 dbewley  staff        2863 Oct 10 1995  pfind.pl
```

El primer comando **ls** nos muestra los permisos de archivo originales. Luego, el comando **chmod** agrega el permiso de ejecución para el propietario (o usuario) del archivo **pfind.pl**. el segundo comando **ls** exhibe los permisos recién modificados.
Para agregar estos permisos a las clases de grupo y otros, usaremos **go+rx** como en el ejemplo siguiente. Recordemos que para ejecutar scripts de Perl, los usuarios deben tener por lo menos los permisos de lectura y ejecución.

```
ls -l pfind.pl
-rwx------   1 dbewley  staff        2863 Oct 10 1995  pfind.pl

chmod go+rx pfind.pl

ls -l pfind.pl
-rwxr-xr-x   1 dbewley  staff        2863 Oct 10 1995  pfind.pl
```

Ahora, cualquier usuario puede leer y ejecutar `pfind.pl`. digamos que se encontró un error grave en `pfind.pl` y no queremos que nadie lo ejecute. Para revocar el permiso de ejecución para todas las clases de usuarios, utilicemos la opción `-x` con el comando `chmod`.

```
ls -l pfind.pl
-rwxr-xr-x   1 dbewley  staff        2863 Oct 10 1995  pfind.pl

chmod a-x pfind.pl

ls -l pfind.pl
-rw-r--r--   1 dbewley  staff        2863 Oct 10 1995  pfind.pl
```

Ahora, todos los usuarios pueden leer `pfind.pl` pero ninguno puede ejecutarlo.

Por norma general debemos hacer tres cosas con el CGI antes de pasarlo al servidor:

1. La **primera línea** debe empezar por `#!` y en ella se indica el path del intérprete Perl. Así, si tenemos instalado el intérprete Perl en el siguiente path `/usr/bin/perl` debemos poner en esa línea lo siguiente `#!/usr/bin/perl`

2. Tras las explicaciones de instalación (o en medio de la explicación de cómo instalar el CGI), hay un apartado dedicado a dar el valor correcto a las variables necesarias. Las variables en Perl se pueden declarar de esta forma

   ```
   $nombre_de_la_variable = "valor_de_la_variable";
   ```

   Aunque las dobles comillas pueden ser en ocasiones comillas simples. Estas variables las podemos clasificar en dos tipos, variables de localización de programas de sistema y variables referentes a nuestro espacio Web. Algunos CGI's necesitan saber dónde está algún programa específico del sistema (por ejemplo `sendmail`, `date` o `ftp`). Eso debemos indicarlo en alguna variable que estará pensada para eso. Por ejemplo si al principio del programa hay algo como `$mailprog='/usr/bin/sendmail';` y nosotros sabemos que sendmail está en `/usr/local/bin/sendmail` entonces debemos modificar esa variable y poner `$mailprog='/usr/local/bin/sendmail';`

3. Luego encontraremos generalmente una serie de variables que se refieren a datos de nuestro espacio Web, por ejemplo donde está algún archivo de datos o bien nuestra dirección de correo. Lo más seguro para asegurar que el CGI funcione a la primera es que demos el path absoluto de los archivos (por ejemplo `$archivo datos="/home/pepe/misdatos/datos.dat"`) en vez de dar paths relativos. Esto es criticado por algunos, ya que es una fuente de información para todos aquellos que puedan ver el código del CGI. Lo que pasa es que si alguien ve el código del CGI es que ya hemos cometido errores de seguridad mucho más graves que ese. Un detalle más: si damos una dirección de correo en alguna variable, debemos hacerlo de la siguiente forma

   ```
   $mi_mail="pepe\@nosedonde.zz";
   ```

Es decir no debemos escribir @, si no escribir \@. Esto debe ser así para que el intérprete Perl no confunda `@nosedonde` con un arreglo (en Perl los arreglos se indican de esa forma).

Tras todos estos puntos, si es que modificamos nuestros propios CGI's debemos **guardarlos en formato ASCII.**

A veces es necesario modificar otros archivos **(plantillas, archivos de datos, etc.)**. Si es así, debemos guardar los cambios de la misma forma.

Para colocar los programas CGI y los demás archivos necesarios al servidor, lo más recomendable es pasarlos (mediante *FTP*). Hay que recordar dos cosas:

1. Debemos pasar el CGI por *FTP* en **modo ASCII.**

2. Y que si indicamos en las variables los directorios dónde debían estar el resto de los archivos debemos ponerlos en esos directorios y no en otros.

Una vez que el CGI está en el servidor, debemos darle permisos de ejecución. Tal vez para los usuarios de Windows sonará raro si vienen del mundo de UNIX. Si es así debe pasar a la sección que son los permisos. En ocasiones debes cambiar también los permisos de algún subdirectorio o archivo. Los permisos se asignan mediante un código de tres números o triadas. Lo más recomendable es que dejemos el segundo a 0 en toda nuestra Web, así nadie que tenga una cuenta en el mismo sistema podrá hackear nuestros archivos.

Por ejemplo debemos dar a los subdirectorios y CGI's permisos 705 y no 755, a los subdirectorios escribibles 707 y no 777 a los archivos HTML 604 y no 644, etc.

Por último debemos **modificar la página que llama al CGI.** Para llamar un CGI desde una página Web, podemos usar los comandos del *Server Side Includes (SSI)*

```
<!--#exec cmd="path y nombre de la rutina"-->
```

O bien

```
<!--#exec cgi="path y nombre de la rutina"-->
```

Simplemente debemos poner eso dentro del *body* de nuestra página Web. Esto es necesario en el caso de que el CGI deba ejecutarse sin intervención de la persona que lee la página (contadores, estadísticas, etc.). Si se hace de esta forma, entonces la página debe tener la extensión **shtml**. También podemos usar un formulario con la instrucción `<FORM ACTION="path y nombre de la rutina" METHOD="POST">`.

4.5.3 ¿Cómo llamar a un CGI desde una página Web? *(04B) (08B)*

Para incluir un CGI en nuestra página, debemos copiarla en nuestra cuenta mediante *ftp* a nuestro servidor. Por regla general los CGI deben tener la extensión `cgi`, pero como los escribimos en Perl llevaran una extención pl. Una vez que los tengamos en nuestra cuenta (y los hayamos modificado convenientemente para que se adapten a nuestras necesidades), debemos darles una serie de permisos. Los permisos de los archivos en UNIX se cambian con la instrucción `chmod`.

Para llamar un CGI desde una página Web, podemos usar el comando `<!--#exec cmd="path y nombre de la rutina"-->` o bien `<!--#exec cgi="path y nombre de la rutina"-->`. Si se hace de esta forma, entonces la página debe tener la extensión **shtml**. También podemos usar un formulario con la instrucción `<FORM ACTION="path y nombre de la rutina">`. Es conveniente que tengamos nuestro CGI en un subdirectorio aparte.

4.5.4 ¿Por qué no funcionan los CGI's en mi PC? *(04B) (08B)*

No vale la pena intentarlo, si no tenemos un servidor Web, no podremos ejecutar localmente los CGI's desde nuestro navegador. Lo más que podremos realizar será instalar el lenguaje en el cual están programados y ejecutarlos como programas normales, pero jamás desde el navegador.

4.5.5 ¿Cómo instalar CGI's en un Windows XP? *(04B) (08B)*

Ante todo debemos asegurarnos de que en el servidor Web esté instalado el compilador del lenguaje que vamos a utilizar para crear los programas. En nuestro caso, el que utilizaremos es Perl 5, y por lo general se instala en "`c:\perl`".

Después, el servidor Web debe estar configurado de forma correcta para que los clientes puedan ejecutar los CGI's. En este caso nos referiremos al sistema Windows XP, y al servidor Apache 2.0.39. En el servidor Apache utilizaremos principalmente dos directorios que son:

- Sitio Web, donde aparecerán las páginas que creamos (`c:\Apache\htdocs\`)
- Directorio `cgi-bin` (`c:\Apache\cgi-bin\`), que es donde debemos dejar nuestros programas CGI.

Ahora veamos cómo instalar nuestro primer programa CGI y para ello utilizaremos los Listados 4.14 Formulario HTML de adición de registros al Libro de Visitas y el Listado 4.15 Un programa CGI para agregar un registro en el Libro de Visitas y exhibir una página HTML que utilizamos en la sección 4.3 Creación de CGI'S.

Primeramente lo que debemos hacer es, poner nuestra página que genera el Listado 4.14 en el directorio `htdocs` del Servidor Apache ya que este lugar es especial para ubicar los

archivos html. Por ejemplo si nuestra página que generamos se llama `Librodevisitas.html` entonces quedaría ubicada en la siguiente ruta: `"c:\Apache\htdocs\Librodevisitas.html"`

Listado 4.14 El formulario HTML de adición de registros al Libro de Visitas

```
<HTML>
<HEAD><TITLE>Libro de visitas</TITLE></HEAD>
<BODY>
<CENTER><H1> Libro de visitas </H1></CENTER>
Rellena los espacios en blanco del formulario. Para poder añadir
tus comentarios debes rellenar, al menos, los campos
correspondientes a tu nombre y los comentarios. Gracias.
<HR>
<FORM METHOD=POST ACTION="/cgi-bin/libro.pl">
<TABLE BORDER=0 CELLPADDING=10>
<TR>
<TD>Tu Nombre:</TD>
<TD><INPUT TYPE=text NAME=name SIZE=30></TD>
</TR>
<TR>
<TD>E-mail:</TD>
<TD><INPUT TYPE=text NAME=email SIZE=40></TD>
</TR>
<TR>
<TD VALIGN=top> Comentarios:</TD>
<TD><TEXTAREA NAME=comentarios COLS=60 ROWS=4></TEXTAREA></TD>
</TR>
</TABLE>
<INPUT TYPE=submit VALUE="Firmar"><INPUT TYPE=reset>
</FORM>
</BODY>
</HTML>
```

El segundo punto que necesitamos realizar es ubicar el archivo que genera el Listado 4.15 En el directorio `cgi-bin` del Servidor Apache puesto que este directorio es especial para ubicar los archivos CGI's con extensión `.pl` e incluso `.cgi`. Por ejemplo si el archivo que generamos se llamara `libro.pl`, entonces quedaría ubicado en la siguiente ruta: `"c:\Apache\cgi-bin\libro.pl"`

Hay algo que no debemos olvidar en el Listado 4.15 utilizamos un archivo para guardar los datos comentados en el Libro de Visitas, y la ruta que tiene es: `"c:\\perl\\datos\\coment.dat"`; en la cual debe realmente existir ese archivo para que el programa no tenga problemas. Ese archivo no tiene nada en particular es como un archivo de texto (`.txt`) sin ningún campo aún, pero, en este caso tiene extensión `.dat`

Listado 4.15 Un programa CGI para agregar un registro en el Libro de Visitas y exhibir una página HTML.

