UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

“DESARROLLO DE SISTEMAS INFORMATICOS A TRAVÉS DE HERRAMIENTAS DE SOFTWARE LIBRE”

T E S I N A

P A R A O B T E N E R E L T Í T U L O D E

I N G E N I E R O E N C O M P U T A C I Ó N

P R E S E N T A :

R O D R I G O A R I A S H U R T A D O

ASESOR: ING. JUAN GASTALDI PÉREZ

2008

FACULTAD DE ESTUDIOS SUPERIORES
ARAGÓN

JEFATURA DE CARRERA
INGENIERÍA EN COMPUTACIÓN

OFICIO Nº: FESAR/JACO/ 41 /07

ASUNTO: Asignación de Jurado

VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

LIC. ALBERTO IBARRA ROSAS
SECRETARIO ACADÉMICO
Presente

Por este conducto me permito presentar a usted el nombre de los profesores que sugiero integren el Sínodo del Examen Profesional del alumno

R O D R I G O A R I A S H U R T A D O

con el trabajo de titulación intitulado

"DESARROLLO DE SISTEMAS INFORMÁTICOS A TRAVÉS DE HERRAMIENTAS DE SOFTWARE LIBRE"

bajo la modalidad de: Seminarios y Cursos de Actualización y Capacitación Profesional

PRESIDENTE	ING. JUAN GASTALDI PÉREZ
VOCAL	ING. SILVIA VEGA MUYTOY
SECRETARIO	ING. ROBERTO BLANCO BAUTISTA
SUPLENTE	ING. JOSÉ MANUEL QUINTERO CERVANTES
SUPLENTE	M. EN C. MARCELO PÉREZ MEDEL

Quiero subrayar que el director de titulación es

ING. JUAN GASTALDI PÉREZ

el cual está incluido con base en lo que reza el Reglamento de Exámenes Profesionales de esta Facultad.

Sin otro particular, me es grato enviarle un cordial saludo.

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
San Juan de Aragón, Estado de México a 06-Feb-08

EL JEFE DE CARRERA

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN U.N.A.M. INGENIERÍA EN COMPUTACION

M. EN C. MARCELO PÉREZ MEDEL

C.c.p. - Ma. Teresa [illegible]a Sánchez. * Jefa del Departamento de Servicios Escolares
- [illegible]ALDI PÉREZ * Asesor
- Interesado.

MPM * vjd

San Juan *de* Aragón, *Edo. de Méx., a 16 de marzo de 2006.*

Lic. Alberto Ibarra Rosas
Secretario Académico
FES Aragón
Presente:

Por este conducto me permito hacer de su conocimiento que el alumno **RODRIGO ARIAS HURTADO**, con número de cuenta 9722591-3 concluyó satisfactoriamente el trabajo de Titulación titulado **"DESARROLLO DE SISTEMAS INFORMATICOS A TRAVÉS DE HERRAMIENTAS DE SOFTWARE LIBRE"**, lo que hago de su conocimiento para que el mismo pueda continuar con los trámites respectivos.

Sin más por el momento, reciba un cordial saludo.

Atentamente.

Ing. Juan Gastaldi Pérez
Director de Tesis

Vo. Bo.

M. en C. Marcelo Pérez Medel
Jefe de la Carrera

Agradecimientos

Haber terminado mi carrera, representó una de las mayores satisfacciones que he experimentado en mi vida, no sólo porque me ha permitido integrarme a la sociedad como un profesionista preparado para enfrentar y resolver problemas, sino que también simboliza el final de una etapa en mi existencia, que estuvo constituida por esfuerzos, fracasos y éxitos, en los cuales, nunca estuve sólo.

Es por esta razón que aprovecho este apartado, para agradecer a todas aquellas personas que con su cariño y fe en mí, lograron mantenerme constante y firme hasta el final. En primera instancia, quiero mencionar a mis padres y hermano, porque reconozco todos los sacrificios que hicieron por mí, en búsqueda de mi bienestar y futuro. Papá, mamá, muchas gracias por su amor, orientación, ejemplo y apoyo, ha sido vital en mi trayecto.
A mi prometida Wendy Mayen, por haber luchado a mi lado y compartido tantos momentos, esfuerzos, sueños que poco a poco hemos ido cumpliendo juntos; por haberme apoyado y alentado a conseguir las cosas que buscaba y por creer siempre en mi y en lo que hago.

Quiero darle las gracias al Ingeniero Juan Gastaldi Pérez, por haberme brindado su amistad y confianza; por todo el apoyo que me ha dado, por su buen humor y mucho más, siempre lo tendré presente. Muchas gracias por haber aceptado ser mi guía en el desarrollo de este trabajo. Gran parte de

Por último quiero agradecer a todos mis sinodales, maestros y compañeros por todo su apoyo, orientación y amistad. No puedo negar que soy orgullosamente Puma.

“Sólo un exceso es recomendable en el mundo: el exceso de gratitud”.
Jean de La Bruyere

Índice General

Introducción

Entre los años sesenta y setenta, era común entre los programadores compartir sus aplicaciones, para que otros desarrolladores pudieran estudiarlas, mejorarlas e incluso, a partir de ellas, crear otros programas. Esto sucedía porque en ese entonces, el software se distribuía gratuitamente con los equipos de cómputo de aquella época.

Pero eso cambió a finales de los años setenta, cuando las grandes compañías empezaron a vender el software individualmente como un producto que permitía darles una funcionalidad extra a los equipos que vendían. A partir de ese momento, las compañías empezaron a vender *Licencias de Software* de sus productos, que en si, era la autorización por parte del titular del derecho de autor, a través de un contrato con el usuario del programa de computadora, para utilizar éste, de manera determinada y bajo condiciones convenidas. Por ejemplo, la licencia señalaba el número total de usuarios que podían utilizar el software, precisaba la existencia de los derechos de modificación y/o redistribución, además, podía marcar una vigencia o plazo para utilizar su producto, entre otras cosas.

Por otro lado, muchas empresas fabricante de software, a cambio de la compra de sus licencias, daban a sus clientes algunos beneficios tal y como la garantía de sus productos y algunas veces ofrecían soporte técnico durante un periodo determinado de tiempo.

Aunque no toda la gente se encontraba muy contenta con esto, tuvieron que formarse el hábito de pagar por una licencia que les permitiera el uso del software que necesitaban.

Pero también hubo personas con una filosofía muy diferente, una mentalidad en contra del tener que pagar por un software para poder utilizarlo. Supongo que todos hemos escuchado algunas vez hablar sobre los Hackers o los Crackers.
Su filosofía entre ellos es diferente, pero básicamente buscan un objetivo en común, que la información siempre este disponible para las personas que la necesiten. Los Crakers por ejemplo, se dedican básicamente a desarrollar a través de ingeniería inversa, programas o modificaciones del código original de un software comercial, para hacer de éste gratuito y poder distribuirlo a cualquier persona, claro, bajo su propia responsabilidad.

Así que para 1984, Richard Stallman, un norteamericano nacido en la ciudad de Nueva Cork, con estudios de Física en la Universidad de Harvard y con una filosofía Hacker, inició un proyecto que tenía como objetivo “retornar al espíritu de cooperación que prevaleció en los tiempos iniciales de la comunidad de usuarios de computadoras”. El proyecto de Stallman, llamado GNU (que significa *GNU No es UNIX)*, que consistía en un sistema operativo totalmente libre, se distribuyó bajo una licencia gratuita, que garantizaba los derechos de ejecución,

modificación, redistribución y al mismo tiempo, evitaba que posteriormente se crearan restricciones a los mismos.

Posteriormente Stallman (1985) creó la **Free Software Foundation** (Fundación de Software Libre), una organización no lucrativa, que permitió coordinar los esfuerzos de miles de desarrolladores que compartían los ideales de la creación de software de libre distribución.

Hoy en día, podemos encontrar en Internet una cantidad inimaginable de proyectos de software libre que se encuentran a cargo de cientos de desarrolladores en todo el mundo. Un buen lugar para empezar y comprobar lo que les digo, es SourceForge:

http://sourceforge.net/

¿Software libre o código abierto?

No resulta fácil definir el término "software libre" o "software de código abierto" en pocas palabras, debido a la multitud de categorías y variantes que existen. Pero tampoco es tan complicado ya que la idea en sí es simple. Cuando hablamos en inglés de software libre (free software) hay una peligrosa ambigüedad debido a que la palabra *free* significa tanto "libre" como "gratis". Por eso se está utilizando mucho el término de "código abierto" (Open source). Afortunadamente en español no tenemos esa ambigüedad y por eso utilizamos básicamente software libre, dado que no induce a pensar en "software gratis".

Antes de entrar en más detalles quiero resaltar que el software libre no tiene por que ser gratis, y de hecho generalmente no lo es, o al menos, no completamente.

A continuación voy a describir los rasgos que caracterizan al software libre:

- *Los usuarios pueden utilizar el software como deseen*, para lo que quieran y en las computadoras que sean.
- *Los usuarios pueden tener el software a su disposición para adecuarlo a sus necesidades*, esto es, poder analizar su funcionamiento, corregir errores, mejorarlo, etc.
- *Lo usuarios pueden distribuir el software a otros usuarios***.** Esta redistribución debe de ser gratuita, o mediante contraprestaciones no fijadas de antemano.
  Quiero resaltar que estamos hablado de libertades y no de obligaciones, es decir, los usuarios son libre de redistribuir el software más no es ninguna obligación que lo hagan. De la misma manera, pueden modificar

el código fuente para adaptar el software sus necesidades pero tampoco es una obligación hacerlo.

- *Los usuarios de una parte del software deben tener acceso al código fuente.* El código fuente de un programa generalmente escrito en un lenguaje de programación de alto nivel, es absolutamente necesario para poder entender la funcionalidad, para poder modificarlo y mejorarlo. Si los programadores tienen acceso al código fuente de un programa pueden analizar y estudiar su funcionamiento para conocerlo más detalladamente y trabajar con él, tal y como el autor lo haría.

  Paradójicamente, si se quiere garantizar esta libertad para un software dado, con la legislación actual, es necesario protegerlo con una licencia que imponga restricciones a la forma de cómo pueda usarse y distribuirse. Esto genera una controversia porque en ciertos círculos, porque se considera que dichas licencias hacen que el software distribuido sea menos libre.

Para completar el punto anterior, quiero mencionar que las licencias garantizan algunas de las libertades básicas de los usuarios (modificación, redistribución, y uso), aseguran algunas condiciones impuestas por los autores (por ejemplo, cita del autor en trabajos derivados) y garantiza que los trabajos derivados sean también software libre. Los autores pueden escoger cómo proteger su software a través de las diferentes licencias de acuerdo al grado con el que deseen cumplir sus objetivos y detalles que quieran asegurar.

Algunas licencias de software libre más comunes son:

- BSD (Berkley Software Distribution)
- GLP (GNU General Public License)
- LGPL (GNU Lesser General Public License)
- MPL (Mozilla Public License)
- Astísticas (una licencia bajo la cual se distribuyó Perl)

Para más información podemos visitar la página oficial de GNU en español:

http://www.gnu.org/philosophy/free-sw.es.html

Objetivo.

Mi objetivo es compartir con aquellas personas que se encuentran dentro del mundo de la computación y el desarrollo de aplicaciones, la posibilidad de utilizar una alternativa de software que satisfaga sus necesidades y les permita obtener mayores frutos y/o beneficios.
Mostrarles que el desarrollo de aplicaciones profesionales a través de software libre, es factible, seguro y confiable.
Teniendo en cuenta lo antes dicho, nos enfocaremos en el desarrollo de una aplicación Web; una “Revista Digital” para ser mucho más específico. Haciendo uso en todo momento de software libre y su extensa documentación, como es el caso de un servidor Web llamado Apache, encargado de gestionar nuestras páginas HTML; una base de datos muy popular en el medio llamada MySQL y otras aplicaciones.

Cabe destacar que este desarrollo nos fue asignado como proyecto final en el “*Diplomado de Desarrollo e Implementación de Sistemas a través de Software Libre”*, el cual esta dividido básicamente en seis módulos:

1. Sistema Operativo Linux
2. Creación de páginas HTML
3. Administración del servidor Apache Web Server.
4. Desarrollo de páginas dinámicas con PHP.
5. Bases de datos bajo Linux.
6. Seguridad computacional.

Por lo que este trabajo pretende llevar el mismo orden y los mismos entregables al final de cada capítulo.

Sistema Operativo Linux

Capitulo I

Capitulo 1: Sistema Operativo Linux

Historia de Linux

Linux ha sufrido una evolución impresionante desde su nacimiento hasta la actualidad. Muchas personas de todo el mundo han participado en su evolución. Linux es una versión de UNIX de libre distribución ya que ha sido registrada bajo los términos de la Licencia Pública General (GNU), General Public Licence o GPL. Esta licencia escrita por Free Software Foundation (FSF), está diseñada para evitar que alguna persona restrinja la distribución del software; aunque en realidad define dos cosas: la obligatoriedad de poner a disposición de todo el mundo el código fuente, y la posibilidad de que cualquiera pueda regalar versiones de éste software.

Su creador es Linus Tovarlds, un finlandés egresado de la Universidad de Helsinki, que se inspiró en Minix, un pequeño UNIX desarrollado por Andrew S. Tanenbaum, para desarrollar este proyecto que intentaba, en un principio explorar el chip 386. Este sistema operativo fue escrito en lenguaje ensamblador.

Una de las primeras declaraciones de su autor, decían:

Linus Tolvard

"*Comencé a utilizar el C tras escribir algunos drivers, y ciertamente se aceleró el desarrollo. En ese punto sentí que mi idea de hacer un "Minix mejor que Minix" se hacía más seria. Esperaba que algún día pudiese recompilar el gcc bajo Linux…*"

"*Dos meses de trabajo, hasta que tuve un driver de discos(con números bugs, pero que parecía funcionar en mi PC) y en un pequeño sistema de archivos. Aquí ya tenía la versión 0.01 (Agosto 1991): no era muy agradable de usar sin el driver de disquetes, y no hacía gran cosa. No pensé que alguien compilaría esta versión*."

No se anunció nada desde la versión 0.01, puesto que su código fuente nunca llego a convertirse en ejecutable, ya que sólo contenía esbozos de lo que sería el núcleo, y se asumía que se tenía acceso a un Minix para poder compilar.

Linus continuó con el desarrollo de lo que hoy es uno de los principales sistemas operativos de mercado mundial, y así un poco más adelante, concretamente, el 5 de octubre de ese mismo año (1991), Linus anunció la primera versión oficial de Linux, la versión 0.02. En esta versión ya se podía ejecutar una shell,

concretamente bash, la shell de GNU y un compilador de C (gcc, compilador de C de GNU).

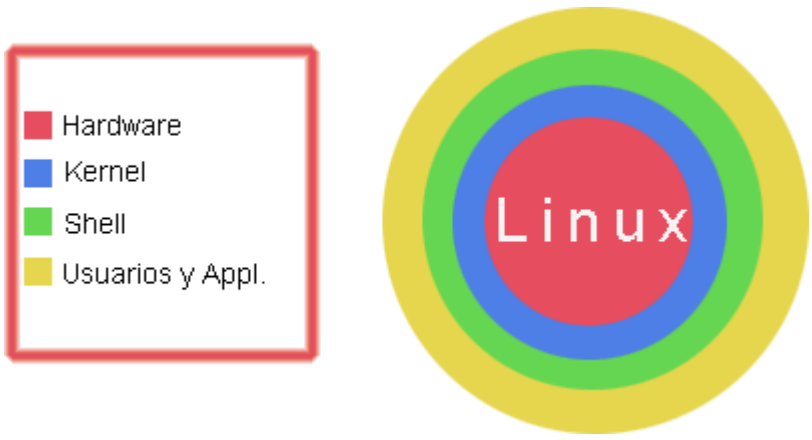
Se puede decir, que en este momento de la evolución de Linux, éste sistema operativo era un juguete para los hackers, ya que no había nada sobre soporte a usuarios, distribuciones, documentación, ni nada parecido.

Tras la versión 0.03, Linux salto a la versión 0.10, al tiempo de que más gente de todo el mundo empezaba a participar en su desarrollo.
Después de numerosas versiones en marzo de 1992 se alcanzó la versión de 0.95, reflejando la esperanza de tener lista la versión oficial, la versión 1.0 que finalmente se anunció a finales de 1993 y a partir de aquí, la evolución de éste sistema operativo ha sido vertiginosa. Se han desarrollado diferentes versiones del mismo (las más conocidas son Debian, RedHat, Su.S.E., Slackware, etc.) involucrando a personas de todo el mundo.

Hoy en día se puede decir que Linux es un clon de UNIX completo, capaz de ejecutar cualquier utilidad desarrollada para UNIX, como entornos gráficos, protocolos de comunicación, editores de texto, servicios de Internet, etc.

Arquitectura de Linux

Linux tiene una arquitectura por niveles o capas. La capa más interna es el hardware, que se compone de los discos, el CPU, memoria y otros dispositivos. Sobre el hardware, encontramos corriendo el sistema operativo. El kernel es el corazón del sistema operativo y es iniciado cada vez que el sistema es iniciado. Maneja todos los recursos del sistema y los presenta al usuario de una manera coherente. Controla el acceso al sistema y a sus archivos, asigna recursos a las distintas actividades que realiza la computadora, administra la memoria de la computadora, maneja todos los dispositivos de E/S, etc. El Shell, mejor conocido como el intérprete de comandos. Es por medio de éste que el usuario se puede comunicar con el sistema. En la capa exterior encontramos las aplicaciones de los usuarios y al usuario mismo.



Características de Linux

Linux es un sistema operativo que posee muchas características que lo hacen uno de los principales sistemas operativos en el mercado. Podemos destacar algunas de ellas:

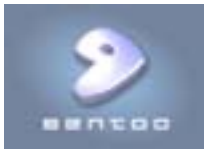
- Multitarea.
- Multiusuario.
- Multiproceso.
- Multiplataforma.
- Es seguro.
- Es robusto, ya que soporta la mayoría de dispositivos de entrada y salida, tiene soporte para la mayoría de los File System existentes y posee una gran cantidad de utilerías para comunicaciones y red.
- Shells programables.
- Portabilidad de sistemas abiertos.

Principales distribuciones

En realidad existen varias ediciones de Linux que están basadas en una versión estándar. Todas las distribuciones utilizan el mismo kernel, aunque puede estar configurado de modo diferente. Lo que sucede es lo siguiente, algunas empresas y grupos obtienen Linux y lo empaquetan con software para Linux con pequeñas diferencias, después de esto, lo distribuyen en CD-ROM o le dan la posibilidad al usuario de descargar una imagen de su distribución desde Internet. Las versiones más recientes pueden incluir versiones actualizadas de programas o incluso, nuevo software.

En la siguiente tabla, se muestran algunas de las distribuciones más populares:

Logo	Distribución	Descripción
	Red Hat	Esta distribución es una de las más famosas por su facilidad de instalación y uso. Creo el sistema de paquetes RPM y aporta gran cantidad de los avances para el escritorio Gnome. Existen dos versiones para Red Hat: • Fedora RedHat (versión gratuita) • Redhat Enterprise (versión comercial con soporte técnico). • http://www.redhat.com/

	Suse Linux	Originalmente es una distribución alemana que está basada en RedHat y actualmente es una de las distribuciones de más rápido crecimiento. Su distribución incluye tanto KDE como Gnomo, WordPerfect, Star Office y KOffice. http://www.suse.de/es/
	Mandrake	Esta es una de las distribuciones más populares ya que cuenta con muchas de las características de Red Hat. Esta distribución se centra en proporcionar mejoras actualizadas, así como una instalación y configuración sencilla de la interfaz de usuario (GUI). http://www.linux-mandrake.com/es/
	Slackware	La distribución de Slackware presta especial atención a continuar ajustándose a UNIX tanto como sea posible. Actualmente solo soporta las plataformas Intel. http://www.slackware.com/
	Debian	Es un proyecto mantenido por un grupo de programadores voluntarios. Es un software no comercial http://www.es.debian.org/
	Ubuntu	Es una distribución que está obteniendo fama rápidamente ya que soporta varias plataformas, ofrece soporte gratuito y cuenta con más de 16000 programas. http://www.ubuntulinux.org/
	Gentoo	Esta distribución de Linux es muy apropiada para ser utilizada en servidores, pero tiene un inconveniente, es una distribución de difícil instalación y requiere de conocimientos previos sobre la utilización de un sistema Linux. A pesar de esto, también es una distribución popular. http://www.gentoo.org/

Existe un sitio en Internet, que contienen la mayoría de las distribuciones Linux en están disponibles para la comunidad y las empresas. Aquí, nosotros

podemos descargas las distribuciones en formato ISO, que posteriormente podemos grabarlas en un CD-ROM e instalarlas en nuestros equipos.

Accedemos al sitio a través de la siguiente liga: http://linuxiso.net

El sistema de archivos

Linux es un sistema muy sólido y con una estructura lógica clara y bien definida, independientemente de cada distribución. La estructura de directorios de un sistema Linux es más o menos la siguiente (cada distribución puede tener pequeñas variaciones, pero en lo básico no cambia nada).

Sistema de Archivos	
bin/	Binarios para el modo consola
boot/	Archivos de arranque del kernel compilado
dev/	Archivos de acceso a dispositivos
etc/	Archivos de configuración del sistema
home/	Carpetas de los usuarios del sistema
lib/	Librerías y módulos del kernel
misc/	Varios
mnt/	Directorio de montaje de unidades
proa/	Directorio de archivos de procesos del sistema
root/	Directorio del usuario administrador
sbin/	Binarios del sistema
tmp/	Directorio de archivos temporales
usr/	Archivos de configuración de aplicaciones y programas de los usuarios del sistema
var/	Bitácoras y archivos de aplicaciones

En el sistema de archivos de Linux podemos encontrar tres tipos de archivos:

- **Archivos ordinarios**: Archivos que sólo contienen datos.
- **Archivos especiales**: Archivos que permiten el acceso a dispositivos como terminales, dispositivos E/S, etc.
- **Directorios**: Contienen información sobre un conjunto de archivos que se emplean para localizar a un archivo por su nombre.

Terminales

Linux no es un entorno gráfico, ni un gestor de ventanas, de hecho, no es nada gráfico, tan sólo se trata de un programa que permite administrar los recursos de nuestra computadora de un modo multiusuario y con una compatibilidad digna de elogio. Sobre él y dada su conectividad, se ejecutan infinidad de aplicaciones, entre ellas los entornos gráficos, éstos como una aplicación más. Por lo que el modo texto, es algo muy unido a Linux (una copia de los sistemas UNIX), ha estas les llamamos terminales y son de las más potentes que existen en el mercado.

Redirecionamiento y utilización de tuberías (pipes)

La entrada y salida estándar, es un concepto que permite al usuario, un manejo más sencillo del flujo de la información de los programas que se ejecutan en su sistema.

Cuando nosotros ejecutamos un programa en nuestra terminal, por ejemplo, imaginemos un script que al ejecutarlo nos muestre cierta información sobre un usuario determinado. En este caso, la **entrada estándar** de nuestro programa vendría desde el teclado y sería el *username* del usuario que queremos consultar. Una vez procesado este dato por nuestro programa, la información resultante se dirigirá a la **salida estándar** que regularmente es nuestro monitor. En caso de que el usuario que ingresemos fuera inválido (porque no estuviera registrado en nuestro sistema), se producirá un error que podremos visualizar en la pantalla de nuestro monitor, y sería encaminado por el **error estándar**. Esto significa que el error y salida estándar comparten el mismo dispositivo salida (monitor), pero la información viaja por diferente canal, es decir, no se mezclan en el trayecto y nosotros podríamos controlar el flujo de esta información por separado.

Ahora bien, como hemos visto existen tres flujos estándar de E/S, los cuales son: **entrada estándar (stdin), salida estándar (stdout) y error estándar (stderr)**.

La gran virtud de este concepto, es que podemos manipular cada uno de estos flujos individualmente y podemos cambiar la dirección hacia un archivo. Tomando el ejemplo anterior, en lugar de ver la información del usuario que solicitamos en pantalla, podemos redireccionar la salida estándar de nuestro programa a un archivo de texto y así, podríamos almacenar esa información con la intención por ejemplo de hacer un reporte.

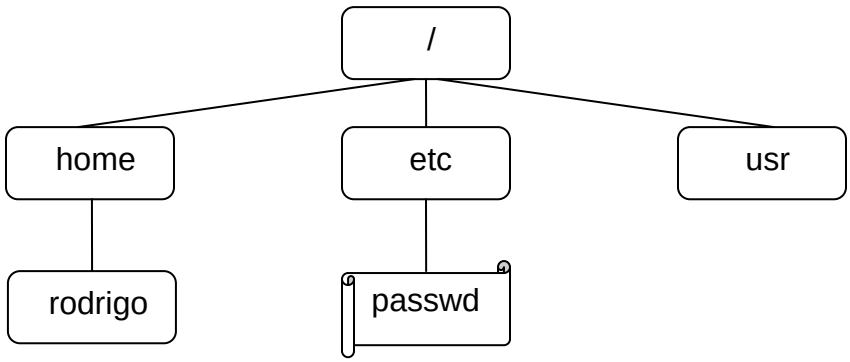
Los descriptores de archivos que utilizaremos para realizar este redirecionamiento son los siguientes:

Símbolo	Significado
<	Toma la stdin de un archivo ($programa < archivo)
>	Envia stdout al archivo. Si el archivo existe y tiene información, la borra primero y después escribe en él. Lo denominan redireccionamiento destructivo.
>>	Envia stdout al archivo. Si el archivo existe y tiene información, inserta la nueva información al final de éste, evitando borrar alguna información previa. Si el archivo no existe lo crea.
2>	Envia el stderr a un archivo. Si este no existe lo crea.

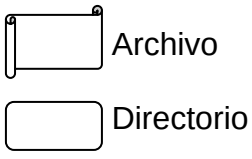
Rutas absolutas y relativas.

Las rutas absolutas y relativas son básicamente dos formas diferentes de establecer una ruta entre dos puntos (origen y destino). Esta ruta nos ayudará a realizar algunas tareas como son el cambiarnos a otro directorio, crear un vínculo entre dos archivos, copiar una archivo en otro directorio, etc.

Vamos a ver un ejemplo para entender mejor éste concepto. Imaginemos la siguiente estructura de archivos:



Donde:



Ahora bien, nosotros deseamos copiar el archivo **passwd** al directorio **rodrigo**. Vamos a apoyarnos del comando **cp** (copy), que es un comando de Linux utilizado para copiar archivos. Su sintaxis es la siguiente:

$> cp [-opciones] <Ruta origen> <Ruta destino>

Necesitamos decirle al comando dónde esta el archivo que vamos a copiar y la ubicación destino para éste. Vamos a realizar este ejemplo utilizando los dos tipos de rutas.