```
#!c:\Perl\bin\Perl.exe

use strict;
    my(%campos);
    my($seg,    $min,    $hora,    $mdia,    $mes,    $ano)    =
(localtime(time))[0..5];
    my($archDatos) = "c:\\perl\\datos\\coment.dat";
    $mes  = zeroFill($mes, 2);
    $hora = zeroFill($hora, 2);
    $min  = zeroFill($min, 2);
    $seg  = zeroFill($seg, 2);
    $campos{'crontiempo'} = "$mes/$mdia/$ano, $hora:$min:seg";
    OptDatosForm(\%campos);
    SalvarDatosForm(\%campos, $archDatos);
    print("Content-type: text/html\n\n");
    print("<HTML>\n");
    print("<HEAD><TITLE>Libro de Visitas</TITLE></HEAD>\n");
    print("<H1><CENTER>Libro de Visitas</CENTER></H1>\n");
    print("<HR>\n");
    LeerDatosForm($archDatos);
    print("</BODY>\n");
    print("</HTML>\n");

sub OptDatosForm {
    my($hashRef) = shift;
    my($buffer) = "";
    if ($ENV{'REQUEST_METHOD'} eq "GET") {
        $buffer = $ENV{'QUERY_STRING'};
    }
    else {
        read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
    }
    foreach (split(/&/, $buffer)) {
        my($llave, $valor) = split(/=/, $_);
        $llave   = decifrarURL($llave);
        $valor = decifrarURL($valor);
        $valor =~ s/(<P>\s*)+/<P>/g;
        $valor =~ s/</&lt;/g;
        $valor =~ s/>/&gt;/g;
        $valor =~ s/&lt;b&gt;/<b>/ig;
        $valor =~ s!&lt;/b&gt;!</b>!ig;
        $valor =~ s/&lt;i&gt;/<b>/ig;
        $valor =~ s!&lt;/i&gt;!</b>!ig;
        $valor =~ s!\cM!!g;
        $valor =~ s!\n\n!<P>!g;
        $valor =~ s!\n! !g;
        %{$hashRef}->{$llave} = $valor;
    }
    $campos{'comentarios'} =~ s!\cM!!g;
    $campos{'comentarios'} =~ s!\n\n!<P>!g;
    $campos{'comentarios'} =~ s!\n!<BR>!g;
}
```

```
sub decifrarURL {
    $_ = shift;
    tr/+/ /;
    s/%(..)/pack('c', hex($1))/eg;
    return($_);
}
sub zeroFill {
    my($temp) = shift;
    my($len)  = shift;
    my($diff) = $len - length($temp);
    return($temp) if $diff <= 0;
    return(('0' x $diff) . $temp);
}
sub SalvarDatosForm {
    my($hashRef) = shift;
    my($archivo)     = shift;
    open(ARCH, ">>$archivo") or die("No se puede abrir el archivo
del Libro de Visitas.");
    print ARCH ("$hashRef->{'crontiempo'}~");
    print ARCH ("$hashRef->{'name'}~");
    print ARCH ("$hashRef->{'email'}~");
    print ARCH ("$hashRef->{'comentarios'}");
    print ARCH ("\n");
    close(ARCH);
}
sub LeerDatosForm {
    my($archivo)     = shift;
    open(ARCH, "<$archivo") or die("No se puede abrir el archivo
del Libro de Visitas.");
    while (<ARCH>) {
        my($crontiempo, $nombre, $email, $comentarios) = split(/~/,
$_);

        print("$crontiempo:              <B>$nombre</B>              <A
HREF=mailto:$email>$email</A>\n");
        print("<OL><I>$comentarios</I></OL>\n");
        print("<HR>\n");
    }
    close(ARCH);
}
```

Al realizar éstos dos pasos, sólo nos resta llamar la página que creamos en cualquier navegador. Por ejemplo: podemos abrir el Internet Explorer y en el campo Dirección ponemos lo siguiente `http://127.0.0.1/Librodevisitas.html`. Al realizar esto, lo que estamos haciendo es llamar a nuestra propia página de Libro de Visitas directamente desde nuestro Servidor Apache, para poder iniciar la puesta en marcha de nuestro primer CGI.

La Figura 4.18 muestra la llamada de nuestra página al Servidor Apache.

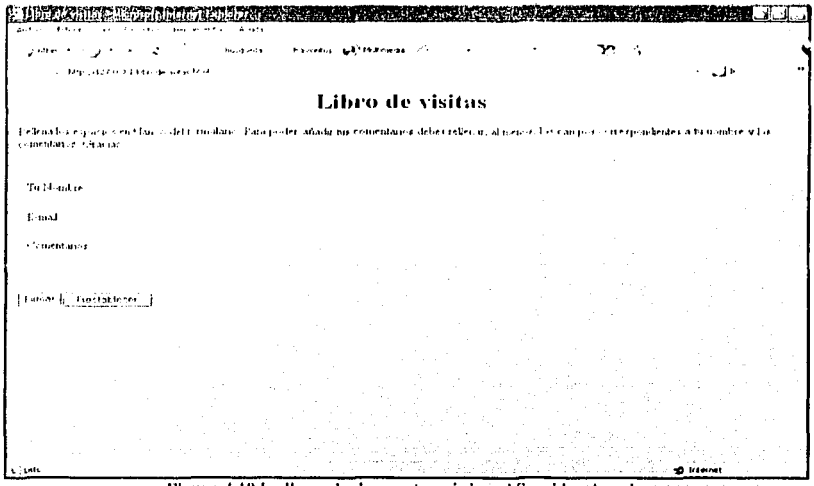


Figura 4.18 La llamada de nuestra página al Servidor Apache

Después de llamar esta página lo que nos resta es llenar los campos de:

- Tu nombre:
- E-mail:
- Comentarios:

Y firmar para invocar a nuestro CGI `libro.pl`, al ponerlo en marcha nos debe generar dos cosas: la primera es una página html con el resultado de nuestros comentarios y la segunda es el archivo de datos `coment.dat`

La Figura 4.19 nos muestra el resultado de poner en marcha al CGI (`libro.pl`)

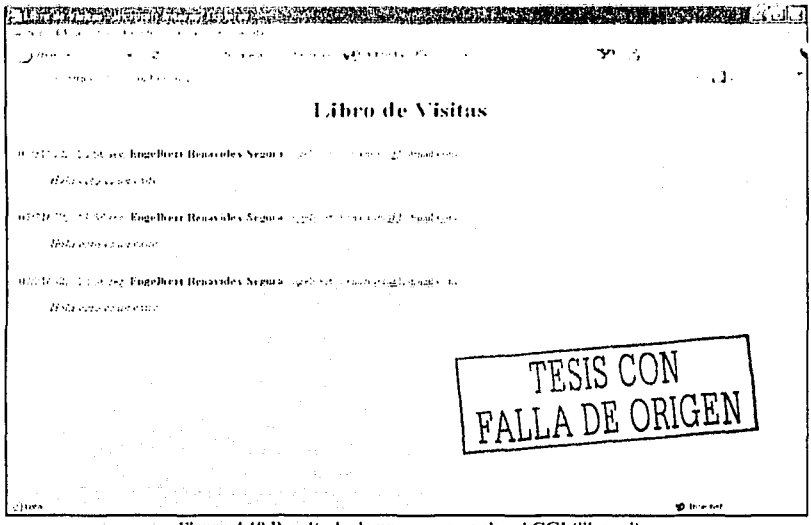


Figura 4.19 Resultado de poner en marcha al CGI (libro.pl)

Sólo hay que hacer un pequeño recordatorio de Perl. No debemos olvidar que la primera línea de todos los programas debe ser:

```
#!c:\perl\bin\perl.exe
```

Ahora veremos otro ejemplo para que quede completamente claro cómo poner en marcha los CGI's

Como vimos lo primero es checar:

- Que el servidor Web esté instalado incluyendo el compilador del lenguaje que vamos a utilizar para crear los programas.
- Configurar el servidor Web para que podamos ejecutar los CGI's.
- Debemos poner las páginas Web creadas en el directorio (`c:\Apache\htdocs\`)
- Debemos poner los CGI's creados en el directorio `cgi-bin` (`c:\Apache\cgi-bin\`)

Después de ver nuevamente los puntos importantes utilizaremos un nuevo From que se presenta en el Listado 4.16. Este From que manejaremos será una página para alojar Noticias y Avisos que puedan ser importantes para nuestro sitio Web.

Y por supuesto necesitaremos crear el CGI que permita publicar dichas Noticias y Avisos, el Listado 4.17 Muestra el CGI que utilizaremos para este otro ejemplo.

Primeramente lo que debemos hacer es, poner nuestra página que genera el Listado 4.16 en el directorio `htdocs` del Servidor Apache ya que este lugar es especial para ubicar los archivos html. Por ejemplo, si nuestra página que generamos se llama `tablon.html`, entonces quedaría ubicada en la siguiente ruta: **"`c:\Apache\htdocs\tablon.html`"**.
Ahora bien en este ejemplo utilizaremos algo nuevo que es el utilizar un password para poder introducir las nuevas noticias si el password no es correcto no podremos agregar ninguna noticia a nuestro Sitio Web.

Listado 4.16 El formulario HTML de adición de Noticias y Avisos

```
<HTML>
<HEAD>
<TITLE>Ejemplo de noticias</TITLE>
<BODY BGCOLOR="#FFFFFF">
<font face='Arial'>
<h2>Introduccion de noticias y avisos</h2>
Introduce las noticias o avisos que consideres pertinentes. <br>
Gracias.<br>
El password en este ejemplo es <b>noticias</b><br>
Si tras introducir la noticia, esta no aparece pulsa la opcion
<I>reload</I> de tu navegador.
<hr>
<center>
<FORM NAME=forma_01 METHOD=post ACTION='/cgi-bin/noticias.pl'>
<table border=0>
<tr><td align=left><b>Password: </B></td>
<td      align=left><input      type="password"      name="password"
size=10></td></tr>
<tr><td align=left><b>Titulo: </B></td>
<td align=left><input type="text" name="titulo" size=50></td></tr>
<tr><td align=left colspan=2><b>Noticia o aviso:</B><br>
      <textarea name=noticia COLS=60 ROWS=6></textarea></td></tr>
      <tr><td align=center colspan=2>
      <input    type=submit    value="Enviar">    <input    type=reset
value="Cancelar"></td></tr>
</table>
</form>
</center>
</DOBY></HTML>
```

El segundo punto que necesitamos realizar es ubicar el archivo que genera el Listado 4.17 En el directorio `cgi-bin` del Servidor Apache puesto que este directorio es especial para ubicar los archivos CGI's con extensión `.pl` e incluso `.cgi`. Por ejemplo si el archivo que generamos se llamara `noticias.pl`, entonces quedaría ubicado en la siguiente ruta: **"`c:\Apache\cgi-bin\noticias.pl`"**

Listado 4.17 Un programa CGI para agregar un registro en el Tablón de Noticias

```
#!c:\Perl\bin\Perl.exe

# Variables

$si_html="1";
$url_noticias = "http://127.0.0.1/noticias.html";
$path_noticias = "c:\\Apache\\htdocs\\noticias.html";
$horario=-21600;
$password ="noticias";
$nopassword= "0";
$num_max= "20";

# Damos la fecha. La intruccion gmtime da el horario de
# Greenwich, de ahi que le sumemos (o restemos) $horario
@meses=("Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio","A
gosto","Septiembre","Octubre","Noviembre","Diciembre");
($sec,$min,$hour,$mday,$mon,$year,$wday,$yday)
=gmtime(time+$horario);
$year=$year+1900;
$date = "$mday de $meses[$mon] de $year a las $hour:$min";
# Tomamos la entrada
read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
# Gestion de entradas
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
   $value =~ s/<!--(.|\n)*-->//g;
   if ($si_html ne "1"){
        $value =~ s/<([^>]|\n)*>//g;
   }
   $FORM{$name} = $value;
}
# Control de errores de entrada
#
&error(2) unless $FORM{'titulo'};
&error(3) unless $FORM{'noticia'};
if ($nopassword eq "1") {
        &error(4) unless $FORM{'password'};
        if ($password ne $FORM{'password'}) {&error(4);}
}
# Leemos el archivo HTML de noticias
&error(1) unless open (FILE,"$path_noticias");
flock(FILE,2);
@LINES=<FILE>;
flock(FILE,8);
close(FILE);
$SIZE=@LINES;
# Abrimos para guardar la noticia
&error(1) unless open (NOTIC,">$path_noticias");
flock(NOTIC,2);
seek(NOTIC, 0, 2);
$n=0;
```

```
for ($i=0;$i<=$SIZE;$i++) {
   $_=$LINES[$i];
   if (/<!--begin-->/) {
      print NOTIC "<!--begin-->\n";
      print                    NOTIC                    "<b>$date<br><font
color='#FF0000'>$FORM{'titulo'}</font></b><br><br>\n";
      print NOTIC "$FORM{'noticia'}<br>\n";
      print NOTIC "<hr>\n";
      $n=1;
   }
   else {

      if (($n eq $num_max)|(/<address>/)) { last; }
      print NOTIC $_;

      if (/<hr>/) {$n++;}
   }
}
print NOTIC "</address></font>";
print NOTIC "</body></html>";
flock(NOTIC,8);
close (NOTIC);
print "Location: $url_noticias\n\n";
exit;
# Subrutinas Control de errores

sub error {
   $tipo=@_[0];
   @er=("","No he podido abrir el archivo","Falta el t&iacute;tulo
de la noticia","Falta la noticia","Password incorrecto");
   print "Content-type: text/html\n\n";
   print "<html><head><title>Error</title></head>\n";
   print "<body bgcolor='#FFFFFF'><font face='Arial'>";
   print "<h2>Error</h2>\n";
   print "$er[$tipo]. Por favor vuelva a intentarlo.\n";
   print "</body></html>";
   exit;
}
```

Al realizar éstos dos pasos, sólo nos resta llamar la página que creamos en cualquier navegador. Por ejemplo: podemos abrir el Internet Explorer y en el campo Dirección ponemos lo siguiente `http://127.0.0.1/tablon.html`. Al realizar ésto, lo que estamos haciendo es llamar nuestra propia página de Ejemplo de Noticias directamente desde nuestro Servidor Apache, para poder iniciar la puesta en marcha de nuestro primer CGI.

La Figura 4.20 muestra la llamada de nuestra página al Servidor Apache.

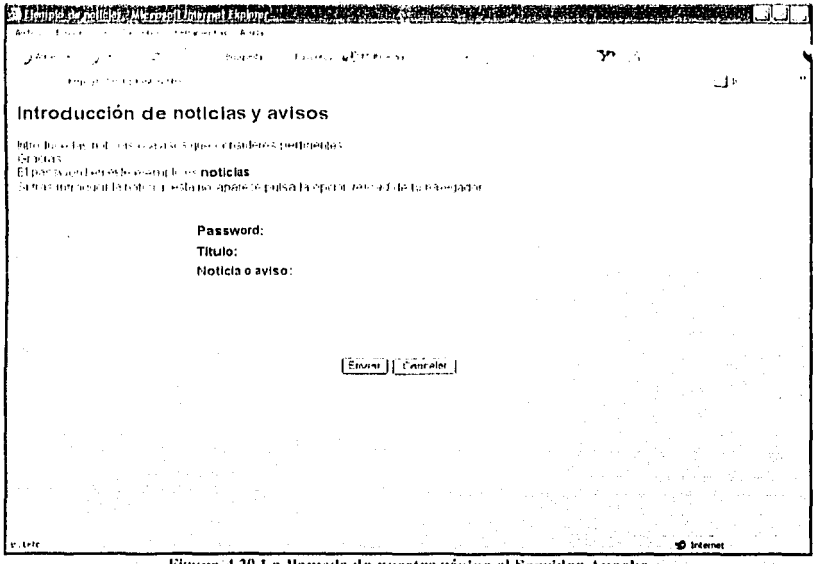


Figura 4.20 La llamada de nuestra página al Servidor Apache

Después de llamar esta página lo que nos resta es llenar los campos de:

- Password:
- Titulo:
- Noticia o aviso:

Y enviar para invocar a nuestro CGI `noticias.pl`, al ponerlo en marcha nos debe generar una página html con el resultado de nuestras noticias y avisos.

La Figura 4.21 nos muestra el resultado de poner en marcha al CGI (`noticias.pl`)

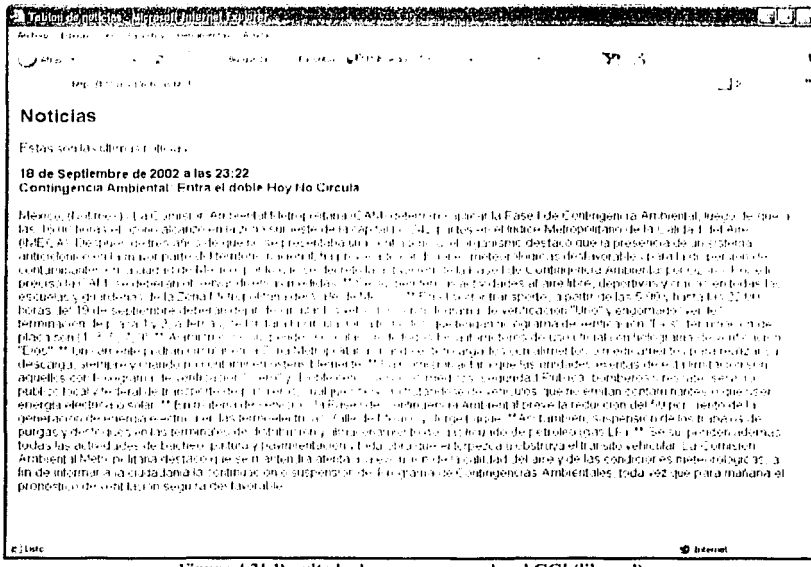


Figura 4.21 Resultado de poner en marcha al CGI (libro.pl)

Una vez realizado todo esto, ya está todo listo para activar tres ventanitas del navegador, un WordPad, comenzar a programar y a volcar nuestras creaciones sin problemas.

No hay que olvidar, que la primera línea de todos los programas debe ser:

```
#!c:\perl\bin\perl.exe
```

Esto, ayudará a nuestros programas a encontrar el intérprete Perl, claro esto es para utilizarlo en sistemas con Windows XP.

CONCLUSIONES

La Lic.en Matemáticas Aplicadas y Computación ha hecho que mi capacidad de análisis sea más amplia; y algunas de las materias que integran la carrera dio las bases y el apoyo para el desarrollo de esta Tesina.

Tales materias son; por mencionar algunas, Programación Avanzada, la cual desarrollo la inquietud de profundizar más los conocimientos en el lenguaje Perl, así como la realización de páginas Web dinámicas. Análisis de Algoritmos, es una materia que abarca los diferentes algoritmos o lógicas que se pueden utilizar al programar en cualquier lenguaje, ayudándonos así a identificar que algoritmo puede ser mas rápido o mas sencillo para resolver cualquier problema y finalmente Programación Básica que da un panorama inicial de que significa programación y la forma en que nos ayuda para resolver problemas.

Ahora bien este estudio será útil para todo aquel que comience a desarrollar aplicaciones o páginas Web dinámicas, y para todo el que quiera adentrarse un poco en lo que es programación utilizando el Interprete Perl.

La utilización de Internet, y su aplicación más importante, el Web, ha tenido una divulgación importante en el medio actual. La publicación de la información en sus diversas maneras ha tomado otra forma por medio de la tecnología. La utilización del CGI propone una forma adecuada, práctica y atractiva para darle a los sitios Web una mayor interacción con el usuario sin agregar costos adicionales.

Además con lo explicado en este documento ya conocemos las bases de la creación de CGI's y lo que podemos lograr con ellos. Si queremos aprender más y hacer algún CGI complejo, el siguiente paso a seguir, es elegir un lenguaje (Perl o C son dos buenas elecciones) y leer un tutorial o un libro dedicado específicamente a la creación de CGI's con ese lenguaje. Además es muy aconsejable leer el código de varios CGI's hechos por expertos, especialmente para aprender las técnicas más comunes, así como aspectos de seguridad en CGI's.

Al comenzar este trabajo solo existían los CGI's, en la actualidad existen ya diversas herramientas para realizar páginas interactivas. Las herramientas a las que me refiero son:

- **Active Server Page (ASP).**
- **Flash**
- **JavaScript**
- **PHP**
- **VBScript**

Aunque en la actualidad existen diversos debates entre los Perl lologos y los IIS lologos sobre cual es la herramienta mas fácil y segura para poder desarrollar sitios Web dinámicos, en lo personal gracias a que he podido comparar estas herramientas me atrevo a asegurar que la herramienta mas segura y fácil de utilizar son los CGI's y en varios años más los seguiremos utilizando.

Glosario de Términos

"A"

Acceso: Término utilizado en la WWW para representar un pedido del navegador para utilizar o ver un archivo dentro de un servidor web. Generalemente, los sitios que tienen contadores, muestran solamente los accesos a sus páginas principales.

ACPI: Advanced Configuration and Power Interfase. Esta especificación permite a Windows controlar la cantidad de energía otorgada a cada periférico, permitiendo encenderlo y apagarlo de ser necesario.

ActiveX: Tecnología desarrollada por Microsoft con el fin de elaborar aplicaciones exportables a la red, las cuales deben ser capaces de operar sobre cualquier plataforma a través de navegadores WWW, de forma que le da dinamismo a las páginas web.

ADSL: Asymmetrical Digital Subscriber Line (Línea de Subscripción Asimétrica Digital) Tecnología de compresión que permite a los hilos telefónicos de cobre convencionales transportar hasta 6 Mbps (megabits por segundo).

AGP: Accelerated Graphics Port. BUS y puerto de alta velocidad para video. Comunica directamente la placa de video con la memoria del sistema para acelerar la transferencia de datos.

Algoritmo: En programación, porción de código del programa que resuelve o ejecuta funciones específicas para la resolución de un problema o un proceso.

ANSI: American National Standard Institute. Instituto Nacional Americano de Estándar.

Ancho de banda: (Bandwidth). Cantidad de información que puede enviarse a través de una conexión. Usualmente, se mide en bits por segundo.

Antivirus: Programa cuya finalidad es prevenir las infecciones producidas por los virus informáticos así como curar las ya producidas. Para que sean realmente efectivos, dada la gran cantidad de virus que se crean continuamente, estos programas deben actualizarse periódicamente

API: Application Program Interface (Interfaz para programas de aplicación) (Conjunto de convenciones de programación que definen como se invoca un servicio desde un programa.

APM: Advanced Power Management. Función del BIOS y de algunos sistemas operativos con la cual se administra el ahorro de energía de algunos dispositivos. Función más antigua que ACPI

Applet: Se llama applet a cualquier programa pequeño hecho en Java que puede referenciarse en una página HTML. Los applet difieren de los programas hechos específicamente para Java en que no serán autorizados a acceder ciertos recursos de la PC local, como archivos y dispositivos de hardware, y no puede comunicarse con otras computadoras conectadas a una red local.

ARPANet: (Advanced Research Projects Agency Network o Red Precursora de la Internet). Fue desarrollada a fines de la década del 60 y principios de los años 70 por el Departamento de Defensa de los Estados Unidos como un prototipo de red extensa.(Ver también: Internet).

@ (arroba): Signo que forma parte de las direcciones de correo electrónico de forma que separa el nombre del usuario de los nombres de dominio del servidor de correo. Su uso en Internet se origina en su frecuente empleo como abreviatura de la preposición inglesa at (en).

ASCII: (American Standard Code For Information Interchange o Código numérico estándar).Utilizado por las computadoras para representar todas las letras mayúsculas y minúsculas del alfabeto, así como también números y signos de puntuación. Existen 128 códigos ASCII, los cuales pueden ser representados mediante números binarios del 0000000 al 1111111.

ASP: Active Server Page. Página de Servidor Activo. Tipo especial de página HTML la cual contiene pequeños programas (también llamados scripts) los cuales son ejecutados en servidores Microsoft Internet Information Server antes de ser enviados al usuario para su visualización en forma de página HTML. Habitualmente esos programas realizan consultas a bases de datos y los resultados de esas consultas determinan la información que se envía a cada usuario específico. Los archivos de este tipo llevan la extensión asp.

ATM: Asynchronous Transfer Mode. Modo de Transferencia Aminoran) Estándar que define la conmutación de paquetes ("celas" o celdas) de tamaño fijo con alta carga, alta velocidad (entre 1.544 Mbps. y 1.2 Gbps) y asignación dinámica de ancho de banda. ATM es conocido también como "paquete rápido"

"B"

Backbone: (Eje central, columna vertebral) Nivel más alto en una red jerárquica. Las redes aisladas ("stub") y de tránsito ("transit") conectadas al mismo eje central están interconectadas.

Banner: Gráfico o imagen, generalmente de forma rectangular inserto en una página web. En general se utiliza para publicidad

BBS: Bulletin Board System. (Tablón de Anuncios Electrónico) PC y programas que habitualmente suministran servicios de mensajería electrónica, archivos de datos y cualquier otro servicio y actividad que pueda interesar al operador del BBS.

Benchmark: Programa especialmente diseñado para evaluar el rendimiento de un sistema.

BIOS: Basic Input Output System. Sistema básico de entrada/salida. Programa residente normalmente en EPROM que controla instrucciones básicas entre el Hardware y el Software.

Bit: (Binary Digit o Dígito Binario). Es un dígito en base 2, es decir, 0 ó 1.Un bit es la unidad más pequeña de información que la computadora es capaz de manejar. El ancho de banda se suele medir en bits por segundo. (Ver también: Ancho de banda, Byte, kilobyte, megabyte).

BITNET: (Because It's Time Network) La implantaron en 1981 las universidades de la Ciudad de Nueva York y la de Yale, con idea de conectar no sólo los departamentos de informática sino todos aquellos que lo desearan, dentro de Estados Unidos. En Europa se creó una red equivalente denominada EARN (European Academic Research Network). El protocolo utilizado, que fué donado por la empresa IBM (International Business Machines), era el NJE (Network Job Entry), muy diferente de cualquier otro. Transmisión de tarjetas perforadas de ochenta columnas. La Universidad que quiera unirse, debe alquilar una línea con otra universidad (y pagar el coste asociado), comprometerse a admitir la conexión por parte de alguna otra universidad en el futuro y a retransmitir gratis todo el tráfico de paso que le llegue. El único coste es la línea alquilada, dado que no hay ningún cargo por el tráfico generado (algo poco habitual en comunicaciones). El servicio básico es la transferencia de archivos y el correo electrónico. Cada archivo que entra en el sistema contiene su destino final, pudiendo seguir un camino cualquiera.

Bookmark: (marca) Señal o recordatorio que los internautas dejan en su aplicación de navegación para marcar un lugar interesante encontrado en la red Internet a fin de poder volver a él posteriormente. (Ver también Browser)

Bps: (Bits per Second). Medida que representa la rapidez con que la información es transferida de una computadora a otra. Un módem de 28.8 Kbps. Es capaz de transferir 28.800 bits por segundo.

Browser: Programa utilizado para acceder y recorrer sitios de la WWW. (Ver también: WWW, Mosaic).

BUS: Elemento integrado a la tarjeta madre que es usado por todos los componentes y el procesador para comunicarse entre sí. El rendimiento global del sistema depende de su velocidad (En MHZ), ya que a través de él viaja toda la información entre los componentes de la PC.

Buffer: Área de la memoria que se utiliza para almacenar datos temporariamente.

Bug: Error de programación que genera fallas en las operaciones de una computadora.

Byte: Unidad de medida de la cantidad de información en formato digital. Usualmente un byte consiste de 8 bits. Un bit es un cero (0) o un uno (1). Esa secuencia de números (byte) puede simbolizar una letra o un espacio (un caracter). Un kilobyte (Kb) son 1024 bytes y un Megabyte (Mb) son 1024 Kilobytes. (Ver también: bit)

C

CC: (Carbon Copy / Courtesy copy) Con Copia. (Encabezado del correo electrónico).