Comenzaremos con las rutas absolutas. En este caso el calificativo *absoluto***,** significa *independiente o que excluye cualquier relación*.
Supongamos que nos encontramos en el sistema de archivos, dentro del directorio *usr*. Como dije anteriormente, nosotros queremos copiar el archivo *passwd* al directorio *rodrigo***,** así que vamos a definir la ruta absoluta de estos archivos.

Para el archivo *passwd,* la ruta absoluta (empezando por el origen) es: **/etc/passwd.** Esto quiere decir que comenzamos por el directorio raiz '*/*', subimos al directorio '*etc'* y dentro de él encontramos el archivo '*passwd'*.

Para el directorio rodrigo, la ruta absoluta sería: **/home/rodrigo/**. Al igual que el archivo passwd, comenzamos por la raiz '**/**', subimos al directorio '*home***'**, finalmente subimos al directorio '*rodrigo***'**.

Por lo tanto, para copiar el archivo a su nueva ubicación, haríamos lo siguiente:

$> cp /etc/passwd /home/rodrigo/

Así de sencillo. Lo importante al momento de utilizar rutas absolutas es **"siempre comenzar desde el directorio raíz (/)".**

Ahora vamos a copiar el archivo utilizando rutas relativas. Seguimos posicionados en el directorio **usr**. Para utilizar las rutas relativas nos vamos a apoyar de dos elementos:

- El punto (.) indica el directorio actual.
- Los dos puntos (..) indica el directorio padre.

Las rutas relativas a diferencia de las rutas absolutas NO comienzan desde el directorio raíz (*/*), sino que se definen a partir del directorio en donde nos encontramos posicionados, o desde la ubicación de un archivo en específico. Utilizamos el ejemplo anterior, la forma de copiar el archivo sería.

Nota: Nos encontramos en el directorio usr. Este es nuestra referencia relativa.

$> cp ./../etc/passwd ./../home/rodrigo

La ruta origen la empezamos con un punto (./), esto significa que a partir del directorio actual (**usr**) definiremos la ruta para llegar al archivo passwd. A continuación colocamos dos puntos (./../) y de esta manera hacemos alusión al directorio padre de nuestro directorio actual, que en este caso es el directorio raíz (/). Como puedes ver, estamos ascendiendo a través del sistema de archivos. Ahora vamos a descender a través de él para llegar al directorio etc, para ello solo colocamos el nombre del directorio (./../etc). Finalmente indicamos el nombre del archivo que se encuentre en este directorio (./../etc/passwd).

Para definir el destino realizamos el mismo análisis que utilizamos para definir el origen.

A simple vista parece más complicado, pero en la vida práctica la utilización de rutas relativas es más común, sobre todo en la creación de aplicaciones Web como lo veremos en capítulos posteriores.

Comandos y utilerías básicas

Linux posee una gran cantidad de comandos y utilerías que nos permiten manipular el flujo de información, la administración del sistema, la programación y creación de scripts en los diferentes tipos de shells, entre otras aplicaciones.

A continuación voy a

A continuación, voy a mencionar algunos de los comandos básicos utilizados en la mayoría de las distribuciones de linux.

Nota: Las opciones de cada comando pueden variar de una distribución a otra.

Comando: **ls**

Lista los archivos existentes en un directorio.

Sintaxis

ls [opciones] [archivo(s) o directorio(s)]

Opciones:

-l Listado largo (permisos, número de links, propietario, grupo, tamaño, fecha y nombre de archivo).

-s Muestra el tamaño
-a Muestra archivos ocultos del tipo .archivo

-F Diferencia directorio (/), ejecutables (*) y links (@)
--color Diferencia colores para directorios, ejecutables y links

Comando: **rm**

Borra archivos

Sintaxis

rm [opciones] <ruta>[archivo(s) o directorio(s)]

Opciones:

-r Borra directorios recursivamente
-f Borra en modo forzado
-i Pide confirmación

Comando: **mkdir**

Crea directorios

Sintaxis

mkdir [opciones] <directorio…>

Opciones:

-p Crea los directorios padre que falten para cada argumento directorio

Comando: **rmdir**

Borra directorios *vacíos*

Sintaxis

rmdir [opciones] <directorio…>

Opciones:

-p Borra los directorios que se encuentren dentro de él y que estén vacíos

Comando: **cd**

Cambia de directorio

Sintaxis

cd [directorio…]

Comando: **pwd**

Muestra la ruta actual del directorio

Sintaxis

pwd

Comando: **du**

Muestra información acerca del espacio ocupado por un directorio y subdirectorios

Sintaxis

du <Archivo(s)>

-h Añade una letra indicativa del tamaño, k (kilobytes), m (megabytes), g (gigabytes).

Comando: **cp**

Copia archivos.

Sintaxis

cp [opciones] <origen> <destino>

-p Conserva los permisos originales
-f Borra archivos destinos si se requiere (sobre escribe)
-r Copia archivos recursivamente

Comando: **mv**

Mueve archivos de lugar o renombra archivos

Sintaxis

mv [opciones] <origen> <destino>

-i Pide confirmación cuando el destino existe

Comando: **date**

Consulta y definición de a hora y la fecha del sistema.

Comando: **cat**

Éste comando le el contenido de un archivo o archivos que se especifican como parámetros y reproduce ese contenido a través del canal de salida estándar. Si no se introduce ningún archivo como parámetro, el comando cat lee el canal de la entrada estándar.

Sintaxis

cat [opciones] <Archivo(s)>

-s Suprime el mensaje de error que aparecería si no existiera el archivo a leer.

Comando: **id**

Este comando proporciona el número de usuario, el número de grupo y el nombre del grupo.

Sintaxis

Id

Comando: **df**

Muestra el espacio en disco en un soporte de datos. Los datos aparecen en KB. Como Linux maneja casi siempre diferentes sistemas de archivos, los despliega por separado.

Sintaxis

df [opciones] [Archivos de dispositivos]

Comando: **jobs**

El shell administra de forma interna mediante un número de identificación todos los procesos inisiados como procesos en segundo plano. Este comando muestra éstos procesos y su identificados correspondiente.

Sintaxis

Jobs [-l] [-p] Número_de_trabajo

Comando: **uname -a**

Muestra a en la salida estándar, información del sistema.

Sintaxis

uname [opciones]

Comando: **ln**

Para cada archivo de un sistema Linux existe una cabecera de archivos en la que se guarda información general para la gestión de este archivo (tamaño, propietario, grupo, permisos, etc.). El nombre de un archivo esta directamente unido a una referencia a la cabecera de archivo correspondiente.

Tras la creación de un archivo, existe una correspondencia 1:1 entre el nombre y la cabecera del archivo. Mediante este comando, pueden asignarse varios nombres a la misma cabecera.

Sintaxis

ln [-f] [-s] <Archivo1> <Archivo2>

Comando: **more**

Este comando nos permite la visualización de un archivo de texto a través de la salida estandar. El comando more detiene la salida de datos tras una página de pantalla y muestra en la última línea de ka oantalla el indicador de comandos:

--More--

La siguiente página de pantalla se visualizará al introducir un espacio en blanco. Si se preciosa la tecla <Enter>, se mostrará solo la siguiente.

Sintaxis

more [opciones] <Archivo(s)>

Comando: **cal**

Si este comando se escribe en el shell sin argumentos, nos muestra el calendario del año en curso.

Sintaxis

cal [mes] [año]

Comando: **man**

El manual en línea contiene una descripción de todos los comandos disponibles en el sistema para los usuarios.

Sintaxis

man [opciones] [sección] comando

Comando: **ps**

El comando *ps* sin singún parámetro muestra los datos referentes a los procesos iniciados desde la terminal del usuario. Los diferentes parámetros que podemos utilizar nos permiten modificar el formato de salida y selección de los procesos.

Sintaxis

ps [opciones]

-a Muestra todos los procesos de los usuarios.
-f Muestra información con referencia a las dependencias entre procesos
-l Creación de un formato largo para desplegar la información de un proceso.
-m Visualización de los procesos de almacenamiento adicional
-x Visualización también de aquellos procesos que no tienen ningún terminal de control

Comando: **chown**

El comando chown permite modificar el propietario de uno o varios archivos. Debe indicarse el número o nombre del futuro propietario. El futuro propietario debe existir dentro del archivo /etc/passwd

Sintaxis

chown <nuevo propietario> <Archivo(s)>

Comando: **touch**

Éste comando permite modificar el momento del último acceso y la última modificación para uno o varios archivos. Si el archivo que se pasa como argumento todavía no existe, se creará. Si el comando touch se ejecuta sólo con un nombre de archivo como parámetro, del nombre del último acceso y la última modificación se establecerá en la fecha actual.

Con la opción –a se modificará sólo el momento del último acceso, con la opción –m sólo el de la última modificación .

Debe seguirse el siguiente formato:

MMDDhhmm [AA]

Sintaxis

touch [-a] [-m] <Archivo(s)>

Comando: **chmod**

Éste comando permite al propietario de los archivos modificar sus permisos lectura, escritura o ejecución. El comando chmod reconoce dos tipos de modificación de derechos:

- Notación simbólica
- Notación octal

Sintaxis

Con la notación simbólica se utiliza la siguiente sintaxis.

chmod [ugoa] [+-=] [rwxsx] <Archivo(s)>

Donde **u** representa al usuario, **g** al grupo, **o** al cualquier usuario y a es una abreviatura de **ugo**. Los derechos pueden añadirse con el símbolo de +, sustraerse con el símbolo de – o definirse como absolutos. Junto a los derechos de acceso r de lectrura, w de escritura y x de ejecución, existe también la letra s para bits de suid y sgid, que pueden introducirse con el comando chmod. El primer parámetro pueden incorporarse varias modificaciones simbólicas separadas por comas.

Por ejemplo: **%chmod go+x archivo1**

En la notación octal, a cada derecho debe de asignársele un valor octal. Pueden utilizarse los siguientes valores octales:

Valor	Representación
0	Sin permisos
1	Sólo ejecución
2	Sólo escritura
3	Escritura y ejecución
4	Sólo lectura

5	Lectura y ejecución
6	Lectura y escritura
7	Todos los permisos

Por ejemplo:
%chmod 754 MiArchivo.txt

Esto nos indica que el usuario tiene todos los permisos sobre el archivo, el grupo posee los derechos de lectura y ejecución y los demás usuario solamente podrán leerlo.

Comando: **expr**

Permite hacer operaciones matemáticas y lógicas.

Sintaxis

expr expresion

Expresión	Resultado
expr 5 + 5	10
expr 10 – 5	5
expr 10 / 2	5
expr 5 * 10	50
expr 5 % 2	1
expr \(5 + 5 \) * 20	200
expr 2 \> 3	0
expr 4 \> 3	1
expr 34 != 34	0
expr 34 \<= 20	0
expr 1 | 0	1
expr 0 \& 1	0

Comando: **wc**

Cuenta el número de palabras, líneas y caracteres de un archivo.

Sintaxis

wc [opciones] <Archivo(s)>

-m Muestra el número de bytes.
-c Muestra el número de caracteres.
-w Muestra el número de palabras..
-l Muestra el número de líneas.

Comando: **cmp**

Compara dos archivos byte por byte.

Sintaxis

cmp [-l] [-s] <archivo1> <archivo2>

-l No muestra nada, sólo da un código de salida.
-s Muestra todos los bytes y valores de todos los bytes que difieran

Comando: **diff**

Compara dos archivos línea por línea

Sintaxis

diff [opciones] <archivo1> <archivo2>

-i Descarta las diferencias entre mayúsculas y minúsculas en el contenido de los archivos.
-e Genera una secuencia adecuada para el editor ed.
-c Produce un listado de diferencias, separadas en tres campos. El primer campo tiene el nombre de los archivos que esta comparando y la fecha de de creación de ambos. El segundo y tercer campo, son el contenido de los archivos que se están comparando. Se utilizan símbolos como **¡ , + , -** para darnos datos acerca del contenido de nuestros archivos.
-b Toma los espacios y tabuladores por igual al momento de comparar.
-w Ignora todos los espacios en blanco (espacios y tabuladores). ejemplo, `if ( a == b )' will compare equal to`if(a==b)'.

Comando: **comp**

Compara los archivos ordenados, **archivo izquiero** y **archivo derecho**, línea por línea.

Sintaxis

comp [opciones] <archivo izquierdo> <archivo derecho>

-1 Suprime las líneas que solo están en el izquierdo.
-2 Suprime las líneas que solo están en el derecho.
-3 Suprime las líneas que aparecen en los dos.

Comando: **find**

Es una herramienta extremadamente potente. Pasa por los directorios especificados y genera una lista de archivos que cumplen los criterios que se han indicado. Los archivos pueden ser comparados por nombre, tamaño, hora de creación, hora de modificación y muchos más criterios. Incluso puede

ejecutarse un comando en los archivos comparados cada vez que se encuentre un archivo.

Sintaxis

find <ruta de acceso> <expresión>

-name archivo	Indica a find cual es el archivo que hay que buscar; este está encerrado entre comillas. Pueden utilizarse caracteres comodín (* y ?).
-perm modo	Compara todos los archivos cuya modalidad concuerda con el valor numérico de modo. Hay que comparar todos los modos, no solo lectura, escritura y ejecución . Si va precedido de un signo de negativo (-), significa todo lo que no sea ese modo.
-type *x*	Compara todos los archivos cuyo tipo sea ***x*** , sea c (significado), b (especial de bloque), d (directorio), p (conducción por nombre), l (enlace simbólico), s (zócalo) o f (archivo regular).
-links n	Compara todos los archivos con un numero **n** de enlaces
-size n	Compara todos los archivos de **n** bloques de tamaño (bloques de 512 bytes, bloques de 1KB si se pone k a continuación de la n).
-group id-grupo	Compara todos los archivos cuyo identificador de grupo es **id-grupo**. Puede ser el valor numérico o el valor de entrada del grupo.
-user id-usuaro	Compara todos los archivos cuyo identificador de usuario es **id-usuario**. Puede ser el valor numérico o el valor de entrada del usuario.
-atime n	Compara todos los archivos a los que se ha accedido durante los últimos **n** días.
-mtime n	Compara todos los archivos que se han modificado durante los **n** días anteriores.
-exec cmd	El comando **cmd** se ejecuta por cada archivo comparado. Se utiliza la notación para indicar donde debe aparecer el nombre del archivo en el comando ejecutado. El comando debe determinarse por na barra inclinada inversa seguida de punto y coma (\;), por ejemplo –exec ls –d {} \; . En este ejemplo el comando ls se ejecuta con el argumento –d y cada archivo se para a ls en el lugar donde se encuentra. {}.
-ok	Similar a –exec con la diferencia que de te pregunta si deseas ejecutar el comando. Espera un una “y” para aceptar o una “n” para cancelar.
-print	Indica la ruta donde se encuentra el archivo.
-a	Funciona como una **AND**

!	Operador **NOT**
-o	Funciona como una **OR**
-newer archivo	Compara todos los archivos que se han modificado más recientemente que **archivo**

Comando: **finger**

Despliega información detallada de los usuarios que se encuentran conectados al sistema.

Sintaxis

finger [opciones] [usuarios]

Si utilizamos el comando sin argumentos, el comando finger nos despliega la siguiente información por cada usuario actualemente conectado al sistema.

- Nombre del usuario (login)
- Nombre del usuario completo
- Terminal
- Tiempo muerto (idle time)
- Hora de conexión.
- Hostname.

Comando: **hostname**

Despliega información sobre el nombre del host al que estamos conectados.

Sintaxis

hostname

Comando: **uname**

Despliega cierta información sobre el sistema operativo en el canal de salida estándar.

Sintaxis

uname [opciones]

-s	Muestra el nombre del sistema operativo (por defecto).
-r	Número de release del sistema operativo.
-v	Numero de versión del sistema operativo
-m	Nombre del hardware del sistema operativo.
-a	Visualiza todo la información sobre el sistema.

Comando: **mail**

Este comando permite el intercambio de mensajes entre usuarios. Sin un nombre de usuario como parámetro, el comando mail busca los mensajes entrantes. Si se solicita un mensaje de usuario como parámetro, se solicitará y enviará el mensaje correspondiente. Los mensajes se guardarán en archivos diferentes para cada usuario. Cada mensaje leído se compiará en otro archivo cuyo nombre se define mediante la variable de shell llamada MBOX (por defecto mbox). El comando mail muestra mediante un indicador de comandos (?) que pueden introducirse ordenes para el manejo y la administración de mensajes. La estructura del comando tienen el siguiente aspecto:

Orden [lista de mensajes] [argumentos]

Cada mensaje tiene un número de mensaje unívoco. Una lista de mensajes se compone de un número de mensaje o de uno de los siguientes caracteres.

Caracteres	Significado
$	El ultimo mensaje leido
*	Todos los mensajes
n-m	Todos los mensajes desde el número “n” al número “m”.
Nombre	Todos los mensajes que provienen del usuario Nombre.

Para la administración de mensajes pueden, entre otros, utilizarse los siguientes comandos. En el resumen de ordenes aparece el lugar de Lista_de_mensajes solo la abreviatura de “NI”:

Comando	Resultado
¡Comando	El comando indicado se ejecutara en el shell independientemente.
=	Muestra el número de mensaje actual.
?	Resumen de comandos.
chdir Directorio	El directorio indicado debe ser del directorio actual.
copy [NI] Archivo	Los mensajes de la lista de mensajes deben copiarse en el archivo. A diferencia del comando save esto no volverá a indicarse.
next [NI]	Visualización del mensaje siguiente. Si se especifica un sector, se mostrará el primer mensaje de dicho sector.
print [NI]	Visualización de los mensajes especificados.
quit	Finalización del comando mail. Los mensajes leidos se guardarán en el archivo local de mensajes (mbox), los no leídos se guardarán en el primer archivo de mensajes.
reply [NI]	Envía un mensaje al remitente.

save [NI] Archivo	Guarda los mensajes en u archivo. Todos los mensajes guardados se eliminarán del archivo de mensajes.
set [Nombre = [Valor]]	Definición de un variable con el comando mail. Esta variable mail no tiene nada en común con las variables del shell.
top [NI]	Visualización de la primera línea del mensaje.
undelete [NI]	Se recupera el mensaje indicado. Solo pueden recuperarse los mensajes borrados en la sección actual.
delete [NI]	Los mensajes deseados se eliminarán.
edit	Para los mensajes definidos se ejecutará el editor definido por la variable EDITOR.
exit	Salida del comando mail.

Estas órdenes no pueden utilizarse durante la entrada de mensajes. Sin embargo, si la línea de entrada comienza con el signo de tilde (~), pueden utilizarse distintas ordenes abreviadas.

La orden abreviada ~f reproduce una lista de las posibles letras de comando que puede aparecer tras el signo tilde.

Si se ejecuta mail con el nombre de usuario como argumento, se envía un mensaje a éste. Para ello se solicitará el “Asunto” y, a continuación, deberá introducirse el texto enviar hasta introducir un punto en un línea vacía.

Administración básica del sistema

Cada distribución de Linux tiene sus propias herramientas para administrar su sistema. Por ésta razón, pretendo mostrar en ésta parte de administración básica de Linux, únicamente elementos de configuración comunes a todas las distribuciones.

Y ya que tocamos el punto de las diferentes distribuciones de Linux, vamos a conocer algunas de ellas.

El usuario root

Para poder realizar modificaciones de administración en el sistema, es necesario tener derechos de acceso, como la contraseña correcta, que permite iniciar sesión como usuario root. Debido a que el superusuario tienen la capacidad de cambiar cualquier elemento del sistema, dicha contraseña debe mantenerse en secreto y en un lugar seguro, cambiarse con frecuencia y facilitarla sólo a aquellos usuarios cuya tarea sea precisamente la administración del sistema.

Si hemos iniciado sesión como un usuario común y posteriormente queremos cambiarnos a la cuenta de root, podemos utilizar dos comandos que nos ayudarán a realizar nuestro propósito: **login y su**.

Veamos un ejemplo utilizando el comando **su**.

```
Rodrigo>su –
Rodrigo>password: *******
Root>
```

El guión medio (-) posterior al comando *su*, le indica a éste, que deberá leer los archivos de configuración del shell dentro del directorio principal del usuario y así, cargar todas las configuraciones y variables de ambiente de dicha cuenta.

El comando *su* también nos permite cambiarnos la cuenta algún usuario del sistema. Para hacer esto, debemos indicarle al comando '*su'* el *username* de la cuenta de usuario, por ejemplo:

```
Rodrigo>su - juan
Rodrigo>password: *******
Juan>
```

El comando **login** es equivalente al comando su -, simplemente escribimos el comando y el *username* de la cuenta de usuario, por ejemplo:

```
Rodrigo>login root
Rodrigo> password: ******
Root>
```

¿Cómo cambiar la contraseña de root u otro usuario?

Únicamente el usuario *root* tiene el poder de cambiar su contraseña y la de cualquier usuario que esté dado alta en el sistema. Con respecto a los demás usuarios, sólo tienen el permiso de utilizar éste comando para cambiar su propia contraseña.

Para poder llevar a cabo el cambio de contraseña, necesitamos utilizar el comando passwd como se muestra en el siguiente ejemplo:

$>passwd [nombre de usuario]

Solamente el usuario root, puede especificar el nombre de un usuario registrado en el sistema, después del comando passwd, ya que sólo él, puede cambiar lo contraseña de cualquier usuario.

Por ejemplo, el administrador quiere cambiar la contraseña del usuario rodrigo, así que lo haría de ésta manera:

```
root> passwd rodrigo
root>New password: **********
root>Retype new password: *********
root>Change successful
```

Niveles de ejecución de Linux

En los sistemas Linux existen siete niveles de ejecución numerados del 0 al 6. Al arrancar el sistema, se entra siempre por el nivel de ejecución predeterminado.
En la siguiente tabla se muestran los diferentes niveles de ejecución con su respectiva descripción:

Nivel de ejecución	Descripción
0	Apaga el sistema por completo (No se configure éste nivel de ejecución como predeterminado).
1	Modo administrativo de un solo usuario. Este nivel de ejecución no permite el acceso a otros usuarios, pero permite el acceso como root a todo el sistema de archivos multiusuario. No se ejecutan scripts de inicio.
2	Nivel de ejecución multiusuario sin servicio de red como NFS, xinetd y NIS. Lo mismo que el nivel tres pero sin red).
3	Este nivel de ejecución el predeterminado en un sistema Linux y se conoce como "*estado modo texto*". Es un modo multiusuario pleno con inicio de sesión en interfaz de línea de comandos. Permite utilizar los servios de red.
4	No se utiliza
5	Modo multiusuario pleno con arranque en una sesión X e inicio de sesión gráfico (igual que el nivel tres pero con inicio de sesión gráfico).
6	Reinicio del sistema. Apaga y vuelve a iniciar el sistema (no se configure éste nivel de ejecución como predeterminado).

Cada nivel de ejecución puede ser útil si surgen problemas en un nivel determinado. Por ejemplo, si la tarjeta gráfica no está instalada correctamente, cualquier intento por iniciar la sesión en el nivel 5 fracasaría, ya que éste nivel arranca inmediatamente la interfaz gráfica. Así que iniciaríamos en el nivel de ejecución 3 y a través de la interfaz de línea de comandos podríamos solventar el problema de la instalación de la tarjeta gráfica.

Cuando se inicia el sistema, el archivo **/etc/inittab** le indica a Linux en que nivel de ejecución debe de iniciar la sesión. Es conveniente darle un vistazo a éste archivo, ya que nos informa sobre los niveles de ejecución y sobre como cambiarlos.
Podemos utilizar el comando **runlevel** para conocer el nivel de ejecución en el que nos encontramos en un momento dado y con el comando **telinit** podemos cambiar de un nivel de ejecución a otro en cualquier momento.

Por ejemplo:

```
root>telinit 2
```

Administración de usuarios

El administrador del sistema debe encargarse de la administración de sus usuarios. Como administrador, puede crear nuevo usuarios, eliminarlos, crear y eliminar grupos de usuarios, cambiar sus permisos de acceso tanto para los usuarios como para los grupos. Además, se tiene el control sobre los archivos de control predeterminados que se copian a cada cuenta en el momento de su creación.

Archivos de configuración de usuario.

Existen determinados archivos predeterminados llamados archivos de configuración, y directorios para configurar la nueva cuenta.
La siguiente tabla contiene una lista de los nombres de ruta.

Directorio o archivos	Descripción
/home	Aquí se encuentra el directorio principal del usuario
/etc/skel	Este directorio contienen los archivos para el shell de inicio de sesión, como el *.bash_profile*, *.bashrc*, y *.bash_logout*. Incluye muchos directorios y archivos de configuración de usuarios, como .kde para KDE y Desktop para Gnome.
/etc/shells	Contiene los shell de inicio de sesión, como bash, Esch, csh, etc.
/etc/passwd	Contiene las contraseñas del usuario.
/etc/group	Contiene el grupo al que pertenece el usuario.
/etc/shadow	Archivo de contraseñas cifrado.
/etc/gshadow	Archivo de contraseñas cifrado para grupos.
/etc/login.defs	Definiciones de inicio de sesión predeterminadas para los usuarios.

Cómo agregar y quitar usuarios con useradd, usermod y userdel.

Linux cuenta con los comandos useradd, usermod y userdel para administrar las cuentas de usuario. Todos estos comandos recogen información a través de opciones que se especifican en la línea de comandos. Si no se especifican los valores, éste comando tomará las opciones predeterminadas.

El comando **useradd** nos permite la creación de una cuenta de usuario, a continuación se mencionan algunas de sus opciones así como un ejemplo de cómo utilizarlas:

Opción	Descripción
-d dir	Define el directorio principal del usuario
-D	Muestra la configuración predeterminada de todos los parámetros.
-e mm/dd/aa	Establece la fecha de expiración para la cuenta (no cuenta con ninguna como omisión)
-f	Establece el número de días que las cuenta permanecerá activa hasta que expire su contraseña.
-g *grupo*	Permite asignar un grupo
-m	Crea el directorio principal del usuario en caso de que no exista.
-M	No crea directorio principal del usuario.
-p *contraseña*	Facilita una contraseña cifrada (crypt o MD5). Si no se especifica argumento, la cuenta queda inhabilitada inmediatamente.
-s *shell*	Configura el shell de inicio de sesión para un nuevo usuario. En Linux el valor por omisión es /bin/bash
-u *id-usuario*	Configura el ID del nuevo usuario. El valor predeterminado el valor más alto utilizado hasta ese momento.

Tabla #. Opciones para useradd y usermod

Ahora vamos a ver un ejemplo que como podríamos crear un usuario utilizando este comando:

```
root> useadd –d /home/rodrigo –g juegos –s /bin/csh –p "@ri@s" rodrigo
```

De acuerdo al ejemplo anterior, definimos el directorio principal del usuario en la siguiente ruta */home/rodrigo*, es miembro del grupo *juegos*, va a utilizar un shell de inicio de sesión llamado *csh* y asignamos una contraseña en ese momento.