CERN: Conseil Europeen pour la Recherche Nucleaire (CERN) (Consejo Europeo para la Investigación Nuclear) Institución europea, situada en Ginebra, que desarrollo, para sus necesidades internas, el primer navegador y el primer servidor WWW. Ha contribuido decisivamente a la difusión de esta tecnología y es uno de los rectores del W3 Consortium, el organismo clave en la difusión y estandarización de WWW. Ver también: "browser","NCSA","W3","WWW".

CGI: (Common Gateway Interface). Conjunto de reglas que describen cómo un servidor web se comunica con un programa dentro de la misma máquina (El "programa CGI").Cualquier programa puede ser un CGI, con tal de que maneje sus entradas y salidas de acuerdo con dichas reglas. Usualmente, cuando se está usando un programa CGI, puede verse "cgi-bin" en el URL del navegador, aunque no siempre sucede así.

CHAT: (Tertulia, conversación, charla) Comunicación simultánea entre dos o más personas a través de Internet. Hasta hace poco tiempo sólo era posible la "conversación" escrita pero los avances tecnológicos permiten ya la conversación audio y video.

Cliente: Se dice que un programa es un "cliente" cuando sirve sólo para obtener información sobre un programa "servidor". Cada programa "cliente" está diseñado para trabajar con uno o más programas "servidores" específicos, y cada "servidor" requiere un tipo especial de "cliente". Un navegador es un programa "cliente".(Ver también: navegador, servidor)

Ciberespacio: Término utilizado por William Gibson en su novela Neuromancer. Se utiliza para describir la totalidad de información disponible en las redes.

Correo Electrónico: Mensaje, usualmente de texto, enviado de una persona a otra a través de Internet o de cualquier otra red. Es posible enviar automáticamente un mismo mensaje a muchos destinatarios.

Cookie: Pequeño archivo de texto que un sitio web coloca en el disco rígido de una computadora que lo visita. Al mismo tiempo, recoge información sobre el usuario.

CPU: Central Processing Unit. Unidad central de procesamiento. Es el dispositivo que contiene los circuitos lógicos que realizan las instrucciones de la computadora.

CSNET: (Computer Science Network) Con idea de ampliar el alcance y llegar a toda la comunidad científica, la fundación Nacional para las Ciencias (National Science Foundation, NSF) estableció la red CSnet, accesible para todos los departamentos de informática de Estados Unidos. Esta red es, en realidad, una metared, es decir, una red formada por máquinas de departamentos que, o bien pertenecen a ARPAnet, o bien se conectan por la red publica X.25, o bien se conectan por teléfono al ordenador central (red denominada Phonenet, similar a Usenet, pero todo el tráfico es entre el ordenador central y otro, no entre dos ordenadores cualesquiera), o bien pertenecen a CYPRESS (red formada por organizaciones que ofrecen un nodo, en concreto una máquina DEC Microvax, conectado por líneas alquiladas a otros nodos, utilizando protocolos propios de ARPAnet). El servicio ofrecido por esta 'red' es correo electrónico principalmente, aunque también son posibles la transferencia de archivos y el acceso remoto, excepto para los que utilizan Phonenet.

En 1989, la red CSnet se unió a BITnet (comentada anteriormente) para formar una red llamada CREN (Corporation for Research and Education Networking).

DARPA: En 1958 se organiza en los EE.UU. la agencia gubernamental de investigación, A.R.P.A (Advanced Research Projects Agency) creada en respuesta a los desafíos tecnológicos y militares de Rusia de la cual surgirán una década más tarde los fundamentos de la futura red global de computadores Internet. La Agencia, bajo contralor del Departamento de Defensa se organizará en forma independiente de la comunidad de investigación y desarrollo militar.

Dialup: (Conexión por línea conmutada) Conexión temporal, en oposición a conexión dedicada o permanente, establecida entre computadoras por línea telefónica normal.

Disco Duro: Unidad de almacenamiento permanente (masivo) de información. Este es el que guarda la información cuando apagamos la computadora. Aquí se guardan la mayoría de los programas y el sistema operativo. Su capacidad de almacenamiento se mide en Megabytes (Mb) o Gigabytes (Gb). 1024 Mb = 1Gb.

Directorio: En D.O.S., una lista de nombres de archivo que contiene toda la información de los archivos almacenados. A partir de Windows 95 este término se reemplazó por CARPETA.

Dirección: Existen tres tipos de dirección de uso común dentro de Internet: "Dirección de correo electrónico" (email address); "IP" (dirección internet); y "dirección hardware"

DNS: Domain Name System (Sistema de Nombres de Dominio) El DNS un servicio de búsqueda de datos de uso general, distribuido y multiplicado. Su utilidad principal es la búsqueda de direcciones IP de sistemas centrales ("hosts") basándose en los nombres de estos. El estilo de los nombres de "hosts" utilizado actualmente en Internet es llamado "nombre de dominio". Algunos de los dominios más importantes son: .COM (comercial- empresas), .EDU (educacion, centros docentes), .ORG (organización sin ánimo de lucro), .NET (operación de la red), .GOB (Gobierno USA) y .MIL (ejercito USA). La mayoría de los países tienen un dominio propio. Por ejemplo, AR (Argentina) .PY (Paraguay), .US (Estados Unidos de América), .ES (España)..AU (Australia), etc..

Dominio: (Domain Name).Nombre único que identifica a un sitio de Internet. Los nombres de dominio tienen 2 o más secciones, separadas por puntos. La sección de la izquierda es la más específica, y la de la derecha, la más general. Una computadora particular puede tener más de un nombre de dominio, pero un nombre de dominio se refiere únicamente a una PC. (Ver también: IP).

D.O.S.: (Disk operating System). Sistema operativo de disco creado por Microsoft para computadoras personales.

Download: (Bajar, descargar) En Internet proceso de transferir información desde un servidor de información a la propia PC.

DVD: Digital Versatile Disc: Disco Versátil Digital. Disco que posee gran capacidad de almacenamiento y sirve para almacenar películas.

EBCDIC: Extended Binary Coded Decimal Interchange. Código ampliado de caracteres decimales codificados en binario para el intercambio de la información. Código de 8 bits empleado por IBM para codificar símbolos alfanuméricos.

ECC: Error Correcting Code. Método para verificar y corregir errores en la información que es leída o remitida a un dispositivo.

ECP: Enhanced Capabilities Port. Una variante similar al EPP, desarrollada por Microsoft y HP. Basicamente para ser utilizado en scanners e impresoras de prestaciones avanzadas.

EIDE: Enhanced Integrated Drive Electronics. Interfase IDE mejorada, desarrollada por Western Digital. Soporta 4 unidades de almacenamiento.

E-mail: Ver correo electrónico.

Emoticon: Simbolo gráfico, que normalmente representa un rostro humano en sus diversas expresiones, mediante el cual una persona puede mostrar su estado de animo en un medio "frio" como es la pantalla de la PC, por ejemplo al comunicarse mediante correo electrónico o al chatear.

Extensiones: Las extensiones de servidor son programas CGI que proveen la implementación del programas de diseño web (por ejemplo FrontPage) en el servidor de hosting. FrontPage se comunica con las extensiones vía HTTP usando un Remote Procedure Call (RPC). Cuando el servidor recibe el comunicado, simplemente lo envía al programa CGI apropiado. Sin las extensiones de FrontPage en el servidor, usted no podrá sacar provecho de los WEBBOTS que vienen con el programa. Tampoco podrá publicar sus páginas a través del FrontPage, listas de tareas, ni muchas otras opciones privilegiadas que vienen con el Editor/Explorador de FrontPage.

Extranet: Parte de una Intranet de acceso disponible a clientes y otros usuarios ajenos a la compañía.

"F"

FAQ: Frequently Asked Question (Preguntas más frecuentes) Referidas a un tema específico, es una lista de las preguntas realizadas con mayor frecuencia y sus respuestas. La recolección de este conjunto de cuestiones se suele realizar en Grupos de Noticias (Usenet Newsgroups) y en servidores de listas (listserv) y reflejan las contribuciones de sus propios usuarios.

Firewall: (Cortafuegos) Sistema que se coloca entre una red local e Internet. La regla básica es asegurar que todas las comunicaciones entre dicha red e Internet se realicen conforme a las políticas de seguridad de la organización que lo instala. Además, estos sistemas suelen incorporar elementos de privacidad, autentificación, etc.

Firewire: Interfase Plug and Play y hot plugging (IEEE-1394, I-LINK). Permite conectar hasta 63 dispositivos con niveles de transferencia de 25MB/seg.

Freeware: (Programas de libre distribución, programas de dominio publico) Programas de computación que se distribuyen a través de la red de forma gratuita.

FSB: Front Side BUS. BUS que sirve para conectar el microprocesador al puente norte del chipset de la placa madre.

FTP: (File Transfer Protocol). Método utilizado para transferir múltiples archivos entre dos sitios de Internet. Existen muchos sitios que permiten a los usuarios ingresar libremente a sus máquinas para obtener programas shareware u otro tipo de información.

"G"

Gateway: (Pasarela) Hoy se utiliza el término "router" (direccionador, encaminador, enrutador) en lugar de la definición original de "gateway". Una pasarela es un programa o dispositivo de comunicaciones que transfiere datos entre redes que tienen funciones similares pero implantaciones diferentes.

Geek: Persona que siente un entusiasmo ilimitado por la tecnología en general y por la Informática e Internet en particular.

GIF: (Graphics Interchange Format). Un formato de archivos (comprimidos) de imágenes. También existen los llamados GIFs Animados, estos permiten de manejar imágenes transparentes e incluso varias imagenes superpuestas que permiten algunos browsers como Netscape y Explorer.

Giga: Prefijo que indica un múltiplo de 1.000 millones, o sea 10^9. Cuando se emplea el sistema binario, como ocurre en informática, significa un múltiplo de 2^{30}, o sea 1.073.741.824.

Gopher: Un servicio de información distribuida que ofrece colecciones jerarquizadas de información en Internet. Gopher utiliza un protocolo simple que permite a un cliente Gopher acceder a información desde cualquier servidor Gopher que este accesible, proporcionándole un único "espacio Gopher" (Gopher space) de información. Están disponibles también versiones de dominio público para cliente y servidor.

GSM: Global System for Mobile communication (Sistema Global para comunicaciones Móviles) Sistema compatible de telefonía móvil digital desarrollado en Europa con la colaboración de operadores, Administraciones Públicas y empresas. Permite la transmisión de voz y datos.

GUI: Graphical User Interface (Interfaz Gráfica de Usuario) Componente de una aplicación informática que visualiza el usuario y a través de la cual opera con ella. Está formada por ventanas, botones, menús e iconos, entre otros elementos

H

Hacker: Una persona que goza alcanzando un conocimiento profundo sobre el funcionamiento interno de un sistema, de una PC o de una red. Este término se suele utilizar indebidamente como peyorativo, cuando sería más correcto utilizar el término "cracker".

Hardware: Son todos los componentes físicos que componen una PC.

Hertz: Hercio.Unidad de frecuencia electromagnética. Equivale a un ciclo por segundo.

Hipertexto: Generalmente, cualquier texto que contiene enlaces hacia otros documentos. Los enlaces son palabras o frases que pueden ser cliqueadas por el lector para visualizar otro documento relacionado.

Hoax: (Engaño, broma) Término utilizado para denominar a rumores falsos, especialmente sobre virus inexistentes, que se difunden por la red, a veces con mucho éxito causando al final casi tanto daño como si se tratase de un virus real.

Home Page: (Página inicial, página principal) Primera página de un servidor WWW.

Host: (sistema central) Computadora que permite a los usuarios comunicarse con otros sistemas centrales de una red. Los usuarios se comunican utilizando programas de aplicación, tales como el correo electrónico, Telnet, WWW y FTP.

Hosting: Espacio para un sitio o página de Internet en uno de los servidores SGI activos. Es decir, es un espacio en un disco rígido de una computadora conectada las 24 hs del día a Internet para que el autor del sitio pueda darse a conocer en la red.

HTML: (HyperText Markup Language).Lenguaje utilizado para crear los documentos de hipertexto que se emplean en la WWW. Los documentos HTML son simples archivos de texto que contienen instrucciones (llamadas tags) entendibles por el Navegador (Browser). (Ver también: Cliente, Servidor, Hipertexto, WWW).

HTTP: (HyperText Transport Protocol). Protocolo utilizado para transferir archivos de hipertexto a traves de Internet. Requiere de un programa "cliente" de HTTP en un extremo y un "servidor" de HTTP en el otro extremo. Es el protocolo más importante de la WWW. (Ver también: Cliente, Servidor, Hipertexto, WWW).

HTTPS: (HyperText Transport Protocol Secured). El protocolo de comunicación seguro empleado por los servidores de WWW con en clave. Esto es usado para trasportar por internet información confidencial como el número de tarjeta de crédito.

"I"

Icono: Símbolo gráfico que aparece en la pantalla de una PC para representar determinada acción a realizar por el usuario, ejecutar un programa, leer una información, imprimir un texto, etc. Un icono hace referencia a un programa o archivo computacional y por lo tanto le permite el acceso al mismo por parte del usuario.

IDS: Internet Domain Survey http://www.isc.org/ds/

IEEE: Institute of Electrical and Electronic Engineers (Instituto de Ingenieros en Eléctricidad y Electrónica) Asociación de profesionales informáticos con base en los EE.UU.

Internet: Conjunto de redes conectadas entre sí, que utilizan El protocolo TCP/IP para comunicarse.

Internet2: Proyecto de interconexión de más de 100 universidades estadounidenses. El objetivo es desarrollar una red de altísima velocidad para la educación y la investigación.

Intranet: Red privada dentro de una empresa que utiliza el mismo software y protocolos empleados en la Internet global, pero que sólo es de uso interno. (VER también: Internet, Red).

IP: (Internet Protocol). Número único que consta de 4 partes separadas por puntos. Cada computadora conectada a Internet tiene un único número de IP. Si la maquina no tiene una IP fija, no está en realidad en Internet, sino que pide "prestado" un IP a un servidor cada vez que se conecta a la Red (usualmente vía módem). (Ver también: Dominio, Internet, TCP IP).

IRda: Infrared Data Association. Permite transmitir datos por medio de luz infrarroja. Estos puertos permiten conectar entre si a 2 dispositivos separados por corta distancia a una velocidad de transferencia similar a la de un puerto paralelo; sin necesidad de utilizar cables.

IRC: Internet Relay Chat (IRC) (Charla Interactiva Internet) Protocolo mundial para conversaciones simultáneas ("party line") que permite comunicarse por escrito entre si a través de ordenador a varias personas en tiempo real. El servicio IRC esta estructurado mediante una red de servidores, cada uno de los cuales acepta conexiones de programas cliente, uno por cada usuario.

ISDN: Integrated Services Digital Network (Red Digital de Servicios Integrados) Tecnología de transmisión que es ofrecida por las compañías telefónicas más importantes. ISDN combina servicios de voz y digitales a través de la red en un sólo medio, haciendo posible ofrecer a los clientes servicios digitales de datos así como conexiones de voz a través de un sólo "cable". Los estándares de la ISDN los especifica la ITU-TSS.

ISO: International Organization for Standardization (Organización Internacional para la Normalización) Organización de caracter voluntario fundada en 1946 que es responsable de la creación de estándares internacionales en muchas áreas, incluyendo la informática y las comunicaciones. Está formada por las organizaciones de normalización de sus países miembro.

ISP: (Internet Service Provider). Ver Proveedor.

ITU: International Telecommunications Union (Unión Internacional de Telecomunicaciones) Agencia de las Naciones Unidas que coordina los diversos estándares nacionales de telecomunicaciones de forma que las personas puedan comunicarse entre si independientemente del país dónde vivan.

Java: Lenguaje de programación orientado a redes. Fue diseñado por Sun Microsystems específicamente para escribir programas que pudieran bajarse y ejecutarse en la computadora local, sin temor que éstos pudieran contener virus o provocar algún daño en los archivos. Las páginas web pueden contener pequeños programas hechos en Java llamados applets, por ejemplo: para producir animaciones u otros efectos vistosos (también se puede programar aplicaciones completas, como calculadoras, clientes de chat, etc.). (Ver también: Applet).

JDK: (Java Development Kit). Aplicaciones de Sun Microsystems que contienen todas las herramientas necesarias para desarrollar, probar y depurar programas y applets hechos en Java. (Ver también: applet, Java).

JavaScript: Lenguaje de programación que soportan los navegadores. Su código se programa directamente dentro de la página HTML, y es interpretado por navegador al leerla. A pesar de su nombre, no tiene nada que ver con Java, ya que los applets creados por este último se bajan, compilan y ejecutan al ser invocados por la página. (Ver también: HTML, Java).

JPEG: Formato gráfico comprimido desarrollado por la 'Join Photographic Expert Group'. El formato JPEG soporta 24 bits por pixel y 8 bits por pixel en imágenes con escala de grises.

Kbps: (Kilobits por segundo) Unidad de medida de la capacidad de transmisión de una línea de telecomunicación. Cada kilobit esta formado por mil bits.

Keyword: (Clave de búsqueda, palabra clave) Conjunto de caracteres que puede utilizarse para buscar una información en un buscador o en un sitio web.

LAN: (Local Area Network). (Red de Area Local). Red de computadoras ubicadas en El mismo ambiente, piso o edificio. (Ver también: Red).

LINUX: Versión de libre distribución del sistema operativo UNIX; fue desarrollada por Linus Torvald

Link: (Enlace/enlazar, vínculo/vincular) Apuntadores hipertexto que sirven para saltar de una información a otra, o de un servidor a otro, cuando se navega por Internet o bien la acción de realizar dicho salto.

Login: Nombre de usuario utilizado para obtener acceso a una computadora o a una red. A diferencia del password, login no es secreto, ya que generalmente es conocido por quien posibilita el acceso mediante este recurso. (Ver también: password)

Majordomo: (Mayordomo) Aplicación que, en los servidores de listas, de encarga de realizar de forma automatizada funciones de gestión tales como altas y bajas de suscriptores a las mismas.

Mailing list: Listas de correo o listas de distribución. Establecen foros de discusión privados a través de correo electrónico. Las listas de correo están formada por direcciones e-mail de los usuarios que la componen. Cuando uno de los participantes envía un mensaje a la lista, ésta reenvía una copia del mismo al resto de usuarios de la lista (inscritos en ella). Las listas pueden ser abiertas: cualquier persona puede subscribirse y participar en ella o cerradas: Existe un dueño y moderador de la lista, que decide quien puede entrar en ella.

MBONE: (Red troncal multimedia) Red de banda ancha y alta velocidad que permite actualmente la realización de audio y videoconferencias entre centenares de usuarios remotos mediante dos canales de video y cuatro de audio.

Mbps: (Megabits por segundo) Unidad de medida de la capacidad de transmisión por una línea de telecomunicación. Cada megabit está formado por un millón de bits.

MCI: Es una unidad Operando de WorldCom, Inc. (NASDAQ: WCOEQ, MCWEQ), es un proveedor principal de voz residencial, messaging avanzados y servicios de las telecomunicaciones comerciales en los Estados Unidos.

Megabyte (MB): 1,048,576 bytes; 1,024 Kilobytes. (Ver también: byte, bit, Kilobyte).

Megahertz: Unidad de medida de la frecuencia de reloj del microprocesador (en millones de ciclos por segundo).

MIME: Multipurpose Internet Mail Extensions (Extensiones Multipropósito del Correo Internet) Conjunto de especificaciones Internet de libre distribución que permiten tanto el intercambio de texto escrito en lenguajes con diferentes juegos de caracteres como el correo multimedia entre ordenadores y aplicaciones que sigan los estándares de correo Internet.

Mirror: (Espejo, replica) Servidor Internet cuyo contenido es una copia exacta de otro. Normalmente este tipo de servidores cuentan con la aprobación del servidor original y sirven para reducir el tiempo de acceso del usuario a servidores situados en lugares muy distantes.

MMX: (Multimedia extensions) nombre dado a las instrucciones contenidas en la segunda generación de procesadores Intel Pentium para mejorar el rendimiento de las aplicaciones multimedia.

Módem: (MOdulator,DEModulator). Dispositivo que se conecta a la computadora y a la línea telefónica y que permite comunicarse con otras computadoras a través del sistema telefónico. Básicamente, los módems sirven a las computadoras de la misma manera que los teléfonos sirven a las personas.