El comando **usermod** permite modificar la cuenta de un usuario. Su utilización es muy sencilla, tanto la sintaxis como sus opciones son idénticas a las del comando useradd.

El comando **userdel** nos permite eliminar una cuenta del usuario del sistema. Se puede utilizar el comando *userdel* para borrar el inicio de sesión de algún usuario o utilizarlo con la opción –r para eliminar también el directorio principal de dicho usuario.

Este ejemplo borraría tanto la cuenta de inicio de sesión como el directorio principal del usuario rodrigo.

```
# userdel –r rodrigo
```

El archivo /etc/passwd

Cuando se agrega un usuario, se agrega una entrada para el mismo en el archivo /etc/passwd, que es bien conocido como el archivo de contraseñas. Cada entrada a éste archivo consta de una línea que contiene varios campos separados por signos de dos puntos. Los campos son los siguientes:

Campo	Descripción
Nombre del usuario	Nombre de inicio de sesión del usuario.
Contraseña	Contraseña cifrada del usuario.
ID del usuario	Número que representa de forma unívoca a cada usuario.
ID del grupo	Número que representa de forma unívoca a cada grupo.
Comentarios	Comentarios o información adicional del usuario.
Directorio principal	Directorio principal del usuario, también conocido como *home directory.*
Shell del usuario	Shell que tendrá el usuario automáticamente cuando inicie sesión.

Es importante conocer éste archivo ya que como administradores podemos modificar la cuenta de un usuario modificando el archivo /etc/passwd directamente o utilizando el comando usermod.

Administración de grupos de usuarios

También podemos administrar grupos con los comandos con los comandos `groupadd, groupmod y groupdel.`

Con el comando `groupadd` podemos crear nuevos grupos. Cuando se crea un grupo en el sistema, éste coloca el nombre del grupo en un archivo /etc/group y asigna al mismo un ID del grupo. Si se encuentra habilitada la seguridad

shadow, se realizan cambios en el archivo /etc/gshadow. En el siguiente ejemplo, el comando `groupadd` crea el grupo desarrollo:

```
# groupadd desarrollo
```

Se puede utilizar el comando groupdel para eliminar el grupo. En el siguiente ejemplo se elimina el grupo desarrollo:

```
# groupdel desarrollo
```

Con el comando groupmod puede cambiar el nombre o ID de un grupo. Por ejemplo, introducimos groupmod –g con el nuevo número ID y el nombre del grupo. Para poder cambiar el nombre del grupo utilizamos la opción –n, por ejemplo, si queremos cambiar el nombre del grupo “desarrollo” por “mp3s”, haríamos lo siguiente:

```
# groupmod –n mp3s desarrollo
```

Sistemas de Archivos

Nosotros podemos ver que todos los archivos del sistema Linux están conectados a través de un árbol de directorios general, pero no significa que todas las partes de ese árbol residan en un mismo dispositivo de almacenamiento como son discos duros o CD-ROMS. Los archivos de un dispositivo de almacenamiento determinado están organizados en lo que se conoce como sistema de archivos. Un sistema de archivos es un dispositivo formateado, con su propio árbol de directorios y archivos. Un árbol de directorios de linux puede albergar varios sistemas de archivos, cada uno sobre un dispositivo de almacenamiento diferente. Por ejemplo, un disco con varias particiones podría tener un sistema de archivos diferente en cada partición. Los propios archivos están organizados en un arbol continuo de directorios, comenzado en el directorio raiz (/) .

Cada sistema de archivos tiene organizado sus archivos en su propio árbol de directorios, esto podríamos verlo como un subárbol de directorios que se adjunta al árbol de directorios principal. El ejemplo más claro podría ser el sistema de archivos de un disquete en Linux. El disquete tiene su propio sistema de archivos y un conjunto de directorios y archivos (subárbol) que se adjuntarán al sistema principal de directorios. Hasta que no sean adjuntados no podremos acceder a los archivos del disquete.
A continuación les mostraré una tabla con los sistemas de archivos más conocidos:

Tipo	Descripción
ext3	Sistema de archivo para linux con soporte para nombre de archivos largos y de gran tamaño. Incluye registro de eventos.
ext2	Sistema de archivos de Linux más antiguo que el anterior. Posee las mismas características que ext3 menos el registro de eventos.
resiserfs	Sistema de archivo para Linux con soporte para nombre de archivos largos, de gran tamaño, registro de eventos y utiliza una técnica que permite rehacer por completo las operaciones realizadas por el sistema de archivos.
txiaf	Sistema de archivos Xiaf
msdos	Sistema de archivos para particiones MS-DOS (16 bits)
vfat	Sistemas de archivos para particiones con Windows 95, 98 y Millennium.
ntfs	Sistema de archivos para Windows NT, Windows 2000, Windows XP y Windows 2003.
smbfs	Sistema de archivos remoto Samba, como NFS.
hpfs	Sistema de archivos para particiones OS/2 de alto rendimiento.
nfs	Sistema de archivos para montar particiones en sistemas remotos.
umsdos	Sistemas de archivos UMS-DOS
swap	Partición o archivo de intercambio en Linux.
sysv	Sistemas de archivos System V para UNIX.
iso9660	Sistemas de archivos para montar un CD-ROM
proc	Sistema de archivos con soporte para el kernel que permite al sistema ejecutar procesos.
devpts	Pseudoterminales UNIX 98 (ttys). Sistema de archivos para la interfaz del kernel.
shmfs y tmpfs	Memoria virtual de Linux, acceso de mantenimiento a memoria compartida POSIX.

¿Cómo montar un sistema de archivos?

Adjuntar un sistema de archivos de un dispositivo de almacenamiento al árbol principal de directorios se denomina comúnmente *montar* el dispositivo. El sistema de archivos se monta en un directorio vacío del árbol de directorios principal de directorios. Una ve realizado esto, podemos cambiarnos a dicho directorio y acceder a sus archivos. Si no existe el directorio tendrá que crearlo expresamente. El directorio de la estructura de archivos al que se adjunta el nuevo sistema de archivos se denomina como *punto de montaje*. Por ejemplo, para acceder un los archivos de un disquete, primero debemos montar el disquete en un punto de montaje como podría ser la siguiente ruta */mnt/floppy*

(este directorio se encuentra en la mayoría de las distribuciones Linux y es un punto de montaje creado precisamente para montar el disquete).

El montaje de un sistema de archivos sólo puede realizarse como un usuario *root* o algún usuario que tenga privilegios de realizar ésta función. Una vez que el sistema de archivos está montado, se agregará una entrada al archivo */etc/mtab* (para algunas distribuciones se utiliza el archivo *mstab*). Ahí podremos encontrar los sistemas de archivos que están montados en el sistema en un momento dado.

¿Cómo montar y desmontar sistemas de archivos con los comandos mount y umount?

Para poder montar un sistema de archivos necesitamos utilizar el comando mount. Normalmente el montaje de particiones como discos duros sólo puede ser realizado por el usuario root, mientras que las unidades de CD-ROM o disco flexible pueden ser montadas por cualquier usuario. En la siguiente tabla podemos ver las opciones del comando mount:

Opciones de mount	Descripción
-f	Simula el montaje de un sistema de archivos. Podemos utilizar esta opción para comprobar si se puede montar un sistema de archivos determinado.
-v	Modo profuso. El comando mount muestra una descripción de las acciones que realiza.
-w	Monta el sistema de archivos con permiso de lectura y escritura.
-r	Monta el sistema de archivos con permiso de sólo lectura.
-n	Monta un sistema de archivos sin insertar una entrada en el mismo mtab o mstab.
-t tipo	Especifica el sistema de archivos que se va a montar. Podemos consultar la tabla para verificar los sistemas de archivos válidos.
-a	Monta todos los sistemas de archivos que se configuran en /etc/fstab
-o lista-de-opciones	Monta un sistema de archivos utilizando una lista de opciones. Se trata de una lista de opciones separadas por comas.

Lista de opciones para utilizar con la opción –o del comando mount.

Opciones	Descripción
async	Indica que todas la E/S al sistema de archivos deben realizarse de forma asíncrona.
auto	Indica que el sistema de archivos puede ser montado con la opción –a.
defaults	Utiliza las opciones predeterminadas: rw, suid, dev, exec, auto, nouser y async.
dev	Interpreta dispositivos especiales de bloques o caracteres en los sistemas de archivos.
kudzu	Comprueba que el dispositivo está instalado y sea accesible
noauto	Indica que el sistema de archivos sólo puede ser montado explícitamente.
exec	Permite la ejecución de archivos binarios.
nouser	Impide que un usuario normal(es decir, no root) pueda montar el sistema de archivos.
remount	Intenta volver a montar un sistema de archivos ya montado. Se utiliza normalmente para cambiar los indicadores de montaje para un sistema de archivos, especialmente para conseguir que se pueda escribir en un sistema de archivos previamente configurado como sólo lectura.
ro	Monta el sistema de archivos como sólo lectura
rw	Monta un sistema de archivos de lectura y escritura
suid	Permite que tengan efecto los bits set-user-identifier o set-group-identifier
sync	Indica que todas las entradas y salidas del sistema de archivos deben realizarse de forma asíncrona.
user	Habilita a los usuarios normales a montar el sistema de archivos. Los usuarios normales siempre tienen activadas las siguientes opciones: noexec, nosuid y nodev.
noexec	No permite la ejecución de archivos binarios en los sistemas de archivos montados.
nosuid	No permite que tengan efecto los bits set-users-identifier o set-group-identifier

El comando mount admite dos argumentos: el dispositivo de almacenamiento a través del cual el sistema operativo accede al sistema de archivos, y el directorio de la estructura de archivos al cual se adjuntará el nuevo sistema de archivos.
El punto de montaje es el directorio del árbol principal de directorios donde deben quedar adjuntados los archivos del dispositivo de almacenamiento. Es dispositivo es un archivo especial que conecta el sistema al hardware. La sintaxis del comando mount es la siguiente:

```
# mount [-opciones] dispositivo punto de montaje.
```

Por ejemplo, si queremos montar un disquete, lo hacemos de la siguiente manera:

```
# mount  /dev/fd0 /mnt/floppy
```

Ahora un CD-ROM (especificando su sistema de archivos):

```
#mount –iso9660 /dev/cdrom /mnt/cdrom
```

El gestor de arranque

En caso de que tengamos en nuestra computadora más de un sistema operativo instalado, necesitamos de alguna herramienta que nos permita seleccionar el sistema operativo que queremos utilizar en un momento dado. Para realizar esto, requerimos de un gestor de arranque.

Los gestores utilizados en Linux son dos:

- LILO (Linux LOader).
- GRUB (GRand Unified Bootloader)

LILO

Este gestor de arranque se encuentra en la mayoría de las distribuciones Linux y se puede instalar como parte del proceso de instalación. LILO tiene actualmente una limitación importante: en discos mayores de 8 gigabytes, la partición de arranque de todos y cada uno de los sistemas operativos presentes en el disco debe estar situada dentro de los 8 gigabytes primeros del disco. Es resto de las particiones puede estar en cualquier otro lugar, incluyendo los gigabytes posteriores a los primeros 8.

GRUB

Este gestor de arranque resuelve el problema que tiene LILO con discos mayores a 8 gigabytes. Permite al usuario seleccionar qué sistema operativo o kernel cargará, en el momento de arrancar el sistema, además, permite pasarle parámetros al kernel. Entre sus principales funciones son[1]:

- *GRUB proporciona un verdadero entorno basado en comandos, pre-sistema operativo, para las máquinas x86.* Esta funcionalidad le otorga al usuario una gran flexibilidad en la carga de sistemas operativos con opciones específicas o con la recopilación de información sobre el sistema. Durante muchos años, las arquitecturas diferentes a x86 han

[1] Funciones del GRUB: http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/s1-grub-whatis.html

usado entornos previos al sistema operativo que permiten arrancar el sistema desde una línea de comandos.

- *GRUB soporta el modo Direccionamiento Lógico de Bloques (LBA).* El modo LBA coloca la conversión de direccionamiento utilizada para buscar archivos en la unidad de disco duro del firmware y se utiliza en muchos discos IDE y en todos los discos duros SCSI. Antes de LBA, los gestores de arranque encontraban la limitación del cilindro 1024 del BIOS, donde el BIOS no podía encontrar un archivo después de ese cabezal de cilindro del disco. El soporte LBA permite que GRUB arranque los sistemas operativos desde las particiones más allá del límite de 1024 cilindros, siempre y cuando el BIOS del sistema soporte el modo LBA. Las mayoría de las revisiones más modernas de la BIOS soportan el modo LBA.
- *GRUB puede leer las particiones ext2.* Esto permite que GRUB acceda a su archivo de configuración, `/boot/grub/grub.conf`, cada vez que el sistema arranca, eliminando la necesidad que tiene el usuario de escribir una nueva versión de la primera etapa del gestor de arranque al MBR en caso de que se produzcan cambios de la configuración. El único caso en el que el usuario necesitaría reinstalar GRUB en el MBR es en caso de que la localización física de la partición `/boot/` se traslade en el disco. Para más detalles sobre la instalación de GRUB en el MBR.

El sistema X Windows

Los sistemas UNIX y Linux utilizan el mismo estándar básico para trabajar con gráficos, conocido como Sistema X Windows, llamado X o X11. Esto significa que en la mayoría de los casos, un programa basado en X puede ejecutarse en cualquier escritorio o administrador de ventanas. Fue desarrollado a mediado de los 80’s como una respuesta a la necesidad de un ambiente grafico que fuera transparente para los sistemas UNIX.

Linux y UNIX no necesitan del sistema X Windows para funcionar, ya que originalmente fueron idealizados para funcionar en modo texto. Hay que aclarar que X Windows es un sistema independiente del sistema operativo y posee una característica que marca una gran diferencia en comparación a la interfaz grafica de otros sistemas operativos, distribuye el procesamiento de aplicaciones, especificando un enlace cliente-servidor, es decir, el cliente de X le dirá qué hacer, al servidor de X, y éste se encargará del cómo hacerlo. Esto no quiere decir que el cliente de X y el servidor de X tengan que estar alojados en la misma máquina, ésta es una ventaja más, el cliente de X puede estar en otra máquina y con otro sistema operativo. ¿Qué tal? Les repito, X Windows es independiente del sistema operativo.

Conclusión

Linux es un sistema operativo con características envidiables que día a día demuestra su gran competitividad y su gran aceptación en todo el mundo.
Este sistema operativo, es la prueba de que no necesariamente tenemos que pagar una suma considerable para obtener un sistema potente, flexible, seguro, estable y confiable que nos ayude a solventar nuestras necesidades. Además, tenemos la oportunidad de seleccionar y probar las diferentes distribuciones de Linux que existen y quedarnos con la que más nos guste. ¿No es genial?
Simplemente el hecho de decir que Linux esta basado en UNIX ya es garantía, y prueba de ello es que cada vez más empresas adoptan este sistema operativo para implementar sus aplicaciones en producción.

El objetivo de éste capítulo fue involucrarnos un poco en el origen, funcionamiento y administración de Linux, para que posteriormente podamos aprovechar las bondades de éste sistema operativo en la gestión de archivos y aplicaciones.

Hay que tener en cuenta que la flexibilidad que nos proporciona Linux, puede convertirse en un factor de dificultad al principio para cualquier usuario inexperto, más sin embargo, puedo asegurar que rápidamente éste usuario adquirirá la capacidad para moverse y realizar casi cualquier tarea a través de las herramientas que nos proporciona éste sistema.

Creación de páginas Web en Linux

Capitulo 2

Capitulo 2: Creación de páginas Web en Linux

¿Qué es HTML?

HTML tiene su origen de las siglas en ingles Hyper Text Markup Language (Lenguaje de Marcado de Hipertexto). Este lenguaje para la creación de páginas Web surgió por la necesidad de divulgar información a través de Internet, aunque nadie se imagino el crecimiento que este tendría con el paso de los años.

Seguramente nos preguntamos por qué es un lenguaje de marcado, y la respuesta es muy sencilla, es por el simple hecho de que HTML se basa en una cantidad considerable de etiquetas predefinidas (tags) que nos permiten diseñar la estructura de nuestras páginas Web y para la creación de hipertexto, que no es otra cosa que texto digital con características especiales para la World Wide Web. Estos documentos Web podemos visualizarlo a través de un navegador de Internet (por ejemplo, Internet Explorer, Netscape Navigator, Mozilla, etc.).

Un punto importante es que *HTML no es considerado como un lenguaje de programación* ya que carece de estructuras de control, entre otras funciones.

¿Origen de HTML?

Fue desarrollado por Tim Berners-Lee en el año de 1986 para el intercambio de hipertexto, pero dada la dificultad que resultaba en ese momento animar al mundo a utilizar un nuevo sistema de información global, HTML fue elegido para parecerse a algunos sistemas basados en SGML (Standard Generalized Markup Language) que habían sido adoptados por la comunidad de la documentación y los para los cuales, la sintaxis de SGML era su preferida.

Aunque para algunas comunidades la adopción de SGML permitió aceptar la Web más fácilmente, SGML resultó muy complejo y no había definido completamente su sintaxis para la creación de documentos Web. Ante el esfuerzo y compromiso de encontrar un lenguaje que tuviera una completa compatibilidad con SGML y fuera de fácil uso, HTML endemonió a los expertos durante mucho tiempo.

Tim Bernes-Lee
Director de la W3C

Tim Berners-Lee fundó en 1994 la W3C[1] (Consorcio World Wide Web), una asociación internacional formada por organizaciones miembro del consorcio, personal y el público en general, que trabajan conjuntamente para desarrollar estándares.

Por lo tanto, la W3C elabora los estándares para HTML y libera las nuevas versiones, para que sean implantadas en los navegadores de Internet y utilizadas por la comunidad de desarrolladores Web.

La siguiente tabla muestra las versiones de HTML que existen hasta el momento:

Versión	Descripción
HTML 1	HTML 1.0 contenía comandos para elementos estándares como encabezamientos, párrafos, referencias de imágenes y por supuesto para enlaces. La especificación para ésta versión ya no se encuentra en la W3C.
HTML 2	Se convirtió en el estándar oficial en noviembre de 1995. En esta versión el navegador de Netscape ya interpretaba Frames (técnica de varias ventanas) y JavaScripts que estaban lejos de ser estándar oficial de HTML. La especificación de ésta versión se encuentra en la W3C y podemos acceder a través de la siguiente dirección: http://www.w3.org/MarkUp/html-spec/html-pubtext.html
HTML 3.2	HTML 3.2 se convirtió en el lenguaje oficial el 14 de enero de 1997. En el número de la versión se puede ver que esta versión tuvo muchos problemas para ser tomada como estándar. El Consorcio W3 desarrollaba al principio la versión 3.0, sin importarle el éxito del navegador de Netscape. HTML 3.0 contenía algunas proposiciones muy interesantes que hasta el momento son ignoradas por los navegadores modernos (por ejemplos comandos para representar fórmulas matemáticas). Sin embargo las proposiciones eran contrarias a la nueva realiad en la WWW. En aquellos tiempos el Consorcio W3, que pertenece al mundo de las ciencias, comenzó a colaborar con productores de software comercial. HTML 3.2 es el resultado de una total revisión de las proposiciones de HTML 3.0, versión que nunca se convirtió en versión oficial. Tablas fueron finalmente reconocidas oficialmente, además de diversos comandos para la marcación física de texto. Los productores de navegadores, entre ellos ya Microsoft, se quedaron esperando la toma de los frames como estándar, pero fue en vano.

[1] Para conocer más acerca de la W3C podemos visitar la siguiente liga: http://www.w3.org

	La especificación de ésta versión se encuentra en la W3C y podemos acceder a través de la siguiente dirección: http://www.w3.org/TR/REC-html32.html
HTML 4	HTML 4.0 fue aprobado el 18 de febrero de 1998 oficialmente como lenguaje oficial. Con HTML 4.0 el Consorcio W3 demostró que esta lista a cooperar con los productores de navegadores comerciales y ve de esta manena ve su papel como gremío cooperativo de estándarización en mercercado multimillonario de WWW. Para los empedernidos científicas este no fue un paso fácil, pero uno muy importante. Pues sólo un Consorcio W3 que sea respetado por los productores de software tiene una existencia segura. Muchas personas reconocen hoy en día el significado que tiene este gremio independiente después del intento de algunos productores de desarrollar e implantar su propio pseudo-estándares. HTML 4.0 reconoce el uso de frames y el ligamento de CSS Style Sheets (hojas de estilo) en HTML. De esta manera casi todos los suplementos de HTML han sido reconocidos como estándares, o por lo menos estan previstos ser estándares de HTML. La especificación de ésta versión se encuentra en la W3C y podemos acceder a través de la siguiente dirección: http://www.w3.org/TR/REC-html40/

Introducción a la creación de paginas HTML

Estructura básica de un documento Web

Como hemos visto anteriormente, los documentos Web necesitan de navegador de Internet que interprete un documento Web y por ende las etiquetas de HTML que utilice el documento. En necesario que utilicemos un navegador de Internet reciente ya que estaremos casi seguro de que utiliza como referencia el estándar de HTML 4 y esto nos permitirá utilizar una mayor cantidad de etiquetas HTML.[2]

[2] Existen etiquetas HTML que fueron desarrolladas para ser interpretadas por un navegador de Internet en específico. Por ejemplo, la etiqueta <blink> que es interpretada por Nestcape Navigator, pero no es reconocida por Internet Explorer.

La creación de un documento Web es una tarea muy sencilla, vamos a ver como es la estructura básica de una página HTML. Podemos utilizar un editor de texto para crear nuestra página HTML, únicamente debemos guardar nuestro documento con la extensión ".html". Ahora bien, veamos cual es la estructura básica de un documento HTML:

```
<HTML>
<HEAD>
    <TITLE>Mi primera página Web </TITLE>
</HEAD>
<BODY>

Este es el cuerpo de mi página. Aquí podemos escribir todo
el texto que queramos.
</BODY>
</HTML>
```

Pagina1.html

Vamos a ver el significado de etiquetas presentadas en el cuadro anterior:

- Una página Web comienza con la etiqueta <HTML> y finaliza con la etiqueta </HTML>. Esto le dice al navegador de Internet que tipo de documento va a procesar y éste reconoce en dónde inicia y dónde termina.
- Dentro de <HEAD> </HEAD> podemos agregar etiquetas específicas, que den información adicional a los buscadores de páginas Web y a los usuarios que las consultan.
- La etiqueta <TITLE> indica el nombre o título de nuestra página Web.
- Por último, todo el texto y las etiquetas que se encuentren dentro de las etiquetas <BODY> </BODY> indican el cuerpo del documento, dicho de otra manera, toda la información que se va a visualizar en nuestra página Web (imágenes, tablas, vinculos, etc.)

La mayoría de las etiquetas de HTML utilizan una etiqueta de cierre. Ésta se utiliza para delimitar el área afectada por la etiqueta. Vamos a ver un pequeño ejemplo para entenderlo más fácilmente.

La dirección completa de la FES Aragón es: Avenida Rancho Seco S/N, colonia Impulsora, Nezahualcóyotl, Estado de México, Código Postal 57130.

Supongamos que éste texto se encuentra dentro de la etiquetas <BODY> </BODY> de nuestra página Web. Si nosotros quisiéramos resaltar con **negritas** el nombre de la escuela, utilizamos la etiqueta <B></B> (que viene del texto **bold**, *negritas*), quedando de ésta manera:

La dirección completa de la ****FES Aragón**** es: Avenida Rancho Seco S/N, colonia Impulsora, Nezahualcóyotl, Estado de México, Código Postal 57130.

HTML no es sensible a mayúsculas o minúsculas, así que la etiqueta <B> o significan lo mismo para nuestro navegador. El resultado obtenido al poner las etiquetas en nuestro navegador de Internet es el siguiente:

La dirección completa de la **FES Aragón** es: Avenida Rancho Seco S/N, colonia Impulsora, Nezahualcóyotl, Estado de México, Código Postal 57130.

Aunque reitero que no todas las etiquetas HTML utilizan una etiqueta de cierre, como por ejemplo <HR> la cual se utiliza para poner una línea horizontal en un documento Web.

Lista de las principales etiquetas Web

No pretendo abordar un curso completo de HTML, por lo tanto, únicamente me limitaré a dar una referencia rápida de las etiquetas más utilizadas en HTML[3]:

Etiquetas para caracteres	
<B>...</B>	Texto en negrita
<BIG>...</BIG>	Ampliación del tamaño de los caracteres
<BLINK>...</BLINK>	Texto parpadeante (Netscape solo)
<EM>...</EM>	Texto en itálico
<FONT color="#XXXXXX"> ...</FONT>	Texto en color donde XXXXXX es un valor hexadecimal
<FONT size=X>...</FONT>	Tamaño de los caracteres donde X es un valor de 1 a 7
<I>...</I>	Texto en itálico
<NOBR>...</NOBR>	Impide las rupturas automáticas de línea de los navegadores
<PRE>...</PRE>	Texto preformateado, o sea con una visualización de todos los espacios y saltos de línea
<SMALL>...</SMALL>	Reducción del tamaño de los caracteres
<STRONG>...</STRONG>	Puesta en negrita del texto
_{...}	Texto en indicio
^{...}	Texto en exponente
<U>...</U>	Texto subrayado

[3] La lista de etiquetas fue tomada de la siguiente página: http://www.ccim.be/ccim328/htmlsp/REF.htm

Etiquetas para el texto.	
<!--...-->	Comentarios ignorado por el navegador
** **	A la línea
<BLOCKQUOTE>... </BLOCKQUOTE>	Citación (introduce un retracto de texto)
<CENTER>...</CENTER>	Centra cada elemento comprendido en la etiqueta
<DIV align=center> ...</DIV>	Centra el elemento encuadrado por la etiqueta
<DIV align=left> ...</DIV>	Alinea el elemento a la izquierda
<DIV align=right> ...</DIV>	Alinea el elemento a la derecha
<Hx>...</Hx>	Título o x tiene un valor de 1 à 7
<Hx align=center>...</Hx>	Título centrado
<Hx align=left>...</Hx>	Título alineado a la izquierda
<Hx align=right>...</Hx>	Titulo alineado a la derecha
<P>...</P>	Nuevo párrafo
<P align=center>...</P>	Párrafo centrado
<P align=left>...</P>	Párrafo alineado a la izquierda
<P align=right>...</P>	Párrafo alineado a la derecha

Etiquetas para listas	
<UL>	Lista no numerada
<LI>	Elemento de lista
</UL>	
<OL>	Lista numerada
<LI>	Elemento de lista
</OL>	
<DL>	Lista de glosario
<DT>...</DT>	Termino de glosario (sin retracto)
<DD>...</DD>	Explicación del termino (con retracto)
</DL>	

Etiquetas para líneas	
<HR>	Línea de separación. Raya horizontal
<HR width="x%">	Anchura de la raya en %
<HR width=x>	Anchura de la raya en píxeles
<HR size=x>	Altura de la raya en píxeles
<HR align=center>	Raya centrada
<HR align=left>	Raya alineada a la izquierda
<HR align=right>	Raya alineada a la derecha
<HR noshade>	Raya sin efecto de sombreado

Etiquetas para enlaces o hipervínculos	
<A href="http://...">...</A>	Enlace hacia una página Web
<A href="mailto:...">...</A>	Enlace hacia una dirección Email
<A href="fichier.htm">...</A>	Enlace hacia la página fichero.htm situada en el mismo directorio
<A name="xyz">...</A>	Definición de una ancla
<A href="xyz">...</A>	Enlace hacia una ancla
<A href="fichier#xyz">...</A>	

Etiquetas para imágenes	
<IMG scr="xyz.gif"> **<IMG scr="xyz.pjg>**	Inserción de una imagen al formato Gif o Jpg (ver enlaces para la dirección)
<IMG ... width=x height=y>	Puesta a la escala de la imagen en pixeles
<IMG ... border=x>	Definición del borde de una imagen con un enlace
<IMG ... alt="votre texte">	Texto alternativo cuando la imagen no esta mostrada
<IMG ... align=bottom>	Alinea la imagen abajo
<IMG ... align=middle>	Alinea la imagen en el medio
<IMG ... align=top>	Alinea la imagen arriba
<IMG ... align=left>	Alinea la imagen a la izquierda
<IMG ... align=right>	Alinea la imagen a la derecha
<IMG ... hspace=x> **<IMG ... vspace=y>**	Espaciamiento horizontal entre la imagen y el texto Espaciamiento vertical entre la imagen y el texto

Etiquetas para frames	
<FRAMESET>...</FRAMESET>	Define una estructura de frames (reemplaza la etiqueta BODY)
<FRAMESET rows="x%,y%,...">	División horizontal de la ventana en %
<FRAMESET cols="x%,y%,...">	División vertical de la ventana en %
<FRAME src="fichier.htm">	Fichero mostrado en una ventana de frames
<NOFRAMES>...</NOFRAMES>	Contenido para los browser no previstos para los frames

Etiquetas para tablas	
<TABLE>...</TABLE>	Definición de una tabla
<TABLE width="x%">	Anchura de la tabla en %
<TABLE width=x>	Anchura de la tabla en pixeles
<TABLE border=x>	Anchura del borde
<TABLE cellpadding=x>	Espacio entre el borde y el texto
<TABLE cellspacing=x>	Espesor de la raya entre las celdas
<TR>...</TR>	Línea de la tabla
<TD>...</TD>	Celda de la tabla
<TD bgcolor="#XXXXXX">	Color de una celda de la tabla
<TD width="x%">	Anchura de columna en %
<TD width=x>	Anchura de columna en pixeles
<TD align=center>	Texto centrado en la celda
<TD align=left>	Texto alineado a la izquierda en la celda
<TD align=right>	Texto alineado a la derecha en la celda
<TD valign=bottom>	Alineación hacia arriba del contenido de la
<TD valign=middle>	celda Centrado vertical del contenido de
<TD valign=top>	una celda Alineación hacia el bajo del contenido de la celda
<TD colspan=x>	Numero de celdas para fusionar
<TD rowspan=x>	horizontalmente Numero de celdas para fusionar verticalmente

Etiquetas para un documento Web	
<HTML>...</HTML>	Principio y fin del fichero Html
<HEAD>...</HEAD>	Zona de encabezamiento de un fichero Html
<TITLE>...</TITLE>	Titulo visualizado por el browser (elemento de HEAD)
<BODY>...</BODY>	Principio y fin del cuerpo del fichero Html
<BODY bgcolor="#XXXXXX">	Color del fondo (en hexadecimal)
<BODY background="xyz.gif">	Imagen del fondo

Proyecto Final: Revista Digital.

Hasta ahora, hemos aprendimos a utilizar y administrar nuestra plataforma Linux que es el sistema operativo que nos proveerá de las herramientas necesarias para manipular los archivos de nuestro proyecto final. Ahora con HTML, ya somos capaces de crear documentos HTML con elementos multimedia como son imágenes y sonidos. Por lo tanto, ya podemos crear la estructura de nuestra Revista Digital, así que vamos a empezar.

¿Qué es un Sistema Administrador de Contenidos?

Un Sistema Administrador de Contenidos (CMS por sus siglas en Ingles) es una aplicación Web desarrollada con el propósito de facilitarnos la creación de un sitio Web o portal, aportándonos las herramientas de gestión de archivos, creación, modificación y borrado de artículos, gestión de usuarios, gestión de plantillas, entre otras cosas. Un ejemplo de un Sistema de Administración de Contenidos famoso y además libre, es *Mambo Server*[4].

La razón por la cual describo a grandes rasgos lo que es un CMS, es porque en teoría, nuestra Revista Digital debe actuar tal, es decir, debe gestionar archivos, usuarios, artículos de una manera fácil e intuitiva para los usuario que la administran e intervienen en su crecimiento.

Nota: Esta aplicación, corresponde al proyecto final del diplomado "Desarrollo e implementación a través de software libre en Linux".

Análisis de requerimientos.

Se requiere una aplicación Web que permita a los usuarios la posibilidad de acceder a los artículos de su interés, la opción de darse de alta en el sistema para poder contribuir, si así lo desean, con nuevos artículos, archivos, vínculos, etc. Además, es necesario que cada usuario tenga acceso a los archivos y documentos que haya publicado, con la opción de borrarlos o modificarlos.

Técnicamente se utilizará MySQL como nuestro gestor de bases de datos, PHP para la programación dinámica de páginas Web, el servidor Web Apache con el módulo de SSL para realizar conexiones seguras y todas estas aplicaciones montadas sobre un servidor Linux.
Para nuestra Revista Digital, he elegido un enfoque orientado a artículos de computación y tecnología. los colores, el logo y elementos multimedia que conformen la revista, tienen que ser adecuados y atractivos para las personas que navegan en ella.

[4] Para más información podemos visitar la página oficial de Mambo Server www.**mamboserver**.com/

Como un requerimiento de diseño, se nos solicitó que la distribución de la información estuviera concentrada en la página inicial, y que ésta, funcionara como una plantilla, con un diseño libre a escoger por nosotros. La distribución de la información quedaría distribuida tal y como se muestra en la figura 3:

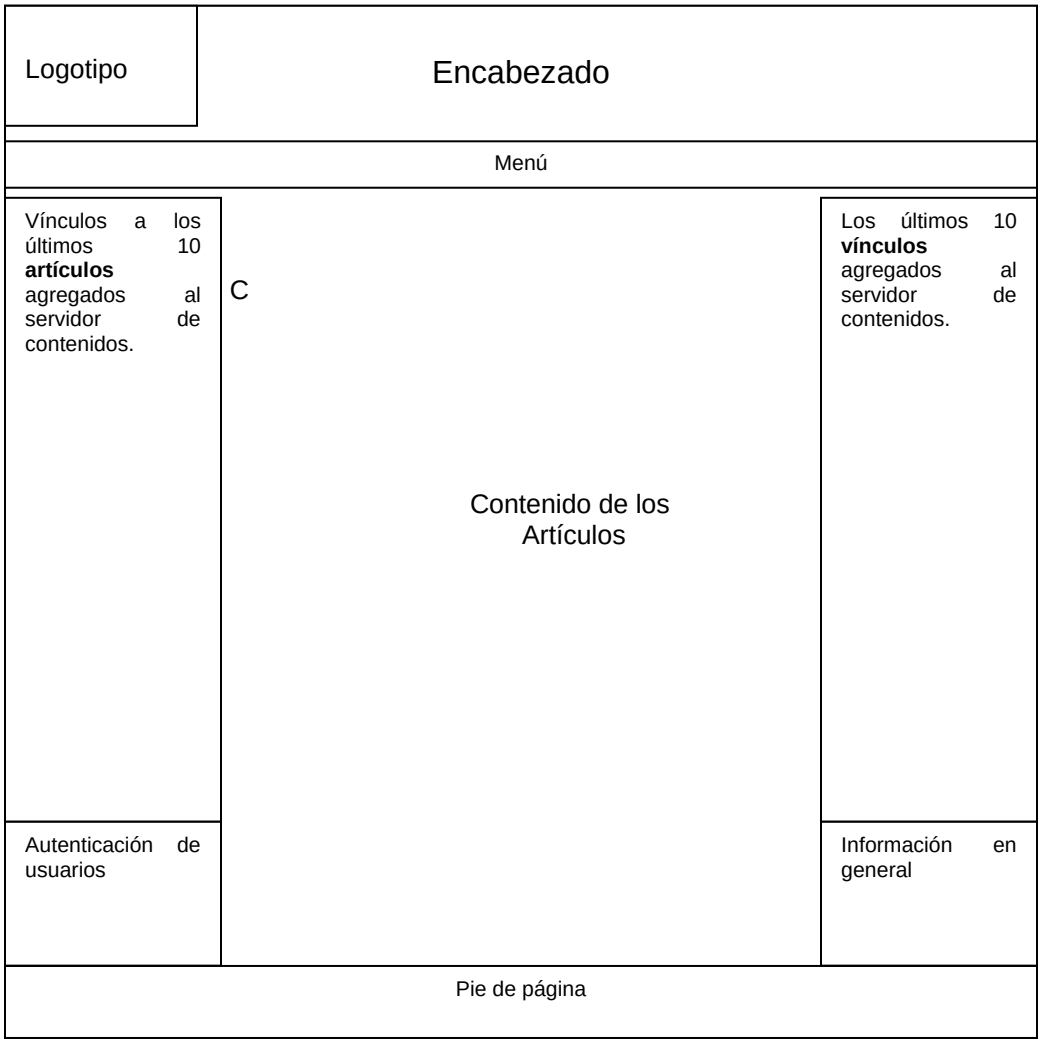


Figura 3. Distribución de la información en la página principal

El motivo de que nuestra página principal tenga ésta estructura y distribución, es por cuestiones de accesibilidad, y facilidad de uso. Por ejemplo, la idea es la siguiente, si un usuario accede a nuestra revista y da un clic sobre alguno de los vínculos del menú del lado izquierdo, el contenido del artículo automáticamente se desplegará en la parte central, dejando todos los demás elementos sin modificación alguna.

Requerimientos de Software

Las herramientas libres que utilizaremos para realizar éste proyecto quedan libres para el programador, aunque se dará una lista de las herramientas que utilizaremos y una breve descripción de cada una de ellas.

Software recomendado	Descripción	Software similar
Linux Slackware 10.1	Sistema operativo Open Sourcecon grandes prestaciones donde residirá nuestra aplicación Web. Para más información: www.slackware.com/	• Linux Redhat • Linux S.u.S.E • Linux Mandrake • Otros
Apache Server	Servidor de aplicaciones Web, que utiliza el protocolo http. Es el encargado de responder a las peticiones de los usuarios a través de Internet. Para más información: www.apache.org/	• Roxen WebServer • Cherokee • Zeus • Otros
PHP	Lenguaje de script que permite al diseñador Web la creación de páginas con contenido dinámico. Compatible con Apache Server. Para más información: www.php.net/	• ASP • JSP
MySQL	Manejador de bases de datos relacionales de gran popularidad y de libre distribución. Para más información: www.mysql.com/	• Postgresql
Open SSL	Secure Socket Layer, servicios de seguridad cifrando los datos intercambiados entre el servidor y el cliente con un mecanismo de cifrado simétrico, típicamente RC4 o IDEA y cifrando la clave de sesión RC4 o IDEA mediante un algoritmo de llave pública, típicamente RSA. Para más información: www.openssl.org/	
NVU	Editor de páginas Web.	• Screem • Quanta Plus •

Requerimientos de hardware

Los requerimientos **mínimos** necesarios para instalar el sistema operativo Linux Slackware 10.2 son los siguientes:

- Procesador 486
- 16MB RAM (32MB sugerido)

- 100-500 MB de disco duro para una instalación mínima y alrededor de 3.5GB para una instalación completa.
- 3.5" floppy drive

Pero nosotros no tenemos ningún problema ya que contamos con una computadora con las siguientes características:

- Procesador Pentium 4
- 256 MB en memoria RAM
- 20 GB en disco duro.
- Unidad de CDROM
- 3.5” floppy drive

Desarrollando la interfaz Web

De acuerdo a los requerimientos del usuario, vamos a comenzar por crear la página principal de nuestra revista electrónica.
Es mucho más sencillo distribuir la información dentro de nuestra página si utilizamos tablas para ordenas los elementos, tal y como se muestra en la siguiente figura:

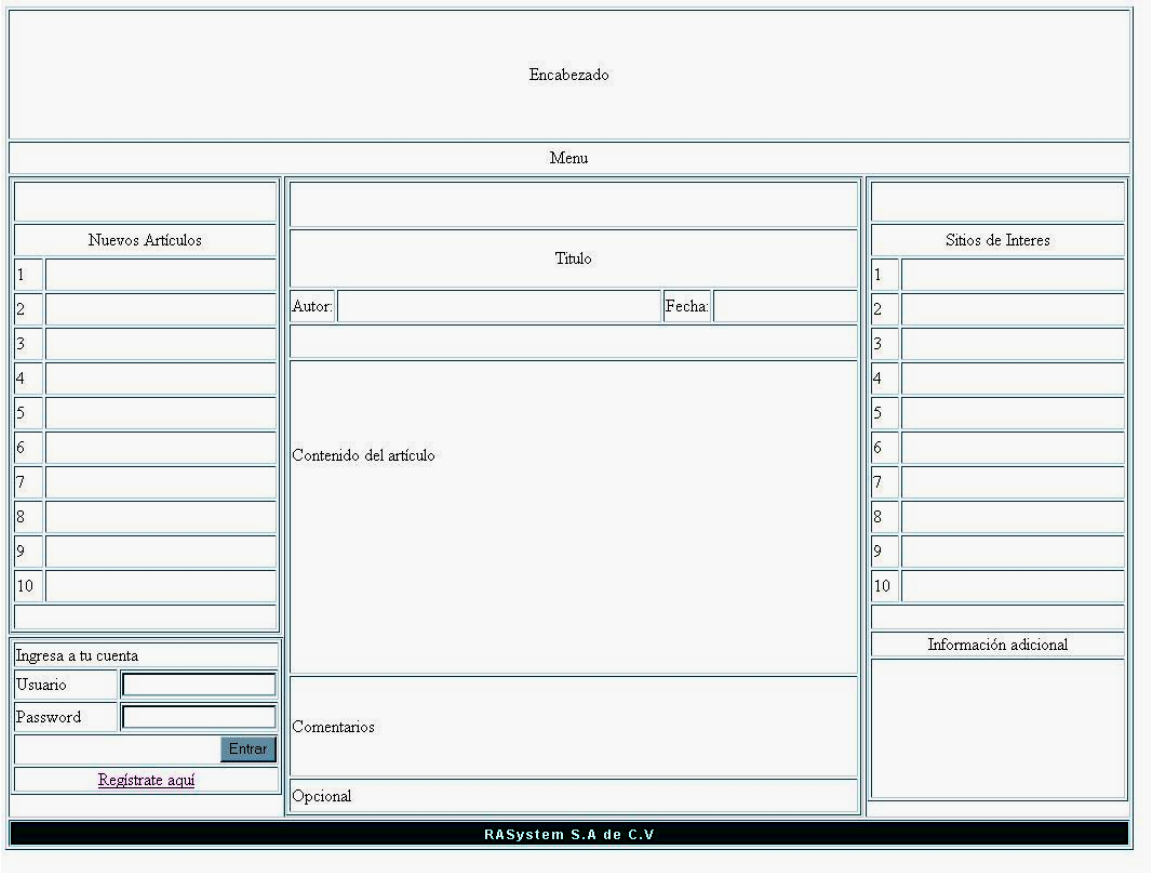


Figura 4: Página principal

Ya teniendo la estructura en tablas, vamos a colocar las imágenes, los textos y los estilos necesarios para darle un aspecto atractivo al usuario que la visita.

Este es el aspecto de nuestra página principal. Ahora podemos continuar con la creación de las páginas que nos permitirán administrar nuestra revista electrónica siguiendo los mismos patrones de diseño.

Las páginas que vamos a requerir para darle la funcionalidad deseada son:

- Una página para el registro de nuevos usuarios.
- Una página para crear nuevos artículos, modificarlos y borrarlos.
- Una página para agregar nuevos vínculos, modificarlos y borrarlos.
- Una página para el administrador que permita eliminar artículos y vínculos.
- Una página para el administrador que permita eliminar usuarios.

Así lucirá la pantalla de inicio.

Una vez que es usuario se ha logeado al sistema, en la parte inferior izquierda, aparecerá un menú exclusivo con funciones para el usuario.

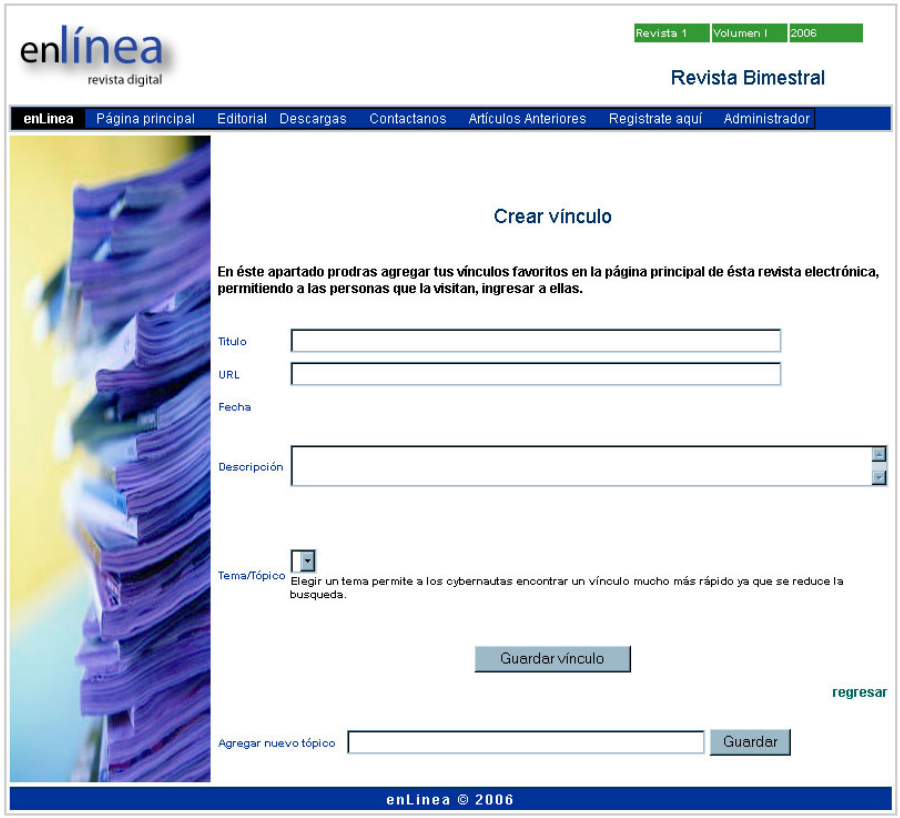
Una página que permita a un usuario darse de alta en el sistema.

Esta información podrá modificarla posteriormente el usuario a través del menú "Perfil" una vez que se haya registrado en el sistema.

Dos de los privilegios que tiene un usuario al momento de registrarse en el sistema, es la posibilidad de crear artículos o agregar vínculos de sitios interesantes en el sistema.

Las siguientes dos imágenes nos muestran la interfaz de usuario para la creación de nuevos artículos y vínculos.


enlínea
revista digital
Revista 1
Volumen I
2006
Revista Bimestral
enLinea
Página principal
Editorial
Descargas
Contactanos
Artículos Anteriores
Registrate aquí
Administrador
Crear artículo
Llena los campos con la información que se te solicita a continuación:
Titulo
Autor(es)
Fecha
Resumen *
Artículo
Referencias bibliográficas *
* Opcionales
Guardar
Limpiar
regresar
enLinea © 2006


enlínea
revista digital
Revista 1
Volumen I
2006
Revista Bimestral
enLinea
Página principal
Editorial
Descargas
Contactanos
Artículos Anteriores
Registrate aquí
Administrador
Crear vínculo
En éste apartado prodras agregar tus vínculos favoritos en la página principal de ésta revista electrónica, permitiendo a las personas que la visitan, ingresar a ellas.
Titulo
URL
Fecha
Descripción
Tema/Tópico
Elegir un tema permite a los cybernautas encontrar un vínculo mucho más rápido ya que se reduce la busqueda.
Guardar vínculo
regresar
Agregar nuevo tópico
Guardar
enLinea © 2006

Una vez que el usuario publique sus documentos, tendrá la posibilidad de modificarlos o borrarlos del sistema en cualquier momento a través del menú “Mis artículos” o “Mis vínculos” de la pantalla principal del usuario.

El administrador del sistema tiene los privilegios de crear, modificar y borrar vínculos creador por él o de alguno de los usuarios del sistema. Para realizar esto se apoya en una página que posee las herramientas necesarias para administrar los artículos.

Nota: Para poder acceder a ésta página se requiere una autentificación de usuario.


enlínea
revista digital
enLinea
Página principal
Editorial
Descargas
Contactanos
Artículos Anteriores
Registrate aquí
Administración de usuarios y contenidos
Usuarios
Selecciona la acción deseada sobre el usuario especificado
Rodrigo Arias Hurtado
Modificar
Eliminar
Ejecutar
Agregar un nuevo usuario
Artículos
En ésta sección prodrás modificar o eliminar un artículo determinado. Selecciona la opción "Todos" en el campo "Autor" para poder ver todos los artículos.
Autor
Todos
Artículo
Conoce RedHat Enterprise SA 4
Acción
Modificar
Borrar
Ejecutar
Vínculos
En ésta sección prodrás modificar o eliminar un vínculo determinado. Selecciona la opción "Todos" en el campo "Autor" para poder ver todos los vínculos..
Autor
Todos
Vínculo
http://www.unam.com.mx
Acción
Modificar
Borrar
Ejecutar
Regresar
enLinea © 2006

Conclusión

En éste capitulo aprendimos a utilizar el Lenguaje de Marcado de Hipertexto para la elaboración de documentos Web.
En sus orígenes, HTML era la única herramienta que nos permitía publicar información en Internet, pero con el paso del tiempo, HTML ha ido evolucionando y se ha ido adaptando a las necesidades de los usuarios, permitiendo la entrada de las aplicaciones multimedia y de aplicaciones dinámicas que ayudan a realizar y simplificar tareas a través de la Web.

Hoy en día, no existe ninguna aplicación en Internet que no haga uso de HTML, a pesar de la gran cantidad de lenguajes que se han ido incorporando a la creación de documentos Web. Por ejemplo, en un capítulo posterior analizaremos un lenguaje de programación par Web llamado PHP, que se mezcla con HTML para otorgarle un dinamismo a nuestras páginas Web, es decir, darles la capacidad de interactuar con el usuario y de mostrar contenido actualizado utilizando siempre una fuente de datos, como podría ser por ejemplo, una base de datos.

Quiero recalcar que sólo hicimos referencia en éste capítulo a las principales etiquetas utilizadas en la elaboración de documentos Web. Si queremos profundizar en éste tema, podemos encontrar muchos libros que hacen referencia a éste lenguaje, así como gran cantidad de información y ejemplo en Internet.

Administración del servidor Web Apache

Capitulo 3

Capitulo 3: Administración del servidor Web Apache

¿Qué es un servidor Web?

Un servidor Web es una aplicación que implementa el protocolo HTTP (Hipertext Transfer Protocol) y está diseñada para despachar peticiones de los usuarios que intentan obtener documentos Web a través de un navegador de Internet.

Servidor Web Apache

Nosotros vamos a aprender a utilizar el servidor Apache Web Server, que es hoy en día el servidor más utilizado en todo el mundo.

Apache Web Server es un software de código abierto que se distribuye en prácticamente todas las distribuciones de Linux, algunas de las características que lo destacan son:

- Robusto, tiene soporte para un gran número de transacciones.
- Configurable para diferentes entornos de trabajo.
- Con un alto nivel de seguridad.
- Disponible para un gran número de plataformas.
- Soporte para servicio de Proxy.
- Soporte para granjas de servidores.
- Soporte para scripting: lenguajes integrados como módulos (por ejemplo, PHP, mod_Perl, etc.).
- Incluye el código fuente del servidor.
- Soporte para accesos restringidos.
- Soporte para SSL.
- Y además es gratuito.

Instalar Apache Web Server

Antes que nada, vamos a descargar el software en el sitio oficial de Apache Web Server:

http://httpd.apache.org/.

Vamos a aprovechar para descargas la versión más actual del servidor Web. No debemos pasar desapercibida la gran cantidad de documentación que existe de Apache, ya que en cualquier momento nos puede servir de gran ayuda.

Una vez que hemos descargado Apache, vamos a descomprimirlo en el directorio */usr/local/src/* con el siguiente comando:

```
$cd /usr/local/src
$> tar –xvzf httpd-2-2-0.tar.gz
$> cd httpd-2-2-0
```

Ahora vamos a instalar el software. Realmente es muy sencillo, basta con sólo ejecutar tres comandos para que nuestro servidor Apache este listo para configurarse y ponerse en marcha.

El primer script que vamos a ejecutar se denomina "./configure". Este script se va a encargar de buscar en nuestra máquina todo el software que necesita para instalar Apache, es decir, localiza las librerías compartidas, el compilador de C, la compatibilidad con otras herramientas, el espacio en disco, el directorio donde se va a instalar, etc.

Vamos a teclear lo siguiente:

```
./configure –-prefix=/usr/local/apache2 –-enable-ssl
```

Con la opción `–prefix=`*`ruta`* le decimos a Apache la ubicación en donde queremos instalar nuestro servidor dentro del sistema.
Con la opción `–enable-ssl` habilitamos la compatibilidad con Secure Socket Layer, que es un sistema de seguridad que veremos en el último capítulo de éste trabajo.

Cuando termine de hacer las configuraciones necesarias, vamos a teclear el comando "make" para compilar toda las información recolectada por el script *configure* y crear un archivo binario de instalación.

Si todo salió bien, tecleamos el comando final: *make install*. Este creará los directorios necesarios y posicionará los archivos creados en su lugar correspondiente

Configuración del Servidor de Web Apache

La configuración del servidor Web Apache se realiza a través del archivo *httpd.conf* ubicado en el directorio *conf* en la ruta donde instalamos nuestro servidor (/usr/local/apache2/conf).

Apache es administrado por más de 150 directivas las cuales permiten que determinadas funcionalidades puedan ser incluidas. Para esto, el administrador controla que directivas estarán disponibles de acuerdo a los módulos que se hayan compilado con el servidor Apache.

A continuación se dará una breve reseña de las directivas más utilizadas dentro del archivo *httpd.conf* del servidor Apache. Para más información al respecto,

podemos visitar el sitio oficial de Apache[1] que cuenta con una excelente documentación en diferentes idiomas.

Grupos de Directivas

La configuración de apache mediante directivas es clasificada en tres grupos:

- **Global Enviroment**
- **Main Server**
- **Virtual Servers (servidores virtuales)**

Nota: Dentro del archivo httpd.conf también se encuentran en éste orden.

Global Enviroment (Ambiente Global)

La sección Global Enviroment administra las directivas generales de operación para Apache.

ServerType

Define el esquema mediante el cual opera apache, las posibles opciones son:

- standalone
- inetd

No se recomienda inetd debido a los recursos de computo que se consumen al obligar al superdemonio a generar un httpd cada vez que se genera una conexión

User

Esta directiva opera para modo standalone y define el ID del usuario mediante el cual apache operara, para que esto funcione se requiere que apache se arranque como root.

Valor por default: **User #-1**
Se puede usar en: **Server config, virtual host**

Group

Esta directiva opera para modo standalone y define el ID grupo mediante el cual apache operara, para que esto funcione se requiere que apache se arranque como root.

Valor por default: **User #-1**

[1] Sitio oficial de Apache Web Server: http://httpd.apache.org/.

Se puede usar en: Server **config, virtual host**

Port

Esta directiva define cual es el puerto en que operara el servidor de Web.

Valor por default: **80**

ServerAdmin

Esta directiva define el correo electrónico del administrador del servidor Web.

ServerAdmin rodrigo.arias@mail.com

DocumentRoot

Esta directiva define la ruta absoluta donde se almacenarán los archivos HTML que se desean publicar.

Valores por default: **/usr/local/apache/htdocs ó /var/www/html**
Se puede usar en: **Server config, virtual host**

MinSpareServers

Esta directiva define cual es la cantidad de demonios que estarán corriendo como mínimo. Se monitorea el numero de conexiones en espera, si esta es menor a “MinSpareServers” se levantan los demonios necesarios.

Valor por Default: **5**

MaxSpareServers

Esta directiva define cual es la cantidad de demonios que estarán corriendo como máximo. Se monitorea el numero de conexiones en espera si esta es mayor a “MaxSpareServers” se eliminan los demonios necesarios.

Valor por Default: **5**

StartServers

Esta directiva define cual es la cantidad de demonios que iniciarán cuando se arranca apache.

Valor por Default: **5**

MaxClients

Esta directiva define cual es la cantidad de máxima de clientes que pueden conectarse simultáneamente.

Valor por Default: **20**

ErrorDocument

En caso de que un error o problema ocurra con Apache, este puede ser configurado para que haga una de las siguientes cosas:

1. Enviar un mensaje de error (default)
2. Enviar un mensaje personalizado
3. Redirigir a un URL local para manejar el problema o error.
4. Redirigir a un URL externo quien manejara el problema o error.

PidFile

PidFile file
Default file**: logs/httpd.pid**

La directiva PidFile define el lugar donde se almacenar el Id del proceso para el demonio raíz.

HostNameLookup

HostNameLookup on|off
Default file: **off**
Se puede usar en: **Server config, virtual host**

Esta directiva habilita la resolución de nombre cuando se establece una conexión. Por razones de rendimiento se sugiere no utilizarla para la resolución de nombres en las bitácoras existe el programa logresolve, también de apache.

<Directory> y <DirectoryMatch>

<Directory dir>
...
</Directory>

La directiva Directory permite aplicar otras directivas a un grupo de directorios, Es importante comentar que dir se refiere a directorios absolutos.

<Directory /> opera en todo el sistema

Options

La directiva options controla que características del servidor están disponibles para un directorio en particular.

Options [+|-] option [[+|-]option]

Puede ser colocado a *option none* en caso de que no se desee ninguna funcionalidad adicional.

- All
  - Todas las opciones se activan excepto MultiViews (default).
- ExecCGI
  - La ejecución de scripts CGIs es permitida.
- FollowSymLinks
  - Las ligas simbólicas son validas dentro de este directorio.

Nota: Aún cuando el servidor siguiera las ligas simbólicas, éste no cambiará la ruta definida por la directiva.

Optiones

- Includes
  - Permite el uso de Server-side includes.
- IncludesNOEXEC
  - Server-side includes son permitidos, pero #exec commands y #exec CGI son desactivados. Es también posible #include virtual CGI scripts desde directorios ScriptAliases
- Indexes
  - Si una URL mapea a un directorio solicitado y no hay DirectoryIndex (index.html) en este directorio, entonces el servidor devolverá un listado de directorio.

CGIs

CGI Common Gateway Interface

Los CGIs son programas que corren en el servidor, reciben parámetros desde el cliente y su salida es enviada al navegador de Internet.

Los CGI's nos permiten generar páginas dinámicas y fueron las primeras alternativas para generar dinamismo a un sitio Web.

ScriptAlias

ScriptAlias /cgi-bin /usr/www/cgi-bin
Se puede usar en: **Server config, virtual host**
Convierte las solicitudes vía URL al PATH absoluto donde residen los CGIs.

AddHandler

AddHandler cgi-scripts .cgi
Default: **No activado**
Se puede usar en**: Server config, virtual host**

AddHandler permite que determinada extensión sea relacionada a un evento en particular, en el caso de los CGIs lo que se indica es que la extensión .cgi queda identificada como un Script.

Accesos Restringidos

Access Control List

A través de la lista de control de acceso podemos indicar que clientes pueden conectarse a nuestro servidor y que es lo que pueden y no puede ver de nuestro sitio Web. Para esto hay diferentes directivas que utiliza Apache.

order

order allow, deny

Permite definir que reglas aplicaran primero para la denegación o acceso a un servicio de Web.

allow
deny

allow from all|host|env=env-variable [host|env=env-variable]
deny from all|host|env=env-variable [host|env=env-variable]

Permite definir un conjunto de clientes a los que se les pueden permitir o negar el acceso al servicio de Web.

La definición de los nombres se puede realizar mediante alguna de las siguientes formas:

- El nombre parcial de un dominio parcial. Ejemplo:
  - atacantes.com.mx
- Una dirección IP.
  - 200.38.166.1
- La pareja red y mascara
  - 10.2.0.0/255.255.255.0

- Un dirección de red definida por Classless Inter-Domain Routing (CIDR)
  - 10.2.2.110/24

AuthUserFile

AuthUserFile /usr/local/apache/htdocs/respaldos/.htpasswd

Indica cual es el archivo que contiene la lista de usuarios y passwords para realizar la autenticación.

AuthGroupFile

AuthGroupFile /dev/null

Define cual es el archivo que contiene la lista de grupos y los usuarios que la conforman para realizar la autenticación.

AuthName

AuthName "*Nombre del recurso protegido*"

Define como se llamará el nombre de autorización para el recurso protegido de forma tal que cuando aparece la caja de dialogo solicitando el password, para poder acceder al recurso.

Limit

Tiene como función definir que tipo de acceso que se tiene hacia el servidor web en determinados recursos.

<Limit GET POST>
</Limit>

Require valid-user

require valid-user backup

Es la directiva que permite definir que usuario tendrá acceso al recurso protegido

Manejo de Sitios Virtuales

Los sitios Web virtuales nos permiten tener varios sitios Web independientes dentro de la misma máquina sin tener que lanzar varias instancias de Apache. Así podemos optimizar recursos.

Hay fundamentalmente dos formas de definir sitios virtuales:

- **Por IP**: se identifica a cada servidor por una IP diferente.
- **Por el nombre del dominio**: el servidor Web escucha peticiones en una única IP y es el nombre de la página en que define qué contenido se va a mostrar.

Ésta última forma es la más sencilla y la que más habitualmente se encuentra. También se pueden definir sitios virtuales utilizando otros puertos (por ejemplo para sitios HTTPs) y mezclando identificación por IP y por nombre.

Los servidores virtuales permiten que un mismo servidor de apache pueda responder a diferentes solicitudes, con lo cual se puede mantener diferentes sitios web.

<VirtualHost nameserver>
</VirtualHost>

Vamos a ver un ejemplo de cómo utilizar un servidor virtual:

```
<VirtualHost 10.200.10.200:80>
   ServerAdmin root@namedomain.com
   ServerName enlinea.com
   DocumentRoot /usr/local/apache2/htdocs/enlinea/
</VirtualHost>

<VirtualHost 10.200.10.200:8080>
   ServerAdmin root@ namedomain.com
   ServerName enlinea.com
   DocumentRoot /usr/local/apache2/htdocs/admin./
</VirtualHost>
```

Ejecutar el servidor Web Apache

Una vez que hemos personalizado nuestro servidor Web, es hora es hora de hacerlo funcionar y para eso, Apache cuenta con dos archivos que realizan esa función:

- apachectl
- httpd

Ambos archivos aceptan como parámetros las opciones: *start, stop, restart, status.*

La diferencia de estos dos archivos es que apachectl fue creado para arrancar manualmente el servidor Web, mientras que httpd es un script creado para incluirse en los archivos de arranque del sistema y de esta manera, arrancar o detener el servidor cada vez que se detenga o arranque el sistema operativo.

Conclusión

Apache Web Server es el principal servidor de páginas HTML que existe actualmente. Su éxito radica en su facilidad de uso, en su gran flexibilidad para agregarle nuevas funcionalidades, su confiabilidad, su seguridad, además de que se encuentra disponible para la mayoría de las plataformas existentes y sobre todo, que es libre.

En este capítulo vimos algunas de las directivas más importantes ha configurar, cuando uno desea realizar una configuración básica de Apache, pero siempre hay que tener en cuenta el tipo de aplicaciones que queremos que trabajen sobre nuestro servidor o que tan sensible es la información que vamos a manejar, por ejemplo, si deseamos montar un negocio en Internet, donde podamos vender equipos de cómputo y software o algún servicio en especial, la seguridad es muy importante y requeriremos de herramientas adicionales como el módulo de Secure SSL, que nos permite codificar la información que transmitimos por Internet, el cual es un punto importante a la hora de realizar transacciones electrónicas.

Por otro lado, quiero resaltar que la compatibilidad de Apache con PHP es excelente y permite varias maneras de integrarlo al servidor de una manera sencilla y muy rápida. Incluso existen programas o paquetes de software que instalan y configuran Apache Web Server, PHP, MySQL y Perl, sobre un servidor Linux; con el propósito de reducir el esfuerzo del administrador, al instalar estos programas.

Diseño de páginas dinámicas con PHP

Capitulo 4

Capitulo 4: Diseño de páginas dinámicas con PHP

¿Qué es PHP?

PHP es un acrónimo que provienen de PHP: Hypertext Pre-processor, siguiendo el estilo marcado por Richard Stallmann, fundado de GNU, y como su nombre lo indica, es un preprocesador de HTML. Esto significa que las páginas .php son procesadas en el servidor Web antes de ser enviadas al navegador del usuario. Podríamos decir que las páginas HTML son construidas “al vuelo” ante cada petición recibida.

Esta filosofía de trabajo ha sido bautizada como páginas activas, es decir, de contenido dinámico que utiliza una fuente de datos, por ejemplo una base de datos, para Mostar siempre contenido actualizado. Desde el punto de vista del diseñador Web, esto aporta una ventaja muy importante. Podemos centrarnos en el diseño del portal o aplicación Web en una primera fase de desarrollo (que es lo que hemos venido haciendo), y posteriormente actualizar la información modificando convenientemente el contenido de la base de datos, ya que PHP reconstruirá la página utilizando para ello las directivas que hayamos utilizado para ello.

Es importante mencionar que PHP es una combinación de diferentes características de C, Java y Perl. Pero mucho más fácil de aprender. Aporta librerías con múltiples funciones predeterminadas para el cálculo matemático, acceso a red, envío de correo, utilización de expresiones regulares y muchas más posibilidades. Una de las principales capacidades de PHP es su interacción con Bases de Datos. Para conectarse una base de datos y obtener información a partir de ella y utilizarla en el diseño de la página Web son necesarias un par de líneas de código. Además está diseñado para trabajar directamente con MySQL, Oracke, SyBase, mSQL, PostgreSQL y cualquier base de datos con soporte para ODBC (por ejemplo: SQL Server).

¿Por qué PHP?

Cuáles son las ventajas de PHP frente a otros lenguajes de programación dinámica de páginas Web como son ASP, JSP, ROXEN, etc.

- Es libre
- Es de código abierto
- Multiplataforma
- Es soportado por la mayoría de los servidores Web: Apache, Roxen, etc.
- Soporta casi cualquier base de datos y compatibles con ODBC.
- Existe mucha documentación.

- Perfecta integración entre Apache, PHP y MySQL: es muy rápido como módulo de Apache.
- Bastante sencillo de aprender y utilizar: sintaxis bien clara y definida.
- Escalable a través de módulos.
- Seguro y rápido como módulo de Apache.

¿Cómo funciona PHP?

Al acercarnos a una nueva tecnología conviene tener en mente de manera básica como funciona, de forma que sea más sencillo posteriormente hacer uso de la misma. Para conocer el funcionamiento de PHP, solo debemos tener en cuenta lo siguiente:

- El código PHP se escribe dentro de la página HTML. Esto quiere decir que una pagina con extensión .PHP (.php3 ó .php4) no es más que una página HTML normal que hace uso de una nueva etiqueta predefinida por PHP y utilizada para incluir nuestras líneas del lenguaje.
- Obviamente, esa página no se envía tal cual al navegador del usuario que visita nuestra página Web, sino que se procesa antes. Por lo tanto, el software servidor donde residen nuestras páginas (Apache Web Server), debe ser capaz de ejecutar PHP, es decir, de decirle a PHP que procese la página.
- Por lo tanto, preprocesar significa que PHP ejecuta las líneas de código que hemos escrito, y que en la salida al navegador devuelve solamente HTML normal. Por supuesto, el "hueco" donde se encontraba nuestro código PHP es "rellenado" por lo que nosotros hayamos decidido cuando programamos esas líneas.

Vamos a ver un ejemplo para terminar de aclarar los conceptos. En el Cuadro 1 se tiene una página de PHP sencilla. Como se puede observar es una página HTML excepto por dos características:

1. La extensión del archivo es **.php**
2. Dentro de la página existe una etiqueta HTML que no nos será familiar **<? ?>.**

```
<HTML>
<HEAD>
<TITLE>Pagina de PHP </TITLE>
</HEAD>
<BODY>
<H1> Un ejemplo sencillo del uso de PHP </H1>

<?
/* la etiqueta "<?" indica comienzo del código PHP */
$fecha= date("Y-m-d");
```

```
PRINT “<B><CENTER> Hoy es: $fecha.</CENTER></B>”;

#Esta línea también es un comentario
?>

</BODY>
</HTML>
```

Cuadro 1. Ejemplo de página .PHP

En efecto, ése es nuestro código escrito en PHP. Lo que va a ocurrir cuando el cliente solicite esta página en el servidor Web, es que éste va a ver una página con extensión .PHP por lo que invocará al pre-procesador pasándole como entrada nuestro archivo.

El pre-pocesador de PHP no tocará nada excepto el contenido de las etiquetas <?. Que será ejecutado, y sustituido por la posible salida. En éste ejemplo vemos como el código PHP que hemos escrito con la función PRINT para escribir algo en HTML. Eso será lo que sustituya al código PHP que habíamos incluido en el archivo original.

En el Cuadro 2 veamos el resultado en el navegador del cliente:

```
Un ejemplo sencillo del uso de PHP
Hoy es: 2006-02-18
```

Cuadro 2: Página HTML que visualiza el navegador del usuario.

Para poder entender rápidamente cualquier código escrito en lenguaje PHP es necesario tener en cuenta que:

1. Todas las líneas de código PHP deben ir definidas entre etiquetas “<?” y “?>”. Existen otras posibles formas de las etiquetas, aunque se comportan exactamente igual, por ejemplo, “<?php” y “?>”.
2. Los comentarios del código van definidos entre “/*” y “*/”, en el caso de ocupar varias líneas, o podemos ocupar simplemente “#” para incluir un comentario de una sola línea.
3. Toda salida que queramos que sea visible en el navegador del usuario debe ser “enviada” utilizando la sentencia PRINT.
4. Casi todos los comandos del lenguaje PHP finalizan en punto y coma “;”.
5. Obviamente cualquier etiqueta HTML incluida en los comandos PRINT será interpretada por el navegador del usuario.
6. Las páginas PHP, es decir las páginas HTML que contengan código PHP, deben ser almacenadas con la extensión *.php*, *php3* o *php4* (actualmente ya esta la versión 5 de PHP).

7. Las funciones de PHP, como por ejemplo date, se definen por su interfaz, al igual que en cualquier otro lenguaje de programación. Así, en el manual oficial de PHO nos indica que la función date tiene la siguiente interfaz en uso:

   ```
   Interfaz: string date (string format, int timestamp)
   ```

   La función puede tomar dos argumentos, pero el último es opcional. El argumento obligatorio define el formato con el que se presentará la fecha actual, y la función devolverá una cadena de texto (*string*) con la fecha obtenida.

Requisitos para poder utilizar PHP

Como ya hemos comentado, el intérprete de PHP es llamado por el servidor Web cada vez que el usuario solicita una extensión *.php* ó similar desde su navegador. Por lo tanto, para que nuestras páginas funcionen adecuadamente deben estar alojadas en un servidor Web con ésta capacidad (nuestro servidor Apache la soporta).

Si tenemos Linux instalador en nuestra computadora, podemos instalar nuestro propio servidor Apache con soporte de PHP y así diseñar las aplicaciones y nuestro servidor de contenido en nuestra propia máquina. El proceso de instalación de PHP se ha simplificado bastante en los últimos tiempos, e incluso algunas distribuciones Linux instalan por defecto Apache con soporte PHP (como es el caso de Slackware).

Instalación de PHP sobre el servidor Apache.

Para poder instalar PHP en nuestra máquina Linux, necesitamos primero descargar el software a través de los siguientes vínculos:

PHP 5: http://www.php.net/downloads.php
HTTPD 2.2: http://httpd.apache.org/download.cgi

Es muy sencillo instalar ambas aplicaciones si seguimos al pie de la letra el archivo de instalación (INSTALL) incluido en el paquete de PHP que descarguemos.

A continuación, podemos ver el procedimiento para instalar PHP y Apache Web Server[1]:

Tu puedes cambiar las ‘xxx’ por la versión adecuada de Apache que estés instalando.

[1] Información obtenida del archivo INSTALL contenido en el paqueta de instalación de PHP.

```
1.  gunzip apache_xxx.tar.gz
2.  tar -xvf apache_xxx.tar
3.  gunzip php-xxx.tar.gz
4.  tar -xvf php-xxx.tar
5.  cd apache_xxx
6.  ./configure --prefix=/usr/local/apache2 --enable-so
7.  make
8.  make install
```

En éste momento el servidor Apache ya está instalado en */usr/local/apache2/.* Para probarlo puedes ejecutar lo siguiente:

```
/usr/local/apache2/bin/apachectl start
```

Detén el servidor Apache para continuar con la instalación de PHP:

```
/usr/local/apache2/bin/apachectl stop.

Continuando con la instalación. Entramos al directorio de php.

9.  cd ../php-xxx
10. ./configure --with-apxs2=/usr/local/apache2/bin/apxs --with-mysql –
with-ssl
11. make
12. make install
13. cp php.ini-dist /usr/local/lib/php.ini
```

Después de haber hecho esto, vamos a configurar nuestro archivo *httpd.conf* ubicado en */usr/local/apache2/conf* para que pueda interpretar los archivos con extención *.php*. Agregamos al archivo las siguientes líneas:

```
AddType application/x-httpd-php .php .phtml
AddType application/x-httpd-php-source .phps
```

Hemos terminado. Ahora si levantamos nuestro servidor Apache:

```
/usr/local/apache2/bin/apachectl start
```

Cuadro 3: Instrucciones de instalación (Apache 2 Shared Module Version)

El lenguaje PHP

PHP es un lenguaje de programación similar a otros lenguajes de propósito general: contiene variables, tipos de datos, sentencias de control, funciones e incluso es orientado a objetos.

Variables

En PHP las variables van precedidas por un signo de dólar ($). El nombre utilizado puede contener letras, números y el carácter subrayado (_). No podemos utilizar espacios o caracteres no alfanuméricos.
Nombres válidos para una variable son, por ejemplo:

```
$x;
$nombre_completo;
$091;
```

Nada más como una pequeña nota para aquellas personas que no hayan programado anteriormente, una variable es la forma que tiene el programador de denominar a una zona de la memoria del programa. Por lo tanto, al definir una variable, y dependiendo del tipo, se reserva un área de la memoria para almacenar su contenido.

Como ya comenté antes, el signo de punto y coma indica el final de una instrucción en PHP. La asignación de valores es también bastante típica y se realiza utilizando el carácter de igual (=).

```
$x = 300;
```

Si queremos hacer uso del valor del contenido en la variable debemos continuar utilizando el signo de dólar para indicar al interprete que la cadena no es una literal. Por tanto:

```
print $x;
```

Es equivalente a

```
print 300;
```

Variables dinámicas

Podemos utilizar el valor de una variable para crear otra con ese nombre. Así, si definimos una variable '$nombre', podemos crear otra cuyo contenido será el contenido de la primera:

```
$nombre = "juan";
```

Es equivalente a

```
$temp = "nombre";
$$temp = "juan"; // PHP crea una variable $nombre cuyo valor es "juan"
```

Sin embargo, debemos tener cuidado a la hora de utilizar variables dinámicas, ya que puede que no consigamos hacer realmente lo que pretendemos. Por ejemplo, si queremos imprimir en pantalla el contenido de la variable $nombre, y dado que la hemos creado a partir del valor de la variable $temp, podríamos pensar que:

```
print “$$temp”;
```

Imprimiría “juan”. Pero lo que obtendremos será la cadena de texto “$nombre”, es decir, un signo de dólar seguido del valor de la variable $temp.

Es necesario que ayudemos a PHP a entendernos correctamente. Para ello debemos utilizar los símbolor “{” y “}”:

```
print “${$temp}”;
```

Ahora sí conseguiremos acceder al valor de la variable $nombre, es decir la cadena de texto “juan”.

La pregunta que ahora nos podemos hacer después de conocer las variables dinámicas es: ¿Para qué sirven esto? La respuesta es sencilla: éste mecanismo nos permitirá crear un número elevado de variables de manera automática en un bucle o utilizando un operador de concatenación.

Asignación de variables por referencia

Normalmente, las variables se asignan por valor. Es decir, cuando asignamos una variable $origen y otra $destino, se almacena una copia del valor de la primera en la segunda. Si cambiáramos el valor $origen, la variable $destino seguirá teniendo el mismo valor:

```
$origen= 2006;
$destino= $origen; //Ahora destino posee el valor 2006.
$origen =150; //hemos cambiado el valor $origen, pero $destino sigue
valiendo 2006.
//Ya que la asignación se ha llevado por VALOR.
print $origen;   //imprime 150
print $destino;  //imprime 2006
```

En PHP, a partir de la versión 4, se puede modificar éste comportamiento, de tal manera que la asignación no se limite a copiar el valor de una variable a otra, sino que las enlace para que un cambio en cualquiera de ellas afecte a la otra:

```
$origen=2006;
$destino=&$origen;  //ahora $destino poseé el valor 2006.
$origen=150; //hemos cambiado el valor de origen y además el del
destino
//Ya que la asignación se ha realizado por referencia.
print $origen;   //imprime 150
print $destino;  //imprime 150
```

El único cambio en el código es el uso del carácter “&” que indica “la dirección de memoria apuntada por”. Es decir que la asignación por referencia consigue que una variable apunte a la misma zona de memoria apuntada por otra.

Tipos de datos

Los tipos de datos permiten indicar en nuestro programa que clase de valor va a almacenar una variable. Escoger bien ese tipo es muy importante a la hora de programar, para así utilizar adecuadamente la memoria disponible.

Algo que me gusto mucho de PHP (y creo que de cualquier lenguaje interpretado), es la flexibilidad para utilizar las variables, ya que se establece el tipo de variables al realizar una asignación. Es decir, una variable puede contener un entero al principio de nuestro programa y posteriormente utilizarla para almacenar una cadena de texto.

La siguiente tabla muestra los tipos de datos que podemos utilizar en PHP:

Tabla1

Operadores y Expresiones

Ahora hemos aprendido a utilizar variables, asignarles un valor e incluso cambiar su tipo de datos. Sin embargo todo lenguaje de programación debe aportar las herramientas necesarias para poder manipular esos valores de forma que obtengamos otros nuevos.

Los operadores nos permiten precisadamente eso, realizar operaciones utilizando las variables de nuestros programas. Atendiendo a su función podemos clasificarlos en operadores de asignación, aritméticos, concatenación, comparación, lógicos…

Tabla2

Sentencias de control y arrays

Hasta ahora, todo el código PHP que hemos escrito se ejecutaba de manera lineal y una única vez cada instrucción. Sin embargo, cualquier lenguaje de programación mínimamente elaborado debe aportarnos sentencias que nos permitan definir comportamientos más complejos:

- Repetir la ejecución de un conjunto de instrucciones (bloque) un determinado número de veces, hasta que se cumpla una condición.
- Ejecutar un bloque únicamente si cumple una condición, etc.

Las sentencias de control nos permite exactamente eso: condicionar la ejecución de varias instrucciones o forzar su ejecución repetitiva.

Sentencias de control condicionales

La mayor parte de los programas necesitan comprobar si se cumplen una serie de condiciones, para ejecutar distinto código en el caso de que no sea así. En realidad, dado que utilizamos PHP para crear páginas HTML dinámicamente, este comportamiento es esencial para el programador.

Vamos a detallar que sentencias de control condicionales se encuentran disponibles en PHP.

Sentencia IF

La sentencia *if* evalúa una expresión; si devuelve un valor trae (verdadero) el bloque de código ejecutado; en caso contrario, el bloqueo se ignora.
El bloque de código es un conjunto de sentencias que se encuentran ubicadas entre los signos { }, por ejemplo:

```
If (expresión)
{
 // Una o varias instrucciones.
}
```

Podemos observar un ejemplo en el siguiente cuadro, en el que hacemos uso del operador de comparación “==” para crear una expresión:

```
$var=”activo”;
if ($var == “activo”)
{
    print “Estado actual ACTIVO”;
}
```

Si el código a ejecutarse se compone de un única sentencia no es necesario utilizar los símbolos {} para encerrar el bloque:

```
If (expresión)
    Sentencia;
```

Es posible que queramos ejecutar cierto código si NO se cumple la condición. En este caso, una solución rápida sería definir la nueva sentencia de control if con la negación de la expresión anterior. Afortunadamente, podemos utilizar la cláusula else:

```
If (expresion)
{
   // Código a ejecutar si se cumple la expresión
}
else
{
  //Código a ejecutar si NO se cumple la expresión.
}
```

Volviendo al ejemplo que he utilizado antes:

```
if ($var == “activo”)
     Print “Estado actual: Activo”;
else
{
   print “ACTIVANDO…”;
   $var = activo;
}
```

Dependiendo del valor de $var, se ejecutará el primer bloque de código o el segundo. Independientemente de la condición, se ejecutará la última sentencia, ya que se encuentra fuera de la de control.

La versión 4 de PHP nos permite definir múltiples condiciones en cascada mediante el uso de la cláusula *elseif*. En este caso, *else* definirá el código a ejecutar si no se cumple ninguna de las condiciones enlazadas:

```
If (expresión1)
{
   //Código a ejecutar si se cumple la expresión1
}
elseif (expresión2)
{
  //Código a ejecutar si NO se cumple expresión1 pero SI expresión2.
}
```

```
else
{
  //Código que se ejecuta si no se ha cumplido ninguna de las expresiones anteriores.
}
```

La cláusula final *else* puede obviarse si no necesitamos ejecutar ningún código por defecto, y podemos enlazar cualquier número de cláusulas *elseif* que precisemos.

El operador interrogación “?” es similar a la sentencia if pero únicamente es capaz de devolver un valor dependiendo de si la expresión es verdadera o falsa:

```
(expresión)?valor_a_devolver_si_la_expresión_es_verdadera
:valor_si_la_expresión_es_falsa ;
```

Por ejemplo:

```
$estado= ($registro == “grabado”)?”Correctamente almacenado”:”Es necesario grabar”;
print “$estado”;
```

La sentencia SWITCH

Al igual que if, nos permite modificar el flujo de nuestro programa dependiendo del valor de una expresión. La diferencia fundamental entre ambas sentencias es que con switch evaluamos una única expresión, pero indicamos que código ejecutar dependiendo de su valor, y con if-elseif-else podemos evaluar múltiples expresiones. El resultado una expresión evaluada dentro de una sentencia if es un valor true o false, mientras que dentro de una sentencia switch el resultado es contrastado contra un conjunto de valores:

```
Switch(expresión)
{
   case resultado1:
      //código ejecutado sólo si expresión es igual a resultado1
      break;
   case resultado2:
      //código ejecutado sólo si expresión es igual a resultado2
      break;
   default:
     //código ejecutado si no se ejecutó ningún bloque anterior.
      break;
}
```

La palabra reservada *break* finaliza la ejecución de la sentencia *switch*. Si no se incluye al final del case, el siguiente case será ejecutado, por lo que es muy importante incluirlo siempre para evitar comportamientos “extraños” en nuestro programa.

Normalmente la expresión que se evalúa en una sentencia *switch* suele ser simplemente una variable y los diferentes case’s nos permiten indicar que hacer dependiendo del valor que contenga:

Sentencia de control WHILE

La sentencia *while* comprobará en primer lugar si la expresión devuelve *true*, en cuyo caso ejecutará el bloque de código. Después volverá a comenzar el proceso, comprobando de nuevo la expresión.
Únicamente cuando sea evaluada como *false* finalizará la sentencia *while*.

```
While (espresión)
{
   //código
}
```

Obviamente, lo normal es modificar partes dentro de la expresión dentro del bloque de código, ya que si no se convertirá en un bucle infinito:

Por ejemplo:

```
$contador=1;
while ($contador <= 5)
{
    print $contador;
    $contador++;
}
```

Sentencia DO-WHILE

Es muy similar a la sentencia *while*, la única diferencia es que primero se ejecuta el bloque de código y después se evalúa la expresión. Por lo tanto es muy útil si queremos que el código se ejecute al menos una vez, independientemente si la expresión resulta falsa.

```
do
{
   //Código
}
while (expresión)
```

Sentencia FOR

Cualquier comportamiento que obtengamos con un *for*, puede producirse sin problemas utilizando una sentencia *while*. La única diferencia es que *for* nos facilita la descripción de bucles de incremento o decremento de un contador debido a su sintaxis:

```
for (asignación de variable; expresión; incremento o decremento)
{
   //código
}
```

Vamos a ver un ejemplo:

```
For ($contador=1; $contador <=5; $contador++)
{
     print $contador;
}
```

La variable $contador es inicializada una única vez, es implementada al inicio de cada ejecución del bucle de manera automática. En este caso concreto, se inicializa la variable igual a 1 en la primera ejecución y posteriormente se comprueba si se cumple la condición (variable menor o igual a 5). Sólo en éste caso se ejecuta el bloque de código. Después se incremente en uno la variable ($contador++) y se vuelve a comprobar la expresión, así hasta que el valor de la comprobación sea *false*.

Uso de las cláusulas BREAK y CONTINUE en bucles.

Las cláusulas *break* y *continue* flexibilizan el uso de los bucles de control, permitiendo modificar su comportamiento según las necesidades.
En concreto, *break* permite forzar el fin de la ejecución de un bucle, aunque no se cumpla la condición (es decir, la expresión utilizada en *while* o *for*). Por ejemplo:

```
for ($contador = 10; $contador >=0; $contador--)
{
    if ($contador == 0 )
         break;
    print $contador;
}

$valor = 5;
```

En el ejemplo podemos observar un bucle *for* que cuenta desde 10 bajando a 0. Dentro del bloque de código incluimos una condición adicional: si contador es igual a cero, *rompemos* inmediatamente el bucle. La ejecución continuará en la siguiente línea después del bucle, en éste caso una asignación de variable.

Por su parte, *continue* permite también forzar el fin de la ejecución, pero solo de la iteración en curso, por lo que el programa continua la ejecución al inicio del bucle, comprobando la expresión:
El resultado de la expresión será: 5 3 2 1 0.

```
for ($contador = 5; $contador >=0; $contador--)
{
  if ($contador == 4)
     continue;
  print “$contador”;
}
```

Al cumplirse la condición adicional ($contador == 4) se ejecuta el *continue*, que finaliza la ejecución de la iteración en curso (no se ejecuta el resto del bloque de código) y vuelve al inicio del *for*.

Arreglos de datos

Arreglos simples

Los *arrays* son listas de variables, o mejor dicho, una variable que puede contener múltiples elementos indexados por un número o una cadena de caracteres. Gracias a ésta característica podemos utilizarlos para almacenar y a la vez mantener organizados datos de nuestra aplicación.

Por defecto los arreglos son listas de valores indexados por número y pueden ser creados de dos modos: utilizando la función del lenguaje array() o directamente utilizando el símbolo [], por ejemplo:

```
$finalistasMundial= array (“México”, “Alemania”, “Brasil”, “Argentina”);
```

Dado que la primera posición de los arrays se referencia con el número 0, podemos acceder al segundo elemento del array usando el índice “1”:

```
print “El ganador es : $finalistasMundia[1]”;  //Imprimiría: El ganador es : Alemania
```

Vemos que el arreglo *finalistasMundial* contiene cuatro valores, ordenados por número. Obviamente también podríamos haber definido cuatro variables simples

para almacenar valores, pero como vamos a ver, el trabajar con arreglos nos facilita enormemente la gestión de datos.
También podemos crear un arreglo sin utilizar la función descrita anteriormente:

```
$finalistasMundial[]=”México”;
$finalistasMundial[]=”Alemania”;
$finalistasMundial[]=”Brasil”;
$finalistasMundial[]=”Argentina”;
```

En éste caso PHP inicializa automáticamente los índices de cada posición. Este método también puede utilizarse para añadir nuevos elementos a un arreglo ya existente.

Arreglos Asociativos

El hecho de indexar los arreglos por número es útil en muchas circunstancias; nos permite ordenar los valores por importancia, por orden de creación, etc. Sin embargo, en ocasiones es significativamente más útil el poder acceder a un elemento del arreglo utilizando como índice una cadena de texto. Esta funcionalidad es mucho más cercana a las bases de datos tradicionales: ¿Qué es preferible, indexar la edad en el campo número 4 o en un campo llamado EDAD? Por tanto, se denominan arrays asociativos a aquellos indexados por cadenas de texto y no por números.

Para definir un arreglo asociativo se utiliza también la función array(), por ejemplo:

$alumno= array (nombre => “Rodrigo”, apellidos => “Arias Hurtado”, carrera => “Ing. en Computación”);

Y así podemos acceder a cualquiera de los campos:

```
print $alumno[apellidos]  //Imprimimos “Arias Hurtado”
```

El nombre del campo se denomina clave (nombre, apellidos, carrera), y a cada una de ellas le corresponde un valor. Al utilizar las claves no es necesario entrecomillarlas, a no ser que se compongan de varias palabras.

Por último podemos crear arreglos asociativos utilizando directamente la asociación:

```
$alumno[nombre]=”Rodrigo”;
$alumno[apellidos]=”Arias Hurtado”;
$alumno[carrera]=”Ing. en Computación”;
```

Funciones útiles para trabajar con arreglos

La ordenación es una acción especialmente útil si trabajamos con arreglos. Sin embargo, es necesario es necesario utilizar las funciones que incluye PHP con mucho cuidado, y diferenciando claramente si las queremos aplicar sobre un arreglo numérico o asociativo, porque los resultados pueden ser los no esperados.

PHP incluye seis funciones de ordenación aplicables a arrays como vemos en la siguiente tabla:

Función	Aplicable a	Comportamiento
sort()	Arreglos numéricos	Ordena alfabéticamente si algún elemento es una cadena de texto, o numéricamente si todos los elementos son números.
Rsort()	Arreglos numéricos	Ordena (ORDEN INVERSO) alfabéticamente si algún elemento es una cadena de texto, o numéricamente si todos los elementos son números.
Asort()	Arreglos asociativos	Ordena de igual manera que sort() utilizando el campo valor, no la clave.
rasort()	Arreglos asociativos	Ordena (ORDEN INVERSO) de igual manera que sort() utilizando el campo valor, no la clave.
Ksort()	Arreglos asociativos	Ordena de igual manera que sort() utilizando el campo clave, no el valor.
krsort()	Arreglos asociativos	Ordena (ORDEN INVERSO) de igual manera que sort() utilizando el campo clave, no el valor.

Integración de PHP con formularios HTML

Esta sección no será de gran ayuda en la realización de nuestro proyecto final, debido a que necesitamos crear algunas páginas que interactúen con los datos enviados vía Web por el usuario, como paso previo a la integración con bases de datos como MySQL.

Trabajando con formularios

Los formularios en HTML, son la principal vía por la que el usuario puede enviar información al servidor HTTP (nuestro servidor Apache). PHP fue diseñado muy especialmente teniendo en cuenta esta vía de comunicación, y como veremos,

ofrece muchas facilidades al programador para gestionar los datos recibidos vía formulario.

Desde el punto de vista del programador en PHP, es necesario poder realizar las siguientes acciones de la manera más sencilla y automatizada posible:

- Acceder a variables de entorno del usuario, enviadas por el navegador.
- Acceder a los campos que componen un formulario, es decir, a lo que ha escrito o seleccionado el usuario.
- Trabajar fácilmente con elementos de selección múltiple, presentes en los formularios HTML.
- Poder incluir en un mismo documento HTML el formulario (FORM) y el código PHP que procesará la información recibida.
- Enlazar varios formularios, almacenando el estado mediante campos ocultos (hidden).
- Redirigir al usuario una nueva página.
- Diseñar formularios (y el código PHP necesario) que permitan subir (UPLOAD) archivos al servidor.

Así que mi propósito que es toquemos todos estos puntos a los largo del capítulo.

Acceso a variables de entorno.

Cuando un usuario visita una página Web de nuestro servidor se establecen variables que describen el entorno existente, tanto del lado del cliente como del servidor. Estas variables de entorno son accesibles desde cualquier parte de nuestro código, ya que el intérprete de PHP las define como *variables globales.*

En la siguiente tabla podemos observar algunas de éstas variables de entorno. Pero además de ellas, PHP crea otras variables globales con información útil para el programador. Por ejemplo, la variable global $PHP_SELF, contiene la ruta completa del script que se está ejecutando. Aunque parece un dato inútil, vamos a ver que es fundamental escribir programas PHP integrados a la perfección con formularios HTML.

Variable	Valor	Ejemplo
$HTTP_USER_AGENT	El nombre y número de la versión del cliente.	Mozilla/4.76 X11;I;Linux2.2.4 i386
$REMOTE_ADDR	La dirección IP del cliente	132.248.62.51
$REQUEST_METED	Indica si la petición fue GET o POST.	POST
$QUERY_STRING	Si método es GET contiene los datos pegados a la URL	Login=rodrigo&tfno=5567
$REQUEST_URI	La dirección completa de la	/catalogo/formas/f.php?lo

	petición incluyendo la cadena de datos.	gin= rodrigo&password=hola123
$http_REFERER	La página desde la que se realizó la llamada.	http://www.unam.com.mx/registro.html

Variables de entorno

Puede ser muy útil saber que existe un arreglo asociativo denominado $GLOBALS, creado por PHP, que contiene todas éstas variables (por ejemplo $GLOBALS[“PHP_SELF”]), así como otras que vayamos definiendo en nuestro programa.

¿Cómo acceder a los datos de un formulario?

Supongamos que queremos, por un lado, una página HTML con un formulario, y por otro diseñar un programa en PHP que trate los datos una vez que el usuario los envíe. Posteriormente veremos como tener en nuestra página PHP el formulario y el código.

```
<HTML>
<head>
<title>Formulario HTML simple</title>
</head>
<body>
<form action=”ejemplo.php” method=”GET”>
     <input type=”text” name=”usuario”>
     <textarea name=”direccion” rows=”5” cols=”40”></textarea>
      <br>
      <input type=”submit” value=”Enviar”>
</form>
</body>
</html>
```

Formulario HTML

En el Cuadro anterior podemos ver un formulario HTML que contiene un campo de texto con el nombre usuario, un área de texto con el nombre dirección y un botón de envío (submit). El formulario define como elemento ACTION un archivo llamado ejemplo.php, que procesará la información.

Dado que sólo ponemos el nombre del archivo, éste debe residir en el mismo directorio que la página HTML que contiene el formulario. Ahora podemos escribir ese archivo *ejemplo.php*:

```
<HTML>
<head>
<title>Script que procesa los datos del formulario</title>
</head>
<body>
<?php
  print " Hola <b> $usuario </b> <p> \n";
  print " Direcci&oacute;n: $direccion <p>\n" ;
?>
</body>
</html>
```

Ejemplo.php

Este código se ejecutará cuando un usuario presione el botón de envío en el formulario HTML. Como vemos, accedemos directamente a las variables *$usuario y $direccion*, que contendrán los datos introducidos por el usuario en el formulario. Como podéis observar, crear programas que accedan a los datos de un formulario es así de simple.

El intérprete de PHP crea variables globales con el nombre de la variable en el FORM antes de ejecutar el programa, por lo que el programador sólo se preocupa de qué hacer con esos valores.

Acceso a entradas múltiples

La etiqueta SELECT permite definir dentro de un formulario entradas donde el usuario debe seleccionar un valor de entre varios posibles. Sin embargo, si se incluye el atributo MULTIPLE a la etiqueta, será posible seleccionar varios valores simultáneamente.

```
<SELECT name="marca" multiple>
```

El script PHP que acceda a la variable *marca* sólo verá un valor simple, normalmente el último seleccionado (aunque no siempre tiene porqué ser el último). Lo interesante de todo esto es qué podemos hacer para cambiar este comportamiento, y con seguir un acceso sencillo desde nuestro código PHP a todos los valores seleccionados por usuario en este campo múltiple. La idea consiste en pedir a PHP que trate las selecciones como parte de un arreglo, utilizando como nombre de variable en el FORM *marca[]*:

```
<HTML>
  <head>
      <title>Formulario HTML con SELECT m&uacute;ltiple</title>
  </head>
<body>
<form action=" script.php" method="POST">
<input type="text" name="usuario">
<br>
<textarea name= "direccion" rows="4" cols="50"></textarea><br>
<select name= "productos[]" multiple>
   <option>Walkman
   <option>Camara
   <option>Ordenata
</select>
<br>
<input type=submit" value="Enviar">
</form>
</body>
</html>
```

En el script PHP que procese este formulario existirá un arreglo $*productos* con todos los elementos seleccionados:

```
<HTML>
   <HEAD>
<TITLE>Script para procesar un formulario con SELECT &uacute;ltiple</TITLE>
  </HEAD>
<BODY>
<?php
print "Hola <b>$usuario</b> <p> \n";
print "Dirección: $dirección <p> \n";
print "Productos elegidos: <p> \n";
print "<OL>\n";
foreach ($productos as $valor)
{
   print "<LI>$valor<br>\n";
}
?>
</BODY>
</HTML>
```

El elemento SELECT no es el único que puede contener múltiples valores. Por ejemplo, si utilizamos varios INPUT de tipo checkbox y les damos el mismo nombre, podemos simular un SELECT. En este caso también podemos utilizar el ejemplo anterior, dándoles un nombre con corchetes para que PHP incluya todos los valores en un arreglo.

```
<input type="checkbox" name="productos[]" value="Mouse">
<input type="checkbox" name="productos[]" value="Teclado">
<input type="checkbox" name="productos[]" value="Monitor">
```

Acceso a todos los campos de un formulario.

Hasta ahora todo lo que hemos visto funciona, pero siempre y cuando el script PHP sepa de antemano los nombres que se han utilizado en el FORM para cada una de las entradas que se desean leer.

Pero normalmente es necesario escribir código que se adapte a cambios en el formulario, o incluso procesar información que provenga de distintos formularios, porque no se puede depender del nombre de las variables. PHP facilita al programador todas las variables que provienen del formulario en un arreglo asociativo. Dependiendo del método utilizado para el envío de los datos al servidor (METHOD=GET ó METHOD=POST) debe utilizarse un arreglo distinto, $HTTP_GET_VARS ó $HTTP_POST_VARS. En el siguiente ejemplo vemos un script que procesa datos enviados desde un formulario con el método GET:

```
<HTML>
<HEAD>
<TITLE> Script que procesa datos con GET </TITLE>
</HEAD>
<BODY>
<?

foreach($HTTP_GET_VARS as $clave=>$valor)
{
   if (gettype ($valor) == "array")
   {
      print "$clave == <br> \n";
        foreach($valor as $contenido)
           print "-- $contenido";
   }
   else
   {
      print "$clave == $valor<br>\n";
   }
}
?>
</BODY>
</HTML>
```

Envío de datos mediante GET con $HTTP_GET_VARS

Este código imprime los nombres y valores de todos los campos del formulario. La función gettype() es utilizada para averiguar si alguno de los campos es un valor múltiple y necesita un tratamiento especial.

Distinguir entre POST y GET

Para que el código escrito sea aún más flexible, debemos poder distinguir si los datos se envían con POST o GET, de forma que sepamos que arreglo asociativo utilizar.

```
<HTML>
<HEAD>
<TITLE> Script que procesado válido para GET y POST </TITLE>
</HEAD>
<BODY>
<?

$PARAMS= (isset($HTTP_POST_VARS)) ? $HTTP_POST_VARS : $HTTP_GET_VARS;
foreach ($PARAMS as $clave => $valor)
{
   if (gettype ($valor) == $array )
   {
      print "$clave == <br>\n";
        foreach ($valor as $contenido)
           print "-- $contenido <br>\n";
   }
   else
   {
      print "$clave == $valor <br> \n";
   }
}

?>
</BODY>
</HTML>
```

Procesador de parámetros enviados mediante GET o POST

Podemos averiguar el método utilizado consultando la variable de entorno $REQUEST_METHOD, que deberá contener la cadena "POST" o "GET".

Dependiendo de su valor sabremos qué arreglo asociativo utilizar: HTTP_GET_VARS ó HTTP_POST_VARS. Sin embargo, no en todos los sistemas funcionará este procedimiento, ya que puede que no se haya definido correctamente $REQUEST_METHOD. Para evitar este problema, lo mejores comprobar si está definido el arreglo $HTTP_POST_VARS que sólo existirá si PHP obtuvo los datos vía POST.

El siguiente script es capaz de procesarla información enviada por cualquier formulario, independientemente del método que haya utilizado:

Como podemos ver, utilizamos el operador *"?"* para evaluar si existe o no el arreglo asociativo HTTP_POST_VARS. En caso de que exista se utiliza, y en

caso contrario se hace uso de HTTP_GET_VARS. A partir de aquí el resto de código es equivalente al tratamiento realizado en los ejemplos anteriores.

Cómo utilizar HTML y código PHP en la misma página

En ciertas circunstancias puede ser útil disponer en la misma página del código PHP y el HTML. Este es el caso si tenemos que mostrar el mismo formulario varias veces al usuario, o simplemente por forma de trabajo, al no tener que editar por separado los archivos.

En cualquier caso, el disponer de código PHP en la página puede ser una traba si trabajamos con diseñadores que sólo conocen HTML, JavaScript y poco más. Para facilitar la integración del diseño Web con la potencia de las páginas dinámicas es conveniente hacer uso en lo posible de llamadas a funciones, que permitirán tener páginas con código HTML bastante limpio.

En todo caso, supongamos que queremos diseñar una página PHP que muestre un formulario, y que además procese ella misma los datos cuando el usuario presione el botón de envío (submit).

Para ello tenemos que tener en cuenta dos puntos:

1. La acción (ACTION) del formulario debe ser $PHP_SELF, es decir, la ruta a la propia página PHP.
2. Dentro de nuestro código será necesario que incluyamos alguna condición que nos permita saber si debemos mostrar el formulario, o el resultado de procesar una petición.

El siguiente código de ejemplo nos permitirá observar estas dos condiciones:

```
<?
$numero_para_adivinar = 25;
$texto = "";
if (! isset ($numero))
/*Esta es la condición. Si no existe $numero es que NO nos invocan
como
procesadores del formulario, sino como página normal*/
{
   $texto = "Intenta adivinar el número ";
}
elseif ( $numero > $numero_para_adivinar ) /*Si existe la variable
$numero es que el usuario le a dado un clic al boton submit del
formulario*/
{
   $texto = "Te has pasado. Elige un número más pequeño.";
}
elseif ( $numero < $numero_para_adivinar)
{
   $texto = "No has llegado. Elige un número más grande.";
}
else
{
   $texto = "Acertaste. Felicidades.";
}
?>
<HTML>
<HEAD>
<TITLE> Formulario y código PHP que lo procesa todo en un archivo
</TITLE>
</HEAD>
<BODY>
<?php print $texto ?>
<FORM ACTION="<?php print $PHP_SELF?>" method="POST">
Número: <input type="text" name="numero">
</FORM>
</BODY>
```

Código PHP y HTML en el mismo archivo

Al inicio del código se realiza una comprobación, si existe la variable $*numero* quiere decir que se está invocando a esta página como programa para procesar un formulario, en caso contrario se está visualizando una página normal. Como vemos, conseguimos separar por completo el código PHP del diseño Web de la página, por lo que podemos mejorarlo o incorporar un formulario hecho por un diseñador con unos cambios mínimos.

Otro punto de vista consisten que toda la página sea un programa PHP y, escribamos HTML mediante la función *echo* del lenguaje:

```
if ( $texto == "hola"){
      echo"<B>Esto son etiquetas HTML<\B> normales. <P>
                        Es decir HTML integrado   en el código PHP. ";
}
```

Utilizar campos ocultos para guardar el estado.

Hay veces en las que necesitamos conocer ciertos valores a lo largo de varios formularios que presentamos al usuario, o en sucesivas pasadas por el mismo formulario. Esto puede conseguirse utilizando los campos ocultos (hidden) de HTML.
Obviamente, no es una forma segura de pasar información de un formulario a otro, ya que puede ser modificada por el usuario, pero si deseamos realizar un intercambio seguro entre páginas Web, podemos hacer uso de las sesiones en PHP.

Para más información sobre el uso de sesiones en PHP podemos consultar la siguiente página:

http://www.php.net/manual/es/features.sessions.php

En el siguiente ejemplo modificamos el script anterior para "arrastrar" el número de intentos que lleva el usuario para adivinar el número:

```
<?php
$numero_para_adivinar = 25;
$texto="";
$intentos = (isset ($intentos)) ? $intentos++:0;
if (! isset ($numero))
//Esta es la condición. Si no existe $numero es que NO nos invocan como
//procesadores del formulario, sino como página normal
{
   $texto = "Intenta adivinar el número ";
}
elseif ( $numero > $numero_para_adivinar )
/*Si existe la variable $numero es que el usuario le a dado un clic al
boton submit del formulario*/
{
   $texto = "Te has pasado. Elige un número más pequeño.";
}
elseif ( $numero < $numero_para_adivinar)
{
   $texto = "No has llegado. Elige un número más grande.";
}
else
{
   $texto = "Acertaste. Muy bien.";
}
?>
<HTML>
<HEAD>
<TITLE> Formulario y código PHP que lo procesa todo en un archivo
</TITLE>
</HEAD>
<BODY>
<?php print $texto ?>
<P> Intentos: <?php print $intentos?></P>
<FORM ACTION="<?php print $PHP_SELF?>" method="POST">
Número: <input type="text" name="numero">
<input type="hidden" name="intentos" value="<?php print $intentos ?>">
</FORM>
</BODY>
</HTML>
```

Transferir archivos al servidor Web usando upload

Un servicio muy utilizado en formularios HTML es permitir al usuario el envío de archivos al servidor Web como parte de la petición. Por ejemplo, cuando utilizamos un lector de correo vía Web y solicitamos anexar un archivo al mensaje que estamos escribiendo, estamos en realidad utilizando un formulario con un campo denominado *file*.

Para crear un formulario HTML que sea capaz de utilizar este tipo de campo es necesario definirlo con el atributo ENCTYPE="multipart/form-data":

```
<HTML>
<head>
<title>Formulario que puede contener un campo FILE<\title>
</head>
<body>
<FORM ENCTYPE="multipart/form-data" ACTION="<? print $PHP_SELF ?>"
METHOD="post">
<input type="hidden" name="MAX_FILE_SIZE" value="30000">
<input type="file" name="archivo">
<br>
<input type="submit" value="UPLOAD">
</FORM>
</body>
</html>
```

El campo MAX_FILE_SIZE permite definir el tamaño máximo admitido del archivo a transferir al servidor. El campo *file* creará un botón en el formulario que permitirá al usuario navegar por sus discos locales para seleccionar el archivo que desea transferir.

En este ejemplo el mismo script recibe la petición, al haber utilizado $PHP_SELF en el campo ACTION. Pero no se ha incluido todavía el código que lo trate. Cuando un archivo es transferido al servidor es almacenado en el directorio /tmp. Su ruta completa es accesible en una variable con el mismo nombre dado al campo *file*, en este caso $*archivo*. Además PHP almacena información sobre el archivo transferido en unas cuantas variables globales que pueden ser leídas desde nuestro código. Los nombres de estas variables se derivan del que hallamos utilizado en el campo file, en nuestro ejemplo $*archivo*.

Sabiendo que existen estas variables ya es posible escribir código PHP que reciba correctamente el archivo y haga algo con él. El siguiente ejemplo comprueba si es un archivo GIF o JPG y en caso de que sea así lo visualiza:

Como vemos en el código, utilizamos la variable que almacena el tipo de archivo transferido para averiguar si es una imagen. En caso afirmativo copiamos el archivo (el tratamiento de archivos se verá en profundidad en posteriores artículos) a un directorio que está por debajo del directorio raíz de nuestro servidor Web, para poder posteriormente enviar al navegador la línea:

```
<img src=\"$url/$nombre_archivo \"><P>
```

Que contendrá la URL de la imagen que queremos que sea visualizada. No es necesario que nos preocupemos de borrar el archivo temporal creado en /tmp, ya que el propio intérprete de PHP lo hace al finalizar la ejecución del script.

Conclusión

PHP es un excelente lenguaje para el desarrollo de páginas Web dinámicas, con miles de seguidores en todo el mundo. ¿Por qué?, las razones son muy sencillas:

1. Es libre.
2. Multiplataforma.
3. Existe una extensa documentación en Internet y códigos disponibles.
4. Tiene una gama muy amplia de funciones para trabajar con las principales bases de datos en el mercado, para el desarrollo de Web Services, manejo de documentos XML, implementación de seguridad y funciones de red, creación de documentos en PDF, etcétera.
5. Es sencillo de utilizar y fácil de aprender.
6. La compatibilidad con MySQL es extraordinaria, ya que ambos programas nacieron casi juntos.
7. Es extensible ya que se pueden integrar nuevos módulos de funcionalidad con facilidad.
8. Un lenguaje en continuo crecimiento.
9. Y más…

El objetivo de éste capítulo era introducirnos al lenguaje de programación PHP, aprender a integrarlo a nuestro servidor Apache y conocer las características básicas del lenguaje.

Bases de datos bajo Linux y su interacción con la Web

Capitulo 5

Capitulo 5: Bases de datos en Linux y su interacción con la Web.

Bases de datos en Linux

Linux es una excelente plataforma para que nosotros podamos instalar y administrar una base de datos que almacene grandes cantidades de información. Este sistema operativo, soporta casi todas las bases de datos que se encuentran actualmente en el mercado, ya sean comerciales o libres. Pero cuando nosotros hablamos de sistemas administradores de de Bases de Datos (RDBMS) libres, nos encontramos que en Linux existen dos muy populares: *MySQL y PostgreSQL.*

Ambos DBMS tienen una gran comunidad en Internet conformada por miles de personas en todo el mundo. Gracias a esto, podemos encontrar una gran cantidad de ejemplos y documentación en general.

Las características de estos RDBMS varían al respecto, haciendo un manejador mucho mejor que el otro en determinadas situaciones. Por ejemplo, MySQL soporta consultas que vinculen a varias bases de datos, pero esto no es posible de hacer si utilizamos PostgreSQL. Por otro lado, PostgreSQL agrega características como *object-oriented, herencia y arrays y reglas declarativas* que MySQL no tiene. Por esta situación nosotros debemos seleccionar cuidadosamente el DBMS que vamos a utilizar para desarrollar nuestra aplicación, considerando el tamaño de la base de datos, el número de usuarios, tiempo de respuesta, entre otras cosas.

Para realizar nuestra Revista Digital nosotros vamos a utilizar MySQL. Creo que es ideal para la aplicación que vamos a desarrollar por las siguientes razones:

- Es software libre.
- Es abierto.
- Es sencillo de utilizar.
- PHP trabaja excelente con MySQL debido a algunas razones como son: disponibilidad de licencias estido (software libre), accesibilidad al código fuente, funcionamiento en las principales plataformas, rendimiento espectacular bajo determinadas condiciones, etc.
- Tiene un manejo de conexiones muy rápido, especialmente ideal para ser utilizado con CGIs o similares.
- A partir de la versión 4.1 soporta subquerys.
- Excelente estabilidad aún con cientos de conexiones simultaneas.
- Las desconexiones son aleatorias y los archivos COR DUMP es muy raro que aparezcan.

- Es más extendido que PostgreSQL en su uso lo cual significa que sea más probado en ambientes de producción.

Así que el siguiente paso será instalar MySQL en nuestro servidor.

Instalación de MySQL en Linux

Realmente hoy en día la instalación de MySQL es muy sencilla, simplemente basta con acceder a la siguiente URL http://dev.mysql.com/downloads/, descargar la última versión de MySQL para Linux y desempaquetar el paquete en la siguiente ruta: */usr/local* (ubicación recomendada).

Debido a que las distribuciones de MySQL ya vienen precompiladas, es mucho más sencilla su configuración. Podemos leer el archivo "INSTALL-BINARY" que vienen junto con la aplicación y seguir los sencillos pasos, tal y como se muestra a continuación:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> cd /usr/local
shell> gunzip < /PATH/TO/MYSQL-VERSION-OS.tar.gz | tar xvf -
shell> ln -s FULL-PATH-TO-MYSQL-VERSION-OS mysql
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root  .
shell> chown -R mysql data
shell> chgrp -R mysql .
shell> bin/mysqld_safe --user=mysql &
```

Instalación básica de MySQL

Una vez que nuestro servidor de bases de datos esté funcionando, será necesario cambiar la contraseña de "root" para poder conectarnos a MySQL y poder crear nuevos usuarios, otorgar permisos, crear bases de datos, etc... Para hacer esto, tecleamos en nuestra Terminal lo siguiente:

```
./bin/mysqladmin -u root password 'new-password'
./bin/mysqladmin -u root -h <nuestro_host> password 'new-password'
```

Ahora nos podemos conectar a MySQL de la siguiente manera:

```
./bin/mysql –u root –p
password: *******
```

```
mysql>
```

Y ahora ya podemos ejecutar instrucciones de SQL para comunicarnos con la base de datos y manipular los datos.

¿Qué es SQL?

SQL equivale a lenguaje de consulta estructurado. Se trata de un sistema para acceder a los sistemas de administración de bases de datos (RDBMS). SQL se utiliza para almacenar y consultar datos desde y hasta una base de datos. Se utiliza en sistemas de bases de datos como MySQL, PostgreSQL, Oracle, Sybase, entre otros.

Existe un estándar ANSI de SQL, y los sistemas de bases de datos como MySQL suelen implementarlo. Sin embargo, existen diferencias sutiles entre el SQL estándar y el SQL de MySQL, que en algunos casos está previsto integrar en el estándar y en otros resultan deliberadas.

Podemos consultar el manual en línea de MySQL donde podemos encontrar una lista completa entre las diferencias de SQL con MySQL y el SQL ANSI de cada versión.

Es importante conocer un poco acerca del lenguaje SQL antes de empezar a trabajar en la creación de la base de datos para nuestra Revista Digital, ya que de ésta manera podremos realizar consultas más óptimas y sencillas.

Creación de la base de datos para nuestro proyecto final

Nuestra aplicación Web va a necesitas de una base de datos para almacenar toda la información referente a nuevos artículos, usuarios y vínculos. Esta base de datos las llamaremos igual que nuestra revista “enlinea” y sobre ella crearemos las siguientes tablas:

1. Artículos
2. Usuarios
3. Vínculos
4. Categorías
5. Relación usuarios y artículos

Los campos de cada tabla tienen que corresponder a la información que pedimos al usuario dentro de nuestros formularios, por ejemplo, en el formulario correspondiente a “Nuevo Usuario”, solicitamos al la persona que se desea registrar en nuestros sistema los siguientes datos: *nombre, apellido paterno, apellido materno, sexo, fecha de nacimiento, dirección, estado, código postal,*

ciudad, e-mail, nombre de usuario para el sistemas y un password. Así que nuestra tabla “usuarios” debe poder almacenar ésta información.

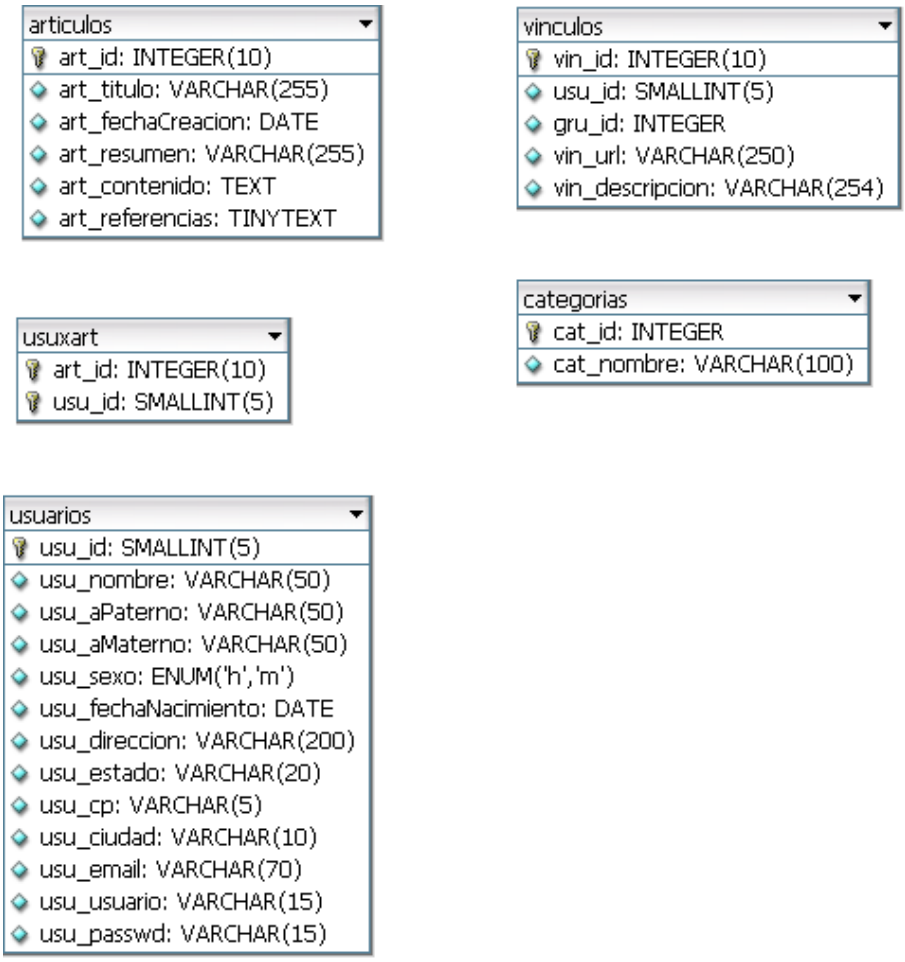


Figura 1: Diseño de la base de datos.

Para poder crear la base de datos una vez que hemos establecido una conexión con MySQL, debemos teclear lo siguiente:

```
mysql> create database enlinea;
mysql> use enlinea;
```

Con la primera línea creamos la base de datos y con la segunda le indicamos al manejador que vamos a empezar a trabajar con ella. Esto nos da la posibilidad de crear tablas, insertar datos, crear consultas, etc.…

Ahora es momento de crear las tablas con sus respectivos campos. Para eso vamos a teclear el siguiente código:

Base Datos “enLinea”

```
CREATE TABLE articulos (
  art_id INTEGER(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  art_titulo VARCHAR(255) NOT NULL,
  art_fechaCreacion DATE NOT NULL DEFAULT '0000-00-00',
  art_resumen VARCHAR(255) NULL,
  art_contenido TEXT NOT NULL,
  art_referencias TINYTEXT NULL,
  PRIMARY KEY(art_id)
);

CREATE TABLE categorias (
  cat_id INTEGER UNSIGNED NOT NULL,
  cat_nombre VARCHAR(100) NOT NULL,
  PRIMARY KEY(cat_id)
);

CREATE TABLE usuarios (
  usu_id SMALLINT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
  usu_nombre VARCHAR(50) NOT NULL,
  usu_aPaterno VARCHAR(50) NOT NULL,
  usu_aMaterno VARCHAR(50) NOT NULL,
  usu_sexo ENUM('h','m') NOT NULL DEFAULT 'h',
  usu_fechaNacimiento DATE NOT NULL DEFAULT '0000-00-00',
  usu_direccion VARCHAR(200) NOT NULL,
  usu_estado VARCHAR(20) NOT NULL,
  usu_cp VARCHAR(5) NOT NULL,
  usu_ciudad VARCHAR(10) NOT NULL,
  usu_email VARCHAR(70) NOT NULL,
  usu_usuario VARCHAR(15) NOT NULL,
  usu_passwd VARCHAR(15) NOT NULL,
  PRIMARY KEY(usu_id)
);

CREATE TABLE usuxart (
  art_id INTEGER(10) UNSIGNED NOT NULL,
  usu_id SMALLINT(5) UNSIGNED NOT NULL,
  PRIMARY KEY(art_id, usu_id)
);

CREATE TABLE vinculos (
  vin_id INTEGER(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  usu_id SMALLINT(5) UNSIGNED NOT NULL,
  gru_id INTEGER UNSIGNED NOT NULL,
  vin_url VARCHAR(250) NOT NULL,
  vin_descripcion VARCHAR(254) NOT NULL,
  PRIMARY KEY(vin_id)
);
```

Después de lo anterior, la estructura de nuestra base de datos ha quedo lista para empezar a insertar datos, realizar consultas, eliminar datos y más.

Consultas a la base de datos

Para hablarle a la base de datos, debemos utilizar uno o más palabras reservadas, así como la información que nosotros queremos manipular, en una instrucción que se envía a la base de datos. En este momento tenemos una base de datos con una tabla llamada “Usuarios”. Esta tabla esta conformada por algunas columnas como son: *usu_nombre, usu_aPaterno, usu_aMaterno, usu_fechaNacimiento*, etc.
Si nosotros quisiéramos saber el nombre completo de los usuarios que cumplen años en el mes de enero, nosotros tendríamos que mandar una instrucción como la que sigue a la base de datos:

```
SELECT usu_nombre, usu_aPaterno, usu_aMaterno FROM usuarios WHERE
MONTH(usu_fechaNacimiento)=1;
```

Los parámetros que siguen al verbo son una lista de parámetros que nosotros queremos mostrar. Estos parámetros son los nombres de las columnas que contienen la información que queremos ver.
Enseguida de FROM está el nombre de la tabla en donde se encuentran localizados los datos que queremos. Y finalmente la cláusula WHERE, la cual limita a través de una condición la información que queremos desplegar.

La cláusula INSERT

Esta cláusula nos permite insertar datos en una tabla. Veamos un ejemplo:

Insertamos una nueva categoría llamada “Computación”.

```
INSERT INTO categorías (cat_id, cat_nombre) VALUES
(“”,”Computación”);
```

La cláusula UPDATE

Esta cláusula nos permite actualizar un registro dentro de una tabla. Veamos un ejemplo:

Cambiamos el nombre de usuario que tenga el identificado 1 por Rodrigo.

```
UPDATE usuarios SET usu_nombre = Rodrigo WHERE usu_id=1;
```

La cláusula DELETE

Esta cláusula nos permite eliminar uno o más registros dentro de una tabla. Veamos un ejemplo:

Eliminamos todos los artículos que tengan un título en blanco.

```
DELETE FROM articulos WHERE art_titulo = “”;
```

Podemos consultar un libro de SQL para poder profundizar más en cada uno de estas cláusulas.

PHP y MYSQL

Para ejecutar una consulta SQL con PHP deben de seguirse unos pasos parecidos a los que se usan para acceder a un archivo:

- Debe abrirse la base de datos
- Se envía la línea de comando SQL a la base de datos.
- Se recibe la respuesta de la base de datos.
- Se cierra la conexión con la base de datos.

Antes de que pueda iniciarse una consulta SQL, deberá crear una conexión con el servidor de bases de datos. Para ello, debemos conocer la siguiente información:

- Nombre del servidor de bases de datos (regularmente llamado “host”).
- Identificador de usuario (username).
- Clave de acceso (password).
- Nombre de la base de datos con la que nos vamos a comunicar.
- Opcionalmente el número de puerto.

Si se quiere acceder a una base de datos con PHP, debemos utilizar el siguiente comando:

```
<?
$conexion=mysq_connect(“host:port”, “username”, “password”);
?>
```

Si la conexión se ha realizado con éxito, PHP almacena en la variable $conexion lo que se llama un “link identifier”, que deberá transmitirse para realizar las operaciones con la base de datos.
Para que MySQL sepa con que base de datos vamos a trabajar, necesitamos elegir una. Todos los comandos que sigan a partir de éste momento se referirán a esa base de datos hasta que se elija otra.

```
<?
mysql_select_db(“base_de_datos”, $conexion);
?>
```

He resumido las funciones MySQL necesarias para trabajar con PHP en el desarrollo de la Revista Digital.

Función	Ejemplo	Descripción
mysql_affect_rows	$cantidad=mysql_affect_rows($vid)	Indica la cantidad de secuencias afectadas por la última operación MySQL.
mysql_close	Mysql_close($vid)	Cierra la conexión con el servidor de bases de datos.
mysql_connect	Mysql_connect($host,$user, $password)	Realiza una conexión con el servidor de MySQL.
mysql_db_query	$resultado=mysql_db_query ($db,$sql,$vid)	Envía una consulta SQL $sql a la base de datos.
mysql_error	$errmsg=mysql_error($vid)	Muestra el texto de error $errmsg de la operación ejecutada con anterioridad.
mysql_fetch_array	$array=mysql_fetch_array($ resultado,$tipo)	Almacena el resultado de un SELECT en un $array, que contienen los valores de la línea. $type indica el tipo de matriz. MYSQL_ASSOC: asociativo. MYSQL_NUM: numérico MYSQL_BOTH: ambos
mysql_field_name	$nombre=mysql_field_name ($resultado, $index)	Indica el nombre de un campo con el índice $index en el resultado de una consulta.
mysql_num_fields	$cantidad=mysql_num_fields($resultado)	Indica la cantidad $cantidad de campos en el resultado de una consulta.
mysql_num_rows	$cantidad=mysql_num_rows($resultado)	Indica la cantidad $cantidad de secuencias en el resultado de una consulta.
mysql_pconnect	Mysql_pconnect($host,$user,$password)	Realiza una concexión persistente en el servidor de MYSQL.

mysql_query	$succ=mysql_query($sql,$vid)	Envía, a través de una conexión abierta con anterioridad, una consulta SQL $sql al servidor de bases de datos.
mysql_select_db	$succ=mysql_select_db($db,$vid)	Elige una base de datos $db

Conclusiones

Como lo he mencionado anteriormente, MySQL se integra perfectamente con Apache Web Server y PHP, aumentado su facilidad de uso y la comunicación entre componentes. Se caracteriza básicamente por la velocidad y la poca cantidad de recursos que consume.

En la actualidad se tienen registrados más de 6 millones de copias de MySQL funcionando[1] y cada vez son más empresas que adoptan este manejador de bases de datos para sus aplicaciones. Por dar algunos ejemplos, estas son algunas de las empresas que utilizan MySQL para sus aplicaciones críticas:

1. Yahoo
2. Google
3. Digg
4. Nokia
5. Wikipedia
6. NASA
7. CNET Network

Por otro lado, posee un gran número de características que lo hacen una excelente elección para el manejo de nuestra información.

Para terminar, éste capitulo nos dio una pequeña probadita de lo que es MySQL. Vimos la forma de instalarlo, algunas funciones básicas para utilizarlo y una lista de funciones de PHP, para su interacción con la Web.

Este es un claro ejemplo, de la gran calidad que tienen muchos productos libres y de su aceptación en el mundo de la computación.

[1] Para más información visitar http://es.wikipedia.org/wiki/MySQL

Introducción a la seguridad en cómputo

Capitulo 6

Capitulo 6: Introducción a la seguridad en cómputo

Seguridad computacional.

La seguridad computacional es un punto que nunca debemos de olvidar a la hora de realizar aplicaciones sobre sistemas informáticos, ya que es la única medida que nos permiten garantizar la confidencialidad, integridad y disponibilidad de los recursos de nuestros sistemas (hardware y software).

En un principio, cuando el hombre requirió diseñar un medio para poder comunicarse con otras computadoras y compartir información, la seguridad no era un factor importante en el diseño de las redes, y esto se debió a que nunca se imaginaron la rapidez y las dimensiones que éstas llegarían a tener, en un plazo de tiempo tan corto.

Ante este crecimiento acelerado de la redes, la seguridad de la información fue un blanco fácil para muchos individuos deshonestos, que aplicaron sus habilidades y conocimientos para introducirse a los sistemas de información con malas intenciones. Fue éste el motivo que dio la pauta para la creación de los principios básicos de la seguridad computacional en las redes, que fue posterior a los principios de la seguridad informática.

Quiero recalcar que no sólo la seguridad computacional se ve comprometida a través de las redes, existen otros factores como son la deficiencia en los equipos de soporte, la ingeniería social, las deficiencias de la administración de una red, los virus, fallas de seguridad en los programas, los famosos Hackers, Crakers, etc; que son un factor que ponen en riesgo el activo más importante de una empresa: *la información*.

Realmente hablar de seguridad informática es un mundo de información, por lo tanto, en éste capítulo vamos a abordar algunos conceptos básicos de seguridad, así como algunas herramientas que nos ayudarán a proteger nuestra información y prevenir diversos ataques a nuestro servidor.

Conceptos básicos de seguridad

Seguridad computacional

El conjunto de políticas y mecanismos que nos permiten garantizar la confidencialidad, la integridad y la disponibilidad de los recursos de un sistema.
En la actualidad, el activo más importante en una organización es la información.

De la cual se derivan los siguientes conceptos.

- **Confidencialidad**: Un sistema posee la propiedad de confidencialidad si, la información manipulada por éste no es disponible ni puesta en descubierto para usuarios, entidades o procesos no autorizados.
- **Disponibilidad**: Un sistema posee la propiedad de integridad si los datos manipulados por éste no son alterados o destruidos por usuarios, entidades o procesos no autorizados.
- **Integridad**: Un sistema posee la propiedad de disponibilidad si, la información es accesible (está disponible) en el momento en que así lo deseen los usuarios, entidades o procesos autorizados.

Activo

Es cada uno de los elementos que cuenta con un valor tangible o intangible para la empresa. Este valor se basa en el papel que desempeña el *activo* en las operaciones diarias. El ataque a algunos de los elementos tangibles o intangibles puede provocar un daño a la empresa o tener un efecto en cascada sobre otros activos. Tal como sucede con la información, un activo puede tener un valor material pequeño (por ejemplo, nuestro servidor Web) pero cuando han sido comprometidos conjuntamente con otros activos, el valor de esta combinación puede ser mucho más elevado. La misión global que los activos tienen encomendada también se debe tener en cuenta a ka hora de calcular el valor del activo.

Amenaza

Tanto los actos como los actores que pueden infligir algún daño a los activos, aprovechándose de las vulnerabilidades, reciben el nombre de amenazas. Las circunstancias y características que rodean al activo objetivo son el factor decisivo en el éxito o el fracaso de una amenaza. Las amenazas afectan a la confidencialidad, integridad y disponibilidad del activo a través de una o más de sus vulnerabilidades.

Agente/Actor de la amenaza y acto de amenaza.

El *actor o agente*, es una persona, grupo, organización o acto que está ejecutando la acción o *acto de amenaza*. Todo esto recibe el nombre genérico de “amenaza”. La recomendación es identificar los posibles o conocidos agentes de amenazas que nos permita analizar los motivos, apoyos y posibilidades, lo que a su vez determina el *nivel de la amenaza*.

Vulnerabilidad

Básicamente una vulnerabilidad es una debilidad que puede ser una característica o defecto inherente asociada con un activo o su entorno que pueden permitir el compromiso del activo o la ejecución de un daño sobre el

mismo. Las vulnerabilidades son activadas o explotadas por amenazas intencionadas o accidentales.

Una vulnerabilidad puede existir debido a la existencia de una o más de las siguientes condiciones:

- Utilización de controles inapropiados.
- Defectos estructurales.
- Administración o procedimientos operativos deficientes.
- Actualizaciones de parches de software perdidos o no empleados.

Impacto

El impacto es una representación del grado conocido o percibido del daño asociado con los activos de la empresa una vez que la amenaza ha tenido consecuencias sobre dichos activos después de valerse de las vulnerabilidades existentes.

Riesgo

El riesgo es una representación del grado de impacto potencial o percibido asociado a los activos de la empresa. Es un grado de “estado” en términos de daño o peligro, una medida de la probabilidad de daño y del grado de impacto que una determinada amenaza tiene sobre la confidencialidad, integridad y disponibilidad de un activo o recurso.

Mecanismos de seguridad

Autenticación

La autentificación se refiere a demostrar la identidad de las entidades involucradas en una transacción y así, evitar que alguien tome la identidad de otro. Generalmente toma dos formas:

- Autentificación del proveedor de bienes o servicios.
- Autentificación del cliente.

Control de acceso

Permite definir quién puede tener acceso a ciertos recursos, dependiendo de los privilegios o atributos que posea. Esto nos permite proteger los recursos del sistema contra el uso no autorizado.

El control de acceso se basa en credenciales o atributos y se aplica a los usuarios y procesos que ya han sido autentificados. ya han sido autentificados.

Confidencialidad

Este mecanismo se encarga de garantizar la privacidad de los datos y se apoya en la criptografía para conseguirlo

Integridad de datos

Los mecanismos que intervienen en la integridad de los datos, permiten proteger los archivos contra ataques activos, por ejemplo, el usuario A desarrolló una aplicación y antes de enviarle ésta, al usuario B, decide obtener su huella digital. Esta huella digital se la enviará antes que nada al usuario B, para que cuando reciba la aplicación desarrollada por el usuario A, pueda corroborar que no ha sufrido ninguna modificación en el camino. Con esto comprobamos la integridad de los datos y los mecanismos en los que nos podemos apoyar para conseguir esto son:

- CRCs
- Huellas digitales.

No repudiación

Garantiza que los actores involucrados en una transacción de información, por cualquier medio que ésta utilice, sean realmente quienes dicen ser y comprueben la identidad del mismo.

Los mecanismos principales para conseguir esto son:

- Los certificados.
- La notarización.
- Las firmas digitales.

Criptografía

La Criptografía (del griego *kryptos*, “oculto” y *graphein*, “escritura”, “escritura oculta”) se encarga de convertir un texto normal y comprensible en un formato incomprensible a menos que se posea un conocimiento secreto.

En épocas recientes la *criptología* se define como la ciencia de usar las matemáticas para cifrar y descifrar información.

El *criptoanalisis* es la ciencia de analizar y romper la comunicación segura mediante el análisis del algoritmo empleado.

La *Criptología* involucra el *criptoanalisis,* la *criptografía* y *esteganografía*.

La *esteganografía* es una rama de la *criptología* que trata la ocultación de mensajes, para evitar que se perciba la existencia del mismo.

Seguridad en Linux

Aunque Linux es un sistema muy robusto e incorpora las características de seguridad comunes en todos los sistemas tipo UNIX, es importante dedicarle tiempo y recursos para conocer cuales son sus debilidades y vías frecuentes de ataque, para que de ésta manera podamos adoptar posteriormente medidas más eficaces para contrarrestarlas.

El primer punto que tenemos que tener en cuenta es que NO existe un sistema 100% seguro, ya que como administradores únicamente podemos aumentar la dificultad para que una persona pueda comprometer nuestro sistema. Tomemos en cuenta que entre mayor sea la importancia del activo que fluya en nuestro sistema, mayor será la atención que nosotros debemos de prestarle, para trabajar en su seguridad.

¿Cómo podemos verificar la integridad de un archivo en nuestro sistema Linux?

En linux podemos encontrar diferentes programas que nos ayudarán verificar la integridad de un archivo y con esto, podemos cerciorarnos de que el archivo no ha sido modificado y sobre todo, que proviene de donde dice ser. El programa más popular para realizar esta comprobación es *md5sum*.
Éste pequeño programa nos permite firmar un archivo (puede ser un programa, un documentos, una imagen de un disco, etc.) y obtener una clave hexadecimal de 32 caracteres, UNICA. De tal manera que si en el archivo se modifica un solo bit, esta clave será completamente diferente a la origina.

Vamos a ver un ejemplo:

Shell> **md5sum** documentoImportante.pdf
ce8e5113f6627f176ff79e774e2fe0c0 documentoImportante.pdf

Este proceso se llama “Message Digest” y es muy común en Internet descargar programas que vienen con su firma MD5 anexa, para que nosotros podamos verificar que la aplicación viene integra.

Utilizando GPG

GPG o GNU Privace Guard es la versión libre del popular administrador de firmas digitales PGP (Pretty Good Privacy) creado por Phil R. Zimmermann.

Actualmente GPG es un proyecto oficial de GNU que viene integrado en la mayoría de las distribuciones de Linux. La principal función de éste software es garantizar la privacidad, autenticidad e integridad de nuestras comunicaciones electrónicas.

GPG utiliza criptografía de clave pública para que los usuarios puedan comunicarse de un modo seguro. En un sistema de claves públicas cada usuario posee un par de llaves, una *clave privada* y *una pública*. Cada usuario debe mantener su clave privada en secreto y jamás debe revelarla a nadie. Por otro lado, la llave pública es precisamente eso, "pública" y por lo tanto, podemos distribuirla libremente a cualquier persona que desee comunicarse con nosotros.

Un ejemplo de su funcionamiento es el siguiente: Una persona desea enviarnos un mensaje secreto. Esta persona conoce nuestra llave pública y la utilizará para cifrar el mensaje que desea enviarnos. El truco es que sólo nuestra llave primaria puede descifrar un mensaje cifrado con nuestra llave pública. ¿Sencillo, no?

Como mencioné anteriormente, GPG acompaña a la mayoría de las distribuciones Linux, pero nosotros podemos descargarlo desde su página oficial:

http://www.gnupg.org/(en)/download/index.html.

Además, podemos encontrarlo para distintas plataformas como son Windows y MAC OS. En todas estas plataformas, cuenta básicamente con una interfaz en modo texto que nos permite interactuar con éste excelente programa.

Vamos a ver algunos ejemplos de cómo utilizar GPG.

1. Para crear nuestro juego de llaves pública y privada, tecleamos el siguiente comando:

   ```
   shell> gpg-gen-keys
   ```

2. Con el siguiente comando podemos ver las llaves públicas de nuestro llavero:

   ```
   shell> gpg --list-keys
   ```

3. Para exportar mi llave pública a un archivo.

   ```
   shell> gpg --export –armor > MiLlavePublica.key
   ```

4. Para importar a nuestro llavero la llave pública de otra persona.

```
shell> gpg --import < LlavePublicaAmigo.key
```

5. Con éste comando firmamos la llave pública de otra persona con el objetivo de marcarla como *confiable*.

   ```
   shell> gpg –sign-key <id>
   ```

6. Con el siguiente comando ciframos un "*Mensaje.txt*" con la clave pública de un usuario, siempre y cuando ésta exista en nuestro llavero.

   ```
   shell> gpg –e Mensaje.txt
   ```

7. Con el siguiente comando, podemos descifrar un mensaje utilizando nuestra llave privada.

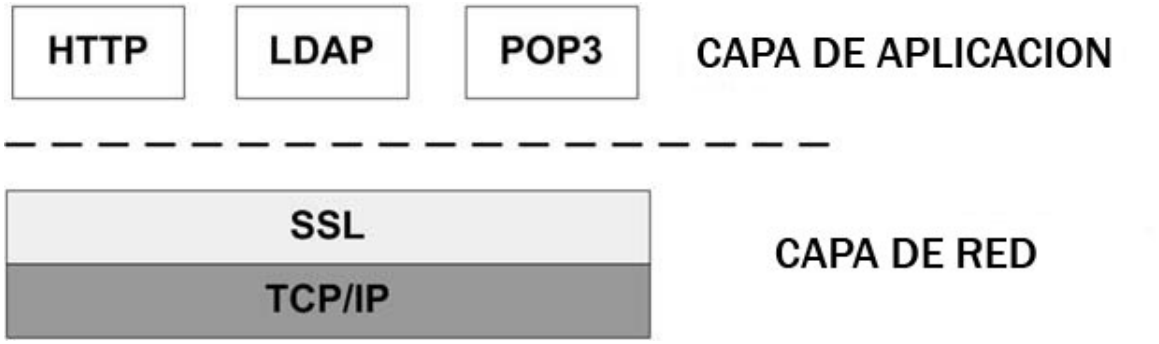
   ```
   shell> gpg  -d Mensaje.txt
   ```

Open Secure Socket Layer

El proyecto OpenSSL es un desarrollo de código abierto con el propósito de desarrollar un grupo de herramientas de SSL y TTL con el nivel de una herramienta profesional. Fue desarrollado por Eric. A. Young y Tim. J. Hudson basado en el código fuente de SSLeay.

Mejor conocido como SSL, fue originalmente desarrollado por Netscape, para proteger la información que es transportada y enviada a las capas de aplicación HTTP, POP3 ó LDAP. SSL fue diseñado para hacer uso de la capa de comunicación TCP, ya que éste le proporciona confiabilidad y una conexión autenticada entre dos puntos a través de la red (por ejemplo, entre el cliente y el servidor).

Regularmente, SSL es utilizado para la protección de datos de un servicio de red, por ejemplo, se utiliza regularmente para trabajar en conjunto con un servidor HTTP y con las aplicaciones clientes. Hoy en día, la mayoría de los servidores Web pueden soportar sesiones SSL, mientras que en los buscadores como Internet Explorer o Netscape Navigator se les ha habilitado el uso de SSL.



SSL trabaja en un esquema de llave pública para el intercambio de llaves de sesión. Las llaves de sesión son utilizadas para cifrar transacciones sobre HTTP y esto dificulta al “atacante” comprometer toda una sesión.

¿Cuáles son los objetivos de SSL?

Los objetivos de SSL son los siguientes:

1. Autenticar al cliente y al servidor uno con otro.
2. Asegura la integridad de los datos.
3. Seguridad en la privacidad de la información.

Apache con soporte para SSL

Yo considero que siempre que vayamos a realizar un sitio Web, en donde la información que manejemos deba ser confidencial, tenemos que apoyarnos de alguna herramienta que establezca y mantenga una conexión segura entre las dos partes (cliente y servidor).

Pensando en nuestra Revista Digital, nosotros podemos utilizar esta liason para poder proteger los datos del usuario al momento de registrarse en nuestro sistema o al momento de realizar algún cambio, como por ejemplo, modificar su contraseña de acceso.

La instalación de Apache con SSL es muy sencilla, simplemente debemos agregar un parámetro para habilitar esta función:

```
shell> gzip -dc httpd-2.0.52.tar.gz  | tar xf -
shell> cd httpd-2.0.52
shell>./configure --prefix=/usr/remote/apache --enable-so --enable-ssl
--with-ssl=/usr/remote/openssl
shell> make
shell> make install

Para finalizar:

shell> apachectl start
shell> apachectl stop
shell> /usr/remote/apache/bin/httpd -l | grep mod_ssl
```

Instalación de Apache con SSL

Algunos conceptos importantes antes de ver el proceso de HTTPs.

Autoridades certificadoras.

El esquema de firmas digitales requiere que alguien autentifique que un individuo es quien dice ser, instituciones gubernamentales o financieras se encargan de emitir certificados digitales en los cuales se integra la llave pública del individuo, piezas de información sobre el individuo o identificadores de la autoridad certificadora quien finalmente se encarga de firmar digitalmente con su llave privada de la CA.

CA

La norma X.509 es el estándar para formatos de certificados con llave pública. Un certificado X.509 consiste de la llave pública de un usuario y la firma de un tercero para la identificación en el bloque de identificación de ese usuario.

¿Cómo generamos un certificado?

Para generar el certificado necesario para realizar la comunicación vía HTTPS se debe realizar:

- Se genera la llave privada

```
shell> openssl genrsa -out server.pem 2048
```

- Se genera un “Certificate signing request” el cúal se enviará a la CA.

```
shell> openssl req -new -key server.key -out cert.csr
```

- O si se utiliza un Certificado auto firmado, se realiza:

```
shell> openssl req -new -x509 -key server.key -out cacert.pem -days
30
```

¿Cuál es el proceso de HTTPs?

Este proceso de comunicación se puede resumir en nueve pasos:

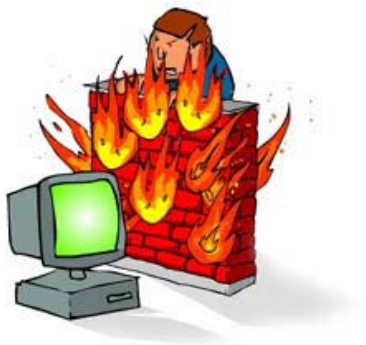
1. El cliente establece una conexión a un servidor que soporta HTTPs.
2. Se negocian los parámetros de comunicación.
    a. El cliente notifica al servidor cuales son los parámetros que soporta y el servidor selecciona cual de ellos utilizará. Entre los parámetros tenemos qué versión de cifrado se utilizará, qué protocolo, etc.
3. El servidor envía al cliente su certificado digital (x509).
4. El cliente utiliza su copia de la llave pública CA, la fecha y el nombre del servidor para validar el certificado.
5. Se realiza un acuerdo de llave. El cliente genera un “Clave pre-maestra“, lo cifra con la llave pública del servidor y se lo envía.
6. El servidor debe descifrar con su llave privada la “Clave pre-maestra” (Autenticación).
7. Tanto el cliente como el servidor deben de generar la “clave maestra” a partir de la “pre-maestra”.
8. El cliente y el servidor intercambian cifrado el Message Digest de la “clave maestra”, para asegurar que obtuvieron lo mismo (integridad y confidencialidad).
9. A partir de la “clave maestra” ambos generan la clave de sesión con la cual se cifrará la comunicación.

Firewall

Un *firewall* o cortafuegos es un dispositivo físico o software implementado en un sistema operativo que nos ayuda a filtrar el trafico de una red (cómo mínimo dos) basándose en unas reglas de filtrado que nosotros determinamos de acuerdo a nuestras necesidades. Un *firewall* actúa entre la capa de transporte y la capa de red, permitiéndonos controlar el destino y el origen de una conexión, el protocolo y el código de control.

Las ventajas de utilizar un *firewall* son las siguientes:

- Nos ayuda a protegernos de las intrusiones.
- Evita que nuestra red se infeste de virus.
- Optimiza los accesos a nuestro sistema.



- Protege nuestra información.
- Nos ayudan a monitorear el tráfico de nuestra red.

Utilizando IPTABLES

Iptables es una aplicación de línea de comandos que apareció a partir de la versión del kernel 2.4.X de Linux y vino a sustituir a otra aplicación llamada *ipchains*. La forma de operar de las iptables es a través de la definición de *reglas de filtrado de paquetes.* Estas reglas son definidas por el administrador del sistema de acuerdo a sus necesidades y políticas de seguridad.

Características principales

Entre sus características principales se encuentran las siguientes:

- Viene dentro del kernel de Linux, así que es parte del sistema operativo.
- Filtrado de paquetes (IPv4 e IPv6)
  - Por IP, puerto, protocolo, etc.
  - Por el estado de los paquetes (connection traking).
- Network Address Translation (NAT)
- Infraestructura flexible y extensible.
- Tiene la capacidad de añadir nuevas funcionalidades mediante plugins o parches.

Un firewall lo podemos utilizar en nuestro hogar, en nuestra empresa y básicamente en cualquier red, ya que nos permiten controlar los accesos hacia dentro y también los internos hacia el exterior (esto último también podemos hacerlo con un servidor Proxy, solamente que las reglas utilizadas son de mucho más alto nivel).

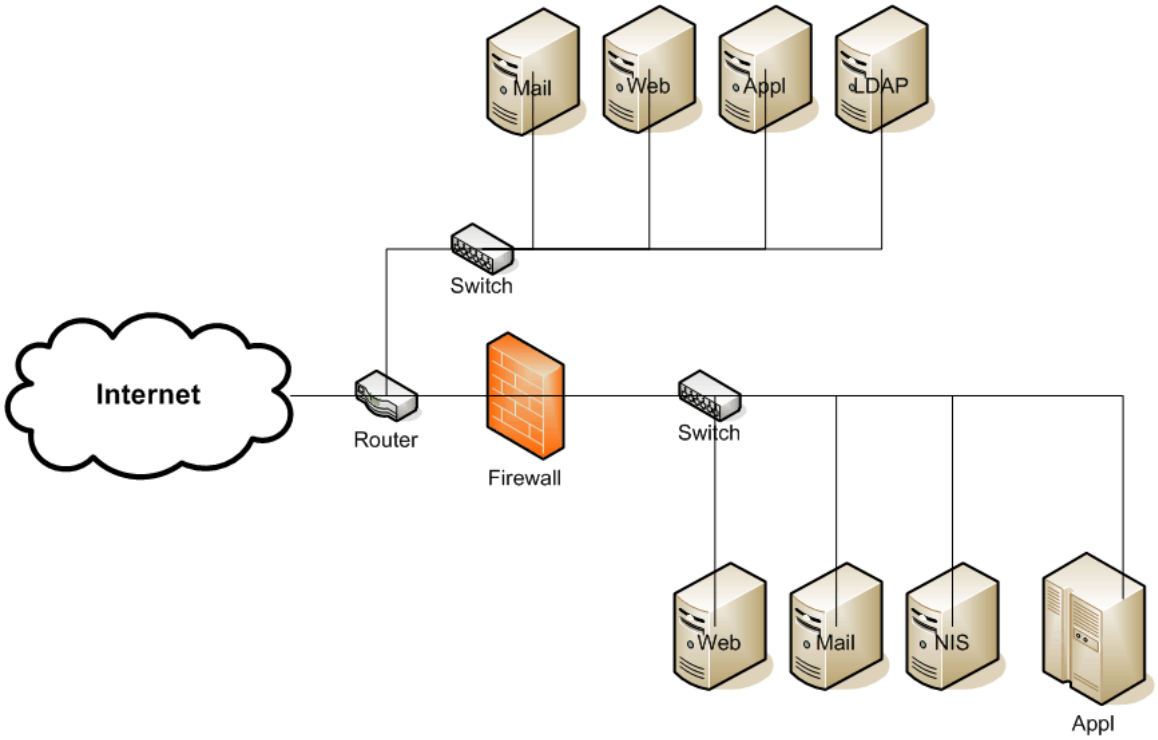


Figura #: Esquema de un firewall entre redes.

En la figura anterior podemos ver como el *firewall* protege solamente un sección de la red, dejando la otra sección expuesta ante la inseguridad de Internet.

Ahora vamos a ver cómo podemos crear algunas reglas de seguridad para implementarlas en las *iptables* de nuestro sistema. Voy a procurar que las reglas sean sencillas de entender para evitar entrar en muchos detalles, pero de cualquier forma, podemos obtener más información sobre el funcionamiento de las *iptables* en la página oficial del proyecto: http://www.netfilter.org/.

Creación de reglas de seguridad

Primero tenemos que hacernos la idea de que utilizar iptables es sencillo. Para poder crear una regla de seguridad, yo recomiendo que primero la escribamos con nuestras palabras, como si fuera un pseudocódigo, esto nos facilitará el trabajo y veremos como la traducción a una regla gramaticalmente correcta de iptables, es muy sencilla.

Quiero resaltar antes de continuar, que es muy importante *el orden* en que pongamos nuestras reglas de seguridad, ya que la aplicación irá ejecutándolas y almacenándolas en el orden en el que las pongamos.

Por ejemplo, nuestra computadora cuenta con un servidor Web, una base de datos en MySQL y además, nos conectamos a ella a través de SSH (Secure

Shell) y FTP. Las personas que se conectan al servidor somos nosotros y una persona que se encarga de darle mantenimiento a la base de datos desde su trabajo, nadie más.

Muy bien, con este antecedente podemos empezar a crear nuestras reglas de seguridad:

1. Localmente le permitimos a nuestro servidor cualquier conexión interna, por ejemplo, al servidor de MySQL.
2. Para nosotros que somos los administradores, debemos darnos todos los privilegios y libertades, por ejemplo, conectarnos a cualquier puerto.
3. Debemos otorgarle el acceso a la persona que le da mantenimiento a la base de datos, pero únicamente al puerto de MySQL (3306), para que pueda conectarse desde su trabajo.
4. Debemos habilitar únicamente los puertos de SSH y FTP.
5. Debemos habilitar el acceso al puerto 80 (WWW), ya que es un servidor Web.
6. Por último, cerramos todo los accesos.

Y cuál sería la traducción de nuestras reglas a *iptables*:

```
#Primero eliminamos cualquier regla que haya sido almacenada con
anterioridad.
Shell>iptables -F
Shell>iptables -X
Shell>iptables -Z
Shell>iptables -t nat -F

# Establecemos políticas por defecto
Shell>iptables -P INPUT ACCEPT
Shell>iptables -P OUTPUT ACCEPT
Shell>iptables -P FORWARD ACCEPT
Shell>iptables -t nat -P PREROUTING ACCEPT
Shell>iptables -t nat -P POSTROUTING ACCEPT

# Empezamos a filtrar
# Todos los privilegios para LOCALHOST
Shell>iptables -A INPUT -i lo -j ACCEPT
# A nuestra IP le damos acceso a todo.
Shell>iptables -A INPUT -s 195.65.34.234 -j ACCEPT

#A la persona encargada le damos acceso desde la IP de su trabajo al
puerto de MYSQL
Shell>iptables -A INPUT -s 231.45.134.23 -p tcp --dport 3306 -j ACCEPT

# habilitamos los puerto TCP de SSH y FTP
Shell>iptables -A INPUT  -p tcp -dport 21 -j ACCEPT
Shell>iptables -A INPUT  -p tcp -dport 22 -j ACCEPT

# Habilitamos el puerto 80 para la WWW
Shell>iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

```
# Y bloqueamos todo lo demás
Shell>iptables -A INPUT -p tcp --dport 20 -j DROP
Shell>iptables -A INPUT -p tcp --dport 21 -j DROP
Shell>iptables -A INPUT -p tcp --dport 3306 -j DROP
Shell>iptables -A INPUT -p tcp --dport 22 -j DROP
Shell>iptables -A INPUT -p tcp --dport 10000 -j DROP
```

Al parecer se complica un poco cuando uno traduce las reglas que hemos pensado al código de las *iptables*, pero en realidad es muy sencillo.

Como mejorar la seguridad en nuestra red

Creo que nunca son suficientes las medidas que podemos tomar para volver nuestra red más segura. Algunas recomendaciones para reforzar nuestra red son las siguientes:

- No utilizar servicios Telcel y FTP que sean inseguros. Podemos utilizar como métodos alternativos SSH o Secure Copy.
- No es recomendable la administración de un sistema basado en Internet.
- Actualizar con la mayor rapidez los parches y fixes emitidos por los fabricantes.
- Emplear tecnologías de seguridad crítica tales como cortafuegos, antivirus, sistemas de detección de intrusos, análisis avanzados de registros y filtros de entrada a la Web.
- Enmascare mediante el uso de un Proxy las conexiones HTTP salientes de usuario utilizando autenticación.
- Minimizar el número de cuentas de usuarios en el sistema, que sean críticos desde el punto de vista operativo.
- Controlar el tráfico de red saliente así como el entrante. Sólo debemos permitir la salida y la entrada de aquellos usuarios que sean estrictamente necesarios para nuestro sistema o red.
- Debemos analizar y separar los anexos peligrosos de los correos electrónicos en los *gateways* de la red.
- Controlar y regular las cuentas de carácter administrativo.
- Utilizar contraseñas muy fuentes para cuentas administrativas y cuantas fuertes para los usuarios normales.
- Efectuar valoración de riesgos para los servicios, sistemas y entornos que sean críticos, como mínimo.

Conclusiones

Creo que la seguridad siempre es un factor importantísimo en el mundo de los negocios y sobre todo en aquellos que se hacer por Internet, ya que existe muchísima gente con habilidades extraordinarias que son capaces de hacernos perder mucho dinero; razón suficiente para dedicarle un tiempo razonable y recursos a la seguridad en nuestras aplicaciones y sistemas.

Como pudimos ver en este capítulo, el tema de seguridad es gigantesco, aún si nos enfocamos a la seguridad en la Web, podríamos escribir varios libros para tocar los aspectos más importantes y temas de prevención.

Para el desarrollo de nuestra aplicación Web, es necesario que conozcamos en la medida de lo posible, las herramientas principales para poder prevenir ataques, pérdida de información, programas malignos, etcétera. Siempre siendo concientes de los riesgos a los que nos enfrentamos. Estar informados es nuestra principal arma en cuestiones de seguridad. También el mantener actualizado nuestro software con los últimos parches, reducirá el riesgo de tener un percance.

También se recomienda contar con planes de contingencia, respaldo de nuestras aplicaciones, implementación de políticas de seguridad, no utilizar software ilegal, entre otras recomendaciones, con el objetivo de tener una plataforma confiable e Integral.

Bibliografía

- KAY, Trevor: "Linux+ Certification Bible."
  Hungry Minds, Inc., Nueva York, 2002, p. 721, ISBN 0-7645-4881-6

- LUMENS Chris, CANTRELL David, JOHNSON Logan: "Slackware Linux Essentials.", Ed. 2da., Slackware Linux, Inc., Canada, 2005, p. 284, ISBN 1-57176-338-4

- MORITSUGU Steve: "UNIX Serie práctica.". Trad. Maribel Martínez Moyano.
  Pearson Educación, Madrid, 2000, p. 1032, ISBN 84-205-29508

- NIEDERST Jennifer: "HTML Pocket Reference, Second Edition", Ed. 2da.
  O'Reilly & Associates, Inc., United States of America, 2002, p. 104, ISBN 0-596-00296-3

- KENNEDY Bill, MUSCIANO Chuck: "HTML & XHTML: The Definitive Guide, 5th Edition", Ed. 5ta. O'Reilly & Associates, Inc., United States of America, 2002, p. 670, ISBN 0-596-00382-X

- MOHAMMED J. kabir: "La Biblia de Servidor Apache".
  Anaya Multimedia-Anaya Interactiva, Madrid, 1999, p. 688, ISBN 84-415-0807-0

- GIL Rubio, Fco. Javier: "Creación de sitios Web con PHP 4."
  McGraw-Hill, Madrid, 2001, p. 547, ISBN 84-481-3209-2

- LERDOF rasmus, TATROE Kevin: "PROGRAMING PHP",
  O'Reilly & Associates, Inc., United States of America, 2002, P.524, ISBN 1-56592-610-2

- WELLING Luke, THOMSON Laura: "Desarrollo Web con PHP y MySQL"
  Anaya Multimedia-Anaya Interactiva, Madrid, 2003, p. 909, ISBN 84-415-1569-7

- GUTIÉRREZ Juan Diego: "MySQL"
  Anaya Multimedia-Anaya Interactiva, Madrid, 2004, p. 304, ISBN 84-415-1683-9

- MASLAKOWSKI Mark, "Aprendiendo MySQL en 21 días"
  Pearson Educación, México, 2001, p. 534, ISBN 970-26-0036-7

- ZAWODNY, Jeremy D.: "MySQL avanzado"
  Anaya Multimedia-Anaya Interactiva, Madrid, 2004, p. 336 , ISBN 84-415-1759-2

- BARRETT Daniel J., BYRNES Robert G., SILVERMAN Richard: "Linux Security Cookbook", O'Reilly & Associates, Inc., United States of America, 2003, p. 332, ISBN 0-596-00391-9

- CLARKE, Justin: "Network Security Tools"
  O'Reilly & Associates, Inc., United States of America, 2005, p. 352, ISBN 0-596-00794-9

- SHEMA, Mike: "Claves Hackers de SItios Web"
  McGraw-Hill, México, 2005, p. ISBN 0-07-222784-2

- [HTTP Server] Apache Web Server
  http://httpd.apache.org/

- [PHP] PHP Net
  http://www.php.net/

- [MySQL] Mysql Developer Zone
  http://www.mysql.org/

- [LINUX] Linux Online
  http://www.linux.org/

- [LC] Características de Linux
  http://www.fismat.umich.mx/~elizalde/curso/node156.html

- [DTIC-SSL]Departamento de Tratamiento de la Información y Codificación
  http://www.iec.csic.es/criptonomicon/ssl.html

- [3W] The World Wide Web: Past, Present and Future
  http://www.w3.org/People/Berners-Lee/1996/ppf.html

- [HTML VER] SELFHTML en español - Versiones HTML
  http://es.selfhtml.org/selfhtml7/tbaf.htm

- [HTML REF] Lista de las principales etiquetas HTML
  http://www.ccim.be/ccim328/htmlsp/REF.htm

- [WEB DESIGN] W3 Schools. Web Developer
  http://www.w3schools.com/

- [OS] The Open Source
  http://www.opensource.org/osd.html

- [3C] Community Created Content
  http://turre.com/images/stories/books/webkirja_koko_optimoitu2.pdf

- [The Rise] THE RISE OFOPEN SOURCE LICENSING
  http://pub.turre.com/openbook_valimaki.pdf

- [PFOSS]Perspectives on Free and Open Source Software
  http://mitpress.mit.edu/books/chapters/0262562278.pdf

- [Proyecto GNU] El proyecto GNU y el software libre
  http://biblioweb.sindominio.net/pensamiento/softlibre/

- [Free Software] Software libre para una sociedad libre
  http://download.savannah.gnu.org/releases/rms-essays/free_software.es.pdf

- [Libro Blanco]Libro Blanco del Software Libre en España (III)
  http://libroblanco.com/joomla/document/III_libro_blanco_del_software_libre.pdf

- [OSO] Open Source Origins
  http://www.charlesriver.com/resrcs/chapters/1584503475_1stChap.pdf

- [CRIPTO] Introducción a la Criptografía
  http://pateame.fciencias.unam.mx/cripto/notas/cripto.pdf

- [GPG] THE GNU PRIVACY GUARD
  http://www.gnupg.org/

- [WIKIPEDIA] WIKIPEDIA Definición de GPG
  http://es.wikipedia.org/wiki/GPG