Mosaic: Primer navegador de WWW, que estuvo disponible tanto para Macintosh como para Windows Unix con la misma interfase. Fue el programa popularizó la Web y en su código fuente se basan los navegadores que más tarde le ganaron terreno: Netscape e Internet Explorer. (Ver también: Navegador, Cliente, WWW).

MP3: Formato de archivos de sonido, notable por su calidad y nivel de compresión (tamaño).

N

Navegador: Programa utilizado para acceder y recorrer sitios de la WWW. (Ver también: WWW, Mosaic).

Netiquette: (Etiqueta de la red) Conjunto de normas dictadas por la costumbre y la experiencia que define las reglas de urbanidad y buena conducta que deberían seguir los usuarios de Internet en sus relaciones con otros usuarios.

Nielsen: ACNielsen, líder mundial en investigación de mercados, información y análisis, fue fundada en 1923 por Arthur Conell Nielsen. Desde entonces ha ido extendiendo los productos y servicios, encontrándose presente en más de 100 países.

NFSNET: Fue la red que remplazo a Arpanet, era una red auspiciada por la Fundación Nacional de Ciencias de EUA.

Nodo: Una computadora conectada a la red. (Ver también: red, Internet, LAN).

News: (Noticias, Grupos de Noticias) Forma habitual de denominar el sistema de listas de correo mantenidas por la red USENET.

"O"

Octeto: Un octeto esta formado por 8 unidades de información (llamadas "bits"). Este término se usa a menudo en vez de "byte" en la terminología de redes porque algunos sistemas tienen "bytes" que no están formados por 8 bits. Ver también:"bit","byte".

OSI: Open Systems Interconnection (Interconexión de Sistemas Abiertos) Conjunto de protocolos diseñados por comités ISO con el objetivo de convertirlos en estándares internacionales de arquitectura de redes de computadoras.

Outsourcing: Contratación de una empresa para desarrollar por medio de ella, en forma tercerizada, tareas determinadas.

"P"

Password: Palabra clave utilizada para obtener acceso a una computadora o a una red. Un password generalmente contiene una combinación de números y letras que no tienen ninguna lógica. (Ver también: Login).

Plug-in: Porción de software que agrega funciones a un programa más grande. Los plug-ins más comunes son los de los navegadores o los de programas para diseño gráfico, como photoshop. Los plug-ins son creados por terceros para agregar funciones que no se encuentran originalmente en los programas.

POP: (Post Office Protocol). Manera en que los programas clientes de correo electrónico obtienen los mensajes de un servidor de correo. Cuando se obtiene una cuenta SLIP o PPP, en general también se obtiene una cuenta POP para poder acceder al servidor de correo y leer los mensajes. (Ver también: SLIP, PPP).

PPP: (Point to Point Protocol). Protocolo que le permite a la computadora usar una línea telefónica y un módem para realzar una conexión TCP/IP y así simular que está realmente dentro de Internet. (Ver también: IP, Internet, SLIP, TCP/IP).

Procesador: Llamado también Microprocesador. Es el chip encargado de ejecutar las instrucciones y procesar los datos que son necesarios para todas las funciones de la computadora. Se puede decir que es el cerebro de la computadora. El estandar del mercado es el fabricado por la empresa INTEL.

"Q"

Queue: (Cola) Conjunto de paquetes en espera de ser procesados

Quote: (Comillas) Generalmente la frase "Quote Of The Day" representa a "Máxima del día", o se refiere a una frase que se encuentra entre comillas.

"R"

RAID: Redundant Array of Independent Disks. Método para almacenar datos en diferentes discos. La información puede ser almacenada en forma redundante (los mismos datos en todas las unidades) o bien repartirse en varios discos que aparecen para el Sistema Operativo como una sola unidad.

RAM: (Random Access Memory) Memoria de acceso aleatorio y de tipo volátil o temporal. Es la memoria de trabajo de una PC.

RAND: "Redes de Comunicación Distribuida" ("On Distributed Communication Networks")
Redes conmutadas por paquetes, sin punto único de interrupción.

Red: Se tiene una red cada vez que se conectan dos o más computadoras de manera que pueden compartir recursos. Al conectar dos o más redes en conjunto, se obtiene una Internet.

Router: Computadora de uso específico que maneja la conexión dos o más redes. Los routers pasan todo el tiempo buscando las direcciones de destino de los paquetes con información y deciden cuál es el mejor camino para enviarlos. (Ver también: Red).

"S"

SGML: Standardized Generalized Markup Language (SGML) (Lenguaje Estandarizado y Generalizado de Marcado) Estándar internacional para la definición de métodos de representación de texto en forma electrónica no ligados a ningún sistema ni a ningún dispositivo. Ver también:"HTML".

Shareware: (Programas compartidos) Dícese de los programas informáticos que se distribuyen a prueba, con el compromiso de pagar al autor su precio, normalmente bajo, una vez probado el programa y/o pasado cierto tiempo de uso. Ver también:"freeware","public domain".

Servidor: Computadora o programa que brinda un servicio específico al "cliente", que se ejecuta en otras computadoras. El término puede referirse tanto a una pieza de software en particular como a una computadora en dónde se ejecuta este tipo de software. (Ver también: cliente, red).

SLIP: (Serial Line Internet Protocol). Un estándar para usar la línea telefónica y un módem para conectarse a Internet. Se está reemplazando gradualmente por El PPP. (Ver también: PPP).

SMP: Symmetric Multiprocessing. Arquitectura mediante la cual se pueden usar varios procesadores para ejecutar múltiples tareas.

SO: (Sistema Operativo). Programa o conjunto de programas que permiten administrar los recursos de hardware y software de una computadora.

Socket: Tipo de conexión (zócalo) entre el microprocesador y la placa madre (socket 5, socket 7, Slot 1 son algunos de los más comunes).

Software: Todos los componentes no físicos de una PC (Programas).

SPP: Standard Parallel Port. Puerto paralelos originales (los primeros que se desarrollaron). Son unidireccionales y sólo fueron pensados para enviar datos a la impresora.

Spam: Correo electrónico no solicitado. Se lo utiliza generalmente para promover productos y/o servicios.

SSL: (Secure Socket Layer) Sistema de seguridad en el cual los mensajes son encriptados de manera que solamente quien los emite y quien los recibe podrán descifrarlos.

"T"

Tarjeta Madre: (Motherboard). Es un circuito integrado con varios microchips y diferentes tipos de ranuras y conectores. En ella se conectan todos los componentes de la computadora incluyendo el procesador. La misma se conecta a la fuente de alimentación.

T3: El sistema más rápido existente para enlaces en internet. T3 es el nombre que tienen las líneas de conexión a Internet con un ancho de banda de 45 MB por segundo, conectadas directamente al backbone de internet (DS3). Un proveedor común de internet tiene líneas de 256k a 4 megas.

TCP/IP: (Transmisor Control Protocol/Internet Protocol). Conjunto de protocolos que definen a la Internet. Fueron originalmente diseñados para el sistema operativo Unix, pero actualmente puede encontrarse en cualquier sistema operativo. (Ver también: IP, Internet, Unix).

Trackball: Dispositivo apuntador similar al mouse en el que se desplaza con la mano, el pulgar o el índice una bola acoplada a una base que permanece fija.

"U"

Ultra-DMA: Tecnología utilizada en los discos rígidos IDE más modernos para elevar la tasa de transferencia teórica máxima hasta 33 MBs. o 66 MBs.

Unix: Sistema operativo que fue diseñado para que lo usaran muchas personas a la vez (es multi-usuario) a la vez tenia incorporado al protocolo TCP/IP. Es el sistema operativo más utilizado por los servidores de Internet. (Ver también: servidor, TCP/IP).

URL: (Uniform Resource Locator). Dirección de algún recurso de Internet que forma parte de la WWW. Ver también: navegador, WWW).

USENET: Conjunto de foros electrónicos llamados grupos de noticias, cuyo mecanismo de funcionamiento es similar a un gran tablón o pizarra electrónica dónde se van sucediendo los mensajes.

USB: Tecnología que facilita la conexión de periféricos a la computadora. Esta reconoce automáticamente los dispositivos nuevos y no hay que insertar una placa controladora para el dispositivo en cuestión, sino que se conecta a la parte trasera de la PC a un enchufe especial (puerto USB). La tarjeta madre debe tener esta tecnología en su CHIPSET para poder conectar dispositivos de este tipo.

UUCP: (Unix to Unix Copy) Aunque se cita como red, en realidad UUCP es un programa que permite copiar archivos vía línea de teléfono desde una máquina UNIX a otra. Según los sistemas UNIX fueron adquiriendo capacidades de envío vía modems y llamadas automáticas, se fueron forjando una serie de redes informales que, basándose en esta funcionalidad situaban como ordenador central un equipo que se conectaba con un grupo de máquinas a avanzadas horas de la noche para intercambiar archivos y correo electrónico. Este tipo de redes creció muy rápidamente dado que para unirse a ellas sólo se necesitaba poseer un sistema UNIX con un modem, algo bastante común en los departamentos de universidad (en este sistema estaba basado Nodo50 desde 1994 hasta 1996). La unión de estas redes dio lugar a la red UUCP.

"V"

V32: Norma internacional para comunicaciones vía modem que permite alcanzar una velocidad de 14.400 baudios.

V34: Norma internacional para comunicaciones vía modem que permite alcanzar una velocidad de 28.800 baudios.

V34+: Norma internacional para comunicaciones vía modem que permite alcanzar una velocidad de 33.600 baudios.

V90: Norma internacional para comunicaciones vía modem que permite alcanzar una velocidad máxima de 55.600 baudios, dependiendo de ciertas condiciones, sobre todo, tipo y calidad de la línea.

VESA: Estándar de modos de video para tarjetas VGA y superiores, que permite programar drivers compatibles con todas las tarjetas graficas que cumplan estas normas, independientemente del chip que incorporen.

VGA: (Video Graphics Array), o dispositivo Grafico de video. Un tipo de tarjeta grafica capaz de obtener hasta 640x480 puntos en 16 colores (en el modelo estándar original).

Virus: Programa que se duplica a si mismo en un sistema informático incorporándose a otros programas que son utilizados por varios sistemas. Estos programas pueden causar problemas de diversa gravedad en los sistemas que los almacenan. Los medios de propagación de los mismos pueden ser a traves de cualquier medio de almacenamiento o a traves de la LAN, o de la misma INTERNET.

VLB: (Vesa Local Bus), un tipo de slot o ranura de expansión de 32 bits capaz de ofrecer hasta 132 MB/s a 33 MHz o 160 MB/s a 40 MHz.

VR: Virtual Reality. Realidad Virtual.

VRAM: Memoria de las placas de video.

VRML: Virtual Reality Modeling Language. Lenguaje para Modelado de Realidad Virtual. Lenguaje para crear mundos virtuales en la Web.

"W"

W3C: World Wide Web Consortium. Organización apadrinada por el MIT y el CERN, entre otros, cuyo cometido es el establecimiento de los estándares relacionados con la WWW

WAIS: Wide Area Information Servers (WAIS) Servidores de Información de Area Amplia Servicio de información distribuida que permite hacer preguntas en lenguaje simple, la búsqueda indexada para obtener información con rapidez y un mecanismo de "retroalimentación de relevancia" que permite que los resultados de una búsqueda inicial repercutan en búsquedas subsiguientes. Existen versiones de dominio publico. Ver también:"archie"."Gopher"

WAP: Wireless Application Protocol. Protocolo inalámbrico de aplicaciones. Permite la comunicación entre dos teléfonos celulares (móviles) a través de Internet y la conexión de cada uno de ellos a Internet.

WAN: (Wide area Net), red de banda ancha. Una red de computadoras de gran tamaño, dispersa por un país o incluso por todo el planeta.

Warez: Software Ilegal. Programas comerciales que han sido puestos a disposición del público a través de Internet de forma ilegal.

Webmaster: Administrador de un sitio de web

Webbot: Son los mecanismos para llamar muchas de las opciones interactivas creadas dentro del FrontPage. Estas opciones se suman a su web a través del FrontPage Editor (Insertar componente de FrontPage). Algunas como "Inclusión" y "Sustitución" permiten ingresar un elemento de la web una vez y luego incluirlo en alguna o todas de las págmas insertando el componente Bot referenciando a ese elemento (por ej.: barra de navegación). Si se le hace un cambio al elemento 'maestro' a utomáticamente cambiarán los elementos de todas las páginas que contengan el Bot referenciado. Otros Bots, tales como buscar, tabla de contenidos y marca de hora, controlan características dinámicas. Estos Bots trabajan detrás de escena para mantener la página al dia para los visitantes.

WinCGI: (Windows Common Gateway Interface). Adaptación de la CGI a programas que funcionan bajo el entorno de Windows. Esto permite construir programas en lenguaje visuales como Visual Basic, y ejecutarlos en un servidor web (que debe soportar WINCGI). (Ver también: CGI, Web).

Wizard: Asistente. Sistema de pantallas interactivas que facilitan el uso de ciertas aplicaciones, por ejemplo las de instalación de los productos en entornos de PC.

WWW: (World Wide Web). Conjunto de recursos que pueden accederse utilizando un Navegador, mediante el protocolo HTTP. (Ver también: HTTP, URL).

"X"

XENIX: Sistema operativo multiusuario y multitarea basado en UNIX.

XGA: (eXtended Graphics Array), o dispositivo grafico extendido. Un tipo de tarjeta grafica capaz de obtener hasta 1024x768 puntos en 16 colores.

XHTML: (EXtended HTML) HTML extendido. Reformulación del HTML 4.0 basada en una aplicación del XML 1.0

XML: (eXtensive Markup Language) Subconjunto del SGML desarrollado con el fin de hacer compatible el propio SGML con la red.

XMS: Memoria extendida. Una forma de acceder a la memoria superior (por encima de los primeros 640 Kb), mediante software como el HIMEM.SYS del sistema operativo.

XT: Tipo de computadora compatible con el modelo denominado de esa forma por IBM. En general, cualquier PC compatible con disco rigido y un procesador 8086 o superior.

"Z"

ZIF: (Zero Insertion Force) (socket), o zócalo de fuerza de inserción nula. Conector de forma cuadrada en el que se instalan algunos tipos de microprocesador, caracterizado por emplear una palan que ayuda a instalarlo sin ejercer presión ("Force") sobre las patillas del chip.

Tabla de Caracteres ASCII

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	◘	BS	040	(	072	H	104	h
009	(tab)	HT	041	)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♫	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	►	DLE	048	0	080	P	112	p
017	◄	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	‼	DC3	051	3	083	S	115	s
020	¶	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↨	ETB	055	7	087	W	119	w
024	↑	CAN	056	8	088	X	120	x
025	↓	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[	123	{
028	(cursor right)	FS	060	<	092	\	124	|
029	(cursor left)	GS	061	=	093	]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	⌂

BIBLIOGRAFÍA

(01B) **ALVARES García, Alonso.** "HTML, CGI, JAVA, servidores -- tecnológia WWW" Anaya Multimedia, Madrid: 1996

(02B) **BOUTELL, Thomas.** "CGI programming in C & Perl" Addison-Wesley, Reading, Mass c1996

(03B) **CHRISTIANSEN, Tom.** "Perl cookbook" Edicion: 1st ed. O'Reilly, Sebastopol, CA : c1998.

(04B) **DAVID Medinets.** "Perl 5 a través de ejemplos" Prentice-Hall Hispanoamericana, S.A. 1997.

(05B) **DEEP, John.** "Developing CGI applications with Perl" J. Wiley, New York: c1996

(06B) **O'FOGLU, Michael.** "PERL 5 : SOLUCIONES INSTANTANEAS" Prenticehal Hispanoamericana, México c 1997

(07B) **PALACIOS Bañares Juan.** "Perl: Páginas Web Interactivas" Alfaomega, México, D.F., c1999

(08B) **PALACIOS Bañares Juan.** "Perl Páginas Web Interactivas" Rama, España, 1999

(09B) **RICH Bowen, Ken Coar.** "Servidor Apache al descubierto" Pearson Educación, S.A. Madrid, 2000

(10B) **ROWE, Jeff.** "Webmaster creación de servidores de bases de datos para Internet con CGI" Prentice-Hall Hispanoamericana, México: c1996.

(11B) **TITTEL Ed.** "Fundamentos de programación con HTML & CGI" Anaya Multimedia, Madrid: 1996

(12B) **TITTEL Ed.** "La Biblia de la programación CGI" Anaya Multimedia, Madrid: 1997

(13B) **WALL, Larry.** "Programming Perl" Edicion: 2nd ed. O'Reilly & Associates, Sebastopol, CA : c1996.

(14B) **WALL, Larry.** "Programming perl" O'Reilly, Sebastopol, california : c1990

(15B) **WALSH, Nancy.** "Learning Perl/Tk" O'Reilly, Beijing ; Sebastopol, CA : 1999.

(16B) **WYKE, R. Allen.** "Guía de referencia para programadores de Perl 5"
Anaya Multimedia, Madrid, c1998
WONG, Clinton. "Web client programming with Perl"
Edicion: 1st ed.
O'Reilly, Cambridge ; Sebastopol, CA : c1997.

(17B) **WYKE, R. Allen.** "The Perl 5 programmer's reference : Windows 95/NT, Macintosh, OS/2 & UNIX /"
Edicion: 1st ed.
Communications Group, Research Triangle Park, NC ; c1997.

REFERENCIAS

(01R) © 2003, O'Reilly & Associates, Inc.<webmaster@oreilly.com> [Citado 02 enero del 2002] Disponible en World Wide Web: <http://perl.oreilly.com>

(02R) (c) 1994 Jordi Adell <jordi@edu.uji.es> Carles Bellver <bellver@si.uji.es> Universitat Jaume I. Castelló *version 0.2.2* ADELL, J. y BELLVER, C. (1995). La Internet como telaraña: el World-Wide Web. [Vol. 2. N. 3. Enero 1995.] Disponible en World Wide Web: <http://tecnologiaedu.us.es/revistas/libros/a5.htm>

(03R) Disponible en World Wide Web: [Citado 02 enero del 2002] <http://www.lpis.com/ayudas/perl.html>

(04R) Área de informática Universidad de Cádiz [en línea]. 14 de Junio de 1996 [Citado 2 enero del 2002] Disponible en World Wide Web: <http://www2.uca.es/FAQ-www/formulario.html>

(05R) AdsoNet 1996 [Citado 12 Noviembre del 2002] Disponible en World Wide Web: <http://www.adso.net/AdsoNet/perl.html>

(06R) Copyright 2002-2003 [Citado 15 enero del 2002] Disponible en World Wide Web: <http://www.perl.org/>

(07R) 2003 ActiveState, a division of Sophos [Citado 3 enero del 2002] Disponible en World Wide Web: <http://www.activestate.com/ActivePerl/>

(08R) Compilation Copyright © 1998-2003 O'Reilly & Associates, Inc. All Rights Reserved [Citado 20 febrero del 2002] Disponible en World Wide Web: <http://language.perl.com/>

(09R) Copyright © Netcraft LTD 2002 [Citado 20 Febrero del 2002] Disponible en World Wide Web: <http://www.netcraft.com/>

(10R) Copyright © 1999-2003, The Apache Software Foundation [Citado 02 enero del 2002] Disponible en World Wide Web: <http://www.apache.org/>

(11R) Copyright © 1999-2002, The Apache Software Foundation [Citado 24 marzo del 2002] Disponible en World Wide Web: <http://httpd.apache.org/>

(12R) CPANJarkko Hietaniemi <cpan@perl.org> [Citado 24 marzo del 2002] Disponible en World Wide Web: <http://www.cpan.org>

(13R) Disponible en World Wide Web: [Citado 13 Agosto 2002] <http://www-cgi.cs.cmu.edu/cgi-bin/perl-man

(14R) Disponible en World Wide Web: [Citado 13 Agosto 2002] <http://theory.uwinnipeg.ca/search/cpan-search.html>

(15R) Disponible en World Wide Web: [Citado 05 Junio 2002] <http://www.scs.leeds.ac.uk/Perl/>

(16R) Copyright © 1998 Steven E. Brenner / <cgi-lib@pobox.com> [Citado 15 Abril del 2002] Disponible en World Wide Web: <http://cgi-lib.berkeley.edu/>

(17R) Text copyright © 1996 Johan Vromans [Citado 03 Abril del 2002] Disponible en World Wide Web: <http://www.rexswain.com>

(18R) Disponible en World Wide Web: [Citado 05 Junio 2002] <http://www.cicei.ulpgc.es/gsi/tutorial_perl/cap1.htm>

(19R) © 1997 por Roberto Da Silva Drumond [Citado 2 mayo del 2002] Disponible en World Wide Web: <http://www.geocities.com/SiliconValley/Park/8603/>

(20R) Disponible en World Wide Web: [Citado 05 Junio 2002] <http://www.servidores.unam.mx/tutoriales/apache.html>

(21R) © Copyright 2003 Arsys Internet S.L. [Citado 15 Abril del 2002] Disponible en World Wide Web: <http://www.arsys.es/soporte/programacion/cgi.htm>

(22R) Disponible en World Wide Web: [Citado 05 Junio 2002] <http://www.dbinternet.com.ar/inter.htm?db0Cil>

(23R) Disponible en World Wide Web: [Citado 05 Diciembre 2002] <http://www.webviva.com/faqcgi/ii.html>

(24R) © 1999 - 2002 - Iván Nieto Pérez [Citado 15 Abril del 2002] Disponible en World Wide Web: <http://www.elcodigo.net/tutoriales/montarwebsite/montarwebsite2.html>

(25R) © 1996-1998 James Marshall Traducido en 1998 por René Alvarez Ultimo Modificado: Abril 18, 1998 Disponible en World Wide Web: [Citado 05 Junio 2002] http://www.jmarshall.com/easy/cgi/spanish.

(26R) © Copyright 1998 Dan Steinman [Citado 05 Marzo del 2003] Disponible en World Wide Web: <http://www.dansteinman.com/dyndns/es/cgicomm.html>

(27R) © Copyright 1997 - 2003 Maestros del Web. [Citado 05 Marzo del 2003] Disponible en World Wide Web: <http://www.maestrosdelweb.com/editorial/articulo.php?cgiintro>

(28R) Copyright © 1998 Klor Universe System Clemente Olivares [Citado 05 Marzo del 2003] Disponible en World Wide Web: <http://www.geocities.com/SunsetStrip/Backstage/6023/cgi1.html>

(29R) Copyright © Grupo Alternativo 2000, 2002. [Citado 06 Enero del 2003] Disponible en World Wide Web: <http://www.mexicoextremo.com.mx/ayuda/cgi-intro.php3